Exercises on *The Art Of Computer Programming*

Juan Marín Noguera

January 6, 2025

Contents

Notes on the Exercises

▸ **1.** [*00*] What does the rating "*M20*" mean?
A mathematically oriented, medium difficulty exercise.

2. [*10*] Of what value can the exercises in a textbook be to the reader?
It helps to internalize contents by applying them.

Chapter 1

Basic Concepts

1.1 Algorithms

1. [*10*] The text showed how to interchange the values of variables m and n, using the replacement notation, by setting $t \leftarrow m$, $m \leftarrow n$, $n \leftarrow t$. Show how the values of the *four* variables (a, b, c, d) can be rearranged to (b, c, d, a) by a sequence of replacements. In other words, the new value of a is to be the original value of b, etc. Try to use the minimum number of replacements.

$t \leftarrow a, a \leftarrow b, b \leftarrow c, c \leftarrow d, d \leftarrow t.$

▸ **5.** [*12*] Show that the "Procedure for Reading This Set of Books" that appears in the preface actually fails to be a genuine algorithm on at least three of our five counts! Also mention some differences in format between it and Algorithm E.

It fails on finiteness (it doesn't have an ending condition), definiteness (the instructions are not unambiguous), and effectiveness (solving the Fermat's theorem on the exercises is not "sufficiently basic that it can be done exactly and in a finite length of time"), and it may have no output (although, this time, this is the output). It also doesn't have a header, an ending mark, or a letter prefix on the steps.

▸ **7.** [*HM21*] Let U_m be the average number of times that step E1 is executed in Algorithm E, if m is known and n is allowed to range over all positive integers. Show that U_m is well defined. Is U_m in any way related to T_m?

On average, it will be the case that $n > m$, so after one step, the two numbers will exchange positions and, after two steps, we'll have the same value for m but n will be the remainder of the original values of n and m, which is uniformly distributed on the integers between 0 and $m - 1$. This means U_m is two plus the average number of times that E1 is executed if m is known and $0 \leq n < m$.

Further, since after one step, the two numbers interchange, it is clear that $U_m = 1 + T_m$.

▸ **9.** [*M30*] Suppose that $C_1 = (Q_1, I_1, \Omega_1, f_1)$ and $C_2 = (Q_2, I_2, \Omega_2, f_2)$ are computational methods. For example, C_1 might stand for Algorithm E as in Eqs. (2), except that m and n are restricted in magnitude, and C_2 might stand for a computer program implementation of Algorithm E. (This Q_2 might be the set of all states of the machine, i.e., all possible configurations of its memory and registers; f_2 might be the definition of single machine actions; and I_2 might be the set of initial states, each including the program that determines the greatest common divisor as well as the particular values of m and n.)

Formulate a set-theoretic definition for the concept "C_2 is a representation of C_1" or "C_2 simulates C_1". This is to mean intuitively that any computation sequence of C_1 is mimicked by C_2, except that C_2 might take more steps in which to do the computation and it might retain more information in its states. (We thereby obtain a rigorous interpretation of the statement, "Program X is an implementation of Algorithm Y.")

We could state that C_2 is a representation of C_1 if there are functions $h : Q_2 \to Q_1$, $g : I_1 \to I_2$, and $j : Q_2 \to \mathbb{N}^*$ such that $h \circ g = \mathrm{Id}_{I_1}$, $\Omega_2 = h^{-1}(\Omega_1)$, and for all $q \in Q_2$, $f_1(h(q)) = h(f_2^{j(q)}(q))$.

1.2 Mathematical Preliminaries

1.2.1 Mathematical Induction

1. [*05*] Explain how to modify the idea of proof by mathematical induction, in case we want to prove some statement $P(n)$ for all *nonnegative* integers—that is, for $n = 0, 1, 2, \ldots$ instead of $n = 1, 2, 3, \ldots$.

We'd prove that $P(0)$ holds and that, for all $n \geq 0$, if $P(0), \ldots, P(n)$ hold, then $P(n+1)$ holds.

▸ **2.** [*15*] There must be something wrong with the following proof. What is it?

"**Theorem:** Let a be any positive number. For all positive integers n, we have $a^{n-1} = 1$. *Proof.* If $n = 1$, $a^{n-1} = a^{1-1} = 0$. And by induction, assuming that the theorem is true for $1, 2, \ldots, n$, we have

$$a^{(n+1)-1} = a^n = \frac{a^{n-1} \times a^{n-1}}{a^{(n-1)-1}} = \frac{1 \times 1}{1} = 1;$$

so the theorem is true for $n + 1$ as well."

For $n = 1$, $a^{(n-1)-1} = a^{0-1}$, but we haven't proved the theorem for $n = 0$, and indeed, $a^{0-1} = a^{-1} = \frac{1}{a} \neq 1$ if $a \neq 1$.

▸ **8.** [*25*]

1. Prove the following theorem of Nicomachus (A.D. c. 100) by induction: $1^3 = 1$, $2^3 = 3 + 5$, $3^3 = 7 + 9 + 11$, $4^3 = 13 + 15 + 17 + 19$, etc.

2. Use this result to prove the remarkable formula $1^3 + 2^3 + \cdots + n^3 = (1 + 2 + \cdots + n)^2$.

1. For $n = 1$, $1^3 = 1$, and the first number of the sequence is $n(n-1)+1 = 1(1-1)+1 = 1$. If these two properties are proved up to a certain number n, then the first number of the sequence for $n+1$ is 2 plus the last number of the sequence for n, that is, since the sequence for n has n numbers, the first number of the sequence for $n+1$ is $n(n-1)+1+2n = n(n-1+2)+1 = (n+1)n+1 = (n+1)((n-1)+1)+1$. Now, $(n+1)^3 = n^3+3n^2+3n+1 = (n(n-1)+1)+(n(n-1)+3)+\cdots+(n(n-1)+2n-1)+2n\cdot n+n^2+3n+1 = (n(n-1)+2n+1)+\cdots+(n(n-1)+2n+2n-1)+n^2+n+2n+1 = (n(n+1)+1)+\cdots+(n(n+1)+2n-1)+(n(n+1)+2n+1)$.

2. $1^3+2^3+\cdots+n^3 = 1+3+\cdots+((n+1)n+1-2) = 1+3+\cdots+((n+1)n-1)$. For each n, $n^2-(n-1)^2 = n^2-n^2+2n-1 = 2n-1$, so $1^3+2^3+\cdots+n^3 = 1+3+\cdots+((n+1)n-1) = (1^2-0^2)+(2^2-1^2)+\cdots+\left(\left(\frac{(n+1)n}{2}\right)^2-\left(\frac{(n+1)n}{2}-1\right)^2\right) = \left(\frac{(n+1)n}{2}\right)^2 = (1+2+\cdots+n)^2$.

13. [*M23*] Extend Algorithm E by adding a new variable T and adding the operation "$T \leftarrow T+1$" at the beginning of each step. (Thus, T is like a clock, counting the number of steps executed.) Assume that T is initially zero, so that assertion $A1$ in Fig. 4 becomes "$m > 0, n > 0, T = 0$." The additional condition "$T = 1$" should similarly be appended to $A2$. Show how to append additional conditions to the assertions in such a way that any one of $A1, A2, \ldots, A6$ implies $T \le 3n$, and such that the inductive proof can still be carried out. (Hence the computation must terminate in at most $3n$ steps.)

At $A3$, we append $T \le 3(n-r)-1$. From $A2$, we have $T = 2$, and since $c = m > 0$ and $d = n > 0$, we have $r < n$ and thus $3(n-r)-1 \ge 3\cdot 1-1 = 2$. We have yet to prove this when coming from $A6$.

At $A4$, we have $T \le 3n$, because, coming from $A3$, $T \le 3(n-r)-1+1 = 3n$. At $A5$, we have $T \le 3(n-r)$. At $A6$, we have $T \le 3(n-d)+1$.

Now, at $A3$, when coming from $A6$, we have $T \le 3(n-d)+2$, but $r < d$, so $n-d > n-r$ and $T \le 3(n-d)+2 \le 3(n-r)-3+2 = 3(n-r)-1$.

▸ **15.** [*HM28*] (*Generalized induction.*) The text shows how to prove statements $P(n)$ that depend on a single integer n, but it does not describe how to prove statements $P(m,n)$ depending on two integers. In these circumstances a proof is often given by some sort of "double induction," which frequently seems confusing. Actually, there is an important principle more general than simple induction that applies not only in this case but also to situations in which statements are to be proved about uncountable sets—for example, $P(x)$ for all real x. This general principle is called *well-ordering.*

Let "$\prec$" be a relation on a set S, satisfying the following properties:

1. Given x, y, and z in S, if $x \prec y$ and $y \prec z$, then $x \prec z$.

2. Given x and y in S, exactly one of the following three possibilities is true: $x \prec y$, $x = y$, or $y \prec x$.

3. If A is any nonempty subset of S, there is an element x in A with $x \preceq y$ (that is, $x \prec y$ or $x = y$) for all y in A.

This relation is said to be a well-ordering of S. For example, it is clear that the positive integers are well-ordered by the ordinary "less than" relation, $<$.

1. Show that the set of *all* integers is not well-ordered by $<$.
2. Define a well-ordering relation on the set of all integers.
3. Is the set of all nonnegative real numbers well-ordered by $<$?
4. (*Lexicographic order.*) Let S be well-ordered by $\prec$, and for $n > 0$, let T_n be the set of n-tuples $(x_1, x_2, \ldots, x_n)$ of elements x_j in S. Define $(x_1, x_2, \ldots, x_n) \prec (y_1, y_2, \ldots, y_n)$ if there is some k, $1 \le k \le n$, such that $x_j = y_j$ for $1 \le j < k$, but $x_k \prec y_k$ in S. Is $\prec$ a well-ordering of T_n?
5. Continuing part 4, let $T = \bigcup_{n \ge 1} T_n$; define $(x_1, \ldots, x_m) \prec (y_1, \ldots, y_n)$ if $x_j = y_j$ for $1 \le j < k$ and $x_k \prec y_k$, for some $k \le \min(m, n)$, or if $m < n$ and $x_j = y_j$ for $1 \le j \le m$. Is $\prec$ a well-ordering of T_n?
6. Show that $\prec$ is a well-ordering of S if and only if it satisfies (i) and (ii) above and there is no infinite sequence $x_1, x_2, x_3, \ldots$ with $x_{j+1} \prec x_j$ for all $j \ge 1$.
7. Let S be well-ordered by $\prec$, and let $P(x)$ be a statement about the element x of S. Show that if $P(x)$ can be proved under the assumption that $P(y)$ is true for all $y \prec x$, then $P(x)$ is true for *all* x in S.

1. $\mathbb{Z}$ is a non-empty subset of $\mathbb{Z}$ and it doesn't have a first element under $<$.
2. Let $a, b \in \mathbb{Z}$, we could say that $a \prec b$ if and only if $|a| < |b|$, or $a = -b < 0$, so the order would be $0, 1, -1, 2, -2, \ldots$. Now:
   (a) If $a \prec b$ and $b \prec c$, then either $a = -b < 0$ so it must be $|a| = |b| \prec |c|$ because it can't be $b = -c < 0$ as $b > 0$, or $|a| < |b|$ so $|a| < |b| \le |c|$. In both cases, $a \prec c$.
   (b) If $x \ne y$, then either $|x| = |y|$ so it must be $x \prec y$ if $y < 0$ or $y \prec x$ if $y > 0$, $|x| < |y|$ so that $x \prec y$, or $|y| > |x|$ so that $y \prec x$. It's clear that $x \prec y$ implies $x \ne y$ and $y \not\prec x$.
   (c) Let $S \subseteq \mathbb{Z}$ be nonempty, then we can find the minimum element of S by looking at the elements of S with minimal absolute value, which form a finite, nonempty subset.
3. No, as the nonempty subset $\{x \in \mathbb{R} : x > 0\}$ doesn't have a first element: for each $x > 0$, $\frac{x}{2} < x$.
4. Yes:
   (a) If $(a_1, \ldots, a_n) \prec (b_1, \ldots, b_n) \prec (c_1, \ldots, c_n)$, let k_1 be the least integer with $a_{k_1} \prec b_{k_1}$, k_2 the least integer with $b_{k_2} \prec c_{k_2}$, and $q := \min\{k_1, k_2\}$, then $a_i = b_i = c_i$ for $0 \le i < q$ and $a_q \prec c_q$, so $(a_1, \ldots, a_n) \prec (c_1, \ldots, c_n)$.

(b) If $(a_1, \dots, a_n) \neq (b_1, \dots, b_n)$, let k be the least integer with $a_k \neq b_k$, then either $a_k \prec b_k$ and so $(a_1, \dots, a_n) \prec (b_1, \dots, b_n)$, or $b_k \prec a_k$ and so $(b_1, \dots, b_n) \prec (a_1, \dots, a_n)$, and we can see that no two of the three possibilities can happen at once.

(c) Let $A \subseteq S^n$ be nonempty. Let A_1 be the set of $a \in A$ with minimal a_1, A_2 the set of $a \in A_1$ with minimal a_2, etc. Then the elements of A_1 are lower than any *other* element in A, the ones in A_2 are lower than any other elements in A_1 and thus in A, etc. But then A_n contains only one element, which is the least element of A.

5. No: if S contains two elements $a \prec b$, then $\{(b), (a, b), (a, a, b), \dots\}$ doesn't have a first element.

6. $\Longrightarrow$] If there were such an infinite sequence $(x_n)_{n \geq 1}$, then the set $\{x_n\}_{n \geq 1}$ wouldn't have a first element.#

$\Longleftarrow$] Let A be a nonempty subset of S, and let $x_1 \in A$. Then we take $x_2 \in A$ with $x_2 \prec x_1$, then $x_3 \in A$ with $x_3 \prec x_2$, etc., and since there's no infinite sequence of this kind in S, then there must be an x_n in the sequence with no candidate for x_{n+1}, meaning that, for all $x \in A$, $x_n \preceq x$. Therefore, x_n is the least element of A.

7. Let P be such a property that $\forall x \in S, (\forall y \prec x, P(y) \implies P(x))$. Assume there's an $x_0 \in S$ such that $P(x_0)$ doesn't hold. This means there's an $x_1 \prec x_0$ such that $P(x_1)$ doesn't hold, which means there's an $x_2 \prec x_1$ such that $P(x_2)$ doesn't hold, etc. Then $(x_n)_n$ is an infinite sequence of elements in S with $x_{j+1} \prec x_j$ for all $j \in \mathbb{N}$, but S is well-ordered by $\prec$ #.

1.2.2 Numbers, Powers, and Logarithms

1. [*00*] What is the smallest positive rational number?

There isn't.

2. [*00*] Is $1 + 0.239999999\dots$ a decimal expansion?

Yes, as long as it means a number x with $1.239999999 \leq x < 1.24$, that is, as long as it doesn't end with an infinite number of nines.

3. [*02*] What is $(-3)^{-3}$?

$(-3)^{-3} = \frac{1}{(-3)^3} = \frac{1}{-27} = -\frac{1}{27}$.

▸ **4.** [*05*] What is $(0.125)^{-2/3}$?

$(0.125)^{-2/3} = (\frac{1}{8})^{-2/3} = \sqrt[3]{(\frac{1}{8})^{-2}} = \sqrt[3]{8^2} = \sqrt[3]{64} = 4$.

5. [*05*] We defined real numbers in terms of a decimal expansion. Discuss how we could have defined them in terms of a binary expansion instead, and give a definition to replace Eq. (2).

A decimal expansion would have the form $x = n + 0.b_1b_2b_3\dots$, where each b_i would be a binary digit, 0 or 1, and this would mean that

$$n + \frac{b_1}{2} + \frac{b_2}{4} + \cdots + \frac{b_k}{2^k} \leq x < n + \frac{b_1}{2} + \frac{b_2}{4} + \cdots + \frac{b_k}{2^k} + \frac{1}{10^k}.$$

6. [*10*] Let $x = m + 0.d_1d_2\dots$ and $y = n + 0.e_1e_2\dots$ be real numbers, Give a rule for determining whether $x = y$, $x < y$, or $x > y$, based on the decimal representation.

We say that $x = y$ if $m = n$ and $d_k = e_k$ for all k, and that $x < y$ if $m < n$ or if $m = n$ and there's a k such that $d_j = e_j$ for $1 \leq j < k$ and $d_k < e_k$.

▸ **11.** [*10*] If $b = 10$ and $x \approx \log_{10} 2$, to how many decimal places of accuracy will we need to know the value of x in order to determine the first three decimal places of the decimal expansion of b^x?

Potentially infinite. Because $10^{\log_{10} 2} = 2$, $10^x = 2.0\dots$ if $x \geq \log_{10} 2$ or $10^x = 1.9\dots$ if $x \leq \log_{10} 2$, but since $\log_{10} 2$ is irrational, its decimal expansion is infinite and, in the worst case that $x = \log_{10} 2$, we can't determine by an algorithm that $x \geq \log_{10} 2$.

12. [*02*] Explain why Eq. (10) follows from Eqs. (8).

Because logarithms are strictly increasing, continuous functions, and we have that $10^{0.30102999} < 2 < 10^{0.30103000}$, we can take logarithms on the expression to get $0.30102999 < \log_{10} 2 < 0.30103000$, and so $\log_{10} 2 = 0.30102999$.

▸ **13.** [*M23*]

1. Given that x is a positive real number and n is a positive integer, prove the inequality $\sqrt[n]{1+x} - 1 \leq x/n$.
2. Use this fact to justify the remarks following (7).

1. Let $f(x) := \sqrt[n]{1+x} - 1 - \frac{x}{n}$, we have to prove that, if n is a positive integer and $x > 0$, then $f(x) \leq 0$. We have

$$f'(x) = \frac{1}{n\sqrt[n]{1+x}} - \frac{1}{n} = \frac{1}{n}\left(\frac{1}{\sqrt[n]{1+x} - 1}\right),$$

   but since $\sqrt[n]{1+x} > 1$ for $x > 0$, we have $f'(x) < 0$ and f is strictly decreasing on $(0, +\infty)$. Since $f(0) = 0$, this proves the result.

2. It's clear that $b^{n+d_1/10+\cdots+d_k/10^k} \leq b^{n+1}$, and then, taken $x = b - 1 > 0$ and $n = 10^k$, we have $\sqrt[10^k]{1+b-1} = b^{1/10^k} \leq (b-1)/10^k$.

15. [*10*] Prove or disprove:

$$\log_b x/y = \log_b x - \log_b y, \qquad \text{if } x, y > 0.$$

Let $\alpha := \log_b x$ and $\beta := \log_b y$, we have

$$\frac{b^\alpha}{b^\beta} = b^\alpha b^{-\beta} = b^{\alpha-\beta},$$

and taking logarithms, we get $\log_b \frac{b^\alpha}{b^\beta} = \log_b \frac{x}{y} = \log_b b^{\alpha-\beta} = \alpha - \beta = \log_b x - \log_b y$.

16. [*00*] How can $\log_{10} x$ be expressed in terms of $\ln x$ and $\ln 10$?
$\log_{10} x = \frac{\log_e x}{\log_e 10} = \frac{\ln x}{\ln 10}$.

▸ **17.** [*05*] What is $\lg 32$? $\log_\pi \pi$? $\ln e$? $\log_b 1$? $\log_b(-1)$?
$\lg 32 = 5$, $\log_\pi \pi = 1$, $\ln e = 1$, $\log_b 1 = 0$. As for $\log_b(-1)$, it would be an x such that $b^x = -1$, which is not a real number most of the time. As a complex number, it would be $\log_b(-1) = \frac{\ln(-1)}{\ln b} = \frac{i\pi}{\ln b}$, since $e^{i\pi} = -1$.

18. [*10*] Prove or disprove: $\log_8 x = \frac{1}{2} \lg x$.
$\log_8 x = \frac{\lg x}{\lg 8}$, but $\lg 8 = 3 \neq 2$, so this is false in the general case (this is only true when $x = 1$).

▸ **19.** [*20*] If n is an integer whose decimal representation is 14 digits long, will the value of n fit in a computer word with a capacity of 47 bits and a sign bit?
The number of possibilities for a number with up to 14 digits is 10^{14}, and the number for 47 bits is $2^{47} \approx 1.3 \cdot 2^{14}$, so it would fit.

20. [*10*] Is there any simple relation between $\log_{10} 2$ and $\log_2 10$?
$\log_{10} 2 = \frac{\log_2 2}{\log_2 10} = \frac{1}{\log_2 10}$, so $\log_{10} 2 \log_2 10 = 1$.

▸ **22.** [*20*] (R. W. Hamming.) Prove that

$$\lg x \approx \ln x + \log_{10} x,$$

with less than 1 % error! (Thus a table of natural logarithms and of common logarithms can be used to get approximate values of binary logarithms as well.)

$$\ln x + \log_{10} x = \frac{\lg x}{\lg e} + \frac{\lg x}{\lg 10} = \lg x \left(\frac{1}{\lg e} + \frac{1}{\lg 10}\right) \cong 0.994 \lg x \approx \lg x,$$

and this gives us about 0.6 % error.

▸ **27.** [*M25*] Consider the method for calculating $\log_{10} x$ discussed in the text. Let x'_k denote the computed approximation to x_k, determined as follows: $x(1-\delta) \le 10^n x'_0 \le x(1+\epsilon)$; and in the determination of x'_k by Eqs. (18), the quantity y_k is used in place of $(x'_{k-1})^2$, where $(x'_{k-1})^2(1-\delta) \le y_k \le (x'_{k-1})^2(1+\epsilon)$ and $1 \le y_k < 100$. Here δ and ϵ are small constants that reflect the upper and lower errors due to rounding or truncation. If $\log' x$ denotes the result of the calculations, show that after k steps we have

$$\log_{10} x + 2\log_{10}(1-\delta) - 1/2^k < \log' x \le \log_{10} x + 2\log_{10}(1+\epsilon).$$

We first prove by induction that

$$x^{2^k}(1-\delta)^{2^{k+1}-1} \le 10^{2^k(n+b_1/2+\cdots+b_k/2^k)} x'_k \le x^{2^k}(1+\epsilon)^{2^{k+1}-1}.$$

For $k=0$, we have $x(1-\delta) \le 10^n x'_0 \le x(1+\epsilon)$, which is given. If this holds up to a certain k, then if $(x'_k)^2 < 10$, we have $b_{k+1}=0$, $(1-\delta)(x'_k)^2 \le x'_{k+1} \le (1+\epsilon)(x'_k)^2$ and

$$\begin{aligned}
x^{2^{k+1}}(1-\delta)^{2^{k+2}-1} &= (x^{2^{k+1}}(1-\delta)^{2^{k+1}-1})^2(1-\delta) \\
&\le (10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x'_k)^2(1-\delta) \\
&\le 10^{2^{k+1}(n+b_1/2+\cdots+b_k/2^k+b_{k+1}/2^{k+1})}x'_{k+1} \\
&\le (10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x'_k)^2(1+\epsilon) \\
&\le (x^{2^k}(1+\epsilon)^{2^{k+1}-1})^2(1+\epsilon) = x^{2^{k+1}}(1+\epsilon)^{2^{k+2}-1}.
\end{aligned}$$

If $(x'_k)^2 \ge 10$, we have $b_{k+1}=1$, $(1-\delta)(x'_k)^2 \le 10x'_{k+1} \le (1+\epsilon)(x'_k)^2$ and

$$\begin{aligned}
x^{2^{k+1}}(1-\delta)^{2^{k+2}-1} &\le (10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x'_k)^2(1-\delta) \\
&\le 10^{2^{k+1}(n+b_1/2+\cdots+b_k/2^k)}10x'_{k+1} \\
&= 10^{2^{k+1}(n+b_1/2+\cdots+b_k/2^k+b_{k+1}/2^{k+1})}x'_{k+1} \\
&\le (10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x'_k)^2(1+\epsilon) \\
&\le x^{2^{k+1}}(1+\epsilon)^{2^{k+2}-1}.
\end{aligned}$$

Then, by taking logarithms on the expression, we get

$$2^k\log x + (2^{k+1}-1)\log(1-\delta) \le 2^k \log' x + \log x'_k \le 2^k \log x + (2^{k+1}-1)\log(1+\epsilon).$$

Finally, subtracting $\log x'_k$ from all sides, using that $-1 < -\log(1-\delta) - \log x'_k = -\log(x'_k(1-\delta))$, because $x'_k(1-\delta) < 10$, using that $-\log(1+\epsilon) - \log x'_k = -\log(x'_k(1+\epsilon)) \le 0$, because $x'_k(1+\epsilon) \ge 1$, and dividing everything by 2^k, we get

$$\log x + 2\log(1-\delta) - \frac{1}{2^k} < \log' x \le \log x + 2\log(1+\epsilon),$$

which is precisely what we wanted to prove.

1.2.3 Sums and products

▸ **1.** [*10*] The text says that $a_1 + a_2 + \cdots + a_0 = 0$. What then, is $a_2 + \cdots + a_0$?

We could set that to be $-a_1$.

2. [*01*] What does the notation $\sum_{1\leq j\leq n} a_j$ mean, if $n = 3.14$?

$a_1 + a_2 + a_3$.

▸ **3.** [*13*] Without using the $\sum$-notation, write out the equivalent of

$$\sum_{0\leq n\leq 5} \frac{1}{2n+1},$$

and also the equivalent of

$$\sum_{0\leq n^2\leq 5} \frac{1}{2n^2+1}.$$

Explain why the two results are different, in spite of rule (b).

$$\begin{aligned}\sum_{0\leq n\leq 5} \frac{1}{2n+1} &= \frac{1}{1}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\frac{1}{9}+\frac{1}{11},\\ \sum_{0\leq n^2\leq 5} \frac{1}{2n^2+1} &= \frac{1}{9}+\frac{1}{3}+\frac{1}{1}+\frac{1}{3}+\frac{1}{9},\end{aligned}$$

as the integers with $0 \leq n^2 \leq 5$ are $\{-2,-1,0,1,2\}$. Rule (b) states that a permutation of the integers satisfying the condition doesn't alter the result, but because $n \mapsto n^2$ is not a permutation of such *integers*, the rule doesn't apply.

4. [*10*] Without using the $\sum$-notation, write out the equivalent of each side of Eq. (10) as a sum of sums for the case $n = 3$.

$a_{11} + a_{21} + a_{22} + a_{31} + a_{32} + a_{33} = a_{11} + a_{21} + a_{31} + a_{22} + a_{32} + a_{33}$.

▸ **5.** [*HM20*] Prove that rule (a) is valid for arbitrary infinite series, provided that the series converge.

Since infinite series are defined for cases where the total number of nonzero elements is at most countable, we might assume that the series are indexed by positive integers. Thus, we'd have to prove that

$$\left(\sum_{i=1}^{\infty} a_i\right)\left(\sum_{j=1}^{\infty} b_j\right) = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} a_i b_j,$$

assuming that the relevant series converge. We have

$$\left(\sum_{i=0}^{\infty} a_i\right)\left(\sum_{j=0}^{\infty} b_j\right) = \sum_{i=0}^{\infty}\left(a_i \sum_{j=0}^{\infty} b_j\right) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} b_j,$$

by entering the second series as a constant into the first series (because it converges) and then entering the element of the first series, considered as a constant, into the inner series.

▸ **9.** [*05*] Is the derivation of Eq. (14) valid even if $n = -1$?

The result is

$$\sum_{0 \le j \le -1} ax^j = 0 = a\left(\frac{1-x^0}{1-x}\right),$$

which is correct. However, the derivation is not correct as the rule (d) cannot be applied, because the applications in the derivation assume $n \ge 0$.

10. [*05*] Is the derivation of Eq. (14) valid even if $n = -2$?

No (see previous exercise).

11. [*03*] What should the right-hand side of Eq. (14) be if $x = 1$?

Mere substitution in the original formula yields an undefined result, but it's clear that

$$\sum_{0 \le j \le n} a \cdot 1^j = \sum_{0 \le j \le n} a = (n+1)a.$$

12. [*10*] What is $1 + \frac{1}{7} + \frac{1}{49} + \frac{1}{343} + \cdots + \left(\frac{1}{7}\right)^n$?

By Eq. (14), this is

$$\sum_{k=0}^{n} \left(\frac{1}{7}\right)^k = \left(\frac{1-\left(\frac{1}{7}\right)^{n+1}}{1-\frac{1}{7}}\right) = \frac{1-\left(\frac{1}{7}\right)^{n+1}}{\frac{6}{7}} = \frac{7}{6}\left(1 - \frac{1}{7^{n+1}}\right).$$

13. [*10*] Using Eq. (15) and assuming that $m \le n$, evaluate $\sum_{j=m}^{n} j$.

$$\sum_{m \le j \le n} j = \sum_{m \le j+m \le n} (j+m) = \sum_{0 \le j \le n-m} (m+j) =$$
$$= m(n-m+1) + \frac{1}{2}(n-m)(n-m+1) = \frac{1}{2}(n(n+1) - m(m-1)).$$

14. [*11*] Using the result of the previous exercise, evaluate $\sum_{j=m}^{n} \sum_{k=r}^{s} jk$.

$$\sum_{j=m}^{n} \sum_{k=r}^{s} jk = \left(\sum_{j=m}^{n} j\right)\left(\sum_{k=r}^{s} k\right) =$$
$$= \frac{1}{4}(n(n+1) - m(m-1))(s(s+1) - r(r-1)).$$

▸ **15.** [*M22*] Compute the sum $1 \times 2 + 2 \times 2^2 + 3 \times 2^3 + \cdots + n \times 2^n$ for small values of n. Do you see the pattern developing in these numbers? If not, discover it by manipulations similar to those leading up to Eq. (14).

Let $\tau(n)$ be such a such sum

$$\tau(1) = 2, \qquad \tau(2) = 2 + 8 = 10, \qquad \tau(3) = 10 + 24 = 34,$$
$$\tau(4) = 34 + 64 = 98, \qquad \tau(5) = 98 + 160 = 258, \qquad \ldots$$

Now,

$$\begin{aligned}
\tau(n) &= \sum_{1 \le k \le n} k2^k && \text{by definition}\\
&= \sum_{0 \le k \le n-1} (k+1)2^{k+1} && \text{by rule (b)}\\
&= 2 \sum_{0 \le k \le n-1} (k+1)2^k && \text{by a special case of (a)}\\
&= 2\left(\sum_{0 \le k \le n-1} k2^k + \sum_{0 \le k \le n-1} 2^k\right) && \text{by rule (c)}\\
&= 2\left(\sum_{0 \le k \le n} k2^k - n2^n + (2^n - 1)\right) && \text{by rule (d) and Eq. (14)}\\
&= 2\left(\sum_{1 \le k \le n} k2^k - n2^n + (2^n - 1)\right) && \text{by rule (d)}
\end{aligned}$$

This means $\tau(n) = 2(\tau(n) - n2^n + 2^n - 1) = 2\tau(n) - n2^{n+1} + 2^{n+1} - 2$, so

$$\tau(n) = n2^{n+1} - 2^{n+1} + 2 = (n-1)2^{n+1} + 2.$$

▸ **17.** [*M00*] Let S be a set of integers. What is $\sum_{j \in S} 1$?

$|S|$.

▸ **20.** [*25*] Dr. I. J. Matrix has observed a remarkable sequence of formulas:

$$9 \times 1 + 2 = 11, \qquad 9 \times 12 + 3 = 111,$$
$$9 \times 123 + 4 = 1111, \qquad 9 \times 1234 + 5 = 11111.$$

1. Write the good doctor's great discovery in terms of the $\sum$-notation.
2. Your answer to part (a) undoubtedly involves the number 10 as base of the decimal system; generalize this formula so that you get a formula that will perhaps work in any base b.
3. Prove your formula from part (b) by using formulas derived in the text or in exercise 16 above.

1. For $n \geq 1$,
$$9 \sum_{j=0}^{n-1} 10^j (n-j) + n + 1 = \sum_{j=0}^{n} 10^j.$$

2.
$$(b-1) \sum_{j=0}^{n-1} b^j (n-j) + n + 1 = \sum_{j=0}^{n} b^j.$$

3. We have
$$\begin{aligned}\sum_{j=0}^{n-1} b^j (n-j) &= n \sum_{j=0}^{n-1} b^j - \sum_{j=0}^{n-1} j b^j \\ &= n\frac{1-b^n}{1-b} - \frac{(n-1)b^{n+1} - nb^n + b}{(b-1)^2} \\ &= n\frac{b^n-1}{b-1} - \frac{(n-1)b^{n+1} - nb^n + b}{(b-1)^2},\end{aligned}$$
so
$$\begin{aligned}&(b-1) \sum_{j=0}^{n-1} b^j (n-j) + n + 1 \\ &= n(b^n-1) - \frac{(n-1)b^{n+1} - nb^n + b}{b-1} + n + 1 \\ &= \frac{n(b^n-1)(b-1) - (n-1)b^{n+1} + nb^n - b + n(b-1) + b - 1}{b-1} \\ &= \frac{nb^{n+1} - nb^n - nb + n - nb^{n+1} + b^{n+1} + nb^n - b + nb - n + b - 1}{b-1} \\ &= \frac{b^{n+1}-1}{b-1} = \frac{1-b^{n+1}}{b-1} = \sum_{j=0}^{n} b^j.\end{aligned}$$

▸ **21.** [*M25*] Derive rule (d) from (8) and (17).

$$\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_j a_j [R(j)] + \sum_j a_j [S(j)] = \sum_j a_j ([R(j)] + [S(j)]) =$$
$$\sum_j a_j ([R(j) \vee S(j)] + [R(j) \wedge S(j)]) = \sum_{R(j) \vee S(j)} a_j + \sum_{R(j) \wedge S(j)} a_j.$$

▸ **22.** [*20*] State the appropriate analogs of Eqs. (5), (7), (8), and (11) for *products* instead of sums.

$$\prod_{R(i)} a_i = \prod_{R(j)} a_j = \prod_{R(p(j))} a_{p(j)}, \qquad \prod_{R(i)} \prod_{S(j)} a_{ij} = \prod_{S(j)} \prod_{R(i)} a_{ij},$$
$$\prod_{R(i)} b_i c_i = \prod_{R(i)} b_i + \prod_{R(i)} c_i, \qquad \prod_{R(j)} a_j \prod_{S(j)} a_j = \prod_{R(j) \vee S(j)} a_j \prod_{R(j) \wedge S(j)} a_j.$$

23. [*10*] Explain why it is a good idea to define $\sum_{R(j)} a_j$ and $\prod_{R(j)} a_j$ as zero and one, respectively, when no integers satisfy $R(j)$.

Zero and one are the neutral terms for sum and product, respectively, so by defining it this way, rule (d) (among others) applies independently of whether there are integers satisfying the properties or not.

▸ **25.** [*15*] Consider the following derivation; is anything amiss?

$$\left(\sum_{i=1}^{n} a_i\right)\left(\sum_{j=1}^{n} \frac{1}{a_j}\right) = \sum_{1\le i\le n}\sum_{1\le j\le n} \frac{a_i}{a_j} = \sum_{1\le i\le n}\sum_{1\le i\le n} \frac{a_i}{a_i} = \sum_{i=1}^{n} 1 = n.$$

First, the change of variable on the second equality is invalid since it changes to a bound variable, not to a free variable. Second, it then converts the double summation into a single summation, which is invalid too.

▸ **29.** [*M30*]

1. Express $\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j} a_i a_j a_k$ in terms of the multiple-sum notation explained at the end of the section.

2. Express the same sum in terms of $\sum_{i=0}^{n} a_i$, $\sum_{i=0}^{n} a_i^2$, and $\sum_{i=0}^{n} a_i^3$ [see Eq. (13)].

1.

$$\sum_{0\le k\le j\le i\le n} a_i a_j a_k.$$

We have

$$\begin{aligned} S := \sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j} a_i a_j a_k &= \sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=j}^{i} a_i a_j a_k \\ = \sum_{i=0}^{n}\sum_{j=i}^{n}\sum_{k=0}^{i} a_i a_j a_k &= \sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=i}^{n} a_i a_j a_k \\ = \sum_{i=0}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} a_i a_j a_k &= \sum_{i=0}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} a_i a_j a_k. \end{aligned}$$

Thus,

$$\begin{aligned}
6S &= \sum_{i=0}^{n}\sum_{j=0}^{i}\left(\sum_{k=0}^{j} a_ia_ja_k + \sum_{k=j}^{i} a_ia_ja_k + \sum_{k=i}^{n} a_ia_ja_k\right) \\
&\quad + \sum_{i=0}^{n}\sum_{j=i}^{n}\left(\sum_{k=0}^{i} a_ia_ja_k + \sum_{k=i}^{j} a_ia_ja_k + \sum_{i=j}^{n} a_ia_ja_k\right) \\
&= \sum_{i=0}^{n}\left(\sum_{j=0}^{i}\left(\sum_{k=0}^{n} a_ia_ja_k + a_ia_j^2 + a_i^2a_j\right) + \sum_{j=i}^{n}\left(\sum_{k=0}^{n} a_ia_ja_k + a_ia_j^2 + a_j^2a_i\right)\right) \\
&= \sum_{i=0}^{n}\left(\sum_{j=0}^{n}\left(\sum_{k=0}^{n} a_ia_ja_k + a_ia_j^2 + a_i^2a_j\right) + \sum_{k=0}^{n} a_i^2a_k + a_i^3 + a_i^3\right) \\
&= \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} a_ia_ja_k + \sum_{i=0}^{n}\sum_{j=0}^{n} a_ia_j^2 + \sum_{i=0}^{n}\sum_{j=0}^{n} a_i^2a_j + \sum_{i=0}^{n}\sum_{k=0}^{n} a_i^2a_k + 2\sum_{i=0}^{n} a_i^3 \\
&= \left(\sum_{i=0}^{n} a_i\right)^3 + \left(\sum_{i=0}^{n} a_i\right)\left(\sum_{i=0}^{n} a_i^2\right) + \left(\sum_{i=0}^{n} a_i^2\right)\left(\sum_{i=0}^{n} a_i\right) \\
&\quad + \left(\sum_{i=0}^{n} a_i^2\right)\left(\sum_{i=0}^{n} a_i\right) + 2\sum_{i=0}^{n} a_i^3 \\
&= \left(\sum_{i=0}^{n} a_i\right)^3 + 3\left(\sum_{i=0}^{n} a_i\right)\left(\sum_{i=0}^{n} a_i^2\right) + 2\left(\sum_{i=0}^{n} a_i^3\right).
\end{aligned}$$

This means

$$S = \frac{1}{6}\left(\sum_{i=0}^{n} a_i\right)^3 + \frac{1}{2}\left(\sum_{i=0}^{n} a_i\right)\left(\sum_{i=0}^{n} a_i^2\right) + \frac{1}{3}\left(\sum_{i=0}^{n} a_i^3\right).$$

30. [*M23*] (J. Binet, 1812.) Without using induction, prove the identity

$$\left(\sum_{j=1}^{n} a_jx_j\right)\left(\sum_{j=1}^{n} b_jy_j\right) = \left(\sum_{j=1}^{n} a_jy_j\right)\left(\sum_{j=1}^{n} b_jx_j\right) + \sum_{1\le j<k\le n} (a_jb_k - a_kb_j)(x_jy_k - x_ky_j).$$

$$\begin{aligned}
\left(\sum_{j=1}^{n} a_jx_j\right)\left(\sum_{j=1}^{n} b_jy_j\right) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_jx_iy_j \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_j(x_jy_i - x_jy_i + x_iy_j) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_jx_jy_i + \sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_j(x_iy_j - x_jy_i) \\
&= \left(\sum_{j=1}^{n} a_jy_j\right)\left(\sum_{j=1}^{n} b_jx_j\right) + \sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_j(x_iy_j - x_jy_i),
\end{aligned}$$

but

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} a_ib_j(x_iy_j - x_jy_i) &= \sum_{1\le j<i\le n} a_ib_j(x_iy_j - x_jy_i) + \sum_{1\le i<j\le n} a_ib_j(x_iy_j - x_jy_i)\\ &= \sum_{1\le i<j\le n} (a_jb_i(x_jy_i - x_iy_j) + a_ib_j(x_iy_j - x_jy_i))\\ &= \sum_{1\le j<k\le n} (a_jb_k - a_kb_j)(x_jy_k - x_ky_j).\end{aligned}$$

▸ **37.** [*M24*] Show that the determinant of Vandermonde's matrix is

$$\prod_{1\le j\le n} x_j \prod_{1\le i<j\le n} (x_j - x_i).$$

We show this by induction. For $n = 1$, this is obvious. For $n > 1$, assuming this holds for $n - 1$, subtracting from each row the previous one multiplied by x_1, expanding by cofactors, factoring out, and applying the induction hypothesis, we have

$$\begin{aligned}\begin{vmatrix} x_1 & x_2 & \cdots & x_n\\ x_1^2 & x_2^2 & \cdots & x_n^2\\ \vdots & \vdots & & \vdots\\ x_1^n & x_2^n & \cdots & x_n^n\end{vmatrix} &= \begin{vmatrix} x_1 & x_2 & \cdots & x_n\\ 0 & x_2(x_2-x_1) & \cdots & x_n(x_n-x_1)\\ \vdots & \vdots & & \vdots\\ 0 & x_2^{n-1}(x_2-x_1) & \cdots & x_n^{n-1}(x_n-x_1)\end{vmatrix}\\ &= x_1\begin{vmatrix} x_2(x_2-x_1) & \cdots & x_n(x_n-x_1)\\ \vdots & & \vdots\\ x_2^{n-1}(x_2-x_1) & \cdots & x_n^{n-1}(x_n-x_1)\end{vmatrix}\\ &= x_1(x_2-x_1)\cdots(x_n-x_1)\begin{vmatrix} x_2 & \cdots & x_n\\ \vdots & & \vdots\\ x_2^{n-1} & \cdots & x_n^{n-1}\end{vmatrix}\\ &= x_1\prod_{2\le j\le n}(x_j-x_1)\prod_{2\le j\le n} x_j \prod_{2\le i<j\le n}(x_j-x_i)\\ &= \prod_{1\le j\le n} x_j \prod_{1\le i<j\le n}(x_j-x_i).\end{aligned}$$

▸ **38.** [*M25*] Show that the determinant of Cauchy's matrix is

$$\prod_{1\le i<j\le n}(x_j-x_i)(y_j-y_i)\Big/\prod_{1\le i,j\le n}(x_i+y_j).$$

We prove it by induction. For $n = 1$, this is obvious. Now assume $n > 1$ and that the hypothesis holds for $n-1$. Multiplying each row i by $\frac{x_1+y_1}{x_i+y_1}$ times the first row, we get

$$\begin{vmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \cdots & \frac{1}{x_1+y_n} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \cdots & \frac{1}{x_2+y_n} \\ \vdots & \vdots & & \vdots \\ \frac{1}{x_n+y_1} & \frac{1}{x_n+y_2} & \cdots & \frac{1}{x_n+y_n} \end{vmatrix} =$$

$$= \begin{vmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \cdots & \frac{1}{x_1+y_n} \\ 0 & \frac{1}{x_2+y_2} - \frac{x_1+y_1}{(x_1+y_2)(x_2+y_1)} & \cdots & \frac{1}{x_2+y_n} - \frac{x_1+y_1}{(x_1+y_n)(x_2+y_1)} \\ \vdots & \vdots & & \vdots \\ 0 & \frac{1}{x_n+y_2} - \frac{x_1+y_1}{(x_1+y_2)(x_n+y_1)} & \cdots & \frac{1}{x_n+y_n} - \frac{x_1+y_1}{(x_1+y_n)(x_n+y_1)} \end{vmatrix} =$$

$$= \frac{1}{x_1+y_1} \begin{vmatrix} \frac{1}{x_2+y_2} - \frac{x_1+y_1}{(x_1+y_2)(x_2+y_1)} & \cdots & \frac{1}{x_2+y_n} - \frac{x_1+y_1}{(x_1+y_n)(x_2+y_1)} \\ \vdots & & \vdots \\ \frac{1}{x_n+y_2} - \frac{x_1+y_1}{(x_1+y_2)(x_n+y_1)} & \cdots & \frac{1}{x_n+y_n} - \frac{x_1+y_1}{(x_1+y_n)(x_n+y_1)} \end{vmatrix}.$$

Now,

$$\frac{1}{x_i+y_j} - \frac{x_1+y_1}{(x_1+y_j)(x_i+y_1)} = \frac{(x_1+y_j)(x_i+y_1)-(x_1+y_1)(x_i+y_j)}{(x_i+y_j)(x_1+y_j)(x_i+y_1)} =$$
$$= \frac{1}{x_i+y_j}\frac{x_1y_1+x_iy_j-x_1y_j-x_iy_1}{(x_1+y_j)(x_i+y_1)} = \frac{1}{x_i+y_j}\frac{(x_i-x_1)(y_j-y_1)}{(x_1+y_j)(x_i+y_1)},$$

so

$$\begin{vmatrix} \frac{1}{x_1+y_1} & \frac{1}{x_1+y_2} & \cdots & \frac{1}{x_1+y_n} \\ \frac{1}{x_2+y_1} & \frac{1}{x_2+y_2} & \cdots & \frac{1}{x_2+y_n} \\ \vdots & \vdots & & \vdots \\ \frac{1}{x_n+y_1} & \frac{1}{x_n+y_2} & \cdots & \frac{1}{x_n+y_n} \end{vmatrix} =$$

$$= \frac{1}{x_1+y_1} \begin{vmatrix} \frac{1}{x_2+y_2}\frac{(x_2-x_1)(y_2-y_1)}{(x_1+y_2)(x_2+y_1)} & \cdots & \frac{1}{x_2+y_n}\frac{(x_2-x_1)(y_n-y_1)}{(x_1+y_n)(x_2+y_1)} \\ \vdots & & \vdots \\ \frac{1}{x_n+y_2}\frac{(x_n-x_1)(y_2-y_1)}{(x_1+y_2)(x_n+y_1)} & \cdots & \frac{1}{x_n+y_n}\frac{(x_n-x_1)(y_n-y_1)}{(x_1+y_n)(x_n+y_1)} \end{vmatrix} =$$

$$= \frac{1}{x_1+y_1}\prod_{i=2}^{n}\frac{x_i-x_1}{x_i+y_1}\prod_{j=2}^{n}\frac{y_j-y_1}{x_1+y_j} \begin{vmatrix} \frac{1}{x_2+y_2} & \cdots & \frac{1}{x_2+y_n} \\ \vdots & & \vdots \\ \frac{1}{x_n+y_2} & \cdots & \frac{1}{x_n+y_n} \end{vmatrix} =$$

$$= \frac{\prod_{j=2}^{n}(x_j-x_1)(y_j-y_1)}{(x_1+y_1)\prod_{i=2}^{n}(x_j+y_1)\prod_{j=2}^{n}(x_1+y_j)} \frac{\prod_{2\le i<j\le n}(x_j-x_i)(y_j-y_i)}{\prod_{2\le i,j\le n}(x_i+y_j)} =$$
$$= \prod_{1\le i<j\le n}(x_j-x_i)(y_j-y_i) \Big/ \prod_{1\le i,j\le n}(x_i+y_j).$$

▸ **45.** [*M25*] A *Hilbert matrix*, sometimes called an $n\times n$ segment of *the* (infinite) Hilbert matrix, is a matrix for which $a_{ij} = 1/(i+j-1)$. Show that this is a special case of Cauchy's matrix, find its inverse, show that each element of the inverse is an integer and show that the sum of all elements of the inverse is n^2. The solution to this problem requires an elementary knowledge of factorial and binomial coefficients, which are discussed in sections 1.2.5 and 1.2.6.

Clearly this is the Cauchy's matrix $a_{ij} = 1/(x_i + y_j)$ where $x_i = i$ and $y_j = j - 1$. Then the elements of the inverse, by exercise 41, are given by

$$\begin{aligned}
b_{ij} &= \left(\prod_{1\le k\le n}(x_j+y_k)(x_k+y_i)\right) \Bigg/ (x_j+y_i)\left(\prod_{\substack{1\le k\le n\\ k\ne j}}(x_j-x_k)\right)\left(\prod_{\substack{1\le k\le n\\ k\ne i}}(y_i-y_k)\right)\\
&= \left(\prod_k (j+k-1)(i+k-1)\right) \Bigg/ (i+j-1)\left(\prod_{k\ne j}(j-k)\right)\left(\prod_{k\ne i}(i-k)\right)\\
&= \frac{(j+n-1)!(i+n-1)!}{(j-1)!(i-1)!} \Bigg/ (i+j-1)(-1)^{i+j}(n-j)!(j-1)!(n-i)!(i-1)!\\
&= (-1)^{i+j}\frac{(j+n-1)!(i+n-1)!}{(i+j-1)(n-j)!(n-i)!(j-1)!^2(i-1)!^2}.
\end{aligned}$$

We have used that

$$\begin{aligned}
\prod_{k\ne j}(j-k) = (-1)^n\prod_{k\ne j}(k-j) = (-1)^n\prod_{j<k\le n}(k-j)\prod_{1\le k<j}(k-j) =&\\
= (-1)^{n+j}(n-j)!(j-1)!,&
\end{aligned}$$

and the same happens to i. Then,

$$\begin{aligned}
b_{ij} &= \frac{(-1)^{i+j}ij}{i+j-1}\,\frac{(i+n-1)!(j+n-1)!}{i!(i-1)!(n-i)!j!(j-1)!(n-j)!}\\
&= \frac{(-1)^{i+j}ij}{i+j-1}\binom{n}{i}\frac{(i+n-1)!}{n!(i-1)!}\binom{n}{j}\frac{(j+n-1)!}{n!}\\
&= \frac{(-1)^{i+j}ij}{i+j-1}\binom{n}{i}\binom{i+n-1}{n}\binom{n}{j}\binom{j+n-1}{n}\\
&= \frac{(-1)^{i+j}ij}{i+j-1}\binom{n}{i}\binom{-i}{n}\binom{n}{j}\binom{-j}{n}\\
&= \frac{(-1)^{i+j}ij}{i+j-1}\binom{-j}{i}\binom{-j-i}{n-i}\binom{-i}{n}\binom{n}{j}\\
&= \frac{(-1)^{i+j}ij}{i+j-1}\binom{i+j-1}{i}\binom{n+j-1}{n-i}\binom{n+i-1}{n}\binom{n}{j}\\
&= (-1)^{i+j}\binom{i+j-2}{i-1}\binom{i+n-1}{i-1}\binom{j+n-1}{n-1}\binom{n}{j} \in \mathbb{Z}.
\end{aligned}$$

Finally, from exercise 44,

$$\sum_{i,j} b_{ij} = \sum_{i=1}^{n} i + \sum_{j=1}^{n}(j-1) = \frac{n(n+1)}{2} + \frac{n(n-1)}{2} = n^2.$$

1.2.4 Integer Functions and Elementary Number Theory

1. [*00*] What are $\lfloor 1.1\rfloor$, $\lfloor -1.1\rfloor$, $\lceil -1.1\rceil$, $\lfloor 0.99999\rfloor$, and $\lfloor \lg 35\rfloor$?
$\lfloor 1.1\rfloor = 1$, $\lfloor -1.1\rfloor = -2$, $\lceil -1.1\rceil = -1$, $\lfloor 0.99999\rfloor = 0$, $\lfloor \lg 35\rfloor = 5$.

▸ **2.** [*01*] What is $\lceil\lfloor x\rfloor\rceil$?
The same as $\lfloor x\rfloor$.

3. [*10*] Let n be an integer, and let x be a real number. Prove that

1. $\lfloor x\rfloor < n$ if and only if $x < n$;
2. $n \le \lfloor x\rfloor$ if and only if $n \le x$;
3. $\lceil x\rceil \le n$ if and only if $x \le n$;
4. $n < \lceil x\rceil$ if and only if $n < x$;
5. $\lfloor x\rfloor = n$ if and only if $x - 1 < n \le x$, and if and only if $n \le x < n + 1$.
6. $\lceil x\rceil = n$ if and only if $x \le n < x + 1$, and if and only if $n - 1 < x \le n$.

1. $\Longrightarrow$] By contraposition, if $x \ge n$, then n is an integer less than or equal to x and so $\lfloor x\rfloor \ge n$.
   $\Longleftarrow$] $\lfloor x\rfloor \le x < n$.
2. It's the contrapositive of the previous statement.
3. Derived from the previous statement replacing x and n with $-x$ and $-n$ and using that $\lfloor -x\rfloor = -\lceil x\rceil$.
4. It's the contrapositive of the previous statement.
5. 1 $\Longrightarrow$ 2] Obviously $n = \lfloor x\rfloor \le x$, and if $x - 1 \ge n$ then it would be $x \ge n+1 \in \mathbb{Z}$ and $\lfloor x\rfloor \ge n + 1 > n\#$.
   2 $\Longrightarrow$ 3] Multiply the inequality by -1 and add $x + n$ to each member.
   3 $\Longrightarrow$ 1] $n \le x$ and any $m \in \mathbb{Z}$ with $m \le x$ has $m < n + 1$ and therefore $m \le n$, so $n = \max\{m \in \mathbb{Z} \mid m \le x\} = \lfloor x\rfloor$.
6. Derived from the previous statement by replacing x and n with $-x$ and $-n$ and using that $-\lceil x\rceil = \lfloor -x\rfloor$.

▸ **4.** [*M10*] Using the previous exercise, prove that $\lfloor -x\rfloor = -\lceil x\rceil$.
Since that would be circular reasoning (I did the previous exercise before reading the statement of this one), we'll prove it directly.

$$\begin{aligned}\lfloor -x\rfloor = n &\iff \forall m \in \mathbb{Z}, (m \le -x \implies m \le n)\\ &\iff \forall m \in \mathbb{Z}, (-m \ge x \implies -m \ge -n)\\ &\iff \forall m \in \mathbb{Z}, (m \ge x \implies m \ge -n) \iff -n = \lceil x\rceil \iff n = -\lceil x\rceil.\end{aligned}$$

▸ **6.** [*20*] Which of the following equations are true for all positive real numbers x?

1. $\left\lfloor\sqrt{\lfloor x\rfloor}\right\rfloor = \lfloor\sqrt{x}\rfloor$;
2. $\left\lceil\sqrt{\lceil x\rceil}\right\rceil = \lceil\sqrt{x}\rceil$;
3. $\left\lceil\sqrt{\lfloor x\rfloor}\right\rceil = \lceil\sqrt{x}\rceil$.

1. True, if $n = \lfloor\sqrt{x}\rfloor$, then $n \le \sqrt{x} < n+1$, so $n^2 \le \lfloor x\rfloor \le x < (n+1)^2$ and so $n \le \sqrt{\lfloor x\rfloor} < n+1$ and $n = \left\lfloor\sqrt{\lfloor x\rfloor}\right\rfloor$.
2. True, if $n = \lceil\sqrt{x}\rceil$, then $n-1 < \sqrt{x} \le n$ and $(n-1)^2 < x \le \lceil x\rceil \le n$, so $n-1 < \sqrt{\lceil x\rceil} \le n$ and therefore $\left\lceil\sqrt{\lceil x\rceil}\right\rceil = n$.
3. False, for example $\left\lceil\sqrt{\lfloor\frac{9}{2}\rfloor}\right\rceil = \lceil\sqrt{4}\rceil = 2$ but $\lceil\sqrt{\frac{9}{2}}\rceil = 3$.

8. [*00*] What are 100 mod 3, 100 mod 7, −100 mod 7, −100 mod 0?
100 mod 3 = 1, 100 mod 7 = 2, −100 mod 7 = −2 mod 7 = 5, −100 mod 0 = −100.

9. [*05*] What are 5 mod −3, 18 mod −3, −2 mod −3?
$5 \bmod -3 = 5 - (-3)\lfloor\frac{5}{-3}\rfloor = 5 + 3(-2) = 5 - 6 = -1$, $18 \bmod -3 = 18 - (-3)(-6) = 0$, $-2 \bmod -3 = -2 - (-3)\lfloor\frac{-2}{-3}\rfloor = -2$.

▸ **10.** [*10*] What are 1.1 mod 1, 0.11 mod .1, 0.11 mod −.1?

$$\begin{aligned} 1.1 \bmod 1 &= 1.1 - \lfloor 1.1\rfloor = .1, \\ 0.11 \bmod .1 &= .11 - .1\lfloor 1.1\rfloor = .01, \\ 0.11 \bmod -.1 &= .11 - (-.1)\lfloor -1.1\rfloor = -.09. \end{aligned}$$

11. [*00*] What does "$x \equiv y \bmod 0$" mean by our conventions?
It means $x = x \bmod 0 = y \bmod 0 = y$.

12. [*00*] What integers are relatively prime to 1?
All of them, since $\gcd\{a, 1\} = 1$ for any $a \in \mathbb{Z}$.

13. [*M00*] By convention, we say that the greatest common divisor of 0 and n is $|n|$. What integers are relatively prime to 0?
Only 1 and −1.

▸ **14.** [*12*] If $x \bmod 3 = 2$ and $x \bmod 5 = 3$, what is $x \bmod 15$?

This is a simple Diophantine equation, there exist $n, m \in \mathbb{Z}$ such that

$$x = 3n + 2 = 5m + 3,$$

so $3n - 5m = 1$ and one solution is $n = 2$ and $m = 1$. In this case $x \bmod 15 = (3n + 2) \bmod 15 = 8$, and the Chinese remainder theorem tells us that this is the only possibility.

15. [*10*] Prove that $z(x \bmod y) = (zx) \bmod (zy)$.

If $z = 0$, then $z(x \bmod y) = 0 = 0 \bmod 0 = (zx) \bmod (zy)$. If $y = 0$, $z(x \bmod y) = zx = (zx) \bmod (zy)$. If $y, z \neq 0$,

$$(zx) \bmod (zy) = zx - zy \left\lfloor \frac{zx}{zy} \right\rfloor = zx - zy \left\lfloor \frac{x}{y} \right\rfloor = z \left(x - y \left\lfloor \frac{x}{y} \right\rfloor \right) = z(x \bmod y).$$

16. [*M10*] Assume that $y > 0$. Show that if $(x - z)/y$ is an integer and if $0 \leq z < y$, then $z = x \bmod y$.

In this case, there exists an integer k such that $x - z = ky$, so $z = x - ky$ and we just have to show that $k = \lfloor \frac{x}{y} \rfloor$. But since $0 \leq z < y$, $0 \leq \frac{z}{y} < 1$ and, multiplying this formula by -1 and adding $\frac{x}{y}$, $\frac{x}{y} - 1 < \frac{x}{y} - \frac{z}{y} = k \leq \frac{x}{y}$, which means that $k = \lfloor \frac{x}{y} \rfloor$ by Exercise 3.

▸ **19.** [*M10*] (*Law of inverses.*) If $n \perp m$, there is an integer n' such that $nn' \equiv 1$ (modulo m). Prove this, using the extension of Euclid's algorithm (Algorithm 1.2.1E).

We already know that the algorithm terminates and that returns $a, b \in \mathbb{Z}$ such that $am + bn = d \coloneqq \gcd\{n, m\}$. But in this case $d = 1$, so $bn = d - am = 1 - am = 1$ (modulo m).

▸ **22.** [*M10*] Give an example to show that Law B is not always true if a is not relatively prime to m.

Let $a = b = 3$, $x = 2$, $y = 4$, and $m = 6$ in Law B, then $ax = 6 \equiv 12 = by$ and $a = 3 = b$ but $x = 2 \not\equiv 4 = y$.

23. [*M10*] Give an example to show that Law D is not always true if r is not relatively prime to s.

If $a = 6$, $b = 12$, $r = 3$, and $s = 6$, in Law D, then $a = 6 \equiv 12 = b$ modulo $r = 3$ and modulo $s = 6$ but not modulo $rs = 18$.

▸ **24.** [*M20*] To what extent can Laws A, B, C, and D be generalized to apply to arbitrary real numbers instead of integers?

Let's check the laws one by one.

Law A If $a, b, x, y, m \in \mathbb{R}$, if $m = 0$ the law is trivial. If not, we have $a - m\lfloor\frac{a}{m}\rfloor = b - m\lfloor\frac{b}{m}\rfloor$ and $x - m\lfloor\frac{x}{m}\rfloor = y - m\lfloor\frac{y}{m}\rfloor$, and for addition

$$\begin{aligned} a + x - m\left\lfloor\frac{a+x}{m}\right\rfloor = b + y - m\left\lfloor\frac{b+y}{m}\right\rfloor &\iff \\ \iff m\left(\left\lfloor\frac{a+x}{m}\right\rfloor - \left\lfloor\frac{b+y}{m}\right\rfloor\right) &= \\ = a + x - b - y = m\left(\left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor - \left\lfloor\frac{b}{m}\right\rfloor - \left\lfloor\frac{y}{m}\right\rfloor\right) &\iff \\ \iff \left\lfloor\frac{a+x}{m}\right\rfloor - \left\lfloor\frac{a}{m}\right\rfloor - \left\lfloor\frac{x}{m}\right\rfloor = \left\lfloor\frac{b+y}{m}\right\rfloor - \left\lfloor\frac{b}{m}\right\rfloor - \left\lfloor\frac{y}{m}\right\rfloor&. \end{aligned}$$

Now,

$$\left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor \le \frac{a+x}{m} < \left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor + 2,$$

so $\lfloor\frac{a+x}{m}\rfloor$ is either $\lfloor\frac{a}{m}\rfloor + \lfloor\frac{x}{m}\rfloor$ or $\lfloor\frac{a}{m}\rfloor + \lfloor\frac{x}{m}\rfloor + 1$, and similarly for b and y. But

$$\begin{aligned} \left\lfloor\frac{a+x}{m}\right\rfloor = \left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor \iff \left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor \le \frac{a+x}{m} < \left\lfloor\frac{a}{m}\right\rfloor + \left\lfloor\frac{x}{m}\right\rfloor + 1 &\iff \\ \iff a + x < m\left\lfloor\frac{a}{m}\right\rfloor + m\left\lfloor\frac{x}{m}\right\rfloor + m &\iff \\ \iff (a \bmod m) + (x \bmod m) = (b \bmod m) + (x \bmod m) < m &\iff \\ \iff \left\lfloor\frac{b+y}{m}\right\rfloor = \left\lfloor\frac{b}{m}\right\rfloor + \left\lfloor\frac{y}{m}\right\rfloor&, \end{aligned}$$

which proves the equality in the last line of the first formula. For subtraction, we have to see that $-x \equiv -y \bmod m$ and so $a - x = a + (-x) \equiv a + (-y) = a - y \pmod{m}$. If $x \bmod m = 0$, then $\frac{x}{m} \in \mathbb{Z}$, so $-\frac{x}{m} \in \mathbb{Z}$ and $-x \bmod m = 0$, and similarly $y \bmod m = 0$ and so $-y \bmod m = 0$. Otherwise $\frac{x}{m} \notin \mathbb{Z}$, so $\lceil\frac{x}{m}\rceil = \lfloor\frac{x}{m}\rfloor + 1$, but by Exercise 4,

$$\begin{aligned} -x \bmod m = -x - m\left\lfloor\frac{-x}{m}\right\rfloor = -x + m\left\lceil\frac{x}{m}\right\rceil = -x + m\left\lfloor\frac{x}{m}\right\rfloor + m &= \\ = m - (x \bmod m) = m - (y \bmod m) = -y \bmod m&. \end{aligned}$$

For multiplication this doesn't hold; for example, if $m = 2.5$, $a = 0.5$, $b = -2$, and $x = y = 2.2$, then $ax \bmod m = 1.1 \bmod 2.5 = 1.1$ but $by \bmod m = -4.4 \bmod 2.5 = 0.6$.

Law B Obviously a and m must be integers, otherwise the statement wouldn't make sense. Even then, however, if $a = 1$, $b = -2$, $x = \frac{4}{3}$, $y = \frac{7}{3}$, and $m = 3$, then $ax = \frac{4}{3} \equiv -\frac{14}{3} = by$ and $a = 1 \equiv -2 = b$, but $\frac{4}{3} \not\equiv \frac{7}{3}$.

Law C Using Exercise 15, for $a, b, m, n \in \mathbb{R}$ with $n \neq 0$,

$$\begin{aligned} a \equiv b \bmod m \iff a \bmod m = b \bmod m \iff \\ \iff an \bmod mn = n(a \bmod m) = n(b \bmod m) = bn \bmod mn \iff \\ \iff an \equiv bn \bmod mn. \end{aligned}$$

Note that the second double implication would not hold in the left direction for $n = 0$.

Law D Here r and s must be integers. Then, if $a, b \in \mathbb{R}$ with $a \equiv b$ modulo r and s, if $r = 0$ or $s = 0$, the statement is obvious, and otherwise $a - r\lfloor \frac{a}{r} \rfloor = b - r\lfloor \frac{b}{r} \rfloor$ and so $a - b = r(\lfloor \frac{a}{r} \rfloor - \lfloor \frac{b}{r} \rfloor) =: d \in \mathbb{Z}$. By Law A, since $a \equiv b$ and $b \equiv b$, $d = a - b \equiv b - b = 0$ modulo both r and s, so from Law D for integers we have $a - b \equiv 0$ modulo rs (assuming $r \perp s$) and so $a \equiv b$ modulo rs.

Thus, Law A holds for addition and subtraction, Law C always holds, and Law D holds if we still maintain $r, s \in \mathbb{Z}$.

25. [*M02*] Show that, according to Theorem F, $a^{p-1} \bmod p = [a$ is not a multiple of $p]$, whenever p is a prime number.

If a is a multiple of p, then so is a^{p-1} and $a^{p-1} \bmod p = 0$. Otherwise $a \perp p$, so we can cancel out in $a^p \equiv a \pmod{p}$ to get $a^{p-1} \equiv 1 \pmod{p}$ and so $a^{p-1} \bmod p = 1 \bmod p = 1$.

▸ **28.** [*M25*] Show that the method used to prove Theorem F can be used to prove the following extension, called *Euler's theorem*: $a^{\varphi(m)} \equiv 1 \pmod{m}$, for *any* positive integer m, when $a \perp m$. (In particular, the number n' in exercise 19 may be taken to be $n^{\varphi(m)-1} \bmod m$.)

First, $\gcd\{x, y\} \equiv \gcd\{x + ky, y\}$ for any $x, y, k \in \mathbb{Z}$, because if $z \mid x$ and $z \mid y$ then obviously $z \mid x + ky$ and if $z \mid x + ky$ and $z \mid y$ then $z \mid x$, so the set of common divisors is the same. It's also clear that if $x \perp z$ and $y \perp z$ then $xy \perp z$, since neither x nor y has common divisors with z and so neither does xy.

With these lemmas, we are ready to prove the statement. Let

$$\{x_1, \dots, x_{\varphi(m)}\} := \{x \in \{0, \dots, m-1\} \mid x \perp m\},$$

then the $x_i a \bmod m$ are all distinct, for if $x_i a \bmod m = x_j a \bmod m$, since $a \perp m$, then $x_i = x_i \bmod m = x_j \bmod m = x_j$. Since $x_i, a \perp m$, $x_i a \perp m$ and also $(x_i a \bmod m) \perp m$, so all the $x_i a \bmod m$ are different and relatively prime to m and therefore $\{x_i a \bmod m\}_i = \{x_i\}_i$. Thus

$$\prod_i x_i \equiv \prod_i (x_i a \bmod m) \equiv \prod_i x_i a \equiv a^{\varphi(m)} \prod_i x_i \pmod{m},$$

and therefore using Law B with the fact that $\prod_i x_i \perp m$ gives us the result.

▸ **34.** [*M21*] What conditions on the real number $b > 1$ are necessary and sufficient to guarantee that $\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$ for all real $x \geq 1$?

It happens if and only if $b \in \mathbb{Z}$. To prove this, observe that, since the logarithm in base $b > 1$ is strictly increasing, $\log_b \lfloor x \rfloor < \log_b x$, so $\lfloor \log_b x \rfloor \neq \lfloor \log_b \lfloor x \rfloor \rfloor$ if and only if $\lfloor \log_b \lfloor x \rfloor \rfloor < \lfloor \log_b x \rfloor$, that is, if there exists an integer k (namely $\lfloor \log_b x \rfloor$) such that $\log_b \lfloor x \rfloor < k \leq \log_b x$, if and only if $\lfloor x \rfloor < b^k \leq x$.

If $b \in \mathbb{Z}$, $b^k \in \mathbb{Z}$ as well, so that would mean that $\lfloor x \rfloor + 1 \leq b^k \leq x\#$.

If $b \notin \mathbb{Z}$, we just have to set $x = b$ and $k = 1$.

▸ **35.** [*M20*] Given that m and n are integers and $n > 0$, prove that

$$\lfloor (x+m)/n \rfloor = \lfloor (\lfloor x \rfloor + m)/n \rfloor$$

for all real x. (When $m = 0$, we have an important special case.) Does an analogous result hold for the ceiling function?

Clearly $\left\lfloor \frac{\lfloor x \rfloor + m}{n} \right\rfloor \leq \left\lfloor \frac{x+m}{n} \right\rfloor$. If this inequality were strict, however, there would be an integer k such that $\frac{\lfloor x \rfloor + m}{n} < k \leq \frac{x+m}{n}$, and so $\lfloor x \rfloor < nk - m \leq x$, but this is impossible because $nk - m \in \mathbb{Z}$.

For the ceiling, clearly $\left\lceil \frac{\lceil x \rceil + m}{n} \right\rceil \geq \left\lceil \frac{x+m}{n} \right\rceil$, but if this inequality were strict, there would be an integer k (namely $\left\lceil \frac{x+m}{n} \right\rceil$) such that $\frac{x+m}{n} \leq k < \frac{\lceil x \rceil + m}{n}$, and so $x \leq nk - m < \lceil x \rceil$, an absurdity because $nk - m \in \mathbb{Z}$.

1.2.5 Permutations and Factorials

1. [*00*] How many ways are there to shuffle a 52-card deck?

$52! = 80658175170943878571660636856403766975289505440883277824000000000000.$

2. [*10*] In the notation of Eq. (2), show that $p_{n(n-1)} = p_{nn}$, and explain why this happens.

$p_{n(n-1)} = n(n-1)\cdots(n-(n-1)+1) = n(n-1)\cdots 2 = n(n-1)\cdots 1 = p_{nn}$. This is because $p_{n(n-1)}$ is the number of ways to take and arrange $n-1$ objects out n and p_{nn} is the number of ways to take and arrange the n objects, which consists of first arranging $n-1$ objects out of the n objects and then adding the one remaining.

3. [*10*] What permutations of $\{1, 2, 3, 4, 5\}$ would be constructed from the permutation 3 1 2 4 using Methods 1 and 2, respectively?

1. **Method 1:** 5 3 1 2 4, 3 5 1 2 4, 3 1 5 2 4, 3 1 2 5 4, 3 1 2 4 5.

2. **Method 2:** 4 2 3 5 1, 4 1 3 5 2, 4 1 2 5 3, 3 1 2 5 4, 3 1 2 4 5.

▶ **4.** [*13*] Given the fact that $\log_{10} 1000! = 2567.60464...$, determine exactly how many decimal digits are present in the number 1000!. What is the *most significant* digit? What is the *least significant* digit?

Given a number x with $n+1$ digits $x_n x_{n-1} \cdots x_1 x_0$, we have $x_n 10^n \le x < (x_n+1)10^n$ and therefore $n + \log_{10} x_n \le \log x < n + \log_{10}(x_n+1)$, with $0 \le \log_{10} X_n < \log_{10}(x_n + 1) \le 1$. Thus, 1000! has 2568 digits and the most significant digit is 4, since $\log_{10} 4 = 0.60205...$ and $\log_{10} 5 = 0.69897....$ Furthermore, since $10 \mid 1000!$, the least significant digit is 0.

▶ **6.** [*17*] Using Eq. (8), write 20! as a product of prime factors.

Numbers up to 20 have prime factors up to 20, so the prime factors are 2, 3, 5, 7, 11, 13, 17, and 19. For the multiplicities,

$$\mu_2 = \sum_{k\ge1} \left\lfloor \frac{20}{2^k} \right\rfloor = 10 + 5 + 2 + 1 = 18, \qquad \mu_{11} = \sum_{k\ge1} \left\lfloor \frac{20}{11^k} \right\rfloor = 1,$$
$$\mu_3 = \sum_{k\ge1} \left\lfloor \frac{20}{3^k} \right\rfloor = 6 + 2 = 8, \qquad \mu_{13} = \sum_{k\ge1} \left\lfloor \frac{20}{13^k} \right\rfloor = 1,$$
$$\mu_5 = \sum_{k\ge1} \left\lfloor \frac{20}{5^k} \right\rfloor = 4, \qquad \mu_{17} = \sum_{k\ge1} \left\lfloor \frac{20}{17^k} \right\rfloor = 1,$$
$$\mu_7 = \sum_{k\ge1} \left\lfloor \frac{20}{7^k} \right\rfloor = 2, \qquad \mu_{19} = \sum_{k\ge1} \left\lfloor \frac{20}{19^k} \right\rfloor = 1,$$

so $20! = 2^{18}3^8 5^4 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$.

7. [*M10*] Show that the "generalized termial" function in Eq. (10) satisfies the identity $x? = x + (x-1)?$ for all real numbers x.

$x + (x-1)? = x + \frac{1}{2}(x-1)x = \frac{1}{2}2x + \frac{1}{2}(x-1)x = \frac{1}{2}(x+1)x = x?$.

9. [*M10*] Determine the values of $\Gamma(\frac{1}{2})$ and $\Gamma(-\frac{1}{2})$, given that $(\frac{1}{2})! = \sqrt{\pi}/2$.

$$(\tfrac{1}{2})! = \tfrac{1}{2}\Gamma(\tfrac{1}{2}) \implies \Gamma(\tfrac{1}{2}) = 2(\tfrac{1}{2})! = \sqrt{\pi},$$
$$-\tfrac{1}{2}\Gamma(-\tfrac{1}{2}) = \Gamma(\tfrac{1}{2}) \implies \Gamma(-\tfrac{1}{2}) = -2\Gamma(\tfrac{1}{2}) = -2\sqrt{\pi}.$$

▶ **10.** [*HM20*] Does the identity $\Gamma(x+1) = x\Gamma(x)$ hold for all real numbers x?

No, as it doesn't hold when $\Gamma(x)$ or $\Gamma(x+1)$ are not defined. However, when they are defined,

$$x\Gamma(x) = x \lim_m \frac{m^x m!}{\prod_{i=0}^m (x+i)} = \lim_m \frac{m^x m!}{\prod_{i=1}^m (x+i)} = \lim_m \frac{(m+1)^x (m+1)!}{\prod_{i=1}^{m+1} (x+i)} =$$
$$= \lim_m \frac{(m+1)^{x+1} m!}{\prod_{i=0}^m (x+1+i)} = \lim_m \left(\frac{m^{x+1} m!}{\prod_{i=0}^m ((x+1)+i)} \left(\frac{m+1}{m}\right)^{x+1} \right) = \Gamma(x+1) \cdot 1.$$

Note that, if $x \in \mathbb{Z}^-$, then the terms from $m = -x$ onward are not defined, so this is not the scenario we are dealing with. For ant other value, this holds.

11. [*M15*] Let the representation of n in the binary system be $n = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$, where $e_1 > e_2 > \cdots > e_r \geq 0$. Show that $n!$ is divisible by 2^{n-r} but not by 2^{n-r+1}.

This is exactly to say that, for $p = 2$, $\mu = n - r$. But r is the sum of all bits of n in base 2, so by Exercise 12, $\mu = \frac{1}{2-1}(n-r) = n - r$.

▸ **12.** [*M22*] (A. Legendre, 1808.) Generalizing the result of the previous exercise, let p be a prime number, and let the representation of n in the p-ary number system be $n = a_k p^k + a_{k-1}p^{k-1} + \cdots + a_1 p + a_0$. Express the number μ of Eq. (8) in a simple formula involving n, p, and a's.

The question makes sense for $n \in \mathbb{N}^*$. Then, given that the previous exercise showed that, for $p = 2$, $\mu = n - r$, where $r := |\{k \in \mathbb{N} \mid a_k \neq 0\}| = \sum_k a_k$, a guess is that $\mu = n - \sum_k a_k$ for any other positive prime integer p as well, that is, $n - \mu = \sum_k a_k$. To prove this,

$$\mu = \sum_{j\geq 1}\left\lfloor \frac{n}{p^j} \right\rfloor = \sum_{j=1}^{k}\sum_{i=j}^{k} a_i p^{i-j} = \sum_{1\leq j\leq i\leq k} a_i p^{i-j} = \sum_{i=1}^{k} a_i \left(\sum_{j=0}^{i-1} p^j\right) = \sum_{i=1}^{k} \frac{a_i(p^i-1)}{p-1},$$

where the latter identity is because of the cyclotomic formula. Then,

$$\begin{aligned}\sum_{i=1}^{k} \frac{a_i(p^i-1)}{p-1} &= \frac{1}{p-1}\left(\sum_{i=1}^{k} a_i p^i - \sum_{i=1}^{k} a_i\right) = \frac{1}{p-1}\left((n-a_0) - \sum_{i\geq 1} a_i\right)\\ &= \frac{1}{p-1}\left(n - \sum_{i\geq 0} a_i\right).\end{aligned}$$

▸ **22.** [*HM20*] Try to put yourself in Euler's place, looking for a way to generalize $n!$ to noninteger values of n. Since $(n+\frac{1}{2})!/n!$ times $((n+\frac{1}{2})+\frac{1}{2})!/(n+\frac{1}{2})!$ equals $(n+1)!/n! = n+1$, it seems natural that $(n+\frac{1}{2})!/n!$ should be approximately $\sqrt{n}$. Similarly, $(n+\frac{1}{3})!/n!$ should be $\approx \sqrt[3]{n}$. Invent a hypothesis about the ratio $(n+x)!/n!$ as n approaches infinity. Is your hypothesis correct when x is an integer? Does it tell anything about the appropriate value of $x!$ when x is not an integer?

The previous examples show $\frac{(n+\frac{1}{m})!}{n!} \approx \sqrt[m]{n} = n^{\frac{1}{m}}$, which is to say that $\frac{(n+x)!}{n!} \approx n^x$ for $x = \frac{1}{m}$. As n gets bigger, the approximation should probably be more accurate for the same value of x, so we could see that $(n+x)! \approx n^x n!$. For a non-negative integer x, $(n+x)! = (n+1)\cdots(n+x)\cdot n!$, so as n gets bigger the approximation is more and more precise, and in fact,

$$\lim_n \frac{n^x n!}{(n+x)!} = \lim_n \frac{n^x}{(n+1)\cdots(n+x)} = \lim_n \prod_{k=1}^{x} \frac{n}{n+k} = 1.$$

Multiplying by $x!$,

$$x! = \lim_n \frac{n^x x! n!}{(n+x)!} = \lim_n (n^x n!) \frac{x!}{(n+x)!} = \lim_n \frac{n^x n!}{(x+1)\cdots(x+n)},$$

which is Euler's factorial formula.

▸ **24.** [*HM21*] Prove the handy inequalities

$$\frac{n^n}{e^{n-1}} \le n! \le \frac{n^{n+1}}{e^{n-1}}, \qquad \text{integer } n \ge 1.$$

(Looking at the answers.) We have

$$\frac{n^n}{n!} = \prod_{k=1}^{n} \frac{n}{k} = \prod_{k=1}^{n-1} \frac{n}{k} = \prod_{k=1}^{n-1}\prod_{j=k}^{n-1} \frac{j+1}{j} = \prod_{1\le k\le j<n} \frac{j+1}{j} = \prod_{j=1}^{n-1} \left(\frac{j+1}{j}\right)^j =$$
$$= \prod_{j=1}^{n-1} \left(1+\frac{1}{j}\right)^j \le \prod_{j=1}^{n-1} \left(e^{1/j}\right)^j = e^{n-1},$$

where for the inequality we use that $1 + x \le e^x$ for all real x. Likewise,

$$\frac{n^{n+1}}{n!} = n\frac{n^n}{n!} = n\prod_{j=1}^{n-1}\left(\frac{j+1}{j}\right)^j = \prod_{j=1}^{n-1}\frac{j+1}{j}\prod_{j=1}^{n-1}\left(\frac{j+1}{j}\right)^j = \prod_{j=1}^{n-1}\left(\frac{j+1}{j}\right)^{j+1} =$$
$$= \prod_{j=2}^{n}\left(\frac{j}{j-1}\right)^j = \prod_{j=2}^{n}\left(1-\frac{1}{j}\right)^{-j} \ge \prod_{j=2}^{n}\left(e^{-1/j}\right)^{-j} = e^{n-1}.$$

1.2.6 Binomial Coefficients

1. [*00*] How many combinations of n things taken $n-1$ at a time are possible?
$\binom{n}{n-1} = \frac{n!}{1!(n-1)!} = n.$

2. [*00*] What is $\binom{0}{0}$?
$\binom{0}{0} = \frac{1!}{1!1!} = 1.$

3. [*00*] How many bridge hands (13 cards out of a 52-card deck) are possible?
$\binom{52}{13} = 635013559600.$

4. [*10*] Give the answer to exercise 3 as a product of prime numbers.

We have $\binom{52}{13} = \frac{52!}{13!39!}$. Using Equation 1.2.5.8:

$$\begin{aligned}
52! &= 2^{49}3^{23}5^{12}7^{8}11^{4}13^{4}17^{3}19^{2}23^{2} \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47,\\
39! &= 2^{35}3^{18}5^{8}7^{5}11^{3}13^{3}17^{2}19^{2} \cdot 23 \cdot 29 \cdot 31 \cdot 37\\
13! &= 2^{10}3^{5}5^{2} \cdot 7 \cdot 11 \cdot 13\\
\hline
\binom{52}{13} &= 2^{4}5^{2}7^{2} \cdot 17 \cdot 23 \cdot 41 \cdot 43 \cdot 47.
\end{aligned}$$

▸ **5.** [*05*] Use Pascal's triangle to explain the fact that $11^4 = 14641$.

$11^4 = (10+1)^4 = \binom{4}{0}10^4 1^0 + \binom{4}{1}10^3 1^1 + \binom{4}{2}10^2 1^2 + \binom{4}{3}10^1 1^3 + \binom{4}{4}10^0 1^4 = 14641$. The pattern works for the number 11 in any given base until the number 10 or higher appears, which in this case happens in the next row.

▸ **6.** [*10*] Pascal's triangle (Table 1) can be extended in all directions by use of the addition formula, Eq. (9). Find the three rows that go on *top* of Table 1 (i.e. for $r = -1, -2$, and -3).

The table is based on the properties that $\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}$ and that $\binom{r}{k} = 0$ for $k < 0$. This implies that $\binom{r}{0} = 1$ for any $r \in \mathbb{Z}$ (really for any $r \in \mathbb{R}$) and that $\binom{r-1}{k} = \binom{r}{k} - \binom{r-1}{k-1}$ (the one below in the table minus the one on the left).

r	$\binom{r}{0}$	$\binom{r}{1}$	$\binom{r}{2}$	$\binom{r}{3}$	$\binom{r}{4}$	$\binom{r}{5}$	$\binom{r}{6}$	$\binom{r}{7}$	$\binom{r}{8}$	$\binom{r}{9}$
-3	1	-3	6	-10	15	-21	28	-36	45	-55
-2	1	-2	3	-4	5	-6	7	-8	9	-10
-1	1	-1	1	-1	1	-1	1	-1	1	-1

8. [*00*] What property of Pascal's triangle is reflected in the "symmetry condition," Eq. (6)?

The symmetry of each row r with center in $k = \frac{r}{2}$.

9. [*01*] What is the value of $\binom{n}{n}$? (Consider all integers n.)

For $n \geq 0$, $\binom{n}{n} = \frac{n!}{n!1!} = 1$. For $n < 0$, by definition $\binom{n}{n} = 0$.

▸ **10.** [*M25*] If p is prime, show that:

1. $\binom{n}{p} \equiv \left\lfloor \frac{n}{p} \right\rfloor \pmod{p}$.
2. $\binom{p}{k} \equiv 0 \pmod{p}$, for $1 \leq k \leq p-1$.
3. $\binom{p-1}{k} \equiv (-1)^k \pmod{p}$, for $0 \leq k \leq p-1$.
4. $\binom{p+1}{k} \equiv 0 \pmod{p}$, for $2 \leq k \leq p-1$.

5. (É. Lucas, 1877.)
$$\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}\binom{n \bmod p}{k \bmod p} \pmod{p}.$$

6. If the representations of n and k in the p-ary number system are
$$\begin{aligned} n &= a_r p^r + \cdots + a_1 p + a_0, \\ k &= b_r p^r + \cdots + b_1 p + b_0, \end{aligned} \quad \text{then} \quad \binom{n}{k} \equiv \binom{a_r}{b_r} \cdots \binom{a_1}{b_1}\binom{a_0}{b_0} \pmod{p}.$$

We assume $n \in \mathbb{Z}$ and $k \in \mathbb{N}$.

1. Using part 5,
$$\binom{n}{p} \equiv \binom{\lfloor n/p \rfloor}{\lfloor p/p \rfloor}\binom{n \bmod p}{p \bmod p} = \binom{\lfloor n/p \rfloor}{1}\binom{n \bmod p}{0} = \left\lfloor \frac{n}{p} \right\rfloor \cdot 1.$$

2. $\binom{p}{k} = \frac{p!}{k!(p-k)!}$, but neither $k!$ nor $(p-k)!$ divide p while $p!$ does, so $p \mid \binom{p}{k}$.

3. We prove it by induction on k. For $k = 0$, $\binom{p-1}{0} = 1 = (-1)^0$, so the equation holds. For $1 \le k \le p-1$,
$$\binom{p-1}{k} = \binom{p}{k} - \binom{p-1}{k-1} \equiv 0 - (-1)^{k-1} = (-1)^k \pmod{p},$$
where for the equivalence we use the previous part of the exercise and the induction hypothesis.

4. Using part 2,
$$\binom{p+1}{k} = \binom{p}{k} + \binom{p}{k-1} \equiv 0 - 0 = 0 \pmod{k}.$$

5. If $k < 0$, then $\binom{n}{k} = 0$ and $\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} = 0$. Now let's prove this for $k \ge 0$. First, note that, if $a, b, c, d \in \mathbb{Z}$, $\frac{a}{b}, \frac{c}{d} \in \mathbb{Z}$, and $a \equiv c$ and $0 \not\equiv b \equiv d \pmod{p}$, then
$$\frac{a}{b} \equiv \frac{c}{d} \pmod{p}.$$
Effectively, let $a = jp + b$ $c = kp + d$ for some integers j and k, then
$$\frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd} = \frac{(jd - bk)p}{bd},$$
which is a multiple of p because it's an integer and, since $bd \nmid p$, it follows that $bd \mid jd - bk$. Now, let $n =: n'p + n_0$ and $k =: k'p + k_0$ with $n', n_0, k', k_0 \in \mathbb{Z}$ and $0 \le n_0, k_0 < p$. Clearly $n_0 \equiv n$ and $k_0 \equiv k$ (modulo p), and then
$$\begin{aligned} \binom{n}{k} &= \prod_{j=1}^{k} \frac{n+1-j}{j} = \left(\prod_{j=1}^{pk'} \frac{n+1-j}{j}\right)\left(\prod_{j=1}^{k_0} \frac{pn' + n_0 + 1 - pk' - j}{j + pk'}\right) \\ &\equiv \left(\prod_{j=1}^{pk'} \frac{n+1-j}{j}\right)\left(\prod_{j=1}^{k_0} \frac{n_0 + 1 - j}{j}\right) = \binom{n}{pk'}\binom{n_0}{k_0} \pmod{p}, \end{aligned}$$

where for the equivalence we used that both $\binom{n}{k}$ and $\binom{n}{pk'}\binom{n_0}{k_0}$ are integers. Now we have to prove that $\binom{n}{pk'} \equiv \binom{n'}{k'}$. This coefficient can be factored as

$$\binom{n}{pk'} = \prod_{j=0}^{k'-1}\left(\prod_{i=1}^{p}\frac{n+1-jp-i}{jp+i}\right) = \prod_{j=0}^{k'-1}\prod_{i=1}^{p}\frac{(n'-j)p+n_0+1-i}{jp+i}.$$

Each of the k' groups of factors has exactly one factor whose numerator is a multiple of p (when $i = s := (n_0+1) \bmod p$), and one whose denominator is. Canceling the p in them and taking congruences on the other factors, we get

$$\binom{n}{pk'} \equiv \prod_{j=0}^{k'-1}\left(\frac{n'-j}{j+1}\frac{\prod_{\substack{1\le i\le p\\ i\neq s}}(n_0+1-i)}{(p-1)!}\right) = \prod_{j=0}^{k'-1}\left(\frac{n'-j}{j+1}\frac{(p-1)!}{(p-1)!}\right) =$$

$$= \prod_{j=1}^{k'}\frac{n'+1+j}{j} = \binom{n'}{k'} \pmod{p}.$$

6. From part 5 by induction on r.

▸ **11.** [*M20*] (E. Kummer, 1852.) Let p be prime. Show that if p^n divides

$$\binom{a+b}{a}$$

but p^{n+1} does not, then n is equal to the number of *carries* that occur when a is added to b in the p-ary number system.

Because of the way the exercise is written, we may assume that both a and b are natural numbers (non-negative integers), and therefore $\binom{a+b}{a} = \frac{(a+b)!}{a!b!}$. Let s_a be the sum of the digits of a in base p and let $\mu_a = \frac{a-s_a}{p-1}$ be the number of times p appears in the prime factorization of a, by Exercise 1.2.5–12. Define s_b, μ_b, s_c, and μ_c in a similar manner, with $c := a+b$. Then the number of times p appears in the prime factorization of $\binom{a+b}{a}$ is precisely

$$\mu_c - \mu_a - \mu_b = \frac{\cancel{c-a-b} - s_c + s_a + s_b}{p-1},$$

but s_c is precisely the sum of s_a plus s_b except that, every time we do a carry in the sum in base p, we have to subtract p and add 1, that is, we subtract $p-1$, so $s_a + s_b - s_c$ is precisely $p-1$ times the number of carries and this proves the result.

▸ **17.** [*M18*] Prove the Chu-Vandermonde formula (21) from Eq. (15), using that $(1+x)^{r+s} = (1+x)^r(1+x)^s$.

If n is negative, either k or $n-k$ is negative for any k so both sides of the equation are 0. Otherwise, for $x \in (-1,1)$,

$$\sum_n \binom{r+s}{n} x^n = (1+x)^{r+s} = (1+x)^r(1+x)^s = \sum_k \binom{r}{k} x^k \sum_m \binom{s}{m} x^m =$$
$$= \sum_k \binom{r}{k} x^k \sum_n \binom{s}{n-k} x^{n-k} = \sum_n \left(\sum_k \binom{r}{k}\binom{s}{n-k} \right) x^n.$$

If r and s are integers, the sums are finite and therefore we have polynomials in x in both side, but since they are both equal in a nonempty open interval, the coefficients are all equal and we have proved the result. If r and s are not integers, since

$$\binom{r+s}{n} - \sum_k \binom{r}{k}\binom{s}{n-k} = 0$$

for all $r,s \in \mathbb{Z}$, and the left hand side is a polynomial on r and s, it follows that the polynomial is 0 everywhere.

▸ **21.** [*05*] Both sides of Eq. (25) are polynomials in s; why isn't that equation an identity in s?

We know the equation to be valid for $n+1$ values of s, namely $0,1,\dots,n$, but while the polynomial on the left has degree n, the one on the right has degree $m+n+1 \geq n+1$, so we cannot apply the reasoning below Eq. (8).

▸ **30.** [*M24*] Show that there is a better way to solve Example 3 than the way used in the text, by manipulating the sum so that Eq. (26) applies.

First, we note that the terms are nonzero when $k \geq 0$ and $m+2k \leq n+k$, which happens precisely when $0 \leq k \leq n-m$, and in particular we may assume $n \geq m$ since otherwise the sum is 0.

We start by negating the upper index (rule G) in the second factor and using symmetry (rule B) on the first, and noting that all factors with negative k are 0:

$$\sum_k \binom{n+k}{m+2k}\binom{2k}{k}\frac{(-1)^k}{k+1} = \sum_{k\geq 0} \binom{n+k}{n-m-k}\binom{-k-1}{k}\frac{1}{k+1}.$$

We have the left hand side of Eq. (26) with $t \mapsto 1$, $r \mapsto -1$, $n \mapsto n-m$, and $s \mapsto 2n-m$, so this is equal to

$$\binom{-1+2n-m-n+m}{n-m} = \binom{n-1}{n-m} = \binom{n-1}{m-1}.$$

▸ **31.** [*M20*] Evaluate

$$\sum_k \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r+k}{m+n}$$

in terms of r, s, m, and n, given that m and n are integers. Begin by replacing

$$\binom{r+k}{m+n} \quad \text{by} \quad \sum_j \binom{r}{m+n-j}\binom{k}{j}.$$

We start assuming that r and s are also integers. The replacement suggested is given directly by Equation 21, and we get

$$\begin{aligned}
S &:= \sum_k \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r+k}{m+n} \\
&= \sum_k \sum_j \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{k}{j} && \text{Eq. (21)} \\
&= \sum_j \binom{r}{m+n-j}\binom{m-r+s}{j}\sum_k \binom{n+r-s}{n-k}\binom{m-r+s-j}{k-j} && \text{Eq. (20)} \\
&= \sum_j \binom{r}{m+n-j}\binom{m-r+s}{j}\sum_k \binom{n+r-s}{k+r-s}\binom{m-r+s-j}{k-j} && \text{Eq. (6)*} \\
&= \sum_j \binom{r}{m+n-j}\binom{m-r+s}{j}\binom{m+n-j}{n-j} && \text{Eq. (22)*} \\
&= \sum_j \binom{r}{m+n-j}\binom{m-r+s}{j}\binom{m+n-j}{m} && \text{Eq. (6)**} \\
&= \binom{r}{m}\sum_j \binom{m-r+s}{j}\binom{r-m}{n-j} && \text{Eq. (20)} \\
&= \binom{r}{m}\binom{s}{n}.
\end{aligned}$$

In (**), we use that, for nonzero addends, $m+n-j \geq 0$ and therefore we can apply Equation 6. In (*), we assume $s \leq m-r$. This is not important, since, by the reasoning about polynomials, this has been proved for infinite values of r and infinite values of s and therefore applies to all r and s.

36. [*M10*] What is the sum $\sum_k \binom{n}{k}$ of the numbers in each row of Pascal's triangle? What is the sum of these numbers with alternating signs, $\sum_k \binom{n}{k}(-1)^k$?

If $n \geq 0$,

$$\sum_k \binom{n}{k} = \sum_k \binom{n}{k} 1^k 1^{n-k} = (1+1)^n = 2^n,$$

$$\sum_k \binom{n}{k}(-1)^k = \sum_k \binom{n}{k}(-1)^k 1^{n-k} = (1-1)^n = 0^n = \delta_{n0}.$$

If $n < 0$ the value is generally undefined, as evidenced by Exercise 6.

37. [*M10*] From the answers to the preceding exercise, deduce the value of the sum of every other entry in a row, $\binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\cdots$.

$$\sum_k \binom{n}{2k} = \sum_k \binom{n}{k}[k \text{ even}] = \sum_k \binom{n}{k}\frac{1+(-1)^k}{2} = \frac{2^n+\delta_{n0}}{2} = \begin{cases} 1, & n=0; \\ 2^{n-1}, & n>0. \end{cases}$$

39. [*M10*] What is the sum $\sum_k \left[{n \atop k}\right]$ of the numbers in each row of Stirling's first triangle? What is the sum of these numbers with alternating signs?

We can use Equation 44:

$$\sum_k \left[{n \atop k}\right] = (-1)^{-n}\sum_k (-1)^{n-k}\left[{n \atop k}\right](-1)^k = (-1)^n(-1)^{\underline{n}} =$$
$$= (-1)^n(-1)(-2)\cdots(-n) = n!,$$

$$\sum_k \left[{n \atop k}\right](-1)^k = \sum_k \left[{n \atop k}\right](-1)^{-k} = (-1)^{-n}\sum_k (-1)^{n-k}\left[{n \atop k}\right]1^k = (-1)^n 1^{\underline{n}} = \delta_{n0} - \delta_{n1}.$$

The first equation makes sense, as it is the number of permutations of n symbols irrespective of the number k of cycles.

42. [*HM10*] Express the binomial coefficient $\binom{r}{k}$ in terms of the beta function defined above. (This gives us a way to extend the definition to all real values of k.)

For integers $r \geq 0$ and $k \neq 0$,

$$\binom{r}{k} = \frac{r!}{k!(r-k)!} = \frac{\Gamma(r+1)}{\Gamma(k+1)\Gamma(r-k)} = \frac{\Gamma(r+1)}{k\Gamma(k)\Gamma(r-k+1)} = (kB(k,r-k+1))^{-1}.$$

We can extend this function to the points where it is not defined by taking limits.

We are not required to evaluate the domain of this function, but we have to see that it matches our definition for $r \in \mathbb{R}$ and $k \in \mathbb{Z}$. For $k \in \mathbb{N}$, we first see that

$$\lim_{(x,y)\to(r,k)} \frac{\Gamma(x-y+1)}{\Gamma(x-k+1)} = \lim_{(x,y)\to(r,k)} \frac{\lim_m \frac{m^{x-y}m!}{(x-y+1)\cdots(x-y+m)}}{\lim_m \frac{m^{x-k}m!}{(x-k+1)\cdots(x-k+m)}} =$$
$$= \lim_{(x,y)\to(r,k)} \lim_m m^{k-y}\prod_{i=1}^m \frac{x+m-y}{x+m-k} = 1,$$

where I don't remember enough of my calculus lessons to know why the last step is correct but it has to do with exchanging limits. With this, if we successively apply the fact that $\Gamma(x+1) = x\Gamma(x)$,

$$\lim_{(x,y)\to(r,k)} \frac{\Gamma(x+1)}{y\Gamma(y)\Gamma(x-y+1)} = \lim_{(x,y)\to(r,k)} \frac{x(x-1)\cdots(x-k+1)}{y\Gamma(y)\frac{\Gamma(x-y+1)}{\Gamma(x-k+1)}} =$$
$$= \frac{r(r-1)\cdots(r-k+1)}{k!}.$$

If $k \in \mathbb{Z}^-$, for $r \notin \mathbb{Z}^-$, $\Gamma(r+1)$ is bounded in an open set around (r,k), and since Γ is continuous and has no zeros, the denominator tends to infinity, so the corresponding limit correctly evaluates to 0. If $k, r \in \mathbb{Z}^-$ I can't find if the value is defined, but if it is then its value is 0.

44. [*HM20*] Using the generalized binomial coefficient suggested in exercise 42, show that

$$\binom{r}{1/2} = 2^{2r+1} \Big/ \binom{2r}{r}\pi.$$

From the definition,

$$\binom{r}{\frac{1}{2}} = \frac{1}{\frac{1}{2}B(\frac{1}{2}, r+\frac{1}{2})} = \frac{2\Gamma(r+1)}{\Gamma(\frac{1}{2})\Gamma(r+\frac{1}{2})}.$$

Now, we know $\frac{\sqrt{\pi}}{2} = (\frac{1}{2})! = \frac{1}{2}\Gamma(\frac{1}{2})$, so $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ and

$$\binom{r}{\frac{1}{2}} = \frac{2r!}{\sqrt{\pi}(r-\frac{1}{2})(r-\frac{3}{2})\cdots(\frac{1}{2})\sqrt{\pi}} = \frac{2r!2^r}{\pi(2r-1)(2r-3)\cdots 1}.$$

We see that $(2r)! = ((2r)(2r-2)\cdots 2)((2r-1)(2r-3)\cdots 1)$. We already have the second factor, and the first one is precisely $2^r r!$, so we multiply both numerator and denominator by this factor to get that

$$\binom{r}{\frac{1}{2}} = \frac{2r!2^r 2^r r!}{\pi(2r)!} = \frac{2^{2r+1}}{\binom{2r}{r}\pi}.$$

▸ **46.** [*M21*] Using Stirling's approximation, Eq. 1.2.5–(7), find an approximate value of $\binom{x+y}{y}$, assuming that both x and y are large. In particular, find the approximate size of $\binom{2n}{n}$ when n is large.

For large x and y,

$$\binom{x+y}{y} = \frac{(x+y)!}{x!y!} \approx \frac{\sqrt{2\pi(x+y)}\left(\frac{x+y}{\mathrm{e}}\right)^{x+y}}{\sqrt{2\pi x}\sqrt{2\pi y}\left(\frac{x}{\mathrm{e}}\right)^x\left(\frac{y}{\mathrm{e}}\right)^y} = \sqrt{\frac{x+y}{2\pi xy}}\left(\frac{(x+y)^{x+y}}{x^x y^y}\right) =$$
$$= \frac{(x+y)^{x+y+1/2}}{\sqrt{2\pi}x^{x+1/2}y^{y+1/2}} = \sqrt{\frac{1}{2\pi}\left(\frac{1}{x}+\frac{1}{y}\right)}\left(1+\frac{y}{x}\right)^x\left(1+\frac{x}{y}\right)^y.$$

Then, for large n,

$$\binom{2n}{n} = \frac{(2n)^{2n+1/2}}{\sqrt{2\pi}n^{2n+1}} = \frac{2^{2n}\sqrt{2}n^{2n}\sqrt{n}}{\sqrt{2\pi}n^{2n}n} = \frac{4^n}{\sqrt{\pi n}}.$$

▸ **48.** [*M25*] Show that

$$\sum_{k\geq 0}\binom{n}{k}\frac{(-1)^k}{k+x} = \frac{n!}{x(x+1)\cdots(x+n)} = \frac{1}{x\binom{n+x}{n}},$$

if the denominators are not zero.

The second equality is obvious. For the first one, since the way the formula is written assumes $n \in \mathbb{N}$, we can prove this by induction. For $n = 0$,

$$\sum_{k\geq 0}\binom{0}{k}\frac{(-1)^k}{k+x} = \frac{1}{x} = \frac{0!}{x}.$$

For $n > 0$,

$$\begin{aligned}\sum_{k\geq 0}\binom{n}{k}\frac{(-1)^k}{k+x} &= \sum_{k\geq 0}\binom{n-1}{k}\frac{(-1)^k}{k+x} + \sum_{k\geq 0}\binom{n-1}{k-1}\frac{(-1)^k}{k+x} = \\ &= \sum_{k\geq 0}\binom{n-1}{k}\frac{(-1)^k}{k+x} + \sum_{k\geq 0}\binom{n-1}{k}\frac{(-1)^{k+1}}{k+1+x} = \frac{1}{x\binom{n-1+x}{n-1}} - \frac{1}{(x+1)\binom{n+x}{n-1}} = \\ &= \frac{(n-1)!}{x(x+1)\cdots(x+n-1)} - \frac{(n-1)!}{(x+1)(x+2)\cdots(x+n)} = \frac{(n-1)!((\cancel{x+}n)\cancel{-x})}{x(x+1)\cdots(x+n)}.\end{aligned}$$

▸ **57.** [*M22*] Show that the coefficient a_m in Stirling's attempt at generalizing the factorial function, Eq. 1.2.5–(12), is

$$\frac{(-1)^m}{m!}\sum_{k\geq 1}(-1)^k\binom{m-1}{k-1}\ln k.$$

Also find q-nomial generalizations of the fundamental identities (17) and (21).

The sum in Stirling's attempt is

$$\begin{aligned} S_n &= \sum_{k\geq 0} a_{k+1}\prod_{j=0}^{k}(n-j) = \sum_{m\geq 1} a_m \prod_{j=0}^{m-1}(n-j) \\ &= \sum_{m\geq 1}\frac{(-1)^m}{m!}\left(\sum_{k\geq 1}(-1)^k\binom{m-1}{k-1}\ln k\right)\prod_{j=0}^{m-1}(n-j) \\ &= \sum_{k\geq 1}\left(\sum_{m\geq 1}(-1)^{m+k}\binom{n}{m}\binom{m-1}{k-1}\right)\ln k. \end{aligned}$$

There are several things happening in the last line. First, note that both sums are bounded, for m must be at most n because otherwise the product would contain a factor $n - n = 0$, and k must be lower than m, and therefore lower than n, because otherwise the factor $\binom{m-1}{k-1}$ would be 0. This allows us to swap the sums and extract a factor that only depends on k. Moreover,

$$\frac{\prod_{j=0}^{m-1}(n-j)}{m!} = \frac{\prod_{j=1}^{m}(n-j+1)}{\prod_{j=1}^{m} j} = \binom{n}{m}.$$

Let A_k be the value of the inner sum for a given k, we know that $A_k = 0$ for $k > n$, and if we prove that $A_k = 1$ for $k \leq n$, we would have

$$S_n = \sum_{k=1}^{n} \ln k = \ln \prod_{k=1}^{n} k = \ln n!$$

Now, we have

$$\begin{aligned}
A_k &= \sum_{m\geq 1} (-1)^{m+k} \binom{n}{m}\binom{m-1}{k-1} \\
&= \sum_{m} (-1)^{m+k} \binom{n}{m}\binom{m-1}{k-1} - (-1)^k \binom{-1}{k-1} && (*) \\
&= \sum_{m} (-1)^{n-m-k} \binom{n}{n-m}\binom{n-m-1}{k-1} + 1 && (**) \\
&= -\sum_{m} (-1)^{n-m} \binom{n}{m}\binom{k-1-n+m}{k-1} + 1 && \text{Rules B, G} \\
&= -\binom{k-1-n}{k-1-n} + 1 = 1, && \text{Eq. (23)}
\end{aligned}$$

where in $(*)$ we expand the domain of m to all integers and subtract the case when $m = 0$ to compensate for that (this is the only nonzero term that wasn't considered already) and in $(**)$ we change m to $n - m$.

For the second part of the exercise, we start by writing the identities (17) and (21) in q-nomial notation. For (17), this would be

$$\binom{r}{k, r-k} = (-1)^k \binom{k-r-1}{k, -r-1}, \quad \text{or} \quad \binom{a+b}{a, b} = (-1)^a \binom{-b-1}{a, -(a+b)-1}.$$

That is, this allows us to change the sum on top to something resembling just one of the numbers that were on the bottom part. To generalize this,

$$\begin{aligned}
\binom{k_1 + \cdots + k_m}{k_1, \ldots, k_m} &= \binom{k_1+k_2}{k_1} \cdots \binom{k_1 + \cdots + k_m}{k_1 + \cdots + k_{m-1}} \\
&= (-1)^{k_1+\cdots+k_{m-1}} \binom{k_1+k_2}{k_1} \cdots \binom{k_1 + \cdots + k_{m-1}}{k_1 + \cdots + k_{m-2}} \binom{-k_m - 1}{k_1 + \cdots + k_{m-1}} \\
&= (-1)^{k_1+\cdots+k_{m-1}} \binom{-k_m - 1}{k_1, \ldots, k_{m-1}, -(k_1 + \cdots + k_m) - 1}.
\end{aligned}$$

We are abusing notation here, since the book only defines multinomial coefficients for $k_i \geq 0$, but we can define them for any $k_i \in \mathbb{Z}$ by using the property that relates them to binomial coefficients, at the cost of symmetry with respect to the order of the k_i.

For (21), this would be

$$\sum_k \binom{r}{k, r-k}\binom{s}{n-k, s-n+k} = \binom{r+s}{n, r+s-n}.$$

That is, in this case we would add up all the numbers on each component. This clearly holds if we place $k, r-k$ and $n-k, s-n+k$ as the two last components of the multinomial coefficient. There must be a better (less silly) generalization but I've already spent way too much time in this exercise.

▸ **60.** [*M23*] We have seen that $\binom{n}{k}$ is the number of combinations of n things, k at a time, namely the number of ways to choose k different things out of a set of n. The *combinations with repetitions* are similar to ordinary combinations, except that we may choose each object any number of times. Thus, the list (1) would be extended to include also *aaa*, *aab*, *aac*, *aad*, *aae*, *abb*, etc., if we were considering combinations with repetition. How many k-combinations of n objects are there, if repetition is allowed?

We can draw a bijection between k-combinations of $\{1, \ldots, n\}$ with repetition and k-combinations of $\{1, \ldots, n+k-1\}$ without repetition by the following assignment:

$$\begin{aligned} 1 \le a_1 \le a_2 \le \cdots \le a_k \le n &\quad\mapsto\quad 1 \le a_1 < a_2 + 1 < \cdots < a_k + k - 1 \le n + k - 1 \\ 1 \le b_1 \le b_2 - 1 \le \cdots \le b_k - k + 1 \le n &\quad\leftarrow\!\shortmid\quad 1 \le b_1 < b_2 < \cdots < b_k \le n + k - 1 \end{aligned}$$

Thus the answer is

$$\binom{n+k-1}{k} = (-1)^k \binom{-n}{k} = \frac{n^{\overline{k}}}{k!}.$$

▸ **62.** [*M23*] The text gives formulas for sums involving a product of two binomial coefficients. Of the sums involving a product of three binomial coefficients, the following one and the identity of exercise 31 seem to be the most useful:

$$\sum_k (-1)^k \binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k} = \frac{(l+m+n)!}{l!m!n!}, \qquad \text{integer } l, m, n \ge 0.$$

(The sum includes both positive and negative values of k.) Prove this identity.

(Looking at the solution.)We may assume that $l \le m, n$, since we can always rename coefficients to make it this way. Then the application of the identity in exercise 31 is actually from right to left: we add a new variable j, change the last factor $\binom{n+l}{n+k}$ to $\binom{n+k}{l-k}$ by symmetry, and apply the aforementioned exercise to the last two factors to get that

$$S := \sum_k (-1)^k \binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k} = \sum_{j,k} (-1)^k \binom{l+m}{k+l}\binom{k+l}{j}\binom{m-k}{l-k-j}\binom{m+n+j}{m+l}.$$

Using that in general $\binom{n}{k} = 0$ for $n \ge 0$ and $k \notin \{0, \ldots, n\}$, nonzero addends here must follow

$$-l \le k \le m, \quad -m \le k \le n, \quad -n \le k \le l, \quad 0 \le j \le l + k, \quad l - m \le j \le l - k;$$

that is, $|k| \le \min\{m, n, l\}$ and $\max\{l-m, 0\} \le j \le l - |k|$, which simplifies to $|k| \le l$ and $0 \le j \le l - |k|$ using the conditions at the beginning and then to only $0 \le j \le l - |k|$.

This means that, in all the binomial coefficients in the right hand side of the above equation, the upper and lower parts are non-negative, so we can substitute them to fractions of coefficients and, after canceling out, we get

$$\begin{aligned} S &= \sum_{0\leq j\leq l-|k|} (-1)^k \frac{(m+n+j)!}{j!(l+k-j)!(l-k-j)!(m-l+j)!(n-l+j)!} \\ &= \sum_{0\leq j\leq l} \sum_{|k|\leq l-j} (-1)^k \binom{2l-2j}{l+k-j} \frac{(m+n+j)!}{(2l-2j)!j!(m-l+j)!(n-l+j)!}, \end{aligned}$$

but then the sum of $(-1)^k\binom{2l-2j}{l+k-j}$ across all k is 0 unless $j=l$, so we just take the term with $j=l$ and $k=0$ to get

$$S = (-1)^0\binom{2l-2l}{l+0-l}\frac{(m+n+l)!}{(2l-2l)!l!(m-l+l)!(n-l+l)!} = \frac{(l+m+n)!}{l!m!n!}.$$

▸ **64.** [*M20*] Show that $\left\{ n \atop m \right\}$ is the number of ways to partition a set of n elements into m nonempty disjoint subsets. For example, the set $\{1,2,3,4\}$ can be partitioned into two subsets in $\left\{ 4 \atop 2 \right\} = 7$ ways: $\{1,2,3\}\{4\}$; $\{1,2,4\}\{3\}$; $\{1,3,4\}\{2\}$; $\{2,3,4\}\{1\}$; $\{1,2\}\{3,4\}$; $\{1,3\}\{2,4\}$; $\{1,4\}\{2,3\}$.

Let $n,m\in\mathbb{N}$, we prove this by induction in n and m. For $n=0$, there is $1=\left\{ 0 \atop 0 \right\}$ way to partition $\emptyset$ into 0 nonempty disjoint subsets and $0=\left\{ 0 \atop m \right\}$ ways to partition it into $m\geq 1$ nonempty subsets, which matches Equation 48. For $m=0$ and $n\geq 1$, there are $0=\left\{ n \atop 0 \right\}$ ways to divide a set of n elements into 0 nonempty subsets.

Now, for the induction step, if $n\geq 1$ and $m\geq 1$, a partition of a set like $\{1,\dots,n\}$ into nonempty subsets can be thought of as a partition of $\{1,\dots,n-1\}$ to which n has been added either to one of the elements of a partition or in a separate singleton set. If the partition of $\{1,\dots,n\}$ has m nonempty subsets, for the first option we might add n to any of the subsets of a partition of $\{1,\dots,n-1\}$ with m subsets, and for the second one, we would need a partition of $\{1,\dots,n-1\}$ with $m-1$ subsets. Thus we would need

$$\left\{ n \atop m \right\} = m\left\{ n-1 \atop m \right\} + \left\{ n-1 \atop m-1 \right\},$$

but this is precisely Equation 46.

▸ **67.** [*M20*] We often need to know that binomial coefficients aren't too large. Prove the easy-to-remember upper bound

$$\binom{n}{k} \leq \left(\frac{n\mathrm{e}}{k}\right)^k, \qquad \text{when } n\geq k\geq 0.$$

From Exercise 1.2.5–21, if $k\geq 1$, then $k!\geq \frac{k^k}{\mathrm{e}^{k-1}}$, so

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq \frac{n(n-1)\cdots(n-k+1)}{\frac{k^k}{\mathrm{e}^{k-1}}} \leq \frac{n^k\mathrm{e}^{k-1}}{k^k} \leq \left(\frac{n\mathrm{e}}{k}\right)^k.$$

For $k=0$,

$$\lim_{k\to 0}\left(\frac{n\mathrm{e}}{k}\right)^k = \lim_{k\to\infty}\sqrt[k]{n\mathrm{e}k} = 1 = \binom{n}{0}.$$

1.2.7 Harmonic Numbers

1. [*01*] What are H_0, H_1, and H_2?
$H_0 = 0$, $H_1 = 1$, $H_2 = \frac{3}{2}$.

▸ **4.** [*10*] Decide which of the following statements are true for all positive integers n:

1. $H_n < \ln n$.
2. $H_n > \ln n$.
3. $H_n > \ln n + \gamma$.

1. False, since $H_2 = \frac{3}{2} > 1 > \ln 2$.
2. True, because the next one is true.
3. True. If this wasn't true for some number n, then it would be

$$0 \geq H_n - (\ln n + \gamma) > \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{12n^4} - \frac{1}{252n^6},$$

but $\frac{1}{2n} > \frac{1}{12n^2}$ and $\frac{1}{12n^4} > \frac{1}{252n^6}$ for any $n \geq 1\#$.

▸ **9.** [*M18*] Theorem A applies only when $x > 0$; what is the value of the sum considered when $x = -1$?
Using Equation 1.2.6(18) in $(*)$ and Exercise 1.2.6–48 in $(**)$,

$$\sum_{k=1}^{n} \binom{n}{k} (-1)^k H_k = \sum_{1 \leq j \leq k \leq n} \binom{n}{k} (-1)^k \frac{1}{j} = \sum_{j=1}^{n} \frac{1}{j} \sum_{k=j}^{n} \binom{n}{k} (-1)^k =$$
$$= (-1)^n \sum_{j=1}^{n} \frac{1}{j} \sum_{k=0}^{n-j} \binom{n}{k} (-1)^k \overset{(*)}{=} (-1)^{\cancel{n}} \sum_{j=1}^{n} \frac{1}{j} (-1)^{\cancel{n}-j} \binom{n-1}{n-j} =$$
$$= \sum_{j=0}^{n} \frac{1}{j+1} (-1)^{-j-1} \binom{n-1}{n-j-1} = -\sum_{j=0}^{n-1} \frac{(-1)^j}{j+1} \binom{n-1}{j} \overset{(**)}{=} \frac{1}{\binom{n}{n-1}} = -\frac{1}{n}.$$

▸ **11.** [*M21*] Using summation by parts, evaluate

$$\sum_{1 < k \leq n} \frac{1}{k(k-1)} H_k.$$

We have

$$\frac{1}{k(k-1)} H_k = \left(\frac{1}{k-1} - \frac{1}{k} \right) H_k = \frac{1}{k-1} \left(H_{k-1} + \frac{1}{k} \right) - \frac{1}{k} H_k,$$

so

$$\sum_{k=2}^{n}\frac{H_k}{k(k-1)}=\sum_{k=2}^{n}\left(\frac{1}{k-1}\left(H_{k-1}+\frac{1}{k}\right)-\frac{1}{k}H_k\right)=1-\frac{1}{n}H_n+\sum_{k=2}^{n}\frac{1}{k(k-1)}=$$
$$=1-\frac{H_n}{n}+\sum_{k=2}^{n}\left(\frac{1}{k-1}-\frac{1}{k}\right)=1-\frac{H_n}{n}+H_{n-1}-H_n+1=$$
$$=1-\frac{1}{n}H_n\cancel{+H_n}-\frac{1}{n}\cancel{-H_n}+1=2-\frac{1}{n}(1+H_n).$$

▸ **12.** [*M10*] Evaluate $H_\infty^{(1000)}$ correct to at least 100 decimal places.

$$H_\infty^{(1000)}=\sum_{k\geq 1}\frac{1}{k^{1000}}=1\pm 0.5\cdot 10^{-100},$$

since 2^{1000} has way more than 100 digits.

▸ **15.** [*M23*] Express $\sum_{k=1}^{n}H_k^2$ in terms of n and H_n.

$$\sum_{k=1}^{n}H_k^2=\sum_{1\leq j\leq k\leq n}\frac{1}{j}H_k=\sum_{j=1}^{n}\frac{1}{j}\sum_{k=j}^{n}H_k=\sum_{j=1}^{n}\frac{1}{j}\left(\sum_{k=1}^{n}H_k-\sum_{k=1}^{j-1}H_k\right)$$
$$=\sum_{j=1}^{n}\frac{1}{j}\left((n+1)H_n-n-jH_{j-1}+j-1\right)$$
$$=\sum_{j=1}^{n}\left(\frac{1}{j}\left((n+1)H_n-n-1\right)-H_{j-1}+1\right)$$
$$=\left((n+1)H_n-n-1\right)H_n-\left(nH_{n-1}-n+1\right)+n$$
$$=(n+1)H_n^2-(n+1)H_n-nH_n+n+n$$
$$=(n+1)H_n^2-(2n+1)H_n+2n.$$

▸ **23.** [*HM20*] By considering the function $\Gamma'(x)/\Gamma(x)$, generalize H_n to noninteger values of n. You may use the fact that $\Gamma'(1)=-\gamma$, anticipating the next exercise.

We have

$$\Gamma'(x)=\lim_{m}\left(\frac{\ln m\cdot m^x m!}{x(x+1)\cdots(x+m)}-\frac{m^x m!\sum_{k=0}^{m}\frac{x(x+1)\cdots(x+m)}{(x+k)}}{(x(x+1)\cdots(x+m))^2}\right)=$$
$$=\Gamma(x)\lim_{m}\left(\ln m-\sum_{k=0}^{m}\frac{1}{x+k}\right)=\Gamma(x)(\ln m-H_{k+m}+H_{k-1}).$$

The fact given in the exercise tells us that

$$-\gamma=\Gamma'(1)=\Gamma(1)\lim_{m}(\ln m-H_{m+1}),$$

so for $n \in \mathbb{Z}^{>0}$,

$$\frac{\Gamma'(n)}{\Gamma(n)} = \lim_m (\ln m - H_{n+m} + H_{n-1}) = H_{n-1} - \gamma,$$

and we can define

$$H_x := \frac{\Gamma'(x+1)}{\Gamma(x+1)} + \gamma$$

for any $x \in \mathbb{C}$ where this expression is defined or can be extended by continuity.

1.2.8 Fibonacci Numbers

1. [*10*] What is the answer to Leonardo Fibonacci's original problem: How many pairs of rabbits are present after a year?

After one month there are $F_3 = 2$ pairs of rabbits, after two months there are $F_4 = 3$, etc., therefore after a year there are $F_{14} = 377$ pairs of rabbits.

▸ **2.** [*20*] In view of Eq. (15), what is the approximate value of F_{1000}? (Use logarithms found in Appendix A.)

$$F_{1000} \approx \frac{\phi^{1000}}{\sqrt{5}} = \frac{10^{\log \phi^{1000}}}{\sqrt{5}} = \frac{10^{1000(\ln\phi/\ln 10)}}{\sqrt{5}} \cong \frac{10^{1000 \cdot 0.48121\ldots \cdot 0.43429\ldots}}{2.23606\ldots} \cong \frac{10^{208.9876..}}{2.23606\ldots}$$
$$\cong \frac{10^{0.9876}}{2.23606} \cdot 10^{208} \cong 4.3 \cdot 10^{208}.$$

▸ **4.** [*14*] Find all n for which $F_n = n$.

$F_0 = 0$, $F_1 = 1$. Then $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, and from here F_n grows strictly faster than n so there are no more such numbers. Thus only 0, 1, and 5 meet this condition.

▸ **7.** [*15*] If n is not a prime number, F_n is not a prime number (with one exception). Prove this and find the exception.

The exception is $F_4 = 3$. For the general rule, let $r, s \in \mathbb{Z}$ with $r \geq 3$ and $s \geq 2$, by Eq. (6),

$$F_{rs} = F_{r+r(s-1)} = F_{r(s-1)}F_{r+1} + F_{r(s-1)-1}F_r = F_r(F_{r(s-1)} + F_{r(s-1)-1}) + F_{r-1}F_{r(s-1)}$$
$$= F_r\left(F_{r(s-1)+1} + F_{r-1}\tfrac{F_{r(s-1)}}{F_r}\right),$$

where the fraction in the last part is an integer. If $r > 2$, then $F_r > 1$ and, since $F_{r(s-1)+1} > 1$, we have a decomposition of F_{rs} as a compound number. Every compound number other than 4 can be expressed as a product of integers rs with $r \geq 3$ and $s \geq 2$.

▸ **13.** [*M22*] Express the following sequences in terms of the Fibonacci numbers, when r, s, and c are given constants:

1. $a_0 = r$, $a_1 = s$; $a_{n+2} = a_{n+1} + a_n$, for $n \geq 0$.
2. $b_0 = 0$, $b_1 = 1$; $b_{n+2} = b_{n+1} + b_n + c$, for $n \geq 0$.

1. We have
$$\begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \end{pmatrix} = \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} s \\ r \end{pmatrix} = \begin{pmatrix} F_{n+1}s + F_n r \\ F_n s + F_{n-1} r \end{pmatrix},$$
so in general $a_n = F_{n-1}r + F_n s$, using the convention that $F_{-1} = 1$.

2. We have
$$\begin{pmatrix} b_{n+1} \\ b_n \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & c \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_n \\ b_{n-1} \\ 1 \end{pmatrix} = \cdots = \begin{pmatrix} 1 & 1 & c \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$
By playing a bit with this matrix, we come to the conclusion that
$$\begin{pmatrix} 1 & 1 & c \\ 1 & & \\ & & 1 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n & (F_{n+2} - 1)c \\ F_n & F_{n-1} & (F_{n+1} - 1)c \\ & & 1 \end{pmatrix}$$
and therefore
$$b_n = F_n + (F_{n+1} - 1)c.$$
For $n = 0$ and $n = 1$, this is easy to check. For $n \geq 2$,
$$b_n = b_{n-1} + b_{n-2} + c = F_{n-1} + (F_n - 1)c + F_{n-2} + (F_{n-1} - 1)c + c = F_n + (F_{n+1} - 1)c.$$

▸ **16.** [*M20*] Fibonacci numbers appear implicitly in Pascal's triangle if it is viewed from the right angle. Show that the following sum of binomial coefficients is a Fibonacci number:
$$\sum_{k=0}^{n} \binom{n-k}{k}.$$

We prove by induction that this sum is precisely F_{n+1}. For $n = 0$ and $n = 1$, this is a simple check. For $n > 1$,
$$\begin{aligned} \sum_{k=0}^{n} \binom{n-k}{k} &= \sum_{k=0}^{n} \binom{n-k-1}{k} + \sum_{k=0}^{n} \binom{n-k-1}{k-1} \\ &= \sum_{k=0}^{n} \binom{n-k-1}{k} + \sum_{k=-1}^{n-1} \binom{n-k-2}{k} \\ &= \sum_{k=0}^{n-1} \binom{(n-1)-k}{k} + \sum_{k=0}^{n-2} \binom{(n-2)-k}{k} = F_n + F_{n-1} = F_{n+1}. \end{aligned}$$

▸ **22.** [*M20*] Show that $\sum_k \binom{n}{k} F_{m+k}$ is a Fibonacci number.

We prove by induction on n that this sum is F_{m+2n}. For $n = 0$ and $n = 1$, this is a simple check. For $n > 1$,

$$\begin{aligned}\sum_k \binom{n}{k} F_{m+k} &= \sum_k \binom{n-1}{k} F_{m+k} + \sum_k \binom{n-1}{k-1} F_{m+k}\\ &= F_{m+2(n-1)} + \sum_k \binom{n-1}{k} F_{m+1+k}\\ &= F_{m+2(n-1)} + F_{m+1+2(n-1)} = F_{m+2+2(n-1)} = F_{m+2n}.\end{aligned}$$

▸ **26.** [*M20*] Using the previous exercise, show that $F_p = 5^{(p-1)/2} \pmod p$ if p is an odd prime.

The previous exercise tells us that

$$2^n F_n = 2 \sum_{k \text{ odd}} \binom{n}{k} 5^{(k-1)/2}.$$

Now, if $n = p$ is prime, $\binom{p}{k} = \frac{p(p-1)\cdots(p-k+1)}{1\cdots k}$ is a multiple of p for $k \in \{1, \ldots, p-1\}$, and $k = 0$ is not odd, so

$$2^{p-1} F_p = \sum_{k \text{ odd}} \binom{p}{k} 5^{(k-1)/2} \equiv \binom{p}{p} 5^{(p-1)/2} = 5^{(p-1)/2} \pmod p,$$

but by Fermat's theorem, $2^{p-1} \equiv 1 \pmod p$, so this simplifies to the desired result.

▸ **29.** [*M23*] (*Fibonomial coefficients.*) Édouard Lucas defined the quantities

$$\binom{n}{k}_{\mathcal{F}} = \frac{F_n F_{n-1} \cdots F_{n-k+1}}{F_k F_{k-1} \cdots F_1} = \prod_{j=1}^{k} \left(\frac{F_{n-k+j}}{F_j} \right)$$

in a manner analogous to binomial coefficients.

1. Make a table of $\binom{n}{k}_{\mathcal{F}}$ for $0 \le k \le n \le 6$.
2. Show that $\binom{n}{k}_{\mathcal{F}}$ is always an integer because we have

$$\binom{n}{k}_{\mathcal{F}} = F_{k-1} \binom{n-1}{k}_{\mathcal{F}} + F_{n-k+1} \binom{n-1}{k-1}_{\mathcal{F}}.$$

1.

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$
0	1						
1	1	1					
2	1	1	1				
3	1	2	2	1			
4	1	3	6	3	1		
5	1	5	15	15	5	1	
6	1	8	40	60	40	8	1

2. We have

$$F_{k-1}\binom{n-1}{k}_{\mathcal{F}} + F_{n-k+1}\binom{n-1}{k-1}_{\mathcal{F}} =$$
$$= F_{k-1}\prod_{j=1}^{k}\frac{F_{n-k+j-1}}{F_j} + F_{n-k+1}\prod_{j=1}^{k-1}\frac{F_{n-k+j}}{F_j} =$$
$$= \frac{1}{F_k}\left(\prod_{j=1}^{k-1}\frac{F_{n-k+j}}{F_j}\right)(F_{k-1}F_{n-k} + F_{n-k+1}F_k),$$

where the last identity is better read backwards, and by Eq. (6),

$$F_n = F_{(n-k)+k} = F_kF_{n-k+1} + F_{k-1}F_{n-k},$$

so the above simplifies precisely to $\binom{n}{k}_{\mathcal{F}}$.

▸ **34.** [*M24*] (*The Fibonacci number system.*) Let the notation $k \gg m$ mean that $k \geq m+2$. Show that every positive integer n has a *unique* representation $n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}$, where $k_1 \gg k_2 \gg \cdots \gg k_r \gg 0$.

If $n \leq 3$, this is easy to check. Otherwise, let k be the greatest number such that $n \geq F_k$. If $n = F_k$, we're done. Otherwise $n > 3$ and $k \geq 4$, and $n - F_k < F_{k-1}$ (otherwise $F_{k+1} \leq n\#$), and it can be expressed this way by induction.

To see that this representation is unique, let $n = F_{k_1} + \cdots + F_{k_r} = F_{j_1} + \cdots + F_{j_s}$. If $n \leq 3$, this is easy to check. Otherwise, if $F_{k_1} = F_{j_1}$ we use induction; otherwise assume $F_{k_1} > F_{j_1} \geq 2$, and we want to prove that

$$F_{j_2} + \cdots + F_{j_s} \leq F_{j_1-2} + \cdots + F_{\lfloor j_1 \bmod 2\rfloor+2} \leq F_{j_1-1} \leq F_{k_1} - F_{j_1}.$$

The first inequality comes from the highest value that $F_{j_2} + \cdots + F_{j_s}$ could possibly have, and the last one from the highest value that $F_{j_1} + F_{j_1-1}$ could. For the middle one we use induction. For $j_1 \in \{2,3\}$, the left hand side is 0. Otherwise

$$F_{j_1-2} + F_{j_1-4} + \cdots + F_{\lfloor j_1 \bmod 2\rfloor+2} \leq F_{j_1-2} + F_{j_1-3} = F_{j_1-1}.$$

▸ **41.** [*M25*] (Yuri Matiyasevich, 1990.) Let $f(x) = \lfloor x + \phi^{-1}\rfloor$. Prove that if $n = F_{k_1} + \cdots + F_{k_r}$ is the representation of n in the Fibonacci number system of exercise 34, then $F_{k_1+1} + \cdots + F_{k_r+1} = f(\phi n)$. Find a similar formula for $F_{k_1-1} + \cdots + F_{k_r-1}$.

Using that each $F_{k_i} = \frac{1}{\sqrt{5}}(\phi^{k_i} + \hat{\phi}^{k_i})$, let $m := F_{k_1+1} + \cdots + F_{k_r+1}$, since $m \in \mathbb{Z}$,

$$f(\phi n) - m = \left\lfloor \frac{1}{\sqrt{5}}\left(\sum_i \phi^{k_i+1} + \sum_i \hat{\phi}^{k_i-1}\right) - \hat{\phi}\right\rfloor - \frac{1}{\sqrt{5}}\left(\sum_i \phi^{k_i+1} - \sum_i \hat{\phi}^{k_i+1}\right)$$
$$= \left\lfloor \frac{1}{\sqrt{5}}\sum_i(\hat{\phi}^{k_i-1} + \hat{\phi}^{k_i+1}) - \hat{\phi}\right\rfloor = \left\lfloor \frac{1+\hat{\phi}^2}{\sqrt{5}}\sum_i \hat{\phi}^{k_i-1} - \hat{\phi}\right\rfloor.$$

Here, when k_i is even, $\hat{\phi}^{k_i-1}$ is negative, and when it's odd, $\hat{\phi}^{k_i-1}$ is positive, so the value of this sum is strictly lower than $\sum_{k\geq 1}\hat{\phi}^{2k} = \frac{\hat{\phi}^2}{1-\hat{\phi}^2}$, and strictly greater than $\sum_{k\geq 1}\hat{\phi}^{2k-1} = \frac{\hat{\phi}}{1-\hat{\phi}^2}$. Now, $\hat{\phi}^2 = \frac{1}{4}(1-\sqrt{5})^2 = \frac{1}{4}(6-2\sqrt{5}) = \frac{1}{2}(3-\sqrt{5})$, so $1-\hat{\phi}^2 = \frac{1}{2}(2-3+\sqrt{5}) = -\frac{1}{2}(1-\sqrt{5}) = -\hat{\phi}$ and therefore the lower bound of what is inside the brackets above is

$$\frac{1+\hat{\phi}^2}{\sqrt{5}}\frac{\hat{\phi}}{1-\hat{\phi}^2} - \hat{\phi} = -\frac{1+\hat{\phi}^2}{\sqrt{5}} - \hat{\phi} = -\frac{5-\sqrt{5}}{2\sqrt{5}} - \frac{1-\sqrt{5}}{2} = -\frac{5-\sqrt{5}+\sqrt{5}-5}{2\sqrt{5}} = 0,$$

and the upper bound is

$$-\hat{\phi}\frac{1+\hat{\phi}^2}{\sqrt{5}} - \hat{\phi} = -\hat{\phi}\left(\frac{5-\sqrt{5}}{2\sqrt{5}}+1\right) = -\hat{\phi}\left(\frac{5+\sqrt{5}}{2\sqrt{5}}\right) = -\hat{\phi}\phi = 1,$$

and since these bounds are never reached, the property has been proven. A similar argument can be made to prove that $F_{k_1-1}+\cdots+F_{k_r-1} = f(n/\phi)$.

1.2.9 Generating Functions

▶ **2.** [*M13*] Prove Eq. (11).

$$\left(\sum_{n\geq 0}\frac{a_n}{n!}z^n\right)\left(\sum_{m\geq 0}\frac{b_m}{m!}z^m\right) = \sum_{n\geq 0}\left(\sum_k \frac{a_k b_{n-k}}{k!(n-k)!}\right)z^n = \sum_{n\geq 0}\frac{1}{n!}\left(\sum_k \binom{n}{k}a_k b_{n-k}\right)z^n.$$

4. [*M01*] Explain why Eq. (19) is a special case of Eq. (21).
Just set $t=0$, then $x = 1+z$.

▶ **6.** [*HM15*] Find the generating function for

$$\left\langle \sum_{0<k<n}\frac{1}{k(n-k)}\right\rangle;$$

differentiate it and express the coefficients in terms of harmonic numbers.
The generating function is

$$G(z) := \sum_{n\geq 0}\sum_{0<k<n}\frac{1}{k(n-k)}z^n = \sum_{n,m\geq 1}\frac{z^{n+m}}{nm} = \left(\sum_{n\geq 1}\frac{z^n}{n}\right)\left(\sum_{m\geq 1}\frac{z^m}{m}\right) = \left(\sum_{n\geq 1}\frac{z^n}{n}\right)^2$$
$$= \left(-\sum_{n\geq 1}\frac{(-1)^{n+1}}{n}(-z)^n\right)^2 = (-\ln(1+(-z)))^2 = \ln(1-z)^2,$$

by Eq. (24). Its derivative is

$$\dot{G}(z) = -\frac{2}{1-z}\ln(1-z) = \frac{2}{1-z}\ln\left(\frac{1}{1-z}\right),$$

so by Eq. (25) with $m=0$ the coefficients are $[z^n]\dot{G}(z) = 2(H_n - H_0)\binom{n}{n} = 2H_n$.

▸ **10.** [*M25*] An *elementary symmetric function* is defined by the formula

$$e_m = \sum_{1\le j_1<\cdots<j_m\le n} x_{j_1}\cdots x_{j_m}.$$

(This is the same as h_m of Eq. (33), except that equal subscripts are not allowed.) Find the generating function for e_m, and express e_m in terms of the S_j in Eq. (34). Write out the formulas for e_1, e_2, e_3, and e_4.

Here $m \le n$ or otherwise we have no elements, so

$$\begin{aligned}G(z) &\coloneqq \sum_{m\ge0} e_m z^m = \sum_{m=0}^{n}\sum_{1\le j_1<\cdots<j_k\le n} x_{j_1}\cdots x_{j_m} z^n = \sum_{a_1,\dots,a_n\in\{0,1\}}\prod_{i=1}^{n}(x_iz)^{a_i}\\ &= \prod_{i=1}^{n}\sum_{a\in\{0,1\}}(x_iz)^a = \prod_{i=1}^{n}(x_iz+1),\end{aligned}$$

as we can regard the sum as taking the product of each subset of $\{x_iz\}_i$. From here,

$$\ln G(z) = \sum_{i=1}^{n}\ln(1+x_iz) = \sum_{i=1}^{n}\sum_{k\ge1}\frac{(-1)^{k+1}}{k}x_i^kz^k = \sum_{k\ge1}\frac{(-1)^{k+1}}{k}S_kz^k,$$

so

$$\begin{aligned}G(z) &= \mathrm{e}^{\ln G(z)} = \exp\left(\sum_{k\ge1}\frac{(-1)^{k+1}}{k}S_kz^k\right) = \prod_{k\ge1}\mathrm{e}^{(-1)^{k+1}S_kz^k/k}\\ &= \prod_{k\ge1}\sum_{j\ge0}\frac{1}{j!}\left(\frac{(-1)^{k+1}S_k}{k}\right)^j z^{jk} = \sum_{m\ge0}\left(\sum_{\substack{j_1,\dots,j_m\ge0\\ j_1+2j_2+\cdots+mj_m=m}}\prod_{k=1}^{m}\frac{1}{j_k!}\left(\frac{(-1)^{k+1}S_k}{k}\right)^{j_k}\right)z^m.\end{aligned}$$

This gives us an explicit formulation for e_m. Note that

$$\prod_{k=1}^{m}(-1)^{(k+1)j_k} = (-1)^{\sum_{k=1}^m kj_k+\sum_{k=1}^m j_k} = (-1)^{m+\sum_k j_k},$$

so

$$e_m = \sum_{\substack{j_1,\dots,j_m\ge0\\ j_1+2j_2+\cdots+mj_m=m}}(-1)^{m+j_1+\cdots+j_m}\prod_{k=1}^{m}\frac{S_k^{j_k}}{k^{j_k}j_k!}.$$

For example, if we call p_{kj_k} to each of the factors of the product in this latest formula, using that each $p_{k0} = 1$ and can therefore be ignored,

$$\begin{aligned}
e_1 &= p_{11} = S_1, \\
e_2 &= p_{12} + p_{21} = \frac{S_1^2}{2} - \frac{S_2}{2} = \frac{1}{2}(S_1^2 - S_2), \\
e_3 &= p_{13} - p_{11}p_{21} + p_{31} = \frac{S_1^3}{6} - S_1\frac{S_2}{2} + \frac{S_3}{3} = \frac{1}{6}(S_1^3 + 2S_3 - 3S_1S_2), \\
e_4 &= p_{14} - p_{12}p_{21} + p_{11}p_{31} + p_{22} - p_{41} = \frac{S_1^4}{24} - \frac{S_1^2}{2}\frac{S_2}{2} + S_1\frac{S_3}{3} + \frac{S_2^2}{8} - \frac{S_4}{4} \\
&= \frac{1}{24}(S_1^4 - 6S_1^2S_2 + 8S_1S_3 + 3S_2^2 - 6S_4).
\end{aligned}$$

▸ **11.** [*M25*] Equation (39) can also be used to express the S's in terms of the h's: We find $S_1 = h_1$, $S_2 = 2h_2 - h_1^2$, $S_3 = 3h_3 - 3h_1h_2 + h_1^3$, etc. What is the coefficient of $h_1^{k_1}h_2^{k_2}\cdots h_m^{k_m}$ in this representation of S_m, when $k_1 + 2k_2 + \cdots + mk_m = m$?

(I had to look up the solution.) We ignore Eq. (39) and focus on Eqs. (37) and (24). There,

$$\begin{aligned}
\sum_{k\geq1}\frac{S_kz^k}{k!} &= \ln G(z) = \ln\left(1 + \sum_{j\geq1}h_jz^j\right) = \sum_{k\geq1}\frac{(-1)^{k+1}}{k}\left(\sum_{j\geq1}h_jz^j\right)^k \\
&= \sum_{k\geq1}\frac{(-1)^{k+1}}{k}\sum_{j_1,\dots,j_k\geq1}h_{j_1}\cdots h_{j_k}z^{j_1+\dots+j_k} \\
&= \sum_{m\geq1}\sum_{\substack{i_1,\dots,i_m\geq0\\ i_1+2i_2+\dots+mi_m=m}}\frac{(-1)^{i_1+\dots+i_m+1}}{i_1+\dots+i_m}\binom{i_1+\dots+i_m}{i_1,\dots,i_m}h_1^{i_1}\cdots h_m^{i_m}z^m,
\end{aligned}$$

so the coefficient is

$$\frac{(-1)^{k_1+\dots+k_m+1}(k_1+\dots+k_m)!}{k_1!\cdots k_m!}.$$

For the last identity, we have regrouped all the terms by the value of $m := j_1 + \cdots + j_k$ and then by $\{i_j := |\{t \mid j_t = j\}|\}_{j=1}^m$, that is, the number of times that each h_j appears rather than the indexes of those that do. When doing this, we note that $k = i_1+\cdots+i_m$ and that $h_{j_1}\cdots h_{j_k} = h_1^{i_1}\cdots h_m^{i_m}$, so all the addends in a given group are the same, and the number of addends is the number of orderings of the k indexes without considering the order of indexes that are equal, of which there are i_1 indexes equal to 1, i_2 equal to 2, etc., and this is a multinomial coefficient.

▸ **12.** [*M20*] Suppose we have a doubly subscripted sequence $\langle a_{mn}\rangle$ for $m, n = 0, 1, \dots$; show how this double sequence can be represented by a *single* generating function of two variables, and determine the generating function for $\langle\binom{n}{m}\rangle$.

A generating function in this case could be something like

$$G(y,z) := \sum_{m,n\geq0}a_{mn}y^mz^n = \sum_{n\geq0}\left(\sum_{m\geq0}a_{mn}y^m\right)z^n = \sum_{m\geq0}\left(\sum_{n\geq0}a_{mn}z^n\right)y^m.$$

In our case, by the binomial theorem,

$$(1+y)^n = \sum_{m\ge 0}\binom{n}{m}y^m,$$

so

$$G(y,z) = \sum_{n\ge 0}(1+y)^n z^n = \frac{1}{1-(1+y)z}.$$

▸ **18.** [*M25*] Given positive integers n and r, find a simple formula for the value of the following sums:

1. $\sum_{1\le k_1<k_2<\cdots<k_r\le n} k_1k_2\cdots k_r$;
2. $\sum_{1\le k_1\le k_2\le\cdots\le k_r\le n} k_1k_2\cdots k_r$.

(For example, when $n=3$ and $r=2$ the sums are, respectively, $1\cdot2+1\cdot3+2\cdot3$ and $1\cdot1+1\cdot2+1\cdot3+2\cdot2+2\cdot3+3\cdot3$.)

1. Following Exercise 10 and then applying Equation (27),

$$\begin{aligned}G(z) &:= \sum_{r\ge 0}\sum_{1\le k_1<\cdots<k_r\le n} k_1\cdots k_r = \prod_{k=1}^{n}(1+kz) = z^n\prod_{k=1}^{n}\left(\frac{1}{z}+k\right)\\ &= z^{n+1}\prod_{k=0}^{n}\left(\frac{1}{z}+k\right) = z^{n+1}\sum_{k\ge 0}\left[{n+1 \atop k}\right]z^{-k}\\ &= \sum_{k=0}^{n+1}\left[{n+1 \atop k}\right]z^{n+1-k} = \sum_{k=0}^{n+1}\left[{n+1 \atop n+1-k}\right]z^k,\end{aligned}$$

so the sum is $\left[{n+1 \atop n+1-r}\right]$.

2. Doing something similar with Equations (36) and (28),

$$\begin{aligned}G(z) &:= \sum_{r\ge 0}\sum_{1\le k_1\le\cdots\le k_r\le n} k_1\cdots k_r = \prod_{k=1}^{n}\frac{1}{1-kz} = \frac{1}{z^n}\sum_{k\ge n}\left\{{k \atop n}\right\}z^k\\ &= \sum_{k}\left\{{k+n \atop n}\right\}z^k,\end{aligned}$$

so the sum is $\left\{{n+r \atop n}\right\}$.

▸ **25.** [*M23*] Evaluate the sum $\sum_k\binom{n}{k}\binom{2n-2k}{n-k}(-2)^k$ by simplifying the equivalent formula $\sum_k[w^k](1-2w)^n[z^{n-k}](1+z)^{2n-2k}$.

By the binomial theorem,

$$(1-2w)^n = \sum_{k\ge 0}\binom{n}{k}(-2)^k w^k, \qquad (1+z)^{2n-2k} = \sum_{j\ge 0}\binom{2n-2k}{j}z^j,$$

so the second formula is indeed equivalent to the first, assuming of course that $n \in \mathbb{N}$ and therefore terms with negative k or $k > n$ are null. Now, $[z^{n-k}](1+z)^{2n-2k} = [z^n](z^k(1+z)^{2n-2k})$, so this is

$$S := [z^n](1+z)^{2n}\sum_k [w^k](1-2w)^n\left(\frac{z}{(1+z)^2}\right)^k = [z^n](1+z)^{2n}\left(1-\frac{2z}{(1+z)^2}\right)^n,$$

where we evaluating the sum by taking $\frac{z}{(1+z)^2}$ as the argument of the generating function in w. Then, we simplify to get

$$S = [z^n]\left(\left((1+z)^2(1-{}^{2z}/{}_{(1+z)^2})\right)^n\right) = [z^n]\left((1+z^2)^n\right) = \binom{n}{n/2}[n \text{ even}].$$

1.2.10 Analysis of an Algorithm

1. [*10*] Determine the value of p_{n0} from Eqs. (4) and (5) and interpret this result from the standpoint of Algorithm M.

For $n = 1$, $p_{10} = 1$, and for $n > 1$,

$$p_{n0} = \frac{1}{n}p_{(n-1)(-1)} + \frac{n-1}{n}p_{(n-1)0} = \frac{n-1}{n}p_{(n-1)0} = \cdots = \frac{n-1}{n}\frac{n-2}{n-1}\cdots\frac{1}{2}p_{10} = \frac{1}{n},$$

so in general the probability that step M4 is never run (which happens when a_n is already the maximum value) is $p_{n0} = \frac{1}{n}$.

4. [*M10*] Give an explicit, closed formula for the values of p_{nk} in the coin-tossing experiment, Eq. (17).

$p_{nk} = \binom{n}{k}p^kq^{n-k}$. To show that this derives from Eq. (17), we see that, when $n = 0$, $p_{0k} = \delta_{k0}$, matching the formula, and then, by induction,

$$p_{nk} = pp_{n-1,k-1} + qp_{n-1,k} = \binom{n-1}{k-1}p^kq^{n-k} + \binom{n-1}{k}p^kq^{n-k} = \binom{n}{k}p^kq^{n-k}.$$

▸ **8.** [*M20*] Suppose that each $X[k]$ is taken at random from a set of M distinct elements, so that each of the M^n possible choices for $X[1], X[2], \dots, X[n]$ is considered equally likely. What is the probability that all the $X[k]$ will be distinct?

We need calculate the number of choices out of those M^n that do not repeat numbers. These choices can be considered as taking a subset of n elements from those M (if $n > M$ then we must necessarily repeat) and then ordering them somehow, so the total number of choices is $\binom{M}{n}n!$, and the probability is

$$\frac{\binom{M}{n}n!}{M^n} = \frac{M(M-1)\cdots(M-n+1)}{M^n} = \frac{M^{\underline{n}}}{M^n}.$$

▸ **11.** [*M15*] What happens to the semi-invariants of a distribution if we change $G(z)$ to $F(z) = z^n G(z)$?

Let $H(z) := z^n$, then $h(t) := \ln H(\mathrm{e}^t) = \ln \mathrm{e}^{nt} = nt$, so $\dot{h}(t) = n$ and $\ddot{h}(t) = \dddot{h}(t) = \cdots = 0$. Therefore

$$\kappa_1 = \dot{h}(0) = n; \qquad \kappa_n = h^{(n)}(0) = 0, \qquad n > 1.$$

Thus, by Theorem A, the mean increases by n and all the other semi-invariants stay the same.

▸ **20.** [*M22*] Suppose we want to calculate $\max\{|a_1 - b_1|, |a_2 - b_2|, \ldots, |a_n - b_n|\}$ when $b_1 \le b_2 \le \cdots \le b_n$. Show that it is sufficient to calculate $\max\{m_\mathrm{L}, m_\mathrm{R}\}$, where

$$m_\mathrm{L} = \max\{a_k - b_k \mid a_k \text{ is a left-to-right maximum of } a_1 a_2 \cdots a_n\},$$
$$m_\mathrm{R} = \max\{b_k - a_k \mid a_k \text{ is a right-to-left minimum of } a_1 a_2 \cdots a_n\}.$$

(Thus, if the a's are in random order, the number of k's for which a subtraction must be performed is only about $2 \ln n$.)

We interpret the concepts of "directional" maximum and minimum as meaning that $a_k \ge a_{k-1}, \ldots, a_1$ or that $a_k \le a_{k+1}, \ldots, a_n$, respectively. Then, if a_j is not a left-to-right maximum, there is a $k < j$ with $a_k > a_j$ and therefore $a_k - b_k > a_j - b_j$ (as $b_k \le b_j$), so $a_j - b_j$ is not a maximum of $\{a_k - b_k\}$. Similarly, if a_j is not a right-to-left minimum, there is a $k > j$ with $a_k < a_j$ and $b_k - a_k > b_j - a_j$.

Thus $m_\mathrm{L} = \max\{a_k - b_k\}$ and $m_\mathrm{R} = \max\{b_k - a_k\}$. Since at least one of them is non-negative, $\max\{m_\mathrm{L}, m_\mathrm{R}\}$ is also non-negative and it is therefore $|a_k - b_k|$ for some k, and it is no lower than any $|a_j - b_j|$ as those are either $a_j - b_j$ or $b_j - a_j$.

▸ **21.** [*HM21*] Let X be the number of heads that occur when a random coin is flipped n times, with generating function (18). Use (25) to prove that

$$\Pr(X \ge n(p + \epsilon)) \le \mathrm{e}^{-\epsilon^2 n/(2q)}$$

when $\epsilon \ge 0$, and obtain a similar estimate for $\Pr(X \le n(p - \epsilon))$.

First, we tackle the edge cases. If $n = 0$, then $X = 0$ and $\Pr(X \ge 0) \le 1$ trivially, so we may consider $n > 0$. If $p = 0$, then $\Pr(X = 0) = 1$ and $\Pr(X \ge n\epsilon) \le \mathrm{e}^{-\epsilon^2 n/2}$ trivially, so we may consider $p > 0$. Finally, if $\epsilon > q$ then $\Pr(X \ge n(p + \epsilon)) = 0 \le \mathrm{e}^{-\epsilon^2 n/(2q)}$ and if $\epsilon = q$ then $\Pr(X \ge n) = p^n \le (\mathrm{e}^{-q})^n \le \mathrm{e}^{-qn/2}$, using the fact that $t \le \mathrm{e}^{t-1}$ for every $t \in \mathbb{R}$, so we may consider $\epsilon < q$ and in particular $q > 0$.

Now let $r = n(p + \epsilon)$. Then $\Pr(X \ge n(p + \epsilon)) \le x^{-n(p+\epsilon)}(q + px)^n$, and we just need to find $x \ge 1$ such that

$$\ln(q + px) - (p + \epsilon) \ln x \le -\frac{\epsilon^2}{2q}.$$

(I had to look at the solution, esp. for the right value of x.) If $x := \frac{p+\epsilon}{p}\frac{q}{q-\epsilon}$, we have

$$\begin{aligned}\ln(q+px)-(p+\epsilon)\ln x &= \ln\left(q+(p+\epsilon)\frac{q}{q-\epsilon}\right)-(p+\epsilon)\ln\left(\frac{p+\epsilon}{p}\frac{q}{q-\epsilon}\right) = \\ &= \ln q-\ln(q-\epsilon)-(p+\epsilon)(\ln q-\ln(q-\epsilon)+\ln(p+\epsilon)-\ln p) = \\ &= (q-\epsilon)\ln\frac{q}{q-\epsilon}-(p+\epsilon)\ln\frac{p+\epsilon}{p},\end{aligned}$$

where we are assuming that $\epsilon < q$. Now, by Eq. 1.2.9(24), let $v := \frac{\epsilon}{q}$,

$$\begin{aligned}(q-\epsilon)\ln\frac{q}{q-\epsilon} &= q(1-v)\ln\frac{1}{1-v} = q(v-1)\ln(1-v) = \\ &= q(v-1)\sum_{k\geq1}\frac{(-1)^{k+1}}{k}(-v)^k = q(1-v)\sum_{k\geq1}\frac{v^k}{k} = q\left(\sum_{k\geq1}\frac{v^k}{k}-\sum_{k\geq2}\frac{v^k}{k-1}\right) = \\ &= q\left(v-\sum_{k\geq2}\frac{v^k}{k(k-1)}\right) = \epsilon-\frac{\epsilon^2}{2q}-\frac{\epsilon^3}{6q^2}-\frac{\epsilon^4}{12q^3}-\cdots\leq\epsilon-\frac{\epsilon^2}{2q}.\end{aligned}$$

In addition,

$$-(p+\epsilon)\ln\frac{p+\epsilon}{p} = (p+\epsilon)\ln\frac{p}{p+\epsilon} = (p+\epsilon)\ln\left(1-\frac{\epsilon}{p+\epsilon}\right)\leq-\epsilon,$$

so $-(p+\epsilon)\ln\frac{p+\epsilon}{p}\leq\epsilon$ and, putting it all together,

$$\ln(q+px)-(p+\epsilon)\ln x\leq\cancel{\epsilon}-\frac{\epsilon^2}{2q}\cancel{+\epsilon}.$$

For the second part, switching the roles of heads and tails, let $Y := n-X$,

$$\Pr(X\leq n(p-\epsilon)) = \Pr(Y\geq n(q+\epsilon))\leq \mathrm{e}^{-\epsilon^2n/(2p)}.$$

▸ **22.** [*HM22*] Suppose X has the generating function $(q_1+p_1z)(q_2+p_2z)\cdots(q_n+p_nz)$, where $p_k+q_k=1$ for $1\leq k\leq n$. Let $\mu = EX = p_1+p_2+\cdots+p_n$.

1. Prove that

$$\begin{aligned}&\Pr(X\leq\mu r)\leq(r^{-r}\mathrm{e}^{r-1})^\mu, &&\text{when } 0<r\leq1;\\ &\Pr(X\geq\mu r)\leq(r^{-r}\mathrm{e}^{r-1})^\mu, &&\text{when } r\geq1.\end{aligned}$$

2. Express the right-hand sides of these estimates in convenient form when $r\approx1$.

3. Show that if r is sufficiently large we have $\Pr(X\geq\mu r)\leq2^{-\mu r}$.

1. For the first equation, using Eq. (24) with $x = r$,

$$\Pr(X \leq \mu r) \leq r^{-\mu r}(q_1 + p_1 r) \cdots (q_n + p_n r) = r^{-\mu r} \prod_k (q_k + p_k r) =$$
$$= r^{-\mu r} \prod_k (1 + p_k(r-1)) \leq r^{-\mu r} \prod_k \mathrm{e}^{p_k(r-1)} = r^{-\mu r}\mathrm{e}^{\mu(r-1)} = (r^{-r}\mathrm{e}^{r-1})^{\mu}.$$

   For the second equation, repeat the same steps but using Eq. (25).

2. When $r \approx 1$, $r \approx \mathrm{e}^{r-1}$, so $(r^{-r}\mathrm{e}^{r-1})^{\mu} \approx r^{\mu(1-r)}$. The inequality still holds for $r < 2$ since, in the previous proof, we can see that $1 + p_k(r-1) \leq r^{p_k}$ by taking the Taylor series of the logarithms:

$$\ln(1 + p_k(r-1)) = p_k(r-1) - \frac{p_k^2(r-1)^2}{2} + \frac{p_k^3(r-1)^3}{3} - \frac{p_k^4(r-1)^4}{4} + \cdots \leq$$
$$\leq p_k(r-1) - \frac{p_k(r-1)^2}{2} + \frac{p_k(r-1)^3}{3} - \frac{p_k(r-1)^4}{4} + \cdots = p_k \ln r.$$

   Using this inequality instead of the one we used gives the proper result.

3. Clearly $1 + p_k(r-1) \leq r$, so $r^{-\mu r}\prod_k(1 + p_k(r-1)) \leq r^{-\mu r} r^n$, and we want to see that this in turn is less than $2^{-\mu r}$ for large enough r. Now,

$$\frac{r^n r^{-\mu r}}{2^{-\mu r}} = r^n \left(\frac{2}{r}\right)^{\mu r},$$

   and since $\left(\frac{2}{r}\right)^{\mu r}$ decreases faster than r^n increases, there exist r_0 such that, for $r > r_0$, this fraction is less than 1.

1.2.11 Asymptotic Representations

1.2.11.1 The O-notation

1. [*HM01*] What is $\lim_{n\to\infty} O(n^{-1/3})$?

0.

▸ **2.** [*M10*] Mr. B. C. Dull obtained astonishing results by using the "self-evident" formula $O(f(n)) - O(f(n)) = 0$. What was his mistake, and what should the right-hand side of his formula has been?

$O(f(n))$ is a set of functions, not a single function, so the two $O(f(n))$ do not cancel out (for example, the first $O(f(n))$ could represent $f(n)$ and the second one could be 0). The right hand side should be another $O(f(n))$.

▸ **4.** [*M15*] Give an asymptotic expansion of $n(\sqrt[n]{a}-1)$, if $a>0$, to terms $O(1/n^3)$.

We can do this by considering the function in terms of $x := \frac{1}{n}$, which would be $\frac{1}{x}(a^x-1)$. Then $a^x = e^{x\ln a} = 1 + x\ln a + \frac{1}{2}x^2\ln^2 a + \frac{1}{6}x^3\ln^3 a + O(x^4\ln^4 a)$, so

$$a^x - 1 = x\ln a + \frac{x^2\ln^2 a}{2} + \frac{x^3\ln^3 a}{6} + O(x^4) = \frac{\ln a}{n} + \frac{\ln^2 a}{2n^2} + \frac{\ln^3 a}{6n^3} + O(n^{-4})$$

and finally

$$n(\sqrt[n]{a}-1) = \ln a + \frac{\ln^2 a}{2n} + \frac{\ln^3 a}{6n^2} + O(n^{-3}).$$

▸ **6.** [*M20*] What is wrong with the following arguments? "Since $n=O(n)$, and $2n=O(n)$, ..., we have

$$\sum_{k=1}^{n} kn = \sum_{k=1}^{n} O(n) = O(n^2)."$$

The second expression does not make sense. It is a sum of n functions each of which has n as a parameter, but if n is a parameter, it cannot be an "external" parameter of the function represented by $O(n)$ as well. In this case, it is invalid to move the big O inside the sum as the range of the sum depends on n and so does the domain of values k can take, so k, which depends on n, cannot be treated like a constant.

▸ **11.** [*M11*] Explain why Eq. (18) is true.

The first identity is because $\sqrt[n]{n} = e^{\ln\sqrt[n]{n}} = e^{\ln n^{1/n}} = e^{\ln n/n}$, taking the power $1/n$ out of the logarithm. The second identity is a direct application of Eq. (12), using that $\ln n/n \to 0$.

▸ **13.** [*M10*] Prove or disprove: $g(n) = \Omega(f(n))$ if and only if $f(n) = O(g(n))$.

$g(n) = \Omega(f(n))$ if and only if there exist n_0 and L such that $|g(n)| \geq L|f(n)|$ for each $n \geq n_0$, but this is the same as saying that, for all such n, $|f(n)| \leq \frac{1}{L}|g(n)|$, which is to say that $f(n) = O(g(n))$. Since arguably this L must be positive (otherwise the statement is trivial), taking $\frac{1}{L}$ is valid, and likewise, if $|f(n)| \leq M|g(n)|$ for each $n \geq n_0$ and some M, we can always make this M positive to make the reverse argument.

1.2.11.2 Euler's summation formula

▸ **4.** [*HM20*] (*Sums of powers.*) When $f(x) = x^m$, the high-order derivatives of f are all zero, so Euler's summation formula gives an *exact* value for the sum

$$S_m(n) = \sum_{0\leq k<n} k^m$$

in terms of Bernoulli numbers. (It was the study of $S_m(n)$ for $m = 1, 2, 3, \ldots$ that led Bernoulli and Seki to discover those numbers in the first place.) Express $S_m(n)$ in terms of Bernoulli *polynomials*. Check your answer for $m = 0$, 1, and 2. (Note that the desired sum is performed for $0 \le k < n$ instead of $1 \le k < n$; Euler's summation formula may be applied with 0 replacing 1 throughout.)

For $k \ge 0$, we have $f^{(k)}(x) = m^{\underline{k}} x^{m-k}$. If $k < m$, this is nonzero except that $f^{(k)}(0) = 0$; if $k = m$, $f^{(m)}(x) \equiv 1$, and if $k > m$, $f^{(k)}(x) \equiv 0$. Using Euler's summation formula, when $m \ge 1$,

$$S_m(n) = \sum_{0 \le k < n} k^m = \int_0^n x^m \mathrm{d}x + \sum_{k=1}^{m} \frac{B_k}{k!} m^{\underline{k-1}} n^{m-k+1} = \frac{n^{m+1}}{m+1} + \sum_{k=1}^{m} \frac{B_k}{m-k+1} \binom{m}{k} n^{m-k+1}.$$

Then,

$$S'_m(n) = n^m + \sum_{k=1}^{m} B_k \binom{m}{k} n^{m-k} = \sum_k B_k \binom{m}{k} n^{m-k} = B_m(x),$$

which means that

$$\int_0^n B_m = (S_m(n) - S_m(0)) = S_m(n)$$

and therefore,

$$S_m(n) = \int_0^n B_m.$$

Now we check the solution, and in particular we check that this works for $m = 0$ as well:

$$\int_0^n B_0 = \int_0^n \mathrm{d}x = n - 0 = n, \qquad S_0(n) = \sum_{0 \le k < n} 1 = n;$$

$$\int_0^n B_1 = \int_0^n (x - \tfrac{1}{2})\mathrm{d}x = \frac{n^2 - n}{2}, \qquad S_1(n) = \sum_{0 \le k < n} k = \frac{n(n-1)}{2};$$

$$\int_0^n B_2 = \int_0^n (x^2 - x + \tfrac{1}{6})\mathrm{d}x = \frac{2n^3 - 3n^2 + n}{6},$$

and we check the last result we use induction. For $n = 0$, $S_2(0) = 0$. For $n \ge 0$,

$$\begin{aligned} S_2(n) &= \frac{1}{6}\left(2(n-1)^3 - 3(n-1)^2 + (n-1)\right) + (n-1)^2 \\ &= \frac{1}{6}\left(2n^3 \cancel{-6n^2} \cancel{+6n} \cancel{-2} - 3n^2 \cancel{+6n} \cancel{-3} + n \cancel{-1} \cancel{+6n^2} \cancel{-12n} \cancel{+6}\right) = \int_0^n B_2. \end{aligned}$$

▸ **9.** [*M25*] Find the asymptotic value of $\binom{2n}{n}$ with a relative error of $O(n^{-3})$, in two ways:

1. via Stirling's approximation;

2. via exercise 1.2.6–47 and Eq. 1.2.11.1–(16).

1. We have,
$$\binom{2n}{n} = \frac{(2n)!}{n!^2},$$
so
$$\ln\binom{2n}{n} = \ln(2n)! - 2\ln(n!) =$$
$$= (2n+\tfrac{1}{2})\ln(2n) - 2n + \sigma + \frac{1}{24n} - (2n+1)\ln n + 2n - 2\sigma - \frac{1}{6n} + O(n^{-3}) =$$
$$= (2n+\tfrac{1}{2})(\ln n + \ln 2) - (2n+1)\ln n - \sigma - \frac{1}{8n} + O(n^{-3}) =$$
$$= (2n+\tfrac{1}{2})\ln 2 - \frac{\ln n}{2} - \sigma - \frac{1}{8n} + O(n^{-3}),$$
and therefore
$$\binom{2n}{n} = \exp(\cdots) = \frac{2^{2n}}{\sqrt{\pi n}}\left(1 - \frac{1}{8n} + \frac{1}{128n^2} + O(n^3)\right).$$

2. (I had to look up the solution.)Exercise 1.2.6–47, when $r = -1/2$, the first identity simplifies to
$$(-1)^k\binom{-\frac{1}{2}}{k} = (-1)^k\binom{-k-1}{k}\Big/4^k = \binom{2k}{k}\Big/4^k.$$
Thus, using Eq. 1.2.6–(21),
$$\binom{2n}{n} = 4^n(-1)^n\binom{-\frac{1}{2}}{n} = 4^n\binom{n-\frac{1}{2}}{n} = 4^n\frac{\Gamma(n+\frac{1}{2})}{\Gamma(n+1)\Gamma(\frac{1}{2})} = 4^k\frac{\Gamma(n+\frac{1}{2})}{n\Gamma(n)\sqrt{\pi}} =$$
$$= \frac{4^n n^{\overline{1/2}}}{n\sqrt{\pi}} = \frac{2^{2n}}{n\sqrt{\pi}}\left(\left[{1/2 \atop 1/2}\right]n^{1/2} + \left[{1/2 \atop -1/2}\right]n^{-1/2} + \left[{1/2 \atop -3/2}\right]n^{-3/2} + O(n^{-5/2})\right).$$
Using Eqs. 1.2.6–(48), (49), and (57),
$$\left[{1/2 \atop 1/2}\right] = 1,$$
$$\left[{1/2 \atop -1/2}\right] = \binom{1/2}{2} = \frac{\frac{1}{2}(-\frac{1}{2})}{2} = -\frac{1}{8},$$
$$\left[{1/2 \atop -3/2}\right] = \binom{1/2}{4} + 2\binom{3/2}{4} = \frac{-\frac{15}{16}}{24} + 2\frac{\frac{9}{16}}{24} = \frac{3}{16\cdot 24} = \frac{1}{128},$$
so in the end,
$$\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}\left(1 - \frac{1}{8n} + \frac{1}{128n^2} + O(n^{-3})\right).$$

1.2.11.3 Some asymptotic calculations

▶ **4.** [*HM10*] Prove Eq. (13).

Since $u \geq 0$, $\ln(1+u) \geq 0$ and $u = v + \ln(1+u) \geq v$. On the other hand, x is fixed with respect to v, so by monotonicity of the integral, if $v \geq 1$,

$$0 \leq \int_r^\infty x\mathrm{e}^{-xv}\left(1+\frac{1}{u}\right)\mathrm{d}v \leq \int_r^\infty x\mathrm{e}^{-xv}\left(1+\frac{1}{v}\right)\mathrm{d}v \leq \int_r^\infty 2x\mathrm{e}^{-xv} =$$
$$= -2\mathrm{e}^{-xv}\Big|_{v=r}^{\infty} = 2\mathrm{e}^{-rx}.$$

This means $\int_r^\infty x\mathrm{e}^{-xv}\left(1+\frac{1}{u}\right)\mathrm{d}v = O(\mathrm{e}^{-rx})$.

▶ **6.** [*HM20*] Prove Eq. (23).

$$(n+\alpha)^{n+\beta} = n^{n+\beta}\left(1+\frac{\alpha}{n}\right)^{n+\beta}.$$

For the second factor,

$$\ln\left(1+\frac{\alpha}{n}\right)^{n+\beta} = (n+\beta)\ln\left(1+\frac{\alpha}{n}\right) = (n+\beta)\left(\frac{\alpha}{n}-\frac{\alpha^2}{2n^2}+O(n^{-3})\right) =$$
$$= \alpha + \frac{1}{n}\left(\alpha\beta - \frac{\alpha^2}{2}\right) + O(n^{-2}),$$

so

$$n^{n+\beta}\left(1+\frac{\alpha}{n}\right)^{n+\beta} = n^{n+\beta}\mathrm{e}^{\alpha}\exp\left(\frac{1}{n}\left(\alpha\beta-\frac{\alpha^2}{2}\right)+O(n^{-2})\right) =$$
$$n^{n+\beta}\mathrm{e}^{\alpha}\left(1+\frac{\alpha}{n}\left(\beta-\frac{\alpha}{2}\right)+O(n^{-2})\right).$$

1.3 MIX

1.3.1 Description of MIX

1. [*00*] If MIX were a ternary (base 3) computer, how many "trits" would there be per byte?

The only power of 3 between 64 and 100 is $3^4 = 81$, so there would be 4 trits.

2. [*02*] If a value to be represented within MIX may get as large as 99999999, how many adjacent bytes should be used to contain this quantity.

Since $\lg 99999999 \in (26, 27)$, we'd need 27 bits, so 5 bytes.

3. [*02*] Give the partial field specifications, $(L : R)$, for the (a) address field, (b) index field, (c) field field, and (d) operation code field for a `MIX` instruction.

(0 : 2), (3 : 3), (4 : 4), and (5 : 5), respectively.

4. [*00*] The last example in (5) is '`LDA -2000,4`'. How can this be legitimate, in view of the fact that memory addresses should not be negative?

Because we're assuming that `r4` is at least 2000.

5. [*10*] What symbolic notation, analogous to (4), corresponds to (6) if (6) is regarded as a `MIX` instruction?

`DIV -80,3`.

▸ **6.** [*10*] Assume that location 3000 contains

$$\begin{array}{|c|c|c|c|c|}\hline + & 5 & 1 & 200 & 15 \\ \hline\end{array}$$

What is the result of the following instructions? (State if any of them are undefined or only partially defined.)

1. `LDAN 3000`;
2. `LD2N 3000(3:4)`;
3. `LDX 3000(1:3)`;
4. `LD6 3000`;
5. `LDXN 3000(0:0)`.

1. Set A to $\begin{array}{|c|c|c|c|c|}\hline - & 5 & 1 & 200 & 15 \\ \hline\end{array}$.
2. Set I2 to $\begin{array}{|c|c|}\hline - & 200 \\ \hline\end{array}$.
3. Set X to $\begin{array}{|c|c|c|c|c|c|}\hline + & 0 & 0 & 5 & 1 & ? \\ \hline\end{array}$ (the last byte is undefined).
4. Undefined.
5. Set X to $\begin{array}{|c|c|c|c|c|c|}\hline - & 0 & 0 & 0 & 0 & 0 \\ \hline\end{array}$.

▸ **9.** [*15*] List all the `MIX` operators that can possibly affect the setting of the overflow toggle. (Don not include floating point operators.)

`ADD`, `SUB`, `DIV`, `INCA`, `INCX`, `DECA`, `DECX`, `NUM`, `JOV`, `JNOV`.

▸ **11.** [*15*] List all the MIX operators that can possibly affect the setting of rI1.

MOVE, LD1, LD1N, INC1, DEC1, ENT1, ENN1.

12. [*10*] Find a single instruction that has the effect of multiplying the current contents of rI3 by two and leaving the result in rI3.

INC3 0,3.

▸ **13.** [*10*] Suppose location 1000 contains the instruction 'JOV 1001'. This instruction turns off the overflow toggle if it is on (and the next instruction executed will be in location 1001, in any case). If this instruction were changed to 'JNOV 1001', would there be any difference? What if it were changed to 'JOV 1000' or 'JNOV 1000'?

There wouldn't be any difference after changing it to 'JNOV 1001' aside from the contents of register J. If it's changed to 'JOV 1000', the only difference would be the timing, but if it's changed to 'JNOV 1000', an infinite loop would occur if the overflow toggle wasn't previously on.

15. [*10*] How many *alphameric characters* are there in a typewriter or paper-tape block? in a card-reader or card-punch block? in a line-printer block?

70 characters in a typewriter or paper-tape block, 80 in a card-reader or card-punch block, and 120 in a line-printer block. This is because each word can hold up to 5 characters.

▸ **18.** [*22*] After the following "number one" program has been executed, what changes to registers, toggles, and memory have taken place? (For example, what is the setting of rI1? of rX? of the overflow an comparison indicators?

```
STZ  1
ENNX 1
STX  1(0:1)
SLAX 1
ENNA 1
INCX 1
ENT1 1
SRC  1
ADD  1
DEC1 -1
STZ  1
CMPA 1
MOVE -1,1(1)
NUM  1
CHAR 1
HLT  1
```

Assuming the program doesn't start at 0000, the result is:

Position	Content					
Address 1	+	0	0	0	0	0
Address 2	+	0	0	0	0	0
rX	-	31	30	30	30	30
rA	-	30	30	30	30	30

Position	Content		
rI1	+	0	2
Overflow	ON		
Comparison	EQUAL		

▸ **19.** [*14*] What is the execution time of the program in the preceding exercise, not counting the `HLT` instruction?

42 units of time.

▸ **21.** [*24*]

1. Can the J-register ever be zero?
2. Write a program that, given a number N in rI4, sets register J equal to N, assuming that $0 < N \leq 3000$. Your program should start in location 3000. When your program has finished its execution, the contents of all memory cells must be unchanged.

1. Maybe at startup, but then only before the first jump is executed.

2.
   ```
   LDX   -1,4
   ENTA  3006
   SLA   3
   INCA  39 ; [|+||3006|0|0|39| -> JMP 3006]
   STA   -1,4
   JMP   -1,4
   STX   -1,4 ; position 3006
   HLT
   ```

▸ **22.** [*28*] Location 2000 contains an integer number, X. Write two programs that compute X^{13} and halt with the result in register A. One program should use the minimum number of `MIX` memory locations; the other should require the minimum execution time possible. Assume that X^{13} fits into a single word.

Minimum time, since the conditions imply $|X| \leq 5$:

```
LD1 2000
LDA 3500,1
```

Then at positions 3495 to 3505 we'd have a table of precomputed results. This uses a total of 2 words for the instructions and 11 words for the table.

Minimum number of memory locations, assuming the program starts at position 0:

```
LDA   2000
ENT1    12
MUL   2000
SLAX     5
DEC1     1
J1NZ     2
```

This uses a total of 6 words for the instructions but no additional memory.

▸ **25.** [*30*] Suppose that the manufacturer of MIX wishes to come out with a more powerful computer ("Mixmaster"?), and he wants to convince as many people as possible of those people now owning a MIX computer to invest in the more expensive machine. He wants to design this new hardware to be an *extension* of MIX, in the sense that all programs correctly written for MIX will work on the new machines without change. Suggest desirable things that could be incorporated in this extension. (For example, can you make better use of the I-field of an instruction?)

Some of the ways that MIX could be extended would be:

1. Make the machine work faster through pipelining and other techniques.

2. Allocate additional memory. A way to do this would be to expand the size of the index registers to be one full word, allowing for 1 Gword (equivalent to 4 GB by modern standards). Then we'd have 8 full-word registers, a bare minimum for modern processors.

3. For normal instructions, instead of having the I-field specify a range, it could be used to specify which of the six fields are being used, since we are warrantied at least six bits, allocating the corresponding floating point instructions in the F-field value that doesn't select any of the fields (the null value). For compatibility, we can't use the trivial binary representation for this, so we have to use a non-trivial mapping.

4. Many more floating-point instructions could be allocated in the null value of the I-field, such as square root, sine, cosine, etc. These could be allocated on instruction code 5 (special).

5. 8-bit compatibility would be desirable, but on a binary MIX, we only have 31 bits per word. One way to achieve compatibility is to add an extra bit per word that's not used by legacy MIX programs and extend the F-field in the following way: if the value in the field is at least 48, we take the 4 least significant bits and divide that into two groups: one for the lower part of the range,which would be specified in bytes, and one for the upper part.

   This would allow operating on bytes, and it would be incompatible with the previous proposal to extend the F-field. For other values of the I-field, the extra bit would be ignored.

6. In the case the 8-bit compatibility is implemented, it'd be nice to have a way to address individual bytes and half-words without case distinctions or code

Name	F	Description
`PSET`	1	Set the beginning of the privileged section to M. By default, it's 0. When an exception occurs, A is saved to the first word in the privileged section, the next word contains the reason of the exception, the next contains a pointer to the instruction that was being run and the next contains the start of the exception-processing code, to be run in privileged mode. Exceptions are disabled by default.
`PGET`	2	Copy the address of the privileged section start to rI1.
`PUSR`	3	Change to user (non-privileged) mode; jump to M and set J to 0.
`VCAL`	4	Change to kernel (privileged) code; like an exception, with the reason being 0 in 0:0.
`VSET`	5	Enable virtual memory, switch to the page table M and flush the TLB. Virtual memory addresses are 12-12-10 and each word in a table contains the address (1:4), an enable/disable flag (0:0), and read-write permissions (least significant 2 bits on 4:4), plus anything the OS wants to put in 5:5. When a page fault occurs, an exception happens with reason 1 in 0:0. Changes in an active page mapping have unspecified behavior when trying to access a page that depends on that before flushing the TLB.
`VFLS`	6	Flush the TLB. The I-field must be 0.
`VDIS`	7	Disable virtual memory and jump to M.
`EINT`	8	Enable/disable interrupts for the device in the I-field, depending on whether the sign is positive or negative. Interrupts are exceptions with reason 3 in 0:0 and the device ID in 1:1.
`AEX`	9	Enable/disable all exceptions, depending on whether the sign is positive or negative. The result of triggering an exception that's not an interrupt is unspecified.
`WAIT`	10	Wait until some device is ready.

Figure 1.1: System instructions for MIXMaster.

modifications. This could be done with a special value 6 (half-word) and 7 (byte) for the F-field in load and store instructions, which would use the full 32-bits of the specified index register.

7. Values 7 and 8 for the I-field could be used to mean registers A and X.

8. Values 16 to 24 for the I-field could be used for register to register arithmetic. Then the I-field minus 16 would be the destination operand and the source operand would be the one specified by the first part of the address with the fields given by the second part, with the same format than an F-field.

9. Memory protection and memory management could be implemented by changing the F-field of the `NOP` operation, with the instructions shown in table 1.1.

   All of those instructions except `VCAL` are illegal in user mode, along with `HLT`, `JBUS`, `IOC`, `IN`, `OUT`, and `JRED`. Instructions that had unspecified behavior now cause an exception with type 4 in 0:0, provided that exceptions are enabled. Nonetheless, the operating system is expected to treat those exceptions and act accordingly, usually by checking for authorization and simulating the effects of the instructions as if they were system calls.

10. Bitwise operations and shifts could be added.

11. An `IOC` number could be used to return the number of bytes per chunk for a device, and another number could be self-describing, to tell the type of the device. This would allow for more device types to be added, such as USB, Ethernet, PCI, video, timer, etc., given a standard for differentiating those.

1.3.2 The `MIX` Assembly Language

1. [*00*] The text remarked that '`X EQU 1000`' does not assemble any instruction that sets the value of a variable. Suppose that you are writing a `MIX` program in which the algorithm is supposed to set the value contained in certain memory cell (whose symbolic name is `X`) equal to 1000. How could you express this in `MIXAL`?

```
ENTA 1000
STA  X
```

▸ **2.** [*10*] Line 12 of program M says '`JMP *`', where `*` denotes the location of that line. Why doesn't the program go into an infinite loop, endlessly repeating this instruction?

Because, at the entry point of the function, the code self-modifies, changing that instruction to a jump to the next instruction of the routine that called program M.

▸ **3.** [*23*] What is the effect of the following program, if it is used in conjunction with Program M?

```
START IN   X+1(0)
      JBUS *(0)
      ENT1 100
1H    JMP  MAXIMUM
      LDX  X,1
      STA  X,1
      STX  X,2
      DEC1 1
      J1P  1B
      OUT  X+1(1)
      HLT
      END  START
```

It reads 100 numbers from tape 0 and prints them sorted in nondecreasing order to tape 1. It first reads the 100 numbers, and then it finds the maximum of the elements, exchanges it with the last element in the list and repeats that with the list made of of all the remaining elements. Finally, it outputs the result. This is known as *bubble sort*, the slowest sensible sorting algorithm, with complexity $O(n^2)$.

3000	+	0	0	18	35
3001	+	2051	0	5	9
3002	+	2050	0	5	10
3003	+	1	0	0	49
3004	+	499	1	5	26
3005	+	3016	0	1	41
3006	+	2	0	0	50
3007	+	2	0	2	51
3008	+	0	0	2	48
3009	+	0	2	2	55
3010	-	1	3	5	4
3011	+	3006	0	1	47
3012	-	1	3	5	56
3013	+	1	0	0	51
3014	+	3008	0	6	39
3015	+	3003	0	0	39
3016	+	1995	0	18	37
3017	+	2035	0	2	52
3018	-	50	0	2	53
3019	+	501	0	0	53

3020	-	1		5	5	8
3021	+	0		0	1	5
3022	+	0		4	12	31
3023	+	1		0	1	52
3024	+	50		0	1	53
3025	+	3020		0	2	45
3026	+	0		4	18	37
3027	+	24		4	5	12
3028	+	3019		0	0	45
3029	+	0		0	2	5
0000	+	2				
1995	+	6	9	19	22	23
1996	+	0	6	9	25	5
1997	+	0	8	24	15	4
1998	+	19	5	4	0	17
1999	+	19	9	14	5	22
2024	+	2035				
2049	+	2010				
2050	+	3				
2051	-	499				

Figure 1.2: Machine code for Program P.

▸ **4.** [*25*] Assemble Program P by hand. (It won't take as long as you think.) What are the actual numerical contents of memory, corresponding to that symbolic program?

See figure 1.2.

7. [*10*]

1. What is the meaning of '4B' in line 34 of Program P?
2. What effect, if any, would be cause if the location of line 15 were changed to '2H' and the address of line 20 were changed to '2B'?

1. It references the address at line 29, the first previous line with label '4H'.
2. Line 14 would reference line 15 instead of line 25 and line 24 would reference line 15 instead of line 12.

▸ **8.** [*24*] What does the following program do? (Do not run it on a computer, figure it out by hand!)

```
* MYSTERY PROGRAM
BUF ORIG *+3000
1H  ENT1 1
    ENT2 0
    LDX  4F
```

```
2H   ENT3 0,1
3H   STZ  BUF,2
     INC2 1
     DEC3 1
     J3P  3B
     STX  BUF,2
     INC2 1
     INC1 1
     CMP1 =75=
     JL   2B
     ENN2 2400
     OUT  BUF+2400,2(18)
     INC2 24
     J2N  *-2
     HLT
4H   ALF  AAAAA
     END  1B
```

An analysis shows that the code is not self-modifying and that it's equivalent to the following pseudo-C:

```
r2 = 0;
x = *"AAAAA";
for (r1 = 1; r1 < 75; r1++) {
      for (or3m75 = 0; or3m75 < r1; or3m75++)
            *r2++ = 0;
      *r2++ = x;
}
for (f2i24p100 = 0; r2i24p100 < 100; r2i24p100++)
      write(PRINTER, r2i24p100 * 24);
```

The program code begins at position 3000, and the programs writes until position

$$(2 + \cdots + 75) - 1 = (1 + \cdots + 74) + 74 - 1 = \frac{74 \cdot 75}{2} + 74 - 1 = \frac{74 \cdot 77}{2} - 1 = 2848,$$

so the code is indeed not self-modifying. For each iteration i from 1 to 74, the code stores $5i$ spaces and then 5 A's, and then the code prints the first 12000 characters from those.

As an aside, the same effect can be obtained with the following actual C code, provided there are no runtime errors:

```
#include <stdio.h>
#include <string.h>

int main()
{
      char buffer[15000];
      int i, j, k = 0;
```

```
     for (i = 1; i < 75; i++) {
          for (j = 0; j < i; j++)
               strcpy(buffer + 5*(k++), "     ");
          strcpy(buffer + 5*(k++), "AAAAA");
     }
     for (i = 0; i < 100; i++) {
          fwrite(buffer + 120*i, 120, 1, stdout);
          putchar('\n');
     }
     return 0;
}
```

▸ **9.** [*25*] Location `INST` contains a `MIX` word that purportedly is a `MIX` instruction. Write a `MIX` program that jumps to location `GOOD` if the word has a valid C-field, valid ±AA-field, valid I-field, and valid F-field, according to Table 1.3.1–1; your program should jump to location `BAD` otherwise. Remember that the test for a valid F-field depends on the C-field; for example, if C $= 7$ (`MOVE`), any F-field is acceptable, but if C $= 8$ (`LDA`), the F-field must have the form $8L + R$ where $0 \leq L \leq R \leq 5$. The "±AA"-field is to be considered valid *unless* C specifies an instruction requiring a memory address and I $= 0$ and ±AA is not a valid memory address.

Note: Inexperienced programmers tend to tackle a problem like this by writing a long series of tests on the C-field, such as '`LDA C`; `JAZ 1F`; `DECA 5`; `JAN 2F`; `JAZ 3F`; `DECA 2`; `JAN 4F`; ...'. This is *not* good practice! The best way to make multiway decisions is to prepare an auxiliary *table* containing information that encapsulates the desired logic. If there were, for example, a table of 64 entries, we could write '`LD1 C`; `LD1 TABLE,1`; `JMP 0,1`'—thereby jumping very speedily to the desired routine. Other useful information can also be kept in such a table. A tabular approach to the present problem makes the program only a little bit longer (including the table) and greatly increases its speed and flexibility.

```
ADDR  EQU 0:2
INDEX EQU 3:3
FIELD EQU 4:4
CODE  EQU 5:5
      LD1  INST(CODE)
      LD3  INST(INDEX) ; Load I-field
      ENTA -6,3        ; Check if I-field > 6
      JAP  BAD
      J3NZ 2F          ; I-field nonzero, skip AA test
      LDX  TABLE,1(0:0)
      ENTX 1(5:5)      ; 1 when AA is address, -1 otherwise
      JXN  2F
      LDA  INST(ADDR)
      JAN  BAD         ; Addr relevant, index 0, addr negative
2H    LDA  INST(FIELD) ; Field test: dispatch
      LD1  TABLE,1(4:5)
      JMP  0,1
BOUND CMPA TABLE,1(3:3); F-field in [0,CONTENTS(TABLE,1(3:3))]
      JL   GOOD
      JMP  BAD
RANSX CMPA =6=         ; F-field in range or 6
      JE   GOOD
RANGE SRAX 5    ; rAX is the F-field
      DIV  =8=  ; rA:rX is the F-field
      CMPA FIVE
      JG   BAD
      CMPX FIVE
      JG   BAD
      STA  *+1(0:2) ; rX -= rA
      DECX *
      JXN  BAD ; If rX < rA, dat's bad
      JMP  GOOD
; Format is Sign(0:0), F-field test (4:5), where Sign is
; positive iff the +/-AA field represents an address and
; F-field test jumps to GOOD if the F-field is valid for the
; given case or to BAD otherwise. Fields 1:3 may contain
; specific information
IODEV EQU  21(0:3),BOUND(4:5) ; Field is I/O device
JMPBY EQU  6(0:3),BOUND(4:5)  ; Field is Jx+
IMMAT EQU  4(0:3),BOUND(4:5)  ; Field is INCx/DECx/ENTx/ENNx
TABLE CON  -GOOD  ; NOP
      CON  RANSX  ; ADD/FADD
      CON  RANSX  ; SUB/FSUB
      CON  RANSX  ; MUL/FMUL
      CON  RANSX  ; DIV/FDIV
      CON  -3(0:3),BOUND(4:5) ; NUM/CHAR/HLT
      CON  -6(0:3),BOUND(4:5) ; SLA/SRA/SLAX/SRAX/SLC/SRC
      CON  GOOD   ; MOVE
      CON  RANGE  ; LDA
      CON  RANGE  ; LD1
      CON  RANGE  ; LD2
      CON  RANGE  ; LD3
      CON  RANGE  ; LD4
      CON  RANGE  ; LD5
      CON  RANGE  ; LD6
      CON  RANGE  ; LDX
      CON  RANGE  ; LDAN
      CON  RANGE  ; LD1N
      CON  RANGE  ; LD2N
      CON  RANGE  ; LD3N
      CON  RANGE  ; LD4N
      CON  RANGE  ; LD5N
      CON  RANGE  ; LD6N
      CON   RANGE  ; LDXN
      CON   RANGE  ; STA
      CON   RANGE  ; ST1
      CON   RANGE  ; ST2
      CON   RANGE  ; ST3
      CON   RANGE  ; ST4
      CON   RANGE  ; ST5
      CON   RANGE  ; ST6
      CON   RANGE  ; STX
      CON   RANGE  ; STJ
      CON   RANGE  ; STZ
      CON   IODEV  ; JBUS
      CON   -IODEV ; IOC
      CON   IODEV  ; IN
      CON   IODEV  ; OUT
      CON   IODEV  ; JRED
      CON   10(0:3),BOUND(4:5)  ; JMP/JSJ/JOV/JNOV/*
      CON   JMPBY  ; JA+
      CON   JMPBY  ; J1+
      CON   JMPBY  ; J2+
      CON   JMPBY  ; J3+
      CON   JMPBY  ; J4+
      CON   JMPBY  ; J5+
      CON   JMPBY  ; J6+
      CON   JMPBY  ; JX+
      CON   IMMAT  ; INCA/DECA/ENTA/ENNA
      CON   IMMAT  ; INC1/...
      CON   IMMAT  ; INC2/...
      CON   IMMAT  ; INC3/...
      CON   IMMAT  ; INC4/...
      CON   IMMAT  ; INC5/...
      CON   IMMAT  ; INC6/...
      CON   IMMAT  ; INC7/...
      CON   RANSX  ; CMPA/FCMP
      CON   RANGE  ; CMP1
      CON   RANGE  ; CMP2
      CON   RANGE  ; CMP3
      CON   RANGE  ; CMP4
      CON   RANGE  ; CMP5
      CON   RANGE  ; CMP6
      CON   RANGe  ; CMPX
FIVE  CON 5
```

1.3.3 Applications to permutations

1. [*02*] Consider the transformation of $\{0, 1, 2, 3, 4, 5, 6\}$ that replaces x by $2x$ mod 7. Show that this transformation is a permutation, and write it in cycle form.

Being a permutation follows from the fact that $\mathbb{Z}_7$ is a field. In cycle form, this would be $(1\,2\,4)(3\,6\,5)$.

2. [*10*] The text shows how we might set $(a, b, c, d, e, f) \to (c, d, f, b, e, a)$ by using a series of replacement operations $(x \leftarrow y)$ and one auxiliary variable t. Show how to do the job by using a series of *exchange* operations $(x \leftrightarrow y)$ and no auxiliary variables.

Since the permutation such that $(a, b, c, d, e, f) \to (c, d, f, b, e, a)$ such that $x_i \to \sigma(x_i)$ is $(a\,c\,f)(b\,d)$, we could use $a \leftrightarrow c, c \leftrightarrow f, b \leftrightarrow d$.

3. [*03*] Compute the product $\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}$, and express the answer in two-line notation.

$$\begin{pmatrix} a & b & c & d & e & f \\ d & b & f & c & a & e \end{pmatrix}.$$

4. [*10*] Express $(a\,b\,d)(e\,f)(a\,c\,f)(b\,d)$ as a product of disjoint cycles.

$(a\,d\,c\,f\,e)$

▸ **5.** [*M10*] Equation (3) shows several equivalent ways to express the same permutation in cycle form. How many different ways of writing that permutation are possible, if all singleton cycles are suppressed?

There are 2 cycles which can be ordered in $2! = 2$ different ways, and since an m-cycle can be written in m different ways, there are $2! \cdot 2 \cdot 3 = 12$ possible ways to write the permutation in cycle form.

7. [*10*] If Program A is presented with the input (6), what are the quantities X, Y, M, N, U, and V of (19)? What is the time required by Program A, excluding input-output?

X is the number of cards of input, namely $\lceil 30/24 \rceil = 2$ if no space is added; then $Y = 29$. $M = 5$, the number of cycles in input, and $N = 7$, the number of different elements. Then, since the output is $(a\,d\,g)(c\,e\,b)(f)$, $U = 3$, the number of cycles in the output, and $V = 1$, the number of singleton cycles.

Then the time required, excluding input-output, is

$$\begin{aligned} &112NX + 304X - 2M - Y + 11U + 2V - 11 = \\ &\qquad = 112 \cdot 7 \cdot 2 + 304 \cdot 2 - 2 \cdot 5 - 29 + 11 \cdot 3 + 2 \cdot 1 - 11 = 2161u. \end{aligned}$$

▸ **8.** [*23*] Would it be feasible to modify Algorithm B to go from left to right instead of from right to left through the input?

Yes, if we're allowed to use an extra step for inverting the result. For this algorithm, we use an auxiliary table $U[1], \ldots, U[n]$ where we obtain the inverse of the permutation. We calculate it by noticing that $(a_{11} \ldots a_{1m_1}) \cdots (a_{k1} \cdots a_{km_k}) = ((a_{km_k} \ldots a_{k1}) \cdots (a_{1m_1} \ldots a_{11}))^{-1}$.

B1. [Initialize.] Set $U[k] \to k$ for $1 \leq k \leq n$. Also, prepare to scan the input from left to right.

B2. [Next element.] Examine the next element of the input. If the input has been exhausted, go to step B5. If the element is a '(', set $Z \leftarrow 0$ and repeat step B2; if it is a ')', go to B4. Otherwise the element is x_i for some i, go on to B'3.

B3. [Change $U[i]$.] Exchange $Z \leftrightarrow U[i]$. If this makes $U[i] = 0$, set $j \leftarrow i$. Return to step B2.

B4. [Change $U[j]$.] Set $U[j] \leftarrow Z$. Return to step B2.

B5. [Invert permutation.] For $1 \leq k \leq n$, set $T[U[i]] \leftarrow i$.

9. [*10*] Both Programs A and B accept the same input and give the answer in essentially the same form. Is the output *exactly* the same under both programs?

No. For example, for input (6), Program A produces $(a\,d\,g)(c\,e\,b)(f)$ while Program B produces $(a\,d\,g)(b\,c\,e)(f)$.

▸ **13.** [*M24*] Prove that Algorithm J is valid.

We prove it by induction. Let σ be the original permutation, it's enough to show that, when entering step J2, or when exiting step J5, for each $j \in \{1, \ldots, n\}$, if $\sigma^{-1}(j) > m$, $X[j] = \sigma^{-1}(j)$, and otherwise $X[j] = -\sigma^{k+1}(j)$ where k is the least nonnegative integer with $\sigma^k(j) \leq m$ (if there were no such number, since $\sigma^{-1}(j) = \sigma^p(j)$ for some $p \geq 0$, then it would be $\sigma^{-1}(j) > m$).

If we enter J2 from J1, every $X[j] < 0$, every $j \leq m = n$, so $k = 1$ and what we have to prove is that $X[j] = -\sigma(j)$, which obviously holds. Now, provided that this holds when we enter J2, we have to show that it holds when we reach J5 and the proof is complete.

After steps J2 and J3, obviously the condition still holds. After these steps, $i = -\sigma(m)$ and j is such that $i = X[j]$. We prove this by induction, showing that, after $k \geq 1$ steps, i is either $-\sigma(m) < 0$ or $\sigma^{-k}(m) > 0$, and in the latter case, $\sigma^{-k}(m), \sigma^{-k+1}(m), \ldots, \sigma^{-1}(m) > m$.

After one iteration, $i = X[m]$; if $i > 0$, $i = \sigma^{-1}(m) > m$; otherwise $i = -\sigma^{p+1}(m)$, where p is the least nonnegative integer with $\sigma^p(m) \leq m$, but since $p = 0$, $i = -\sigma(m)$. Now assume that this still holds after k iterations, and that $i > 0$ so there's a $(k+1)$-th iteration. Then $j = \sigma^{-k}(m) > 0$ and we assign $X[j]$ to i. If $i > 0$, $i = \sigma^{-1}(\sigma^{-k}(m)) = \sigma^{-k-1}(m) > m$; otherwise $i = -\sigma^{p+1}(\sigma^{-k}(m))$, where p is the least nonnegative integer with $\sigma^{p-k}(m) \leq m$, but $\sigma^{-k}(m), \ldots, \sigma^{-1}(m) > m$ and $\sigma^0(m) = m$, so $p = k$ and $i = -\sigma^{k+1}(\sigma^{-k}(m)) = -\sigma(m)$.

Now, at the beginning of step J4, $i = -\sigma(m)$ and j is such that, if k is the least nonnegative integer with $\sigma^k(j) \leq m$, then $\sigma^{k+1}(j) = \sigma(m)$. Then the new value of $X[j]$ is $X[-i] = X[\sigma(m)] = -\sigma^{p+1}(m)$, where p is the least positive integer with $\sigma^p(m) \leq m$, the new value of $X[-i] = X[\sigma(m)]$ is $m = \sigma^{-1}(\sigma(m))$ and the new value of m is $m-1$. Since $\sigma^{-1}(\sigma(m)) > m-1$, the condition holds for $X[\sigma(m)] = X[-i]$:. Also, $X[j] = -\sigma^{p+1}(m) = -\sigma^{p+k+1}(j)$, $j, \sigma(j), \dots, \sigma^{k-1}(j), \sigma^k(j) = m, \sigma^{k+1}(j) = \sigma(m), \dots, \sigma^{p+k}(j) = \sigma^{p-1}(m) > m-1$, and because $\sigma^{p+k+1}(j) = \sigma^p(m) \leq m$, and $\sigma^p(m) \neq m$ because otherwise it would be either $p \leq k$ and $\sigma^{k-p}(j) = m\#$, or $p > k$ and $\sigma^p(j) = \sigma^{p-k}(m) = j \leq m\#$, then we have $\sigma^{p+k+1}(j) = \sigma^p(m) \leq m-1$, and the condition holds for $X[j]$.

Now the only thing left to prove is that step J3 always ends, for which it suffices to prove that the cycle that m is in contains an element t with $X[t] < 0$, which happens if and only if it contains a t with $\sigma^{-1}(t) \leq m$, if and only if there's an element not greater than m, but m is such an element.

▸ **26.** [*M24*] Extend the principle of inclusion and exclusion to obtain a formula for the number of elements that are in exactly r of the subsets $S_1, S_2, \dots, S_M$. (The text considers only the case $r = 0$.)

For the sake of abbreviation, let E_r be the number of elements that are exactly in r of the subsets and let

$$T_r \coloneqq \sum_{1 \leq j_1 < \dots < j_r \leq M} |S_{j_1} \cap \dots \cap S_{j_r}|,$$

so that $T_0 = N$ (we use the convention that the intersection of no subsets is the intended "domain" set), $T_M = |S_1 \cap \dots \cap S_M|$, and for $r > M$, $T_r = 0$.

Obviously $E_r = 0$ for $r > M$, and $E_M = T_M$. After calculating E_{M-1} and E_{M-2}, we formulate the hypothesis that

$$E_r = \sum_{k=r}^{M} (-1)^{k-r} \binom{k}{r} T_k$$

for $0 \leq r \leq M$, which matches what we know about E_M and E_0. We prove this by induction from $r = M$ to 0. We have already established the base case. For $r < M$, we start by pointing out that, for $r \leq k \leq M$, T_r counts the elements that appear exactly k times a total of $\binom{k}{r}$ times, corresponding to choosing the r indexes from those k indexes that contain the element. Thus $T_r = \sum_{k=r}^{M} \binom{k}{r} E_k$, so

$$\begin{aligned} E_r &= T_r - \sum_{k=r+1}^{M} \binom{k}{r} E_k = T_r - \sum_{k=r+1}^{M} \binom{k}{r} \sum_{j=k}^{M} (-1)^{j-k} \binom{j}{k} T_j \\ &= T_r - \sum_{j=r+1}^{M} \left(\sum_{k=r+1}^{j} (-1)^{j-k} \binom{j}{k} \binom{k}{r} \right) T_j, \end{aligned}$$

but by Eq. 1.2.6–(23),

$$\sum_{k=r+1}^{j} (-1)^{j-k} \binom{j}{k} \binom{k}{r} = \sum_{k=r}^{j} (-1)^{j-k} \binom{j}{k} \binom{k}{r} - (-1)^{j-r} \binom{j}{r} = \cancel{\binom{0}{r-j}}^{=0} - (-1)^{j-r} \binom{j}{r},$$

and so

$$E_r = T_r + \sum_{j=r+1}^{M} (-1)^{j-r}\binom{j}{r}T_j = \sum_{j=r}^{M}(-1)^{j-r}\binom{j}{r}T_j.$$

▸ **34.** [*M25*] (*Transposing blocks of data.*) One of the most common permutations needed in practice is the change from $\alpha\beta$ to $\beta\alpha$, where α and β are substrings of an array. In other words, if $x_0x_1\ldots x_{m-1} = \alpha$ and $x_mx_{m+1}\ldots x_{m+n-1} = \beta$, we want to change the array $x_0x_1\ldots x_{m+n-1} = \alpha\beta$ to the array $x_mx_{m+1}\ldots x_{m+n-1}x_0x_1\ldots x_{m-1} = \beta\alpha$; each element x_k should be replaced by $x_{p(k)}$ for $0 \le k < m+n$, where $p(k) = (k+m) \bmod (m+n)$. Show that every such "cyclic-shift" permutation has a simple cycle structure, and exploit that structure to device a simple algorithm for the desired rearrangement.

Let $d \coloneqq \gcd\{m,n\}$, then $p = C_0C_1\cdots C_{d-1}$ where

$$C_j \coloneqq (j\,(m+j)\,(2m+j)\,\ldots\,((\tfrac{n}{d}-1)m+j)) \pmod n.$$

Effectively, each of the cycles above contains only elements with the same value modulo d, so the cycles are disjoint, and for any $k \in \{0,\ldots,n-1\}$, let p and q be the quotient and remainder of k/d and $am+bn=d$ be a Bézout identity, then

$$k = pd + q = pam + pbn + q \equiv pam + q \equiv (pa \bmod n/d)m \pmod n.$$

We can use this in the following algorithm.
Algorithm T. (*Transpose blocks of data.*)

1. [Initialize.] Set $d \leftarrow m$, $N \leftarrow m+n$, and $j \leftarrow -1$.
2. [Calculate d.] Set $n \leftarrow n \bmod d$. If $n \ne 0$, set $d \leftrightarrow n$ and repeat this step.
3. [Set up cycle.] Increment j by 1. If $j = d$, terminate the algorithm. Otherwise set $k \leftarrow j$ and $s \leftarrow x_j$.
4. [Iterate.] Set $l \leftarrow (k+m) \bmod (m+n)$. If $l = j$, set $x_k \leftarrow s$ and go to the previous step. Otherwise set $x_k \leftarrow x_l$, $k \leftarrow l$, and repeat this step.

1.4 Some Fundamental Programming Techniques

1.4.1 Subroutines

1. [*10*] State the characteristics of subroutine (5), just as (4) gives the characteristics of Subroutine 1.3.2M.

Calling sequences:	`JMP MAX100` if $n = 100$, or `JMP MAXN`.
Entry conditions:	For `MAXN`, rI3 $= n$; $n \ge 1$.
Exit conditions:	rA $= \max_{k=1}^{n}$ `CONTENTS(X` $+ k$`)` $=$ `CONTENTS(X` $+$ rI2`)`; rI3 $= 0$.

2. [*10*] Suggest code to substitute for (6) without using the `JSJ` instruction.

```
MAX50    ENT3 50
         STJ  EXIT
         JMP  MAXN
EXIT     JMP  *
```

▸ **4.** [*21*] White a subroutine that generalizes `MAXN` by finding the maximum value of $\mathtt{X}[a], \mathtt{X}[a+r], \mathtt{X}[a+2r], \ldots, \mathtt{X}[n]$, where r and n are parameters and a is the smallest positive number with $a \equiv n \pmod r$, namely $a = 1 + (n-1) \bmod r$. Give a special entrance for the case $r = 1$. List the characteristics of your subroutine, as in (4).

```
MAXIMUM ENT1 1
MAXMOD  STJ  9F
        JMP  2F
1H      CMPA X,3
        JGE  3F
2H      ENT2 0,3
        LDA  X,3
3H      DEC3 0,1
        J3P  1B
9H      JMP  *
```

$$\left.\begin{array}{ll}\text{Calling sequences:} & \mathtt{JMP\ MAXIMUM} \text{ if } r = 1, \text{ or } \mathtt{JMP\ MAXMOD}.\\ \text{Entry conditions:} & \text{rI3} = n;\ \text{for } \mathtt{MAXMOD},\ \text{rI1} = r;\ n, r \geq 1.\\ \text{Exit conditions:} & \text{rA} = \max_{k=1}^{\lfloor (n-1)/r \rfloor} \mathtt{CONTENTS}(\mathtt{X} + n - kr) = \mathtt{CONTENTS}(\mathtt{X} + \text{rI2});\\ & \text{rI3} \leq 0.\end{array}\right\}$$

▸ **7.** [*20*] Why is self-modifying code now frowned upon?

Modern computers have separate instruction and data caches for efficiency, and typically several instructions are being processed at once, so modifying one of these instructions means emptying the instruction cache and pipeline, severely degrading performance.

These computers also support a security measure that works by preventing pieces of code intended to be executed from being written to, in order to avoid attackers executing code by overriding some buffer due to a bug in the application, and self-modifying code would need to have that protection disabled.

Third, typically these computers are running several processes or threads which may share code, and if the code is self-modifying we would need to keep several copies of it, which wastes memory.

Finally, self-modifying code is difficult to reason about unless it is restricted to a few well known patterns, but these patterns can be easily replaced by constructions like stacks which do not suffer from the problems above and allow techniques like recursion.

1.4.2 Coroutines

1. [*10*] Explain why short, simple examples of coroutines are hard for the author of a textbook to find.

Coroutines appear when there are several parts of the program coordinating among them, where each part performs a higher level task. While simple coroutines appear in higher level languages such as Python (in the form of generators), in languages such as assembly where they is no builtin support for them, they are more complex and slower than equivalent code that doesn't use coroutines, so their utility is not evident from these simple cases.

▸ **2.** [*20*] The program in the text starts up the `OUT` coroutine first. What would happen if `IN` were the first to be executed—that is, if line 60 where changed from '`JMP OUT1`' to '`JMP IN1`'?

The first parsed character would be ignored, since the `OUT` coroutine doesn't expect a character to be loaded already when it begins. Here `IN` acts effectively like a generator for `OUT`.

▸ **6.** [*22*] Give coroutine linkage analogous to (1) for the case of *three* coroutines, `A`, `B`, and `C`, each of which can jump to either of the other two. (Whenever a coroutine is activated, it begins where it last left off.)

In principle, this would require us to extend the code to *six* coroutine linkages, like below:

```
BTOA  STJ  BX     ATOB  STJ  AX     ATOC  STJ  AX
      JMP  AX           JMP  BX           JMP  CX
CTOA  STJ  CX     CTOB  STJ  CX     BTOC  STJ  BX
AX    JMP  A1     BX    JMP  B1           JMP  C1
```

We can then elide the code for jumps that never happen. When the number of coroutines is large, typically each one only communicates with the next or they communicate with a central, coordinating coroutine that decides which to call next based on some protocol (maybe coroutines store the index of the next coroutine to run in some register, or coroutines are polled in order based on which has input data available, for example.)

1.4.3 Interpretive Routines

1.4.3.1 A `MIX` simulator

▸ **3.** [*22*] Write the `MOVE` routine, which is missing from the program in the text (operation code 7).

The following is not tested.

```
MOVE    J3Z  CYCLE
        ENTA 0,3      ; Calculate time
        INCA 1,3
        STA  TIME(0:2)
        LD1  A1REG     ; Load destination position
        LDA  SIGN1     ; Check positive sign
        JAP  1H
        J1NZ MEMERROR
        STZ  SIGN1(0:0)
1H      J3Z  CYCLE     ; Check loop variable
        CMP1 =BEGIN=   ; Check destination
        JGE  MEMERROR
        CMP5 =BEGIN=   ; Check source
        JGE  MEMERROR
        LDA  0,5       ; Move value
        STA  0,1
        DEC3 1         ; Next iteration
        INC1 1
        INC5 1
        ST1  I1REG
        JMP  1B
```

▸ **5.** [*24*] Determine the time required to simulate the LDA and ENTA operators, compared with the actual time for MIX to execute these operators directly.

It takes 2 cycles to run an LDA instruction and 1 cycle to run an ENTA instruction. However, it takes 95 cycles to simulate LDA, plus 5 if the sign is not included in the fields to load, and 52 to simulate ENTA, plus 2 if the value to load is not zero, and we also have to add 15 cycles to both if the index field is nonzero. This *more or less* matches the solution but I'm not spending any more effort on *counting cycles.*

1.4.3.2 Trace Routines

3. [*10*] The previous exercise suggests having the trace program write its output onto tape. Discuss why this would be preferable to printing it directly.

The output of the tracer would be large, and so it would need to be analyzed by another program in order to focus on interesting parts or gather some statistics.

▸ **4.** [*25*] What would happen if the trace routine were tracing *itself*? Specifically, consider the behavior if the two instructions ENTX LEAVEX; JMP *+1 were placed just before ENTER.

Unfortunately this wouldn't work, as the registers A and J of the tracing tracing program and the traced traced program would be stored in the same addresses.

▸ **7.** [*25*] Discuss how to write an efficient *jump trace* routine, which emits much less output than a normal trace. Instead of displaying the register contents, a jump trace simply records the jumps that occur. It outputs a sequence of pairs $(x_1, y_1), (x_2, y_2), \ldots,$

meaning that the program jumped from location x_1 to y_1, then (after performing the instructions in locations $y_1, y_1 + 1, \ldots, x_2$) it jumped from x_2 to y_2, etc.

We could save this information right at the `JUMP` label. At this point, `PREG` contains the current instruction and `INST1` contains the instruction to jump to, so it should be easy to move those to values to a buffer and call a subroutine to write to tape when the buffer is full.

If the code wasn't self-modifying, we could just scan up to the next jump instruction, modify that one, and run up to that point, and the solution from the book specifies just that except that it allows `STJ` to modify jump calls by handling this case separately during the scan. Since modern processors can raise an exception when modifying their own code (W^X), it should be possible to save all the jump calls in some table and substitute them by calls to a debugger routine before running, accepting that the program may detect that it's being debugged by looking at its own jump calls.

1.4.4 Input and Output

1. [*05*]

1. Would sequence (3) still be correct if the `MOVE` instructions were placed before the `JBUS` instruction instead of after it?
2. What if the `MOVE` instructions were placed after the `IN` command?

1. No, because they would be reading the old data.
2. No, because the new data would overwrite the old data while we are reading it. It would probably turn out okay since the new data is probably being read at a slower pace than the `MOVE` instructions, but it's not wise to bet on it.

2. [*10*] The instructions '`OUT 1000(6); JBUS *(6)`' may be used to output a tape block in an unbuffered fashion, just as the instructions (1) did this for input. Give a method analogous to (2) and (3) that buffers this output, by using `MOVE` instructions and an auxiliary buffer in locations 2000–2099.

Every time a new block of data is ready in locations 1000–1099, we want to clear it to allow other operations to write on it and write it to tape in the meantime, making sure that the previous block has finished writing before that. Therefore we could just run the following after each block is ready:

```
ENT1 2000
JBUS *(6)
MOVE 1000(50)
MOVE 1050(50)
OUT  2000(6)
```

▸ **3.** [*22*] Write a buffer-swapping output subroutine analogous to (4). The subroutine, called `WORDOUT`, should store the word in rA as the next word of output, and if a buffer is full it should write 100 words onto tape unit `V`. Index register 5 should be used to refer to the current buffer position. Show the layout of buffer areas and explain what instructions (if any) are necessary at the beginning and end of the program to ensure that the first and last blocks are properly written. The final block should be filled out with zeros if necessary.

The following code is not tested.

```
WORDOUT STJ  1F
        STA  0,5
        INC5 1
2H      LDA  0,5
        DECA SENTINEL
1H      JAZ  *
        OUT  -100,5(V)
        LD5  1,5
        JMP  2B
OUTBUF1 ORIG *+100
        CON  SENTINEL
        CON  OUTBUF2
OUTBUF2 ORIG *+100
        CON  SENTINEL
        CON  OUTBUF1
```

At the start of the program, we would need to `ENT5 OUTBUF1`. At the end, we have to fill the current block with zeroes and write it *unless* we are at the beginning of the block so we have already written what we had to, and then we had to wait for the remaining write to complete. We can do this with the following code:

```
        DEC5 OUTBUF1
        J5Z  9H
        DEC5 102
        J5Z  9H
1H      STZ  0,5
        INC5 1
        LDA  0,5
        DECA SENTINEL
        JANZ 1B
        OUT  -100,5(U)
9H      JBUS *(5)
```

Knuth's version is 2 words shorter and very slightly faster by comparing addresses rather than contents in a couple places.

4. [*M20*] Show that if a program refers to a single I/O device, we might be able to cut the running time in half by buffering the I/O, in favorable circumstances; but we can never decrease the running time by more than a factor of two, with respect to the time taken by unbuffered I/O.

See the next exercise.

▸ **5.** [*M21*] Generalize the situation of the preceding exercise to the case when the program refers to n I/O devices instead of just one.

Let's assume that reads and writes on devices are so slow that the time taken for the CPU to run the I/O operations is negligible. Let's also assume that the time it takes for the CPU to complete an iteration of the program is the same time that it takes for each device to read or write a block, that this time is the same for all devices, and that each iteration reads or writes a block on each device at the beginning which is only needed by the next iteration, if at all. Moreover, we assume that there are enough iterations so that the time needed for initialization and finalization is negligible.

Then the unbuffered case takes $MN(n+1)$ cycles, where M is the number of iterations and N is the number of cycles per iteration, while the buffered case takes MN, so in this optimal case, the time is divided by $n+1$.

On the other hand, if we have a general program where the CPU takes N_0 cycles and the kth device takes N_k cycles for $k = 1, \dots, n$, then the unbuffered version takes $N_0 + N_1 + \cdots + N_n$ cycles and the buffered version takes at least $\max\{N_0, \dots, N_n\} \geq \frac{N_0+\cdots+N_n}{n+1}$ cycles, so we cannot go lower than this limit.

▸ **9.** [*21*] A program that leads to the buffer contents shown in figure 1.3 may be characterized by the following list of times:

$$A,\ 1000,\ R,\ 1000,\ A,\ 1000,\ R,\ 1000,\ A,\ 1000,\ R,\ 1000,\ A,\ 1000,\ R,\ 1000,$$
$$A,\ 1000,\ R,\ 5000,\ A,\ 7000,\ R,\ 5000,\ A,\ 7000,\ R,\ 5000,\ A,\ 7000,\ R,\ 5000,$$
$$A,\ 1000,\ R,\ 1000,\ A,\ 2000,\ R,\ 1000.$$

This list means "assign, compute for 1000 u, release, compute for 1000 u, assign, ..., compute for 2000 u, release, compute for 1000 u." The computation times given do not include any intervals during which the computer might have to wait for the output device to catch up (as at the fourth "assign" in figure 1.3). The output device operates at a speed of 7500 u per block.

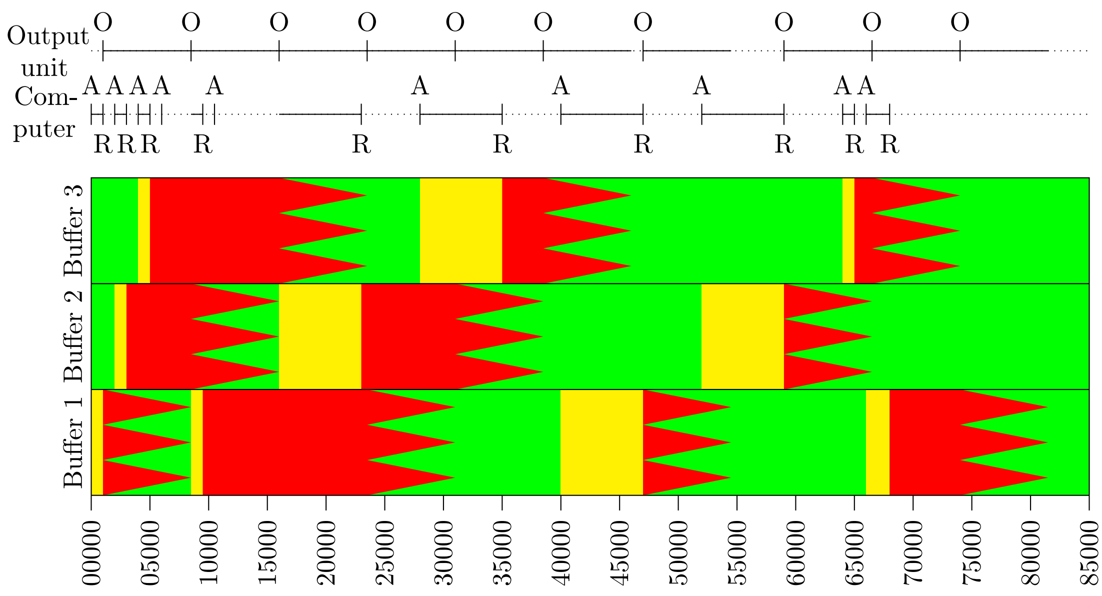


Figure 1.3: Output with two buffers (exercise 9).

The following chart specifies he actions shown in figure 1.3 as the time passes:

Time	Action	Time	Action
0	`ASSIGN(BUF1)`	38500	`OUT BUF3`
1000	`RELEASE, OUT BUF1`	40000	`ASSIGN(BUF1)`
2000	`ASSIGN(BUF2)`	46000	Output stops.
3000	`RELEASE`	47000	`RELEASE, OUT BUF1`
4000	`ASSIGN(BUF3)`	52000	`ASSIGN(BUF2)`
5000	`RELEASE`	54500	Output stops.
6000	`ASSIGN` (wait)	59000	`RELEASE, OUT BUF2`
8500	`BUF1` assigned, `OUT BUF2`	64000	`ASSIGN(BUF3)`
9500	`RELEASE`	65000	`RELEASE`
10500	`ASSIGN` (wait)	66000	`ASSIGN(BUF1)`
16000	`BUF2` assigned, `OUT BUF3`	66500	`OUT BUF3`
23000	`RELEASE`	68000	`RELEASE`
23500	`OUT BUF1`	69000	Computation stops.
28000	`ASSIGN(BUF3)`	74000	`OUT BUF1`
31000	`OUT BUF2`	81500	Output stops.
35000	`RELEASE`		

The total time required was therefore 81500 u; the computer was idle from 6000–8500, 10500–16000, and 69000–81500, or 20500 u altogether; the output unit was idle from 0–1000, 46000–47000, and 54500–59000, or 6500 u.

Make a time-action chart like the above for the same program, assuming that there are only *two* buffers.

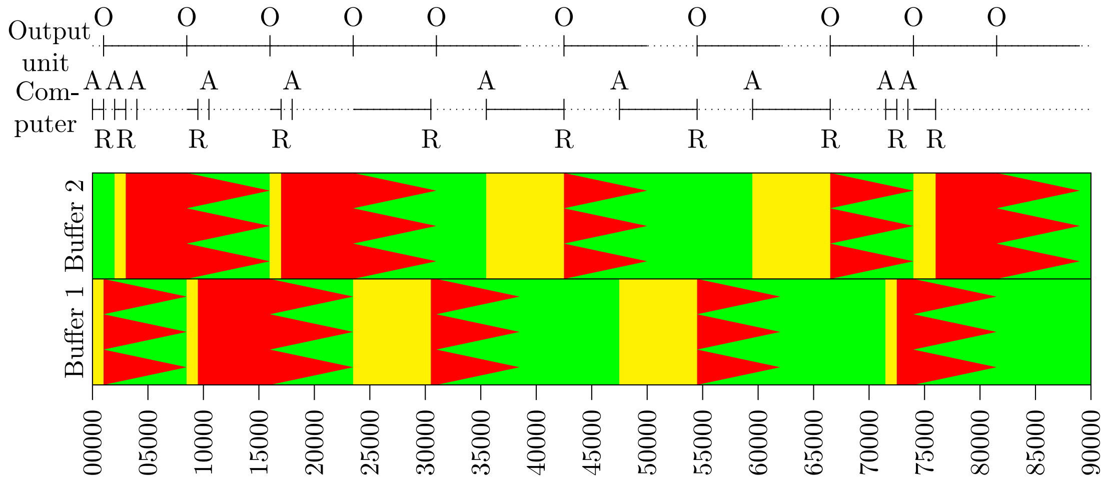


Figure 1.4: Output with three buffers (see exercise 9).

We can do this graphically as shown in figure 1.4, giving us the following chart:

Time	Action	Time	Action
0	`ASSIGN(BUF1)`	42500	`RELEASE, OUT BUF2`
1000	`RELEASE, OUT BUF1`	47500	`ASSIGN(BUF1)`
2000	`ASSIGN(BUF2)`	50000	Output stops.
3000	`RELEASE`	54500	`RELEASE, OUT BUF1`
4000	`ASSIGN` (wait)	59500	`ASSIGN(BUF2)`
8500	`BUF1` assigned, `OUT BUF2`	62000	Output stops.
9500	`RELEASE`	65500	`RELEASE, OUT BUF2`
10500	`ASSIGN` (wait)	71500	`ASSIGN(BUF1)`
16000	`BUF2` assigned, `OUT BUF1`	72500	`RELEASE`
17000	`RELEASE`	73500	`ASSIGN` (wait)
18000	`ASSIGN` (wait)	74000	`BUF2` assigned, `OUT BUF1`
23500	`BUF1` assigned, `OUT BUF2`	76000	`RELEASE`
30500	`RELEASE`	77000	Computation stops.
31000	`OUT BUF1`	81500	`OUT BUF2`
35500	`ASSIGN(BUF2)`	89000	Output stops.
38500	Output stops.		

▸ **14.** [*20*] Suppose the computational program does not alternate between `ASSIGN` and `RELEASE`, but instead gives the sequence of actions `...ASSIGN...ASSIGN...RELEASE...RELEASE`. What effect does this have on the algorithms described in the text? Is it possibly useful?

It just means we would need at least two buffers or the program would stall. It might be useful if the algorithm needs more data at a time than what fits in one buffer, so that it doesn't have to copy the data elsewhere.

▸ **15.** [*22*] Write a complete `MIX` program that copies 100 blocks from tape unit 0 to tape unit 1, using just three buffers. The program should be as fast as possible.

The following programs have not been tested. We need to use the buffers in a circle, checking input and output in a loop to see if the devices are ready and if we can move to the next buffer. The following program in pseudo-C might be illustrative:

```
void aread(char *buf), awrite(char *buf);  // IN, OUT
bool done_read(), done_write();  // JBUS/JRED
char buf[3][100];
char *cin = buf[0], *cout = buf[0], *next(char *buf);
int n = 100;  // Remaining blocks to input

aread(cin);
while (1) {
        if (done_read() && next(cin) != cout) {
                        if (--n == 0)
                                break;
                        cin = next(cin);
                        aread(cin);
                }
        }
        if (done_write() && next(cout) != cin) {
                        cout = next(cout);
                        awrite(cout);
        }
}
do  // Output remaining blocks
        awrite(cout = next(cout));
while (cout != cin);
```

We may translate this as follows:

```
        IN    BUF1(0)
        ENTA  100
        ENT1  BUF1  ; cin
        ENT2  BUF1  ; cout
READ    JBUS  WRITE(0)
        CMP2  100,1
        JEQ   WRITE
        DECA  1
        JAZ   EOUT
        LD1   100,1
        IN    0,1(0)
WRITE   JBUS  READ(1)
        CMP1  100,2
        JEQ   READ
        LD2   100,2
        OUT   0,2(1)
        JMP   READ
EOUT    LD2   100,2
        CMP2  100,1
        JAZ   END
        OUT   0,2(1)
        JMP   EOUT
END     HLT
BUF1    ORIG  *+100
        CON   BUF2
BUF2    ORIG  *+100
        CON   BUF3
BUF3    ORIG  *+100
        CON   BUF1
```

Now, the behavior of this program is always the same: first the input advances, then the output advances, and so on, so we might do better (and with a lower chance of bugs) just by coding it manually and using the blocking properties of `IN` and `OUT` with respect to other operations in the same device:

```
        IN   BUF1(0)
LOOP    ENTA 33
        IN   BUF2(0)
        OUT  BUF1(1)
        IN   BUF3(0)
        OUT  BUF2(1)
        IN   BUF1(0)
        OUT  BUF3(1)
        DECA 1
        JANZ LOOP
        JBUS *(0)
        OUT  BUF1(1)
        JBUS *(1)
        HLT
BUF1    ORIG *+100
BUF2    ORIG *+100
BUF3    ORIG *+100
```

1.5 MMIX

1.5.1 Description of MMIX

1.5.2 The MMIX Assembly Language

1.6 Some Fundamental Programming Techniques (MMIX)

1.6.1 Subroutines

1.6.2 Coroutines

1.6.3 Interpretive Routines

Chapter 2

Information structures

2.1 Introduction

1. [*04*] In the situation depicted in (3), what is the value of

1. `SUIT(NEXT(TOP))`;
2. `NEXT(NEXT(NEXT(TOP)))`?

1. 4.
2. Λ.

2. [*10*] The text points out that in many cases `CONTENTS`(`LOC`(`V`)) = `V`. Under what conditions do we have `LOC`(`CONTENTS`(`V`)) = `V`?

`V` must be a valid memory address different from Λ.

▸ **5.** [*21*] Give an algorithm that essentially undoes the effect of exercise 4: Assuming that the pile is not empty and that its bottom card is face down, your algorithm should remove the bottom card and make `NEWCARD` link to it. (This algorithm is sometimes called "cheating" in solitaire games.)

1. If `NEXT`(`TOP`) = Λ, let `NEWCARD` ← `TOP`, `TOP` ← Λ and terminate.
2. `X` ← `TOP`, `Y` ← `NEXT`(`TOP`).
3. If `NEXT`(`Y`) ≠ Λ, let `X` ← `Y`, `Y` ← `NEXT`(`Y`) and repeat this step.
4. Set `NEWCARD` ← `Y` and `NEXT`(`X`) ← Λ.

6. [*06*] In the playing card example, suppose that `CARD` is the name of a variable whose value is an entire node as in (6). The operation `CARD` ← `NODE(TOP)` sets the fields of `CARD` respectively equal to those of the top of the pile. After this operation, which of the following notations stands for the suit of the top card?

1. `SUIT(CARD);`
2. `SUIT(LOC(CARD));`
3. `SUIT(CONTENTS(CARD));`
4. `SUIT(TOP)?`

2 and 4.

▸ **7.** [*04*] In the text's example `MIX` program, (5), the link variable `TOP` is stored in the `MIX` computer word whose assembly language name is `TOP`. Given the field structure (1), which of the following sequences of code brings the quantity `NEXT(TOP)` into register A? Explain why the other sequence is incorrect.

1. `LDA TOP(NEXT)`
2. `LD1 TOP`
   `LDA 0,1(NEXT)`

2, since the first loads the field `NEXT` from the memory word `TOP`, that only contains the address, instead of from the node that `TOP` points to.

▸ **8.** [*18*] Write a `MIX` program corresponding to steps B1–B3.

```
NEXT   EQU   4:5
LLEN   STJ   CHECK
       ENTA  0          ; B1
       ENT1  TOP
2H     J1Z   *          ; B2
       INCA  1          ; B3
       LD1   0,1(NEXT)
       JMP   2B
```

2.2 Linear lists

2.2.1 Stacks, Queues, and Deques

1. [*06*] An input-restricted deque is a linear list in which items may be inserted at one end but removed from either end; clearly an input-restricted deque can operate either as a stack or as a queue, if we consistently remove all items from one of the two ends. Can an output-restricted deque also be operated either as a stack or as a queue?

Yes. If output happens on the left, we can use a deque as a queue by inserting on the right or as a stack by inserting on the left.

▸ **2.** [*15*] Imagine four railroad cars positioned on the input side of the track in Fig. 1, numbered 1, 2, 3, and 4, from left to right. Suppose we perform the following sequence of operations (which is compatible with the direction of the arrows in the diagram and does not require cars to "jump over" other cars): (i) move car 1 into the stack; (ii) move car 2 into the stack; (iii) move car 2 into the output; (iv) move car 3 into the stack; (v) move car 4 into the stack; (vi) move car 4 into the output; (vii) move car 3 into the output; (viii) move car 1 into the output.

As a result of these operations the original order of the cars, 1 2 3 4, has been changed into 2 4 3 1. *It is the purpose of this exercise and the following exercises to examine what permutations are obtainable in such a manner from stacks, queues, or deques.*

If there are six railroad cars numbered 1 2 3 4 5 6, can they be permuted into the order 3 2 5 6 4 1? Can they be permuted into the order 1 5 4 6 2 3? (In case it is possible, show how to do it.)

For 3 2 5 6 4 1, we push 1, push 2, push 3, pop 3, pop 2, push 4, push 5, pop 5, push 6, pop 6, pop 4, and pop 1. We can't get the permutation 1 5 4 6 2 3: first, we'd need to push 1 and pop it immediately since, otherwise, it couldn't be the first car to go out; then we need to push trains 2 to 5 without popping anything before the 5, so that no car is between the 1 and the 5, but then we must pop 3 before popping 2, so the order wouldn't be respected.

▸ **5.** [*M28*] Show that it is possible to obtain a permutation $p_1 p_2 \ldots p_n$ from $1\,2 \ldots n$ using a stack if and only if there are no indices $i < j < k$ such that $p_j < p_k < p_i$.

$\Longrightarrow$] Assume, by contradiction, that there are such indexes. When the next element to pop is p_i, p_i is either at the top of the stack or in the input road. In the second case, the only valid option is to push all elements from the input until reaching p_i and then pop p_i, so in either case, p_j and p_k end up in the stack after popping p_i. Then p_k has to remain in the stack until the next element is p_k, and in particular it has to remain after popping p_j, but since higher elements in the stack have higher values (this is easily proved by induction on the size of the stack), p_k is over p_j and thus it must be popped before popping p_j, giving an incorrect order to the output.

$\Longleftarrow$] Let's consider the following algorithm for generating such permutation: we set $m \leftarrow 1$, and for $j \leftarrow 1$ to n, if $p_j \geq m$, we push elements m to p_j, pop p_j and set $m \leftarrow p_j + 1$, and otherwise we just pop p_j. In this algorithm, m represents the next element to be pushed and j is such that p_j is to be popped, so $j \leq m$ at the beginning of every iteration. Thus, if $p_j \geq m$, the car is still in the input road and we have to push elements m to p_j, and otherwise p_j is in the stack.

We show that, in the latter case, if p_j weren't at the top of the stack, there would be indices $i < j$ and $k > j$ such that $p_j < p_k < p_i$. If p_j is in the stack but not as the top element, the latest operations have been pushing $m', m'+1, \ldots, p_i$ for some $i < j$ (m' is the value of m when processing i) and, after that, to popping $p_i, p_i - 1, \ldots, p_j + c$ for some $c \in \{2, \ldots, j - i\}$, since at least p_i is popped and at least $p_j + 1$ is not popped (otherwise, either p_j would have been popped too or it would be at the top).

Let k be the element such that $p_k = p_j + 1$, so that $p_j < p_k < p_i$. We have $i < j$ because p_i was popped earlier than p_j, and we have to show that $j < k$. If it were $k < j$, p_k would have already been popped, clearly before i, but then it would be $k < i$ and, because the value of m when processing k would be $m'' \leq m' \leq p_j < p_k$, we'd have pushed p_j when processing k, not when processing $i\#$. Since $k \neq j$, we conclude that $j < k$.

6. [*00*] Consider the problem of exercise 2, with a queue substituted for a stack. What permutations of $1\,2\,\ldots\,n$ can be obtained with use of a queue?

Just one; a queue would be represented by a bare straight line.

▸ **7.** [*25*] Consider the problem of exercise 2, with a deque substituted for a stack.

1. Find a permutation of 1 2 3 4 that can be obtained with an input-restricted deque, but it cannot be obtained with an output-restricted deque.
2. Find a permutation of 1 2 3 4 that can be obtained with an output-restricted deque but not with an input-restricted deque. [As a consequence of (1) and (2), there is definitely a difference between input-restricted and output-restricted deques.]
3. Find a permutation of 1 2 3 4 that cannot be obtained with either an input-restricted or an output-restricted deque.

1. 4 1 3 2 (input all from the right, output 4 right, 1 left, 3 right, then 2). With an output-restricted deque, to output the 4, we'd first have to put 1, 2, and 3 in the deque and (here lies the difference) in the order of the permutation. But, after inserting the 1, we'd have to insert the 2 at the right and then we'd have no place for the 3.
2. 4 2 1 3 (input 1 wherever, 2 left, 3 right, 4 left, then output all to the left). With an input-restricted deque, to output the 4, we'd first have to put 1, 2, and 3 in the deque and (here lies the difference) in the order of the input. Then we'd have to output 2, which is just between 1 and 3.
3. 4 2 3 1. With an input-restricted deque, to output the 4, we'd have to input 1, 2, 3 first, but then we'd have to output the 2 which lies in the middle. With an output-restricted deque, we'd have to input 1, 2, 3 first in the order 2, 3, 1, so we'd have to put the 2 to the left of the 1 but then we couldn't place the 3 in the middle.

▸ **14.** [*26*] Suppose you are allowed to use only stacks as data structures. How can you implement a queue efficiently with two stacks?

Let A and B be two stacks (elements go from the top of A to the bottom and from the bottom of B to the top). To enqueue an element, we push it to A (we could push it to B if B is empty). To deque an element, if B is empty, we do $x \Leftarrow A, B \Leftarrow x$ until A is empty; if B is still empty, the queue is empty, and otherwise we pop the element from B. The time needed to copy n elements from A to B is generally amortized as we won't need to copy for the following n deletions.

2.2.2 Sequential Allocation

▸ **1.** [*15*] In the queue operations given by (6a) and (7a), how many items can be in the queue at one time without `OVERFLOW` occurring?

$\mathtt{M} - 1$, since `R` and `F` range between 1 and `M` and $\mathtt{R} = \mathtt{F}$ is used to signal an empty queue. Since only the relative value of the two indexes matters, this leaves us with a number of elements between 0 and $\mathtt{M} - 1$.

▸ **2.** [*22*] Generalize the method of (6a) and (7a) so that it will apply to any deque with fewer than `M` elements. In other words, give specifications for the other two operations, "delete from rear" and "insert at front."

$$\mathtt{Y} \Leftarrow \mathtt{X} \text{ (delete from rear):} \begin{cases} \text{if } \mathtt{F} = \mathtt{R}, \text{ then } \mathtt{UNDERFLOW}; \\ \mathtt{Y} \leftarrow \mathtt{X}[\mathtt{R}]; \\ \text{if } \mathtt{R} = 1, \text{ then } \mathtt{R} \leftarrow \mathtt{M}, \text{ otherwise } \mathtt{R} \leftarrow \mathtt{R} - 1. \end{cases} \tag{2.1}$$

$$\mathtt{X} \Leftarrow \mathtt{Y} \text{ (insert at front):} \begin{cases} \mathtt{X}[\mathtt{F}] \leftarrow \mathtt{Y}; \\ \text{if } \mathtt{F} = \mathtt{1}, \text{ then } \mathtt{F} \leftarrow \mathtt{M}, \text{ otherwise } \mathtt{F} \leftarrow \mathtt{F} - 1. \\ \text{if } \mathtt{F} = \mathtt{R}, \text{ then } \mathtt{OVERFLOW}; \end{cases} \tag{2.2}$$

6. [*10*] Starting with the memory configuration shown in Fig. 4, determine which of the following sequences of operations causes overflow or underflow:

1. I_1;
2. I_2;
3. I_3;
4. $I_4I_4I_4I_4I_4$;
5. $D_2D_2I_2I_2I_2$.

Only 1, 2, and 4 are valid; 3 causes an overflow and 5 causes an underflow in the second instruction.

▸ **19.** [*16*] (*0-origin indexing.*) Experienced programmers learn that it is generally wise to denote the elements of a linear list by `X`[0], `X`[1], ..., `X`[$n - 1$], instead of using the more traditional notation `X`[1], `X`[2], ..., `X`[n]. Then, for example, the base address $\mathtt{L}_0$ in (1) points to the smallest cell of the array.

Revise the insertion and deletion methods (2a), (3a), (6a), and (7a) for stacks and queues so that they conform to this convention. In other words, change them so that the list elements will appear in the array `X`[0], `X`[1], ..., `X`[$\mathtt{M} - 1$], instead of `X`[1], `X`[2], ..., `X`[`M`].

For the stack, since we want `T` to be non-negative, we have to consider $\mathtt{T} = 0$ to represent an empty stack, so `T` still ranges from 0 to `M` but now `T` represents one past the top element. For the queue, we don't need much adaptation.

$$\mathtt{X} \Leftarrow \mathtt{Y} \text{ (insert into stack)}: \begin{cases} \text{if } \mathtt{T} = \mathtt{M}, \text{ then } \mathtt{OVERFLOW}; \\ \mathtt{X[T]} \leftarrow \mathtt{Y}; \\ \mathtt{T} \leftarrow \mathtt{T} + 1. \end{cases} \tag{2.3}$$

$$\mathtt{Y} \Leftarrow \mathtt{X} \text{ (delete from stack)}: \begin{cases} \text{if } \mathtt{T} = 0, \text{ then } \mathtt{UNDERFLOW}; \\ \mathtt{T} \leftarrow \mathtt{T} - 1; \\ \mathtt{Y} \leftarrow \mathtt{X[T]}. \end{cases} \tag{2.4}$$

$$\mathtt{X} \Leftarrow \mathtt{Y} \text{ (insert into queue)}: \begin{cases} \mathtt{R} \leftarrow (\mathtt{R} + 1) \bmod \mathtt{M}; \\ \text{if } \mathtt{R} = \mathtt{F}, \text{ then } \mathtt{OVERFLOW}; \\ \mathtt{X[R]} \leftarrow \mathtt{Y}. \end{cases} \tag{2.5}$$

$$\mathtt{Y} \Leftarrow \mathtt{X} \text{ (delete from queue)}: \begin{cases} \text{if } \mathtt{F} = \mathtt{R}, \text{ then } \mathtt{UNDERFLOW}; \\ \mathtt{F} \leftarrow (\mathtt{F} + 1) \bmod \mathtt{M}; \\ \mathtt{Y} \leftarrow \mathtt{X[F]}. \end{cases} \tag{2.6}$$

2.2.3 Linked Allocation

▶ **1.** [*10*] Operation (9) for popping up a stack mentions the possibility of UNDERFLOW; why doesn't operation (8), pushing down a stack, mention the possibility of OVERFLOW?

Because you can allocate anywhere so this possibility doesn't exist unless you run out of memory, a condition already handled by $\mathtt{P} \Leftarrow \mathtt{AVAIL}$.

▶ **5.** [*24*] Operations (14) and (17) give the effect of a queue; show how to define the further operation "insert at front" so as to obtain all the actions of an output-restricted deque. How could the operation "delete from rear" be defined (so that we would have a general deque)?

Inserting at front can be handled with operation (8), just like with a stack. Deleting from rear is more difficult. One could use doubly linked lists, which are covered in section 2.2.5. Alternatively, if we do not want to change the structure, we have to iterate from the front, with a pointer called I:

$$\begin{cases} \text{If } F = \Lambda, \text{ then } \mathtt{UNDERFLOW}; \\ \mathtt{Y} \leftarrow \mathtt{INFO(R)}, \mathtt{P} \leftarrow \mathtt{LOC(F)}; \\ \text{while } \mathtt{LINK(P)} \neq \mathtt{R} \text{ repeat } \mathtt{P} \leftarrow \mathtt{LINK(P)}; \\ \mathtt{AVAIL} \Leftarrow \mathtt{R}, \mathtt{R} \leftarrow \mathtt{P}, \mathtt{LINK(P)} \leftarrow \Lambda. \end{cases}$$

▶ **7.** [*23*] Design an algorithm to "invert" a linked linear list such as (1), that is, to change its links so that the items appear in the opposite order. [If, for example, the list (1) were inverted, we would have FIRST linking to the node containing item 5; that node would link to the one containing item 4; etc.] Assume that the nodes have the form (3).

We use auxiliary pointers P, C, and N.

1. Set $\mathtt{P} \leftarrow \Lambda$ and $\mathtt{C} \leftarrow \mathtt{FIRST}$.
2. If $\mathtt{C} \neq \Lambda$, set $\mathtt{N} \leftarrow \mathtt{LINK(C)}$, $\mathtt{LINK(C)} \leftarrow \mathtt{P}$, $\mathtt{P} \leftarrow \mathtt{C}$, $\mathtt{C} \leftarrow \mathtt{N}$, and repeat.
3. $\mathtt{FIRST} \leftarrow \mathtt{P}$.

▸ **11.** [*24*] The result of topological sorting is not always completely determined, since there may be several ways to arrange the nodes and to satisfy the conditions of topological order. Find all possible ways to arrange the nodes of Fig. 6 into topological order.

First we draw it in a way that simplifies reasoning, with all arrows pointing downwards or to the right:

```
1 ---> 3 ---> 7 ---> 4
              |      |
              v      |
       9 ---> 5      |
       |      |      |
       v      v      v
       2 ---> 8 ---> 6
```

Then we explore possibilities by following the idea in algorithm T but backtracking. We suppress the arrows for compactness.

137492586	137495286	137924586	137925486	137925846	137942586
137945286	137952486	137952846	137954286	139274586	139275486
139275846	139724586	139725486	139725846	139742586	139745286
139752486	139752846	139754286	192374586	192375486	192375846
193274586	193275486	193275846	193724586	193725486	193725846
193742586	193745286	193752486	193752846	193754286	912374586
912375486	912375846	913274586	913275486	913275846	913724586
913725486	913725846	913742586	913745286	913752486	913752846
913754286	921374586	921375486	921375846		

▸ **17.** [*21*] What output does Algorithm T produce if it is presented with the input (18)?

We follow the algorithm with the following table:

	0	1	2	3	4	5	6	7	8	9
QLINK/COUNT	1	9	7	2	5	8	0	4	6	3
TOP		3.	8.	7.	6.	8.	Λ	4, 5.	6.	5, 2.
	N:	0	F:	0	R:	6	P:	Λ		

To get the output $1 \to 9 \to 3 \to 2 \to 7 \to 4 \to 5 \to 8 \to 6 (\to 0)$.

▸ **20.** [*24*] Algorithm T uses F, R, and the QLINK table to obtain the effect of a queue that contains those nodes whose COUNT field has become zero but whose successor relations have not yet been removed. Could a stack be used for this purpose instead of a queue? If so, compare the resulting algorithm with Algorithm T.

Yes, because those are nodes that we can already print next after the ones that we have already printed. The differences would be:

1. In T4, we would do $\mathtt{T} \leftarrow 0$ and, for each $1 \le k \le n$ with $\mathtt{COUNT}[k] = 0$, $\mathtt{QLINK}[k] \leftarrow \mathtt{T}$ and $\mathtt{T} \leftarrow k$.

2. In T5, we would output the value of T rather than F, and instead of P $\leftarrow$ TOP[F] we would do P $\leftarrow$ TOP[T]. We would do T $\leftarrow$ QLINK[T] right after T5 instead of doing T7.
3. In T6, the two assignments involving R would instead be QLINK[SUC(P)] $\leftarrow$ T and T $\leftarrow$ SUC(P).

Then we wouldn't need QLINK[0]. With input (18), this would work as follows:

	1	2	3	4	5	6	7	8	9
QLINK/COUNT	0	1	0	0	4	0	0	4	1
TOP	3.	8.	7.	6.	8.	Λ	4, 5.	6.	5, 2.
	N:	0	T:	0	P:	Λ			

Output: $9 \to 2 \to 1 \to 3 \to 7 \to 5 \to 8 \to 4 \to 6 (\to 0)$.

▸ **29.** [*21*]

1. Give an algorithm to "erase" an entire list like (1), by putting all of its nodes on the AVAIL stack, given only the value of FIRST. The algorithm should operate as fast as possible.
2. Repeat part 1 for a list like (12), given the values of F and R.

1. If FIRST $= \Lambda$, we don't have to do anything. Otherwise, let P $\leftarrow$ FIRST, if LINK(P) $\neq \emptyset$ then P $\leftarrow$ LINK(P) and repeat, finally set LINK(P) $\leftarrow$ AVAIL and AVAIL $\leftarrow$ FIRST.
2. Just set LINK(R) $\leftarrow$ AVAIL and AVAIL $\leftarrow$ F. This works even if the queue is empty.

2.2.4 Circular Lists

▸ **3.** [*20*] What does operation (3) do if PTR_1 and PTR_2 are both pointing to nodes in the *same* circular list?

It splits the list into two parts; the one that goes from the right of PTR_1 to PTR_2 is preserved but the other one becomes a memory leak. If the two pointers are equal, nothing happens, just PTR_2 is set to Λ.

▸ **5.** [*21*] Design an algorithm that takes a circular list such as (1) and reverses the direction of all the arrows.

This is similar to exercise 2.2.3.7.

1. Set P $\leftarrow \Lambda$ and C $\leftarrow$ PTR.
2. If C $\neq \Lambda$, set N $\leftarrow$ LINK(C), LINK(C) $\leftarrow$ P, P $\leftarrow$ C, C $\leftarrow$ N, and repeat.

7. [*10*] Why is it useful to assume that the ABC fields of a polynomial list appear in decreasing order?

Many operations on polynomials require us to match monomials with equal exponents of the variables. By having a total order on them it's easy to match these in linear time.

▸ **8.** [*10*] Why is it useful to have `Q1` trailing one step behind `Q` in Algorithm A?

We need `Q` to advance in order to check if the next monomial has greater, lower, or equal coefficient than the one pointed to by `P`, and if it's lower, we need `Q1` to insert the one pointed to by `P` right before the one pointed to by `Q`.

▸ **9.** [*23*] Would Algorithm A work properly if $\texttt{P} = \texttt{Q}$ (i.e., both pointer variables point at the same polynomial)? Would Algorithm M work properly if $\texttt{P} = \texttt{M}$, if $\texttt{P} = \texttt{Q}$, or if $\texttt{M} = \texttt{Q}$?

Algorithm A would work properly, doubling the polynomial being pointed to (this is assuming no coefficient is 0). As for Algorithm M, it would work for $\texttt{P} = \texttt{M}$ since they're not mutated. If $\texttt{P} = \texttt{Q}$ it wouldn't work, because if `M` has more than one monomial then the addition of `P` times the second monomial or later (the second iteration of step M2) would add the wrong value to `Q`. If $\texttt{M} = \texttt{Q}$ it wouldn't work either as, for example, if initially $\texttt{M} = \texttt{Q} = 1$ and $\texttt{P} = -1$, then addition in the first iteration of M2 would free the current monomial pointed to by `M` and in the next iteration of M1, `M` would point to free memory with unpredictable consequences. In cases where the constant coefficient of `P` is not -1, it amazingly works, as writes to `Q` are always behind where we read from `M` when considering monomials of `P` with positive degree and, for the constant coefficient, it writes to the same position where `M` is but only to the `COEF` field—unless `P` is -1 in which case the node pointed to by `M` is freed.

▸ **10.** [*20*] The algorithms in this section assume that we are using three variables x, y, and z in the polynomials, and that their exponents individually never exceed $b - 1$ (where b is the byte size in `MIX`'s case). Suppose instead that we want to do addition and multiplication of polynomials in only one variable, x, and to let its exponent take on values up to $b^3 - 1$. What changes should be made to Algorithms A and M?

None, as they work out of the box if we consider the `ABC` field to hold a single coefficient. The sums and comparisons made on this field in the algorithms are equally valid independently of which of the two interpretations we want to give to this field, assuming no overflow.

▸ **17.** [*22*] What advantage is there in representing polynomials with a circular list as in this section, instead of with a straight linear linked list terminated by Λ as in the previous section?

Steps A2 and A5 don't need special cases to handle the end of a list, only A3 does. Other than that there's not much of a difference.

▸ **18.** [*25*] Devise a way to represent circular lists inside a computer in such a way that the list can be traversed efficiently in both directions, yet only one link field is used per node.

If the link field combines the location of the next and previous fields with some monoidal operation like XOR, and we have an item x_i, then knowing the location of x_{i+1} allows us to get the location of x_{i-1} and vice versa, so knowing the location of two consecutive items allows us to traverse the list in both directions.

2.2.5 Doubly Linked Lists

▸ **2.** [*22*] Explain why a list that is singly linked cannot allow efficient operation as a general deque; the deletion of items can be done efficiently at only one end of a singly linked list.

One could, in fact, implement it reasonably efficiently using the trick from Exercise 2.2.4.18. If we assume a normal linked list, however, deletion of an element requires knowing the link to the previous element. We know this for the head of the list, as we just have to change the pointer to the head, but we do not know it for the tail, and if we cached a pointer to the second-to-last element to do this, then we would need to update this pointer to point to the new second-to-last element, whose address we cannot retrieve efficiently.

▸ **3.** [*22*] The elevator system described in the text uses three call variables, `CALLUP`, `CALLCAR`, and `CALLDOWN`, for each floor, representing buttons that have been pushed by the users in the system. It is conceivable that the elevator actually needs only one or two binary variables for the call buttons on each floor, instead of three. Explain how an experimenter could push buttons in a certain sequence with this elevator system to *prove* that there are three independent binary variables for each floor (except the top and bottom floors).

We need an experiment that proves that, for any given floor that is not the top or bottom one, the elevator behaves differently for each of the 8 possible combinations of the 3 buttons.

Let us denote the states by the values of `CALLUP`, `CALLCAR` and `CALLDOWN` in sequence, so for example only `CALLUP` would be 100 and all buttons at once would be 111.

We first note that the three buttons are not completely independent: 101 and 111 are equal, because the elevator would stop in both directions and then one of `CALLUP` and `CALLDOWN` is cleared at the same time as `CALLCAR` is, so there's no observable difference, but this state and all others are pairwise different, as we shall see.

000 is trivially different from all the other states since, if the elevator is in another floor, any other combination would take it to our floor. If the elevator is in the lower floor and we call 001 in the current and above floors, it would go to the floor above and then to the current one, the opposite of what would happen if in the current floor we would call any combination involving `CALLUP` or `CALLCAR`. The opposite experiment (calling 100 in the current and below floors with the elevator in the one above, and then changing what we call in the current floor) shows that 100 is also different from all the others.

To see that 010 and 101 are different, we press `CALLCAR` in the current floor with the elevator in the one above and, while it's going down, press `CALLUP` in the floor below and tell it to go two floors up. Our design wouldn't stop in our floor when going up, but it would if we had pressed both `CALLUP` and `CALLDOWN` since `CALLUP` would still be 1 after stopping when going down. Actually, this shows that 010 and 011 are different from 101, 110, and 111, and the opposite experiment shows that 010 and 110 are different from 101, 011, and 111.

▸ **11.** [*21*] (*A sparse-update memory.*) The following problem often arises in *synchronous* simulations: The system has n variables $\texttt{V}[1], \ldots, \texttt{V}[n]$, and at every simulated step new values for some of them are calculated from the old values. These calculations are assumed to be done "simultaneously" in the sense that the variables do not change to their new values until after all assignments have been made. Thus, the two statements

$$\texttt{V}[1] \leftarrow \texttt{V}[2] \qquad\qquad \text{and} \qquad\qquad \texttt{V}[2] \leftarrow \texttt{V}[1]$$

appearing at the same simulated time would interchange the values of $\texttt{V}[1]$ and $\texttt{V}[2]$; this is quite different from what would happen in a sequential calculation.

The desired action can of course be simulated by keeping an additional table $\texttt{NEWV}[1], \ldots, \texttt{NEWV}[n]$. Before each simulated step, we could set $\texttt{NEWV}[k] \leftarrow \texttt{V}[k]$ for $1 \le k \le n$, then record all changes of $\texttt{V}[k]$ in $\texttt{NEWV}[k]$, and finally, after the step we could set $\texttt{V}[k] \leftarrow \texttt{NEWV}[k]$, $1 \le k \le n$. But this "brute force" approach is not completely satisfactory, for the following reasons:

1. Often n is very large, but the number of variables changed per step is rather small.
2. The variables are often not arranged in a nice table $\texttt{V}[1], \ldots, \texttt{V}[n]$, but are scattered throughout memory in a rather chaotic fashion.
3. This method does not detect the situation (usually an error in the model) when one variable is given two values in the same simulated stop.
4. Assuming that the number of variables changed per step is rather small, design an efficient algorithm that simulates the desired actions, using two auxiliary tables $\texttt{NEWV}[k]$ and $\texttt{LINK}[k]$, $1 \le k \le n$. If possible, your algorithm should give an error stop if the same variables is being given two different values in the same step.

The idea, of course, would be to make a linked list with the LINK table. If we assume the table is initially set to 0, we could use -1 to signify the end of the linked list.

1. [Initialize.] (This step only happens at the beginning of the simulation.) Set $\texttt{LINK}[k] \leftarrow 0$, $1 \le k \le n$, $\texttt{Q} \leftarrow -1$.
2. [Advance time.] Advance to the next step of the synchronous simulation.
3. [Run the current state.] For each calculation that needs to be done, if k is the variable to be updated, raise an error if $\texttt{LINK}[k] \neq 0$, calculate the new value, store it in $\texttt{NEWV}[k]$, set $\texttt{LINK}[k] \leftarrow \texttt{Q}$ and $\texttt{Q} \leftarrow k$.
4. [Update variables.] If $\texttt{Q} > 0$, set $\texttt{V[Q]} \leftarrow \texttt{NEWV[Q]}$, $\texttt{Q2} \leftarrow \texttt{LINK[Q]}$, $\texttt{LINK[Q]} \leftarrow 0$, $\texttt{Q} \leftarrow \texttt{LINK[Q]}$, and repeat this step. Otherwise go to 2.

▸ **12.** [*22*] Why is it a good idea to use doubly linked lists instead of singly linked or sequential lists in the simulation program of this section?

This is because we often delete nodes from the middle of the list without knowing the predecessor or successor, like an user who was tired of waiting, and doubly linked lists are the only kind that allows doing this efficiently.

2.2.6 Arrays and Orthogonal Lists

▸ **.** [*21*] Formulas (5) and (6) have been derived from the assumption that $0 \leq \mathtt{I}_r \leq d_r$ for $1 \leq r \leq k$. Give a general formula that applies to the case $l_r \leq \mathtt{I}_r \leq u_r$, where l_r and u_r are any lower and upper bounds on the dimensionality.

$$\mathtt{LOC}(\mathtt{A}[\mathtt{I}_1, \mathtt{I}_2, \dots, \mathtt{I}_k]) = \mathtt{LOC}(\mathtt{A}[l_1, l_2, \dots, l_k]) + \sum_{1\leq r\leq k} a_r(\mathtt{I}_r - l_r) =$$
$$= \left(\mathtt{LOC}(\mathtt{A}[l_1, l_2, \dots, l_k]) - \sum_{1\leq r\leq k} a_r l_r\right) + \sum_{1\leq r\leq k} a_r \mathtt{I}_r,$$

where

$$a_r = c \prod_{r<s\leq k} (u_s - l_s + 1).$$

▸ **6.** [*24*] Consider the "tetrahedral arrays" $\mathtt{A}[i,j,k]$,$\mathtt{B}[i,j,k]$, where $0 \leq k \leq j \leq i \leq n$ in $\mathtt{A}$, and $0 \leq i \leq j \leq k \leq n$ in $\mathtt{B}$. Suppose that both of these arrays are stored in consecutive memory locations in lexicographic order of the indices; show that $\mathtt{LOC}(\mathtt{A}[\mathtt{I},\mathtt{J},\mathtt{K}]) = a_0 + f_1(\mathtt{I}) + f_2(\mathtt{J}) + f_3(\mathtt{K})$ for certain functions f_1, f_2, f_3. Can $\mathtt{LOC}(\mathtt{B}[\mathtt{I},\mathtt{J},\mathtt{K}])$ be expressed in a similar manner?

Let c be the size of an entry, $a_0 := \mathtt{LOC}(\mathtt{A}[0,0,0])$, and $x(i,j,k)$ be the number of positions from a_0 to $\mathtt{LOC}(\mathtt{A}[i,j,k])$, so that $\mathtt{LOC}(\mathtt{A}[i,j,k]) = a_0 + cx(i,j,k)$, then

$$x(i,j,k) = x(i,0,0) + (x(i,j,0) - x(i,0,0)) + (x(i,j,k) - x(i,j,0)),$$

but

$$x(i,j,k) - x(i,j,0) = k,$$
$$x(i,j,0) - x(i,0,0) = \sum_{p=0}^{j-1}(x(i,p+1,0) - x(i,p,0)) = \sum_{p=0}^{j-1}(p+1) = \sum_{p=1}^{j} p = \frac{j(j+1)}{2},$$

and $x(i,0,0)$ is already a function of i, so in fact locations can be put in this form.

Now let $b_0 := \mathtt{LOC}(\mathtt{B}[0,0,0])$ and $y(i,j,k) := \frac{\mathtt{LOC}(\mathtt{B}[0,0,0]) - \mathtt{LOC}(\mathtt{B}[i,j,k])}{c}$, the functions, if they exist, would depend on n, since $y(0,1,1) = y(0,0,n) + 1 = n + 1$, but still

$$y(i,j,k) = y(i,n,n) + (y(i,j,n) - y(i,n,n)) + (y(i,j,k) - y(i,j,n))$$

with

$$y(i,j,k) - y(i,j,n) = n - k,$$
$$y(i,j,n) - y(i,n,n) = \sum_{p=j+1}^{n} (y(i,p-1,n) - y(i,p,n)) = \sum_{p=j+1}^{n} (1 + y(i,p,p) - y(i,p,n))$$
$$= \sum_{p=j+1}^{n} (n-p+1) = \sum_{p=1}^{n-p} p,$$

and $y(i,n,n)$ is already a function of i (and n).

▸ **12.** [*20*] What are `VAL(Q0)`, `VAL(P0)`, and `VAL(P1)` at the beginning of step S7, in terms of the notation a, b, c, d used in (13)?

`VAL(P1)` $= d$, `VAL(Q0)` $= c$, `VAL(P0)` $= b/a$.

▸ **13.** [*22*] Why were circular lists used in Fig. 14 instead of straight linear lists? Could Algorithm S be rewritten so that it does not make use of the circular linkage?

Circular lists are used as an optimization and simplification: since many lists have to be traversed many times, it is advantageous to not have to reset the pointer to the list every time as the pointer is already situated at the beginning of the list by the time the previous iteration ends.

▸ **24.** [*25*] (*The sparse array trick.*) Suppose you want to use a large array for random access, although you won't actually be referring to very many of its entries. You want `A`$[k]$ to be zero the first time you access it, yet you don't want to spend the time to set every location to zero. Explain how it is possible to read and write any desired elements `A`$[k]$ reliably, given k, without assuming anything about the actual initial memory contents, by doing only a small fixed number of additional operations per array access.

One could have an auxiliary array `L` containing pairs of index and value, and have `A` contain pointers to `L`. If the pointer is within bounds and `L` contains the index k in that position, then the value contained is the value of `A`$[k]$, otherwise the value is 0. `L` could actually be the size of `A` but have an upper limit n with the number of elements actually allocated, which is considered the upper bound and which increases whenever a value that has not been yet allocated is written to.

2.3 Trees

4. [*01*] True or false: In a conventional tree diagram (root at the top), if node X has a *higher* level number than node Y, then node X appears *lower* in the diagram than node Y.

True.

5. [*02*] If node A has three siblings and B is the parent of A, what is the degree of B?

4.

▸ **6.** [*21*] Define the statement "X is an mth cousin of Y, n times removed" as a meaningful relation between nodes X and Y of a tree, by analogy with family trees, if $m > 0$ and $n \geq 0$. (See a dictionary for the meaning of these terms in regard to family trees.)

It means that there exists a node X' such that either X is the nth ancestor of X' or vice versa (that is, the ancestor n levels below, where for $n = 0$ we would take $X' = X$), X' and Y are at the same level, the $(m + 1)$th ancestor of X' and Y is the same, and the mth ancestor isn't.

▸ **8.** [*03*] What binary tree is not a tree?

By the definitions in this book, the empty binary tree. Actually binary trees and trees are very separate concepts and therefore none is.

9. [*00*] In the two binary trees of (1), which node is the root (B or A)?

By the current conventions, A.

13. [*10*] Suppose that node X is numbered $a_1.a_2.\cdots.a_k$ in the Dewey decimal system; what are the Dewey numbers of the nodes in the path from X to the root (see exercise 3)?

They would be $a_1.a_2.\cdots.a_k; a_1.a_2.\cdots.a_{k-1}; \dots; a_1.a_2; a_1$.

▸ **15.** [*20*] Invent a notation for the nodes of binary trees, analogous to the Dewey decimal notation for nodes of trees.

We could call the root % and, for a given nonempty node called λ (a string), its left child could be called λL and its right child λR.

17. [*01*] If Z stands for Fig. 19 regarded as a forest, what node is parent($Z[1,2,2]$)?

parent($Z[1,2,2]$) = parent(children of node 1.2.2) = E.

18. [*08*] In List (3), what is $L[5,1,1]$? What is $L[3,1]$?

$L[5,1,1] = (2)$, $L[3,1]$ doesn't exist.

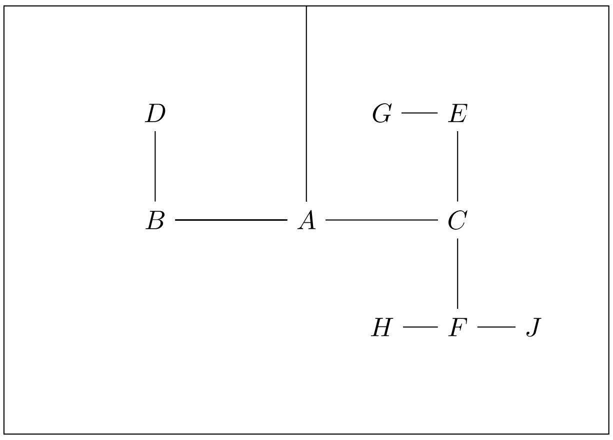
▸ **20.** [*M21*] Define a *0-2-tree* as a tree in which each node has exactly zero or two children. (Formally, a 0-2-tree consists of a single node, called its root, plus 0 or 2 disjoint 0-2-trees.) Show that every 0-2-tree has an odd number of nodes; and give a one-to-one correspondence between binary trees with n nodes and (ordered) 0-2-trees with $2n+1$ nodes.

We can prove that 0-2-trees have an odd number of nodes by induction on the depth of the tree: for depth 0, we just have one node with no children, and for depth greater than 0, we have the root and two children whose subtrees have each an odd number of nodes, so the main tree also has an odd number of nodes.

For the bijection, an empty binary tree corresponds to a node with 0 children, and any other binary tree corresponds to a node whose children are the left and right nodes. This is clearly bijective, and we can prove that the bijection sends binary trees with n nodes to ordered 0-2-trees with $2n+1$ nodes by induction on n. For $n = 0$, this is obvious; for $n > 0$, let m be the number of nodes in the left subtree, clearly $0 \leq m < n$ and the right subtree has $n - m - 1$ nodes, $0 < n - m - 1 \leq n$, so by the induction hypothesis on the subtrees the corresponding 0-2-tree has $1 + (2m+1) + (2(n-m-1)+1) = 1 + 2m + 1 + 2n - 2m - 2 + 1 = 2n + 1$ nodes.

▸ **22.** [*21*] Standard European paper sizes A0, A1, A2, ..., An, ... are rectangles whose sides are in the ratio $\sqrt{2}$ to 1 and whose areas are 2^{-n} square meters. Therefore if we cut a sheet of An paper in half, we get two sheets of A$(n+1)$ paper. Use this principle to design a graphic representation of binary trees, and illustrate your idea by drawing the representation of 2.3.1–(1) below.

One idea would be to represent an empty binary tree as nothing and a nonempty binary tree as dividing the sheet in two by a line parallel to the short side, interrupted by the label of the root node, and then draw the left and right subtrees in the left/top and right/bottom sides of the paper. In fact, one might not even need to draw the whole line, drawing only the line from the label of the parent node (or from the top of the sheet for the root) to the own label is enough and would probably be clearer.



2.3.1 Traversing Binary Trees

1. [*01*] In the binary tree (2), let `INFO(P)` denote the letter stored in `NODE(P)`. What is `INFO(LLINK(RLINK(RLINK(T))))`?

H.

▸ **4.** [*20*] The text defines three basic orders for traversing a binary tree; another alternative would be to proceed in three steps as follows:

1. Visit the root,
2. traverse the right subtree,
3. traverse the left subtree,

using the same rule recursively on all nonempty subtrees. Does this new order bear any simple relation to the three orders already discussed?

Yes, this is the same as postorder but backwards.

▶ **10.** [*20*] What is the largest number of entries that can be in the stack at once, during the execution of Algorithm T, if the binary tree has n nodes? (The answer to this question is very important for storage allocation, if the stack is being stored consecutively.)

There can be up to n nodes in the stack, in the case that all the right links are null and the tree is just a linked list using the `RLINK`.

▶ **13.** [*24*] Design an algorithm analogous to Algorithm T that traverses a binary tree in *preorder*, and prove that your algorithm is correct.

1. [Initialize.] Set stack `A` empty, and set the link variable `P` $\leftarrow$ `T`.
2. [`P` $= \Lambda$?] If `P` $= \Lambda$, go to step 4.
3. [Visit `P`.] Visit `NODE(P)`. Do `A` $\Leftarrow$ `RLINK(P)`, `P` $\leftarrow$ `LLINK(P)`, and return to step 2.
4. [Visit right hand side.] If `A` is empty, the algorithm terminates. Otherwise set `P` $\Leftarrow$ `A` and return to step 2.

We now prove by induction on the number of nodes that, if we start in step 2, we traverse the subtree starting at `P` in preorder, leave the stack unchanged, and proceed to step 4, which clearly would show that the algorithm is correct.

For an empty tree, this is obvious. For a tree with n nodes, that we start with the root is obvious, we do that at step 3 with the stack empty. After that step, setting `P` $\leftarrow$ `LLINK(P)` and returning to step 2 means that we get to step 4 after traversing the left subtree, which by the induction hypothesis would leave us on step 4 with the stack like it was before but adding the node `RLINK(P)`. Then step 4 clearly means to traverse `RLINK(P)` and leave `A` like it was before starting processing `P`.

▶ **16.** [*22*] The diagrams in Fig. 24 help to provide an intuitive characterization of the position of `NODE(Q$)` in a binary tree, in terms of the structure near `NODE(Q)`: If `NODE(Q)` has a nonempty right subtree, consider `Q` = `$P`, `Q$` = `P` in the upper diagrams; `NODE(Q$)` is the "leftmost" node of that right subtree. If `NODE(Q)` has an empty right subtree, consider `Q` = `P` in the lower diagrams; `NODE(Q$)` is located by proceeding upward in the tree until after the first upward step to the right.

Give a similar "intuitive" rule for finding the position of `NODE(Q` $*$ `)` in a binary tree in terms of the structure near `NODE(Q)`.

If `NODE(Q)` has a nonempty left subtree, consider `Q` = $*$`P` in the upper diagrams; then `Q`$*$ = `P` is the left child of `Q`. Otherwise, if `NODE(Q)` has a nonempty right subtree, `Q`$*$ is the right child of `Q`. Otherwise both subtrees are empty and `Q`$*$ can be found by going upward until finding a node that is a left child (which could be `Q` itself) and taking its right sibling.

▸ **17.** [*22*] Give an algorithm analogous to Algorithm S for determining `P*` in a threaded binary tree. Assume that the tree has a list head as in (8), (9), and (10).

If `P` points to a node of a threaded binary tree, this algorithm sets `Q` $\leftarrow$ `P*`.

1. [Is the left subtree nonempty?] If `LTAG(P)` $= 1$, set `Q` $\leftarrow$ `LLINK(P)` and terminate the algorithm.

2. [Search to the right.] Set `Q` $\leftarrow$ `RLINK(P)`. If `RTAG(P)` $= 1$, terminate the algorithm. Otherwise set `P` $\leftarrow$ `Q` and repeat this step.

28. [*00*] After Algorithm C has been used to make a copy of a tree, is the new binary tree *equivalent* to the original, or *similar* to it?

It's equivalent (and similar).

▸ **30.** [*22*] Design an algorithm that threads an unthreaded tree; for example, it should transform (2) into (10). *Note:* Always use notations like `P*` and `P$` when possible, instead of repeating the steps for traversal algorithms like Algorithm T.

This algorithm threads an unthreaded tree `T`. The fields `LTAG` and `RTAG` in the nodes of `T` are considered to initially contain arbitrary values that we do not need to preserve.

1. [Initialize list head and variables.] Get `P` $\Leftarrow$ `AVAIL` and set `LTAG(P)` $\leftarrow$ `RTAG(P)` $\leftarrow 0$, `LLINK(P)` $\leftarrow$ `T`, `RLINK(P)` $\leftarrow$ `P`, `T` $\leftarrow$ `P`, and `Q` $\leftarrow$ `P$`. *(We'll use* `Q` *as a pointer to the node being considered and* `P` *and* `R` *as pointers to the next and previous nodes, respectively. We only ever compute the next node in inorder from the last node calculated this way, which makes it trivial to use algorithm T as a coroutine.)*

2. [Are we over?] If `RLINK(Q)` $=$ `Q`, the algorithm terminates. *(This loop only happens in the list head.)*

3. [Step.] Set `R` $\leftarrow$ `Q$`. If `LLINK(Q)` $= \Lambda$, set `LTAG(Q)` $\leftarrow 0$ and `LLINK(Q)` $\leftarrow$ `P`, otherwise set `LTAG(Q)` $\leftarrow 1$. Similarly, if `RLINK(Q)` $= \Lambda$, set `RTAG(Q)` $\leftarrow 0$ and `RLINK(Q)` $\leftarrow$ `R`, otherwise set `RTAG(Q)` $\leftarrow 1$. Finally set `P` $\leftarrow$ `Q`, `Q` $\leftarrow$ `R`, and return to step 2.

▸ **37.** [*24*] (D. Ferguson.) If two computer words are necessary to contain two link fields and an `INFO` field, representation (2) requires $2n$ words of memory for a tree with n nodes. Design a representation scheme for binary trees that uses less space, assuming that *one* link and an `INFO` field will fit in a single computer word.

If we allow for a second null value (call it λ, which could be e.g. a pointer to a specific word in the program code or to a sentinel constant, or 4001 in MIX, or 1 in MMIX), then we could implement the solution in the book, which is that, if `LINK(P)` $= \Lambda$, then `LLINK(P)` $=$ `RLINK(P)` $= \Lambda$, and otherwise `LLINK(P)` $=$ `LINK(P)` and `RLINK(P)` $=$ `LINK(P)` $+ 1$ (or $+$`sizeof(int)`), except that if `LINK(LINK(P))` $= \lambda$ or `LINK(LINK(P)` $+ 1) = \lambda$ then `LLINK(P)` or `RLINK(P)` is respectively Λ.

2.3.2 Binary Tree Representation of Trees

▸ **1.** [*20*] The text gives a formal definition of $B(F)$, the binary tree corresponding to a forest F. Give a formal definition that reverses the process; in other words, define $F(B)$, the forest corresponding to a binary tree B.

$F(B)$ is the forest such that $B(F(B)) = B$, so $F = B^{-1}$. Now we shall give an explicit definition of F and see that B and F are, in fact, inverses.

Given a binary tree B:

1. If B is empty, $F(B)$ is the empty forest.
2. Otherwise, let root(P) be the root node of a given nonempty binary tree, left(P) be its left subtree, and right(P) be its right subtree. Then, if T_1 is the tree whose root is root(T) and whose children are the trees of the forest $F(\text{left}(B))$, then $F(B) = (T_1, F(\text{right}(B)))$, that is, the forest made up of the tree T_1 followed by the trees in $F(\text{right}(B))$.

We can prove that they are inverses by structural induction, which is a case of strong induction in the number of nodes.

$B(F(B)) = B$ is obvious if B is empty; otherwise $F(B) = ((\text{root}(B); F(\text{left}(B))); F(\text{right}(B)))$ is made up of a tree T_1 with root(B) as root and the trees in $F(\text{left}(B))$ as children, and then the trees in $F(\text{right}(B))$, and so $B(F(B))$ has root(T_1) = root(B) as its root, its left subtree is $B(F(\text{left}(B))) = \text{left}(B)$, and its right subtree is $B(F(\text{right}(B))) = \text{right}(B)$.

$F(B(F)) = F$ is obvious for an empty forest F; if $F = (T_1, \ldots, T_n)$ is nonempty, then $B(F)$ has root(T_1) as its root, $B(T_{11}, \ldots, T_{1m})$ as its left subtree, where $T_{11}, \ldots, T_{1m}$ are the subtrees of T_1, and $B(T_2, \ldots, T_n)$ as the right subtree, so $F(B(F))$ is made up of a tree T_1' whose root is root$(B(F))$ = root(T_1) and whose children are the trees in $F(\text{left}(B(F))) = F(B(T_{11}, \ldots, T_{1m})) = (T_{11}, \ldots, T_{1m})$ (therefore $T_1' = T_1$), followed by the trees in $F(\text{right}(B)) = F(B(T_2, \ldots, T_n)) = (T_2, \ldots, T_n)$.

▸ **2.** [*20*] We defined Dewey decimal notation for forests in Section 2.3, and for binary trees in exercise 2.3.1–5. Thus the node "J" in (1) is represented by "2.2.1", and in the equivalent binary tree (3) it is represented by "11010" *(this is a 1 for the root following by zero or more 0s or 1s, where 0 means moving to the left subtree and 1 means moving to the right)*. If possible, give a rule that directly expresses the natural correspondence between the Dewey decimal notations.

One can get the forest notation from the binary tree notation by switching the 1 at the start by a 0 and then counting the number of 1s after each 0 and adding 1 to that number. In this case, we would change "11010" to "01010", then count "1.1.0" and add 1 to each number to get "2.2.1".

▸ **13.** [*26*] Write a `MIX` program for the `COPY` subroutine (which fits in the program of the text between lines 063–104).

We have 40 lines. We use Algorithm 2.3.1C to copy the tree in rI1. In C5, to take the next node in preorder, we follow the `RLINK` up to and including that of a node with `RTAG` = 0 (positive sign), unless the node has a left child. In C4, when inserting a node to the left, we set RTAG = 1 and RLINK to the parent node, In C2, when inserting a node to the right, we set RTAG = 1 and RLINK to the previous RLINK of the parent node.

We also need to copy the INFO of the HEAD, which is easily achieved by going from C1 to C3 rather than to C4. This code is slightly less efficient than the one by Knuth, and it has not been tested.

```
        ST2  1F
        ST3  2F
        JMP  ALLOC         ; C1 [Initialize]
        ENT2 0,3
        ST2  0,2(RLINK)    ; RLINK(U) <- U
3H      LDA  1,1           ; C3 [Copy INFO]
        STA  1,2
        LDA  0,1(TYPE)
        STA  0,2(TYPE)
4H      LDA  0,1(LLINK)    ; C4 [Anything to the left?]
        JAZ  5F
        JMP  ALLOC         ; R <= AVAIL
        ST3  0,2(LLINK)    ; LLINK(Q) <- R
        ENNA 0,2
        STA  0,3(RLINKT)
        LD1  0,1(LLINK)
        ENT2 0,3
5H      LD1N 0,1(RLINKT) ; C5 [Advance]
        LD2  0,2(RLINK)
        J1P  5B
        ENN1 0,1
6H      CMP2 0,2(RLINK)    ; C6 [Test if complete]
        JEQ  2F            ; Finish when Q = RLINK(Q)
        LDA  0,1           ; C2 [Anything to the right?]
        JAN  3B
        JMP  ALLOC         ; R <= AVAIL
        LDA  0,2(RLINKT) ; RLINK(Q) <- R
        STA  0,3(RLINKT)
        ST3  0,2(RLINKT)
        JMP  3B
ALLOC   STJ  8F
        LD3  AVAIL
        J3Z  OVERFLOW
        LDX  0,3(LLINK)
        STX  AVAIL
        STZ  0,3(LLINK)
        JMP  *
2H      ENT3 *
        ENT1 0,2
1H      ENT2 *
```

▸ **14.** [*M21*] How long does it take the program of exercise 13 to copy a tree with n nodes?

There are 20 cycles for the initialization, then 11 for every node including the header until `JAZ 5F`, then 21 for each left node, plus 5 to go back in C5 if needed (once for each left node or for the header), plus 1 to arrive at the right node (including the header), then 2 cycles per node (including the header at the end) to test for termination, 20 for each right node, and 3 for termination.

In total, we spend 63 cycles, plus 39 cycles for each left child and 34 for each right child. As said before, this is just slightly less efficient than Knuth's code.

▸ **18.** [*25*] An oriented tree specified by n links $\mathtt{PARENT}[j]$ for $1 \le j \le n$ implicitly defines an ordered tree if the nodes in each family are oriented by their location. Design an efficient algorithm that constructs a doubly linked circular list containing the nodes of this ordered tree in preorder. For example, given

$$\begin{array}{rcccccccc} j = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \mathtt{PARENT}[j] = & 3 & 8 & 4 & 0 & 4 & 8 & 3 & 4 \end{array}$$

your algorithm should produce

$$\begin{array}{rcccccccc} \mathtt{LLINK}[j] = & 3 & 8 & 4 & 6 & 7 & 2 & 1 & 5 \\ \mathtt{RLINK}[j] = & 7 & 6 & 1 & 3 & 8 & 4 & 5 & 2 \end{array}$$

and it should also report that the root node is 4.

The way to do this would be to look at the indices in decreasing order, and for each index we insert the child node and its subtree to the right of the parent node, as follows:

1. [Initialize.] Set $\mathtt{LLINK}[j] \leftarrow \mathtt{RLINK}[j] \leftarrow j$ for $1 \le j \le n$, then let $j \leftarrow n$.
2. [Set parent.] If $\mathtt{PARENT}[j] = 0$, set $r \leftarrow j$. Otherwise set $p \leftarrow \mathtt{PARENT}[j]$, $\mathtt{LLINK}[\mathtt{RLINK}[p]] \leftarrow \mathtt{LLINK}[j]$, $\mathtt{RLINK}[\mathtt{LLINK}[j]] \leftarrow \mathtt{RLINK}[p]$, $\mathtt{RLINK}[p] \leftarrow j$, and $\mathtt{LLINK}[j] \leftarrow p$.
3. [Are we done?] Decrease j by 1. If $j = 0$, terminate and report r as the root node. Otherwise go to step 2

First, each element is in a circular list by its own. Then, in step 2, we place the circular list of the child to the right of the circular list of the parent. To see that this produces a preorder, we see by induction that, at the end of step 2, we have the preorder of each connected component in the "subforest" whose edges are $(k, \mathtt{PARENT}[k])$ for $j \le k \le n$, each preorder in a separate circular list.

For $j = n$, this is easy to check. For $1 \le j < n$, at the beginning of the step, j will not be connected with $\mathtt{PARENT}[j]$ in the graph, so all the nodes in its connected component will be descendants of j, and j will be the root of a subtree. Thus connecting j as the leftmost child of $\mathtt{PARENT}[j]$ would affect the preorder of the subtree $\mathtt{PARENT}[j]$ is in by adding the preorder of the child's subtree precisely to the right of its parent.

Furthermore, since we add children from right to left, the order of the subtrees is always preserved.

▸ **20.** [*M22*] Prove that if u and v are nodes of a forest, u is a proper ancestor of v if and only if u precedes v in preorder and u follows v in postorder.

$\Longrightarrow$] Trivial.

$\Longleftarrow$] Certainly u cannot be a descendant of v. If u and v were in different tree, their relative ordering would be decided by the order of those trees, not by whether we use preorder or postorder. Similarly, if r is the closest common ancestor of u and v and $r \neq u$, let u_1 be the child of r whose subtree contains u and v_1 that whose subtree contains v, then $u_1 \neq v_1$, but u_1 and v_1 and therefore u and v would always have the same order if the tree is ordered.

2.3.3 Other Representations of Trees

▸ **1.** [*20*] If we had only `LTAG`, `INFO` and `RTAG` fields (not `LLINK`) in a level order sequential representation like (8), would it be possible to reconstruct the `LLINK`s? (In other words, are the `LLINK`s redundant in (8), as the `RLINK`s are in (3)?)

Yes, although the traversal might be less efficient. First, we would have to collect the roots of the trees to form the first level. These roots are precisely the first family, that is, the first sequence of nodes up to an including one with `RTAG` = 1. We count the number of such nodes that have children, which are the ones with `LTAG` = 0. For the second level, we take as many families as nodes with children in the first level. Each of those families corresponds to the children of one of those nodes, in order. For the third level, we take as many families as nodes with children in the second level, and so on.

▸ **3.** [*24*] Modify Algorithm 2.3.2D so that it follows the ideas of Algorithm F, placing the derivatives it computes as intermediate results on a stack, instead of recording their locations in an anomalous fashion as is done in step D3. (See exercise 2.3.2–21.) The stack may be maintained by using the `RLINK` field in the root of each derivative.

1. [Initalize.] Set `P` $\leftarrow$ `Y$`.
2. [Differentiate.] Set $d \leftarrow$ `DEGREE(P)`. If $d = 0$, set `ARGS` $\leftarrow 0$, otherwise set $t \leftarrow$ $\texttt{RLINK}^{d-1}$`(LLINK(DY))`, `DY` $\leftarrow$ `RLINK(`t`)`, and `RLINK(`t`)` $\leftarrow \Lambda$. Then perform the routine `DIFF[TYPE(P)]`. (This routine uses `P` and `ARGS` as its arguments and it is in change of freeing the nodes of `ARGS` that will not be part of the result, which it should store in `Q`.)
3. [Save link.] Set `RLINK(Q)` $\leftarrow$ `LLINK(DY)`, `LLINK(DY)` $\leftarrow$ `Q`, and `P` $\leftarrow$ `P$`.
4. [Done?] If `P` = `Y`, terminate. Otherwise go back to step 2.

▸ **6.** [*24*] Suppose that the nodes of an *oriented* forest have three link fields, `PARENT`, `LCHILD`, and `RLINK`, but only the `PARENT` link has been set up to indicate the tree structure. The `LCHILD` field of each node is Λ and the `RLINK` fields are set as a linear list that simply links the nodes together in some order. The link variable `FIRST` points to the first node, and the last node has `RLINK` = Λ. Design an algorithm that goes through these nodes and fills in the `LCHILD` and `RLINK` fields compatible with the `PARENT` links, so that a triply linked tree representation like that in Fig. 26 is obtained. Also, reset FIRST so that it now points to the root of the first tree in the representation.

1. [Initialize.] Set C $\leftarrow$ FIRST and FIRST $\leftarrow \Lambda$ (we haven't found any root yet).
2. [Done?] If C $= \Lambda$, terminate.
3. [Add child.] Set $p \leftarrow$ PARENT(C) and N $\leftarrow$ RLINK(C). If $p = \Lambda$, set RLINK(C) $\leftarrow$ FIRST and FIRST $\leftarrow$ C, otherwise set RLINK(C) $\leftarrow$ FCHILD(p) and FCHILD(p) $\leftarrow$ C. (In assembly we might just set p to some function of LOC(FIRST) whenever $p = \Lambda$.)
4. [Advance.] Set C $\leftarrow$ N and go to the previous step.

▸ **11.** [*24*] (*Equivalence declarations.*) Several compiler languages, notably FORTRAN, provide a facility for overlapping the memory locations assigned to sequentially stored tables. The programmer gives the compiler a set of relations of the form X[j] $\equiv$ Y[k], which means that variable X[$j+s$] is to be assigned to the same location as variable Y[$k+s$] for all s. Each variable is also given a range of allowable subscripts: "ARRAY X[l : u]" means that space is to be set aside in memory for the table entries X[l], X[$l+1$], ..., X[u]. For each equivalence class of variables, the compiler reserves as small a block of consecutive memory locations as possible, to contain all the table entries for the allowable subscript values of these variables.

For example, suppose we have ARRAY X[0 : 10], ARRAY Y[3 : 10], ARRAY A[1 : 1], and ARRAY Z[-2 : 0], plus the equivalences X[7] $\equiv$ Y[3], Z[0] $\equiv$ A[0], and Y[1] $\equiv$ A[8]. We must set aside 20 consecutive locations

$$\begin{array}{cccccccccccccccccccc}
 & & & & & X_0 & X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9 & X_{10} & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
Z_{-2} & Z_{-1} & Z_0 & A_1 & & & & & & & & & Y_3 & Y_4 & Y_5 & Y_6 & Y_7 & Y_8 & Y_9 & Y_{10}
\end{array}$$

for these variables. (The location following A[1] is not an allowable subscript value for any of the arrays, but it must be reserved anyway.)

The object of this exercise is to modify Algorithm E so that it applies to the more general situation just described. Assume that we are writing a compiler for such a language, and the tables inside our compiler program itself have one node for each array, containing the fields NAME, PARENT, DELTA, LBD, and UBD. Assume that the compiler program has previously processed all the ARRAY declarations, so that if ARRAY X[l : u] has appeared and if P points to the node for X, then

$$\texttt{NAME(P)} = \text{“X”}, \qquad \texttt{PARENT(P)} = \Lambda, \qquad \texttt{DELTA(P)} = 0,$$
$$\texttt{LBD(P)} = l, \qquad \texttt{UBD(P)} = u.$$

The problem is to design an algorithm that processes the equivalence declarations, so that, after this algorithm has been performed,

PARENT(P) $= \Lambda$ means that locations X[LBD(P)], ..., X[UBD(P)] are to be reserved in memory for this equivalence class;

PARENT(P) $=$ Q $\neq \Lambda$ means that location X[k] equals location Y[$k+$DELTA(P)], where NAME(Q) $=$ "Y".

For example, before the equivalences listed above we might have the nodes

P	NAME(P)	PARENT(P)	DELTA(P)	LBD(P)	UBD(P)
α	X	Λ	0	0	10
β	Y	Λ	0	3	10
γ	A	Λ	0	1	1
δ	Z	Λ	0	-2	0

After the equivalences are processed, the nodes might appear thus:

α	X	Λ	$*$	-5	14
β	Y	α	4	$*$	$*$
γ	A	δ	0	$*$	$*$
δ	Z	α	-3	$*$	$*$

("$*$" denotes irrelevant information.)

Design an algorithm that makes this transformation. Assume that inputs to your algorithm have the form $(\mathtt{P}, j, \mathtt{Q}, k)$, denoting $\mathtt{X}[j] \equiv \mathtt{Y}[k]$, where NAME(P) = "X" and NAME(Q) = "Y". Be sure to check whether the equivalences are contradictory; for example, $\mathtt{X}[1] \equiv \mathtt{Y}[2]$ contradicts $\mathtt{X}[2] \equiv \mathtt{Y}[1]$.

1. [Read input.] Read a line $(\mathtt{P}, j, \mathtt{Q}, k)$, but if there are no more lines, terminate the algorithm.

2. [Find roots.] If $\mathtt{PARENT(P)} \neq \Lambda$, set $j \leftarrow j + \mathtt{DELTA(P)}$, $\mathtt{P} \leftarrow \mathtt{PARENT(P)}$, and repeat this step. If $\mathtt{PARENT(Q)} \neq \Lambda$, set $k \leftarrow k + \mathtt{DELTA(Q)}$, $\mathtt{Q} \leftarrow \mathtt{PARENT(Q)}$, and repeat this step.

3. [Merge trees.] If $\mathtt{P} \neq \mathtt{Q}$, set $\mathtt{PARENT(P)} \leftarrow \mathtt{Q}$ $\mathtt{DELTA(P)} \leftarrow k - j$, $\mathtt{LBD(Q)} \leftarrow \min\{\mathtt{LBD(Q)}, \mathtt{LBD(P)}+\mathtt{DELTA(P)}\}$, $\mathtt{UBD(Q)} \leftarrow \max\{\mathtt{UBD(Q)}, \mathtt{UBD(P)}+\mathtt{DELTA(P)}\}$. Otherwise, if $j \neq k$, notify a contradiction and terminate the algorithm. Finally go to step 1.

▸ **17.** [*25*] Algorithm F evaluates a "bottom-up" locally defined function, namely, one that should be evaluated at the children of a node before it is evaluated at the node. A "top-down" locally defined function f is one in which the value of f at a node x depends only on x and the value of f at the *parent* of x. Using an auxiliary stack, design an algorithm analogous to Algorithm F that evaluates a "top-down" function f at each node of a tree. (Like Algorithm F, your algorithm should work efficiently on trees that have been stored in *postorder* with degrees, as in (9).)

1. [Initialize.] Set the stack to the single element (∞, Λ), and let P point to the *last* ode of the forest in post-order.

2. [Evaluate f.] Let (d, v) be the value at the top of the stack. Evaluate $f(\mathtt{NODE(P)}, v)$ and save the result as VALUE(P). Then push (DEGREE(P), VALUE(P)) to the stack.

3. [Update the stack.] Pop the element (p, v) from the stack. If $p = 0$, repeat this step. Otherwise push $(p - 1, v)$ into the stack.

4. [Advance.] If P is the first node in postorder, terminate the algorithm. Otherwise set P to its predecessor in postorder and return to step 2.

2.3.4 Basic Mathematical Properties of Trees

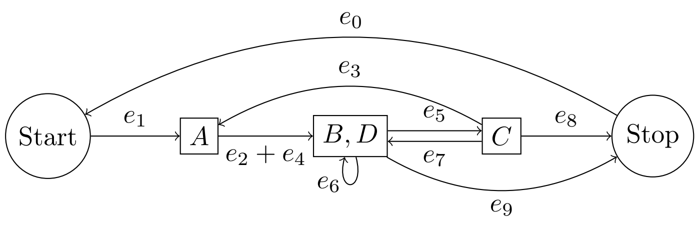
2.3.4.1 Free trees

▸ **4.** [*M20*] Let G' be a finite free tree in which arrows have been drawn on its edges $e_1, \ldots, e_{n-1}$; let $E_1, \ldots, E_{n-1}$ be numbers satisfying Kirchhoff's law (1) in G'. Show that $E_1 = \cdots = E_{n-1} = 0$.

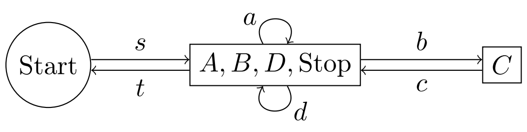
We prove this by induction on n. For $n = 1$ it is trivial. For $n > 1$, there always exists a node of degree 1 (in an oriented tree, this would be a leaf). Let e_r be the only edge connected to that node, clearly $E_r = 0$, so removing e_r leaves us with an isolated node and a free tree of degree 0 which follows Kirchhoff's law and has $n - 1$ nodes, but then by the induction hypothesis all the other E_k are also 0.

▸ **8.** [*M25*] When applying Kirchhoff's first law to program flow charts, we usually are interested only in the *vertex flows* (the number of times each box of the flow chart is performed), not the edge flows analyzed in the text. For example, in the graph of exercise 7, the vertex flows are $A = E_2 + E_4$, $B = E_5$, $C = E_3 + E_7 + E_8$, $D = E_6 + E_9$.

If we group some vertices together, treating them as one "supervertex," we can combine edge flows that correspond to the same vertex flow. For example, edges e_2 and e_4 can be combined in the flow chart above if we also put B with D:



(Here e_0 has also been added from Stop to Start, as in the text.) Continuing this procedure, we can combine $e_3 + e_7$, then $(e_3 + e_7) + e_8$, then $e_6 + e_9$, until we obtain the *reduced flow chart* having edges $s = e_1$, $a = e_2 + e_4$, $b = e_5$, $c = e_3 + e_7 + e_8$, $d = e_6 + e_9$, $t = e_0$, precisely one edge for each vertex in the original flow chart:



By construction, Kirchhoff's law holds in this reduced flow chart. The new edge flows are the vertex flows of the original; hence the analysis in the text, applied to the reduced flow chart, shows how the original vertex flows depend on each other.

Prove that this reduction process can be reversed, in the sense that any set of flows $\{a, b, \ldots\}$ satisfying Kirchhoff's law in the reduced flow chart can be "split up" into a set of edge flows $\{e_0, e_1, \ldots\}$ in the original flow chart. These flows e_j satisfy Kirchhoff's law and combine to yield the given flows $\{a, b, \ldots\}$; some of them might, however, be negative. (Although the reduction procedure has been illustrated for only one particular flow chart, your proof should be valid in general.)

(I had to look up the solution, this one is difficult.) We only have to show that there exists an assignment in the original flowchart that produces any given set of flows in the reduced flowchart.

Since we get from one to the other by reductions, we only have to prove that we can reverse reductions. In these reductions, there are two arrows e_1 from vertex or supervertex a to b and e_2 from a to c, to get an edge $f = e_1 + e_2$ from a to b, c, and reverting means recovering e_1 and e_2 from f and the other edges.

If $b = c$, we can just make up e_1 and set e_2 to $f - e_1$. If $a = b \neq c$, then e_2 must be such that Kirchhoff's law is preserved for c, and we can just set $e_1 = f - e_2$. The situation $a = c \neq b$ is symmetrical. Finally, if the three nodes are distinct, setting e_1 and e_2 such that they follow Kirchhoff's law for b and c makes $e_1 + e_2$ follow Kirchhoff's law in b, c in the reduced flowchart.

2.3.4.2 Oriented trees

▸ **4.** [*M20*] The concept of *topological sorting* can be defined for any finite directed graph G as a linear arrangement of the vertices $V_1 V_2 \dots V_n$ such that init(e) precedes fin(e) in the ordering for all arcs e of G. (See Section 2.2.3, Figs. 6 and 7.) Not all finite directed graphs can be topologically sorted; which ones can be? (Use the terminology of this section to give the answer.)

Only the ones with no directed cycles, except possibly for cycles of length 1. If G has a cycle (of length 2 or more), the nodes in the cycle cannot be topologically ordered; otherwise the transitive closure of the edges of the graph is a partial ordering between the vertices, so they can be topologically ordered.

▸ **7.** [*M22*] True or false: A directed graph satisfying properties (a) and (b) of the definition of oriented tree, and having no oriented cycles, is an oriented tree.

True for finite graphs. If there were cycles, they would be oriented because no vertex is the initial vertex of two arcs, and a graph with no cycles and exactly one edge less than the number of vertices is a free tree. Then the way to orient the edges such that (a) and (b) follow is unique, which can be proved by induction on the distance of a node to the root, and we know that this way follows (c).

False for infinite graphs. A graph whose vertices are $R, V_1, V_2, \dots, V_n, \dots$ and whose edges are $V_1 \to V_2 \to V_3 \to \dots$ follows (a) and (b) and has no cycles but it doesn't follow (c).

▸ **13.** [*M24*] Prove that if R is a root of a (possibly infinite) directed graph G, then G contains an oriented subtree with the same vertices as G and with root R. (As a consequence, it is always possible to choose the free subtree in flow charts like Fig. 32 of Section 2.3.4.1 so that it is actually an *oriented* subtree; this would be the case in that diagram if we had selected e''_{13}, e''_{19}, e_{20}, and e_{17} instead of e'_{13}, e'_{19}, e_{23}, and e_{15}.)

We prove by induction that there is a family of oriented subtrees $\{T_n\}_{n\geq 0}$ of G with root R such that T_n contains exactly all the vertices V in G such that there is an oriented path from V to R of length at most n and, furthermore, $T_{n-1} \subseteq T_n$ for every

$n \geq 1$. It is easy to check that the union of this family of trees is a tree with all the vertices in G.

For $n = 0$, we just take the trivial oriented tree, which only contains R. For $n \geq 1$, and for every node V such that the shortest oriented path from V to R has length n, we choose one such path $VV_1 \cdots V_{n-1}R$, which we can do for all such vertices because of the axiom of choice. Since V_1 is in T_{n-1}, we just have to add V and the arc (V, V_1) to T_{n-1}, for all such V.

▸ **16.** [*M24*] In a popular solitaire game called "clock," the 52 cards of an ordinary deck of playing cards are dealt face down into 13 piles of four each; 12 piles are arranged in a circle like the 12 hours of a clock and the thirteenth pile goes in the center. The solitaire game now proceeds by turning up the top card of the center pile, and then if its face value is k, by placing it next to the kth pile. (The numbers $1, 2, \ldots, 13$ are equivalent to A, $2, \ldots, 10$, J, Q, K.) Play continues by turning up the top card of the kth pile and putting it next to *its* pile, etc., until we reach a point where we cannot continue since there are no more cards to turn up on the designated pile. (The player has no choice in the game, since the rules completely specify what to do.) The game is won if all cards are face up when the play terminates.

Show that the game will be won if and only if the following directed graph is an oriented tree: The vertices are $V_1, V_2, \ldots, V_{13}$; the arcs are $e_1, e_2, \ldots, e_{12}$, where e_j goes from V_j to V_k if k is the *bottom* card in pile j after the deal.

(In particular, if the bottom card of pile j is a "j", for $j \neq 13$, it is easy to see that the game is certainly lost, since this card could never be turned up. The result proved in this exercise gives a much faster way to play the game!)

First we note that this graph already follows conditions (a) and (b) of the definition of an oriented tree, so it will be a tree if an only if V_{13} is a root, if and only if there are no cycles (see Exercise 7). We also note that, since there are only 4 cards of each denomination, and we only take a card from a pile when we find its denomination except from one from V_{13} at the start of the game, the only case where we cannot continue is after finding the last card with face value K.

$\Longrightarrow$] Assume that the game is not an oriented tree, then there is a cycle $V_{k_1} \cdots V_{k_m}$. We can assume that the k_1th pile is the first whose bottom card we turn up. Since clearly $k_1 \neq 13$, this means that we had already turned up all cards with denomination k_1, but we had not turned up $k_m\#$.

$\Longleftarrow$] Assume we lost the game, which means that we have turned up all the cards with face value 13 and that, furthermore, we have turned up as many cards from the kth pile as cards with face value k. This means that, if

$$U := \{V_j \mid \text{the } j\text{th pile's bottom card is down}\},$$

then $V_{13} \notin U$ and each vertex of U has one outgoing edge that points to another vertex in U, as we have already turned up all the cards with face value k for $V_k \notin U$. Thus V_{13} is not a root.

▸ **24.** [*M20*] Let G be a connected digraph with arcs $e_0, e_1, \ldots, e_m$. Let $E_0, E_1, \ldots, E_m$ be a set of positive integers that satisfy Kirchhoff's law for G; that is, for each vertex V,

$$\sum_{\mathrm{init}(e_j)=V} E_j = \sum_{\mathrm{fin}(e_j)=V} E_j.$$

Assume further that $E_0 = 1$. Prove that there is an oriented walk in G from $\mathrm{init}(e_0)$ such that edge e_j appears exactly E_j times, for $1 \le j \le m$, while edge e_0 does not appear.

We apply Theorem G to the graph whose vertices are those of G and whose edges are the e_j repeated E_j times. This is valid, since we could as well place an intermediate vertex among each edge to avoid repeating edges, and it would give us an Eulerian cycle in that graph that goes through e_0 once. Coalescing the repeated edges into one gives us back graph G and a cycle though G that goes though edge e_j exactly E_j times, and we just have to remove the only appearance of e_0.

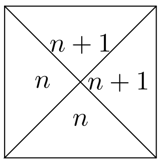
2.3.4.3 The "infinity lemma"

▸ **1.** [*M10*] The text refers to a set S containing finite sequences of positive integers, and states that this set is "essentially an oriented tree." What is the root of this oriented tree, and what are the arcs?

The vertices are the elements of S, the root is $\emptyset$, and the arcs go from $(x_1, \ldots, x_n)$ to $(x_1, \ldots, x_{n-1})$, for every $(x_1, \ldots, x_n) \in S \setminus \{\emptyset\}$.

▸ **3.** [*M23*] If it is possible to tile the upper right quadrant of the plane when given an *infinite* set of tetrad types, is it always possible to tile the whole plane?

No. For example, our tiles might be of the form



for $n \in \mathbb{N}$. Then in the (x, y) square we might place the tile with $n = \max\{x, y\}$ and this would tile the upper right quadrant, but there's obviously no way to tile the whole plane.

▸ **6.** [*M23*] (Otto Schreier.) In a famous paper, B. L. van der Waerden proved the following theorem:

> *If k and m are positive integers, and if we have k sets $S_1, \ldots, S_k$ of positive integers with every positive integer included in at least one of these sets, then at least of the sets S_j contains an arithmetic progression of length m.*

(The latter statement means there exist integers a and $\delta > 0$ such that $a + \delta$, $a + 2\delta$, ..., $a + m\delta$ are all in S_j.) If possible, use this result and the infinity lemma to prove the following stronger statement:

> *If k and m are positive integers, there is a number N such that if we have k sets $S_1, \dots, S_k$ of integers with every integer between 1 and N included in at least one of these sets, then at least one of the sets S_j contains an arithmetic progression of length m.*

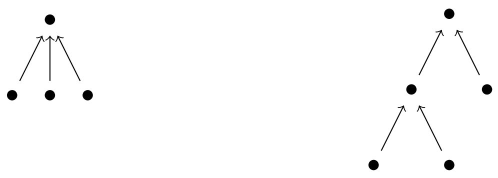
It is enough to prove it when the set that contains every integer between 1 and N is S_1. We define a tree whose vertices are the tuples $(S_1, \dots, S_k)$ of finite sets of integers such that, if $n = \max \bigcup_{i=1}^{k} S_i$, every positive integer up to n is in one of the sets, and such that no set contains an arithmetic progression of length m. The edges go from one such tuple to $(S_1 \setminus \{n\}, \dots, S_k \setminus \{n\})$, so the root is $(\emptyset, \dots, \emptyset)$ and n is the height of the node in the tree.

Since each vertex contains a finite number of children ($2^k - 1$, corresponding to the ways of adding the next number to one or more of the sets), it follows that if this tree were infinite, there would be an infinite path. By van der Waerden theorem, the component-wise union of this path would give us a tuple $(S_1, \dots, S_k)$ such that some S_j contains an arithmetic progression of length m, so by construction one of the nodes in the path would follow this condition.#

Thus the tree is finite and we just need to take N as one plus the depth of the tree.

2.3.4.4 Enumeration of trees

▸ **5.** [*M25*] (A. Cayley.) Let c_n be the number of (unlabeled) oriented trees having n leaves (namely, vertices with in-degree zero) and having at least two subtrees at every other vertex. Thus $c_3 = 2$, by virtue of the two trees



Find a formula analogous to (3) for the generating function

$$C(z) = \sum_n c_n z^n.$$

Every tree has at least one leaf, so $c_0 = 0$. This includes subtrees, so $c_1 = 1$ as, if the root weren't also a leaf, it would have at least two children and therefore two trees. For $n > 1$, the root has a tree and various subtrees. Let j_k be the number of subtrees with k leaves, $1 \leq k < n$, for those subtrees we may choose up to

$$\binom{c_k + j_k - 1}{j_k}$$

possibilities, since these are combinations with repetition (see exercise 1.2.6–60), so

$$c_n = \sum_{\substack{j_1,\dots,j_{n-1} \geq 0 \\ j_1 + 2j_2 + \cdots + (n-1)j_{n-1} = n}} \binom{c_1 + j_1 - 1}{j_1} \cdots \binom{c_{n-1} + j_{n-1} - 1}{j_{n-1}}.$$

Note that this is like (2) except that we have n under the sum instead of $n-1$. Thus, using the same identity as in the derivation of (3),

$$\begin{aligned}
C(z) &= c_0 + c_1 z + \sum_{n\geq 2} \sum_{\substack{j_1,\dots,j_{n-1}\geq 0 \\ j_1+2j_2+\dots+(n-1)j_{n-1}=n}} \prod_{k=1}^{n-1} \binom{c_k+j_k-1}{j_k} z^{kj_k} \\
&= z + \sum_{(j_n)_n\in\mathbb{N}^{(\mathbb{N}^*)}} \prod_{n\geq 1} \binom{c_n+j_n-1}{j_n} z^{nj_n} - \sum_{n\geq 1} c_n z^n - 1 \\
&= \prod_{n\geq 1}\sum_{j\geq 1} \binom{c_n+j-1}{j} z^{nj_n} - C(z) + z - 1 \\
&= \frac{1}{(1-z)^{c_1}(1-z^2)^{c_2}\cdots(1-z^n)^{c_n}\cdots} - C(z) + z - 1,
\end{aligned}$$

where $\mathbb{N}^{(\mathbb{N}^*)}$ is the set of sequences of natural numbers starting at j_1 and with a finite amount of nonzero coefficients; the subtracted term $\sum_{n\geq 1} c_n z^n$ is there to cancel the terms which correspond to nodes having just one child, and likewise for -1 for the term with all zeroes.

Thus,

$$C(z) = \frac{1}{2}\left(\prod_{n\geq 1}\frac{1}{(1-z^n)^{c_n}} + z - 1\right).$$

▸ **10.** [*M22*] Prove that a free tree with n vertices and two centroids consists of two free trees with $n/2$ vertices, joined by an edge. Conversely, if two free trees with m vertices are joined by an edge, we obtain a free tree with $2m$ vertices and two centroids.

$\Longrightarrow$] Let u and v be the two centroids, with weight m. If u had weight m from an edge that wasn't the one that leads to v, then v would have weight at least $m+1$, owing to the m nodes that stem from that edge in u and u itself.# By a similar argument, if there were an intermediate node $w \neq u, v$ in the path connecting u with v, its weight must necessarily be at most $m-1$#. Therefore u and v are the only centroids, they are connected with an edge and the subtrees that result from removing that edge have m nodes each.

$\Longleftarrow$] Let u and v be the two nodes that are newly connected, then u has weight m because of its connection with v, v has weight m because of its connection with u, and any other node has weight at least $m+1$ because of their connection to u and v.

14. [*10*] True or false: The last entry, $f(V_{n-1})$, in the canonical representation of an oriented tree is always the root of that tree.

True. After each removal step, we still have a tree with the same root, and by the end the tree has no edges and therefore it only has one node, which is the root. Just before that, it only has one edge, which must connect a direct child of the root to the root.

▸ **21.** [*M20*] Enumerate the number of labeled oriented trees in which each vertex has in-degree zero or two. (See exercise 20 and exercise 2.3–20.)

These are precisely the 0-2-trees in exercise 2.3–20. We know 0-2-trees always have an odd number of nodes, say $2m+1$ nodes. By the construction in the text, these trees correspond to sequences of $2m$ nodes where each node that appears does so exactly twice. There are $\binom{2m+1}{m}$ ways to choose which nodes *do* appear in the sequence, and $\frac{(2m)!}{2^m}$ ways to arranging them (we can think of arranging $2m$ elements and then discarding the relative order of equal pairs of elements). This gives us

$$\binom{2m+1}{m}(2m)! \Big/ 2^m$$

labeled oriented 0-2-trees.

2.3.4.5 Path length

▸ **3.** [*M24*] An extended binary tree with m external nodes determines a set of path lengths $l_1, l_2, \dots, l_m$ that describe the lengths of paths from the root to the respective external nodes. Conversely, if we are given a set of numbers $l_1, l_2, \dots, l_m$, is it always possible to construct an extended binary tree in which these numbers are the path lengths in some order? Show that this is possible if and only if $\sum_{j=1}^{m} 2^{-l_j} = 1$.

We assign each external node an interval of real numbers as follows: start with $[l, u) = [0, 1)$; then, for each edge in the path from the root to the node, if it's a left edge, set $[l, u) \leftarrow [l, \frac{l+u}{2})$, and if it's a right edge, set $[l, u) \leftarrow [\frac{l+u}{2}, u)$, so the interval's length is 2^{-l}, l being the path length. It's easy to see that these intervals are disjoint and that, for each $x \in [0, 1)$, there is a special node whose interval contains x. With this in mind:

$\Longrightarrow$] This is precisely the sum of the lengths of the intervals, which is 1 because $[0, 1)$ is the disjoint union of these intervals.

$\Longleftarrow$] We just have to sort the path lengths in increasing order, assign consecutive intervals of length 2^{-l_k} starting from 0 and converting the intervals to paths (more precisely, to sequences of left/right turns), which we can do since the increasing order ensures that the starting point l_k of an interval $[l_k, u_k)$ is a multiple of the length $u_k - l_k$. These paths are all different and none is a prefix of another one, so they define the leaves of a binary tree.

▸ **4.** [*M25*] (E. S. Schwartz and B. Kallick.) Assume that $w_1 \le w_2 \le \cdots \le w_m$. Show that there is an extended binary tree that minimizes $\sum w_j l_j$ and for which the terminal nodes in left to right order contain the respective values $w_1, w_2, \dots, w_m$.

Let T be an extended binary tree that minimizes $\sum w_j l_j$. For $i < j$, if $w_i < w_j$, then $l_i \ge l_j$ in that tree, as otherwise we could reduce the weight path length by swapping their positions as

$$(w_i l_j + w_j l_i) - (w_i l_i + w_j l_j) = (w_i - w_j)(l_j - l_i) < 0\#.$$

Thus we might assume $l_i \ge l_j$ for all $i < j$. This means lengths $l_1, \dots, l_m$ are in decreasing order, so the proof in the previous exercise gives us a tree whose nodes, in the order of the tree, are $w_m, \dots, w_1$ with lengths $l_m, \dots, l_1$, and we just have to swap left and right edges in that tree.

▸ **12.** [*M20*] Suppose that a node has been chosen at random in a binary tree, with each node equally likely. Show that the average size of the subtree rooted at that node is related to the path length of the tree.

Let $V_1, \dots, V_m$ be the internal nodes, with path lengths $l_1, \dots, l_m$. The average size of the subtrees is the sum of the number of nodes in each subtree divided by m, but in this sum, each node V_k is counted $l_k + 1$ times, one for the subtree generated by each node in the path from the root to V_k including both ends of the path. Thus this average is precisely

$$\frac{1}{m}\sum_k (l_k + 1) = \frac{1}{m}(I + m) = \frac{I}{m} + 1,$$

where I is the internal path length of the tree.

2.3.4.6 History and bibliography

▸ **1.** [*21*] Find a simple one-to-one correspondence between binary trees with n nodes and dissections of an $(n + 2)$-sided convex polygon into n triangles, assuming that the sides of the polygon are distinct.

Let $v_0, \dots, v_{n+1}$ be the vertices of the polygon. We start at the root and, if the left subtree has k nodes ($0 \le k < n$) and the right subtree has $n - k - 1$ nodes, we take out the triangle $\operatorname{conv}\{v_0, v_1, v_{k+2}\}$. This leaves us with a polygon of $k+2$ nodes $v_1, \dots, v_{k+2}$ when the left subtree is not empty, and one of $n - k + 1$ nodes $v_{k+2}, \dots, v_{n+1}, v_0$ when the right subtree is not empty, and we just have to dissect these two smaller polygons recursively, for example, by renaming v_{k+2} to v_0 in the polygon for the left subtree and $v_{k+2}, \dots, v_{n+1}$ to $v_1, \dots, v_{n-k}$ in the polygon for the right subtree.

To get back the original binary tree, we just have to see which other vertex is in the triangle that contains the edge between v_0 and v_1, take it out, and then "decode" the two remaining polygons if they exist, something we can do for arbitrary dissections of the polygon.

2.3.5 Lists and Garbage Collection

▸ **1.** [*M21*] In Section 2.3.4 we saw that trees are special cases of the "classical" mathematical concept of a directed graph. Can Lists be described in graph-theoretic terminology?

Yes, we can consider Lists as ordered trees where, in general, only the leaves have labels. If we allow Lists to be referenced from multiple places, then we can consider them as graphs where outgoing arcs from a given node are ordered and such that there is a distinguished vertex R such that there is at least one path from R to any other vertex. These Lists can be used to reason about computer memory in higher level programming languages.

6. [*00*] The quantitative discussion at the end of this section says that the cost of garbage collection is approximately c_1N + c_2M units of time; where does the "c_2M" term come from?

From scanning the memory in search of nodes to collect.

2.4 Multilinked Structures

1. [*00*] Considering COBOL data configurations as tree structures, are the data items listed by a COBOL programmer in preorder, postorder, or neither of these orders?

They are in preorder.

2. [*10*] Comment about the running time of Algorithm A.

It takes a constant amount of time for each data item read (assuming that accessing the symbol table takes constant time). There is also a loop to get out of a nested record, but each iteration of the loop takes out an item from the stack so we may consider that as part of processing such item. Therefore the running time is proportional to the number of data items in the input.

8. [*10*] Under what circumstances is "MOVE CORRESPONDING α TO β" exactly the same as "MOVE α TO β", according to the definition in the text?

When α or β is an elementary item.

▸ **11.** [*23*] What additional links or changes in the strategy of the algorithms of the text could make Algorithm B or Algorithm C faster?

(I had to look up the solution.) One could make Algorithm C faster by storing sibling nodes in the table in lexicographical order, so step C3 would have a linear rather than quadratic complexity.

▸ **13.** [*24*] Give an algorithm to substitute for Algorithm A when the Data Table is to have the format shown in exercise 12.

We remove the instructions that set PARENT, CHILD, SIB, and NAME, as those fields do not exist now (in particular, A6 disappears), and we implement Q $\Leftarrow$ AVAIL in step A2 by sequential allocation. We set the SCOPE of an entry when we remove the corresponding item from the stack in step A5. (Note that we do the same in A5 when L1 = L as when L1 > L, so we could just do those instructions unconditionally, then after removing the current stack item and before loading the next one we check if L1 = L and if so we go to A7, then if L1 $\geq$ L we repeat, otherwise we report the error.)

For this to work, we have to get the stack empty by the time the algorithm finishes, which we can do by setting L $\leftarrow$ 0 when the input is exhausted in step A2 and then terminate in step A7 when L = 0.

2.5 Dynamic Storage Allocation

▸ **5.** [*18*] Suppose it is known that `N` is always 100 or more in Algorithm A. Would it be a good idea to set $c = 100$ in the modified step A4$'$?

No, because while we cannot use that storage *right now*, it might be worth it after the next allocated block gets freed.

▸ **6.** [*23*] (*Next fit.*) After Algorithm A has been used repeatedly, there will be a strong tendency for blocks of small `SIZE` to remain at the front of the `AVAIL` list, so that it will often be necessary to search quite far into the list before finding a block of length `N` or more. For example, notice how the size of the blocks essentially increases in Fig. 42, for both reserved and free blocks, from the beginning of memory to the end. (The `AVAIL` list used while Fig. 42 was being prepared was kept sorted by the order of location, as required by Algorithm B.) Can you suggest a way to modify Algorithm A so that

1. short blocks won't need to accumulate in a particular area, and
2. the `AVAIL` list may still be kept in order of increasing memory locations, for purposes of algorithms like Algorithm B?

Yes, we could arrange the free space as a circular list and start each search where the previous one left off. In more precise terms, in A1 we would change `Q` $\leftarrow$ `LOC(AVAIL)` to `Q` $\leftarrow$ `AVAIL`, in A4 we would add `AVAIL` $\leftarrow$ `Q`, and the test in A2 would be `Q` $=$ `AVAIL` rather than `P` $= \Lambda$ and it would be skipped in the first iteration. Other algorithms that deal with memory management might need to be modified as well to avoid invalidating `AVAIL`. For example, algorithm B would have to deal with this when freeing the block just before it.

7. [*10*] The example (1) shows that first-fit can sometimes be definitely superior to best-fit. Give a similar example that shows a case where best-fit is superior to first-fit.

memory request	*available areas, first-fit*	*available areas, best-fit*
—	1300, 1200	1300, 1200
1100	200, 1200	1300, 100
250	200, 950	1050, 100
1000	stuck	50, 100

▸ **12.** [*20*] Modify Algorithm A so that it follows the boundary-tag conventions (7)–(9), uses the modified step A4' described in the text, and also incorporates the improvement of exercise 6.

Algorithm A' (*Next-fit method with boundary-tag conventions*). Let the heap be stored as in (7)–(9), c be as in step A4' in the text, and let `PTR` be a variable whose value persists among runs of this algorithm and which starts with the value `AVAIL`. (We could just get rid of the list head in `LOC(AVAIL)` and use this `Q` instead of `AVAIL`, but let's follow convention (9).) For this algorithm `N` is two more than the number of words we want to allocate.

A1. [Initialize.] Set $\texttt{P} \leftarrow \texttt{PTR}$.

A2. [Is SIZE enough?] If $\texttt{SIZE(P)} \geq \texttt{N}$, go to 2.5.

A3. [End of list?] Set $\texttt{P} \leftarrow \texttt{LINK(P)}$. If $\texttt{P} = \texttt{PTR}$, the algorithm terminates unsuccessfully; there is no room for a block of N consecutive blocks. Otherwise go back to 2.5.

A4. [Reserve N.] Set $\texttt{K} \leftarrow \texttt{SIZE(P)} - \texttt{N}$, but if $\texttt{K} < c$, set $\texttt{K} \leftarrow 0$. Then, set $\texttt{L} \leftarrow \texttt{P} + \texttt{K}$ and $\texttt{TAG(L)} \leftarrow \texttt{TAG(P} + \texttt{SIZE(P)} - 1) \leftarrow$ "+". Finally, if $\texttt{K} = 0$, set $\texttt{LINK(LINK(P}+1)) \leftarrow \texttt{LINK(P)}$ and $\texttt{LINK(LINK(P)} + 1) = \texttt{LINK(P} + 1)$; otherwise set $\texttt{SIZE(L)} \leftarrow \texttt{N}$, $\texttt{SIZE(P)} \leftarrow \texttt{SIZE(L} - 1) \leftarrow \texttt{K}$, and $\texttt{TAG(L} - 1) \leftarrow$ "−". The result is an allocation of size $\texttt{SIZE(L)} - 2 \geq \texttt{N} - 2$ that spans from $\texttt{L} + 1$ to $\texttt{L} + \texttt{SIZE(L)} - 1$.

▸ **15.** [*24*] Show how to speed up Algorithm C at the expense of a slightly longer program, by not changing any more links than absolutely necessary in each of four cases depending on whether $\texttt{TAG(P0} - 1)$, $\texttt{TAG(P0} + \texttt{SIZE(P0))}$ are plus or minus.

1. [Initialize.] Set $\texttt{S} \leftarrow \texttt{SIZE(P0)}$ and $\texttt{PU} \leftarrow \texttt{P0} + \texttt{S}$.
2. [Check lower bound.] If $\texttt{TAG(P0} - 1) =$ "+", go to step 5.
3. [Move to lower area.] Set $\texttt{S} \leftarrow \texttt{S} + \texttt{SIZE(P0} - 1)$ and $\texttt{P0} \leftarrow \texttt{P0} - \texttt{SIZE(P0} - 1)$.
4. [Remove upper area.] If $\texttt{TAG(PU)} =$ "−", set $\texttt{P1} \leftarrow \texttt{LINK(PU)}$, $\texttt{P2} \leftarrow \texttt{LINK(PU} + 1)$, $\texttt{LINK(P1} + 1) \leftarrow \texttt{P2}$, $\texttt{LINK(P2)} \leftarrow \texttt{P1}$, and $\texttt{S} \leftarrow \texttt{S} + \texttt{SIZE(PU)}$. Go to step 7.
5. [Allocate.] If $\texttt{TAG(PU)} =$ "+", set $\texttt{LINK(P0)} \leftarrow \texttt{AVAIL}$, $\texttt{LINK(P0}+1) \leftarrow \texttt{LOC(AVAIL)}$, $\texttt{LINK(AVAIL} + 1) \leftarrow \texttt{P0}$, and $\texttt{AVAIL} \leftarrow \texttt{P0}$.
6. [Merge upper area.] Otherwise let $\texttt{P1} \leftarrow \texttt{LINK(P0)} \leftarrow \texttt{LINK(PU)}$, $\texttt{P2} \leftarrow \texttt{LINK(P0} + 1) \leftarrow \texttt{LINK(PU}+1)$, $\texttt{LINK(P1}+1) \leftarrow \texttt{LINK(P2)} \leftarrow \texttt{P0}$, and $\texttt{S} \leftarrow \texttt{SIZE(P0)}+\texttt{SIZE(PU)}$.
7. [Update size.] Set $\texttt{TAG(P0)} \leftarrow \texttt{TAG(P0} + \texttt{S} - 1) \leftarrow$ "−" and $\texttt{SIZE(P0)} \leftarrow \texttt{SIZE(P0} + \texttt{S} - 1) \leftarrow \texttt{S}$.

17. [*10*] What should the contents of LOC(AVAIL) and LOC(AVAIL) + 1 be in (9) when there are no available blocks present?

They should both point to LOC(AVAIL).

▸ **18.** [*20*] Figures 42 and 43 were obtained using the same data, and essentially the same algorithms (Algorithms A and B), except that Fig. 43 was prepared by modifying Algorithm A to choose best-fit instead of first-fit. Why did this cause Fig. 42 to have a large available area in the *higher* locations of memory, while in Fig. 43 there is a large available area in the *lower* locations?

Both algorithms allocate at the end of the block of free memory that they choose, and in the beginning, both will allocate from higher to lower locations, leaving a large area at the end.

However, as soon as some of the areas allocated are freed, the best fit method will tend to use those newly freed areas from the higher locations since they will tend to be smaller, whereas the first fit method will prefer to allocate in the large lower area.

Eventually the first-fit method fills up the large area and starts allocating at other areas in the lower part, leaving the higher locations mostly free as the first chunks are freed, whereas the best fit method continues to avoid filling up this large area.

Note that this works because most allocations have a limited lifetime; in other algorithms it might be the case that some of the first allocations are never freed until the end of the algorithm while many other chunks appear and disappear quickly. In this case, the first fit method would still show some allocations in the higher locations, although more sparse than those in lower locations.

▸ **19.** [*24*] Suppose that blocks of memory have the form of (7), but without the `TAG` or `SIZE` fields required in the last word of the block. Suppose further that the following simple algorithm is being used to make a reserved block free again: $\texttt{Q} \leftarrow \texttt{AVAIL}$, $\texttt{LINK(P0)} \leftarrow \texttt{Q}$, $\texttt{LINK(P0+1)} \leftarrow \texttt{LOC(AVAIL)}$, $\texttt{LINK(Q+1)} \leftarrow \texttt{P0}$, $\texttt{AVAIL} \leftarrow \texttt{P0}$, $\texttt{TAG(P0)} \leftarrow$ "−". (This algorithm does nothing about collapsing adjacent areas together.)

Design a reservation algorithm, similar to Algorithm A, that does the necessary collapsing of adjacent free blocks while searching the `AVAIL` list, and at the same time avoids any unnecessary fragmentation of memory as in (2), (3), and (4).

The following algorithm reserves a block of size $\texttt{N} \geq 2$ in the first available area while collapsing adjacent free blocks to find such area. In order to avoid unnecessary fragmentation, it allocates in the beginning of the block that is found rather than in the end.

1. [Initialize.] Set $\texttt{P} \leftarrow \texttt{AVAIL}$.

2. [Collapse free blocks.] Let $\texttt{PU} \leftarrow \texttt{P} + \texttt{SIZE(P)}$. If $\texttt{TAG(PU)} =$ "−", then set $\texttt{P1} \leftarrow \texttt{LINK(PU)}$, $\texttt{P2} \leftarrow \texttt{LINK(PU + 1)}$, $\texttt{LINK(P1 + 1)} \leftarrow \texttt{P2}$, $\texttt{LINK(P2)} \leftarrow \texttt{P1}$, $\texttt{SIZE(P)} \leftarrow \texttt{SIZE(P)} + \texttt{SIZE(PU)}$, and repeat this step.

3. [Do we have enough space?] If $\texttt{SIZE(P)} < \texttt{N}$, go to step 5.

4. [Advance.] Set $\texttt{P} \leftarrow \texttt{LINK(P)}$. If $\texttt{P} = \texttt{LOC(AVAIL)}$, terminate unsuccessfully; otherwise go to step 2.

5. [Reserve block.] Set $\texttt{P1} \leftarrow \texttt{LINK(P)}$, $\texttt{P2} \leftarrow \texttt{LINK(P + 1)}$, $\texttt{LINK(P1 + 1)} \leftarrow \texttt{P2}$, $\texttt{LINK(P2)} \leftarrow \texttt{P1}$, and $\texttt{TAG(P)} \leftarrow$ "+″. $\texttt{PF} \leftarrow \texttt{P} + \texttt{N}$,

6. [Reclaim extra space.] Let $\texttt{K} \leftarrow \texttt{SIZE(P)} - \texttt{N}$ and $\texttt{PU} \leftarrow \texttt{P} + \texttt{SIZE(P)}$. If $\texttt{K} = 0$, or $\texttt{K} = 1$ and $\texttt{TAG(P1)} =$ "+", then terminate the algorithm. Otherwise, set $\texttt{P1} \leftarrow \texttt{AVAIL}$ and $\texttt{P2} \leftarrow \texttt{LOC(AVAIL)}$ if $\texttt{TAG(PU)} =$ "+", $\texttt{P1} \leftarrow \texttt{LINK(PU)}$ and $\texttt{P2} \leftarrow \texttt{LINK(PU + 1)}$ otherwise. Finally set $\texttt{LINK(P1 + 1)} \leftarrow \texttt{LINK(P2)} \leftarrow \texttt{PF} \leftarrow \texttt{P} + \texttt{N}$, $\texttt{LINK(PF)} \leftarrow \texttt{P1}$, $\texttt{LINK(PF + 1)} \leftarrow \texttt{P2}$, $\texttt{SIZE(PF)} \leftarrow \texttt{K}$, $\texttt{TAG(PF)} \leftarrow$ "−", and $\texttt{SIZE(P)} \leftarrow \texttt{N}$. The resulting block is at position `P`.

20. [*00*] Why is it desirable to have the `AVAIL`[k] lists in the buddy system doubly linked instead of simply having straight linear lists?

To be able to remove arbitrary elements from the lists efficiently in step S2 of the liberation algorithm.

▸ **22.** [*21*] The text repeatedly states that the buddy system allows only blocks of size 2^k to be used, and exercise 21 shows this can lead to a substantial increase in the storage required. But if an 11-word block is needed in connection with the buddy system, why couldn't we find a 16-word block and divide it into an 11-word piece together with two free blocks of sizes 4 and 1?

We could indeed modify the algorithms to work like this, although they would get quite complicated and slower (more blocks would have to be marked as reserved for the liberation algorithm to work).

23. [*05*] What is the binary address of the buddy of the block of size 4 whose binary address is 011011110000? What would it be if the block were of size 15 instead of 4?

011011110100 and 011011100000, respectively.

▸ **25.** [*22*] Criticize the following idea: "Dynamic storage allocation using the buddy system will never reserve a block of size 2^m in practical situations (since this would fill the whole memory), and, in general, there is a maximum size 2^n for which no blocks of greater size will ever be reserved. Therefore it is a waste of time to start with such large blocks available, and to combine buddies in Algorithm S when the combined block has a size larger than 2^n."

While this is correct when we actually know the program and the size of the allocations it can make is bounded for any valid input, there are many situations where this is not the case, for example when we are writing a library or when the size of some allocation depends on the input; then the program would spuriously fail for inputs that it could otherwise have processed at least in a some computer.

▸ **26.** [*21*] Explain how the buddy system could be used for dynamic storage allocation in memory locations 0 through $\mathtt{M}-1$ even when `M` does not have the form 2^m as required in the text.

If m is an integer such that $2^m \leq \mathtt{M} < 2^{m+1}$, the memory could be split into pieces of size 2^k, $0 \leq k \leq m$, no more than one piece of the same size. For example, for locations 0 to 3999, we would initially have one free area at 0 of size 2048, another one at 2048 of size 1024, another one at 3072 of size 512, another one at 3584 of size 256, another one at 3840 of size 128, and another one at 3968 of size 32.

▸ **36.** [*20*] A certain lunch counter in Hollywood, California, contains 23 seats in a row. Diners enter the shop in groups of one or two, and a glamorous hostess shows them where to sit. Prove that she will always be able to seat people immediately without splitting up any pairs, if no customer who comes alone is assigned to any of the seats numbered 2, 5, 8, ..., 20, provided that there never are more than 16 customers present at a time. (Pairs leave together.)

A single customer can always seat, and they can avoid the “forbidden” seats. Now, if a pair comes, there were at most 14 customers already there. If seats 22 and 23 are free, then we can sit the pair there. Otherwise there are at most 13 customers in the seats 1–21, so if we divide the seats in 7 groups of 3 (1–3, 4–6, etc.), at least one of these groups has no more than one seat taken, and it cannot be the middle one, so that group has two consecutive free seats where we can seat the pair.