The Theory of Computation
Bernard M.E. Moret
Addison-Wesley-Longman, 1998

Solution Manual
Chapter 2, Text Problems

Exercise 2.1

We need to show that an acyclic connected (undirected and finite) graph has exactly one fewer vertices than edges.

We proceed by induction on the number of vertices of the graph. We actually prove first that any acyclic connected graph with n vertices and at least one vertex of degree 1 has $n-1$ edges, then prove that any acyclic graph must have at least one vertex of degree less than 2.

So let us first consider connected graphs with at least one vertex of degree 1. The basis is vacuously true for a graph of $n = 1$ vertices, since such a graph cannot have any edges and is connected by default. (If you dislike starting with a basis that does not display the property very convincingly, consider a graph of 2 vertices: the two vertices must be connected by an edge for the graph to be connected and that is the only edge possible, so we have 1 edge and 2 vertices, thereby obeying the conclusion.) Assume then that all graphs on $n-1$ or fewer vertices obey the theorem and consider a connected acyclic graph $G = (V, E)$ of n vertices. If G has a vertex of degree 1, then let v be such a vertex and consider the graph G' obtained by removing from G the vertex v and its associated edge; G' is connected and has $n-1$ vertices so that it obeys the inductive hypothesis. Thus G' has $n-2$ edges and thus G must have $n-1$ edges, as desired.

What if all vertices of G have degree 2 or higher? We claim that this cannot occur, because any such graph would have a cycle. Again, we prove this by induction. The theorem is vacuously true again for graphs with one vertex or two vertices, since a degree of 2 is not possible in such graphs. (Again, if you dislike vacuous bases, consider a graph on 3 vertices where every vertex has degree 2: there is only one such graph, the complete graph on 3 vertices, K_3, and it has a cycle.) Assume then that such is true of all graphs of up to $n-1$ vertices and consider a graph G of n vertices where all vertices have degree at least 2. Let v be a vertex of lowest degree (which is at least 2) and u and w be two vertices to which v is connected. Two cases arise. If there is an edge between u and w, then G has a cycle (the three vertices u, v, and w form a cycle), as desired. If there is no edge in G between u and w, consider the new graph G' obtained from G by removing vertex v from G and adding the edge from u to w. G' has $n - 1$ vertices, all of degree 2 or higher, and thus obeys the inductive hypothesis, so that it has a cycle. If this cycle does not involve the edge $\{u.w\}$, it is also present in G and we are done. If the cycle uses edge $\{u.w\}$, then G has almost the same cycle, where the edge $\{u, w\}$ (which does not exist in G) is replaced by the two edges $\{u, v\}$ and $\{v, w\}$.

Exercise 2.2

We prove the the chromatic number of K_n is in two steps: first we show that n colors suffice, then we show that n colors are needed. That n colors suffice is rather trivial: since every vertex will be given its own separate color, we can never have any edge with identically colored endpoints. To verify that n colors are necessary, we use contradiction: if we use fewer than n colors, then there will be at least two identically colored vertices; but there is an edge between any two vertices in a complete graph, so that the edge between these two vertices will violate the requirements of a legal coloring.

To see that a tree with at least two vertices always has chromatic number 2, consider the simple coloring strategy: color the root red; then color the children black; then color the grandchildren red; and so on, alternating colors as we alternate odd and even levels. The result is a legal coloring because the only edges present in the tree are between adjacent levels—which always have differently colored vertices. By our previous result, two colors are needed as soon as the graph includes a K_2—which is just a single edge. Any tree with two or more vertices has at least one edge and thus requires two colors.

Exercise 2.3

The infinite number of tour buses, each of infinite size, is just the same as the infinite number of rows, each of infinite size, in our table of rationals. The solution is thus to use the dovetailing technique used for the rational numbers.

Basically, we will conceptually order all tourists on the buses (assign them numbers 1, 2, 3, etc.), then we will be in the situation described in Example 2.1, where we have a full *Infinite Hotel* and an infinite number of new guests and simply move each existing guest in room i to room $2i$ and assign new guest j to room $2j - 1$. Dovetailing on the buses is exactly like dovetailing on rows: we take the first tourist on the first bus, then the second tourist on the first bus and the first tourist on the second bus, then the third tourist on the first bus, the second tourist on the second bus, and the first tourist on the third bus, etc.

Exercise 2.4

Our function is

$$\begin{cases} f(1) = 1 \\ f(2n) = f(N) + 1 \\ f(2n+1) = 1/f(2n) \end{cases}$$

We claim that it maps even natural numbers to positive rational numbers with a value greater than 1 and odd natural numbers to positive rational numbers with a value less than 1—with 1 being mapped to 1. First note that any values produced must be positive: we either add 1 or take the (multiplicative) inverse, but never affect the sign of the result nor have any way to reduce the result to 0 if started with a nonzero value. That being the case, $f(2n)$ must be larger than 1 for any $n > 1$, since it is defined as $f(n) + 1$; and thus also $f(2n + 1)$ must be less than 1 for any $n > 1$, since it is the inverse of $f(2n)$.

To verify that it is a bijection, we need to verify injectivity and surjectivity. Let us start with injectivity. Let i and j be two positive integers; we claim that $f(i)$ and $f(j)$ can only be equal if i and j are. From our remarks above, $f(i)$ and $f(j)$ can only be equal if i and j are both odd or both even. Suppose that i and j are both even, say $i = 2m$ and $j = 2n$; then we have $f(i) = f(2m) = f(m) + 1$ and $f(j) = f(2n) = f(n) + 1$, so that $f(i)$ and $f(j)$ can only be equal if $f(n)$ and $f(m)$ are. Suppose that i and j are both odd, say $i = 2m + 1$ and $j = 2n + 1$; then we have $f(i) = f(2m + 1) = 1/f(2m)$ and $f(j) = f(2n + 1) = 1/f(2n)$, so that $f(i)$ and $f(j)$ can only be equal if $f(2n)$ and $f(2m)$ are and, in turn, these two can only be equal if $f(n)$ and $f(m)$ are. Thus, in both cases, we have reduced the arguments by a factor of 2 and still have the same implication; in other words, we have an inductive step! The basis is clearly true (for instance, we can observe $f(1) \neq f(2)$).

To verify surjectivity, we need to show that, for any rational number a/b (a reduced fraction), there is a positive integer n with $f(n) = a/b$. The easiest way to prove such a statement is to solve the second question in the problem: how is f reversed? Then, given a/b, we can reverse f on it and obtain x. So, given some reduced fraction a/b, we begin by testing whether it is larger than or less than 1: if larger, it was obtained by mapping an even integer, otherwise it was obtained by mapping an odd integer. Each step in the mapping can be reversed directly: if a/b is larger than 1, the step is reversed by subtracting 1, otherwise it is reversed by taking the multiplicative inverse. The result is the following simple recursive procedure:

```
function invert{fraction}
  case fraction of
  = 1: return(1)
  > 1: return(2*invert(reduce(fraction-1)))
  < 1: return(1+invert(1/fraction))
```

(where the function `reduce` returns the fraction in reduced terms). The remaining question is whether or not this procedure always converges; if it does, then it is clear that what it returns is indeed the pre-image of the given fraction, so that it does indeed implement f^{-1}, so that also f is surjective.

To see that it converges, note first that the current value of the fraction is always strictly positive: we start with a strictly positive value and each step keeps it strictly positive. Next, note that the inversion step (for fractions less than 1) must be preceded and followed by one or more subtraction steps: inverting a fraction less than 1 yields a fraction larger than 1. (On the other hand, we can have many consecutive subtraction steps.) Thus at least one out of two steps is a subtraction step and each such step reduces the numerator of the fraction. Since numerators and denominators are integers, the number of times they can be decremented before reaching 1 is bounded, so that the procedure must converge.

Exercise 2.5

By inspection, this function is monotonic in both x and y (all terms are positive). To verify that it is a pairing function, we need to verify injectivity and surjectivity. As in Exercise 2.4, we shall verify surjectivity by showing that we can invert the pairing.

We thus begin by showing injectivity. The key to injectivity is to realize that the function creates a perfect square, then adds x to it; thus, intuitively, the function is injective because, between two consecutive perfect squares, x determines the value of the function—and then, knowing x, the value of y is also fixed. Formally, we begin by verifying that x cannot be large enough to bridge the difference between two consecutive perfect squares. An increase of 1 in the term being squared causes an increase of $2y + 2\lfloor \frac{x+1}{2} \rfloor + 1$ in the value of the function; this increase is indeed always at least as large as x, so that x cannot bridge the difference between consecutive perfect squares. That being the case, the function is injective: for two pairings to yield the same result, they must have the same perfect square and thus the same x; but then they have the same y.

The observation about perfect squares also show that the function is easily reversed: find the largest perfect square that does not exceed the value of the function; the difference is the value of x. Then take the root of the perfect square and subtract the floor of $\frac{x+1}{2}$ to obtain y. Thus the inversion given in the problem is correct and the function is surjective. Hence we have a pairing function.

Exercise 2.6

The first thing to note about our definition of the projections functions is that the recursion terminates. The index of recursion is the superscript (the j in Π_i^j). We have three families of base cases, Π_i^0, Π_i^1, and Π_i^2, and a recursive step, Π_i^j for $j > 2$. The recursive step always reduces the superscript by one, so termination is assured.

It thus remains to show that our definition properly reverses pairing; again, as in the statement of the exercise, note that the projection function may return any value whatsoever when its indices are not compatible with the reversal of a pairing—given Π_i^j, we need only worry about the range $1 \le i \le j$. Moreover, since the base cases are all clearly correct (the only interesting case is for $j = 2$), we can also assume $j > 2$. So let $z = \langle x_1, \ldots, x_{n-2}, x_{n-1}, x_n \rangle_n$ for some $n > 2$ (thus x_1 and x_{n-2} might be the same entry); we want to prove that we have $\Pi_i^n(z) = x_i$ for $1 \le i \le n$. Our proof is by induction: values of n up to 2 give us the basis and we now want to show that the statement hold for some $n > 2$ if it holds for $n - 1$ and all lower values. By definition of pairing, we have

$$z = \langle x_1, \ldots, x_{n-2}, x_{n-1}, x_n \rangle_n = \langle x_1, \ldots, x_{n-2}, \langle x_{n-1}, x_n \rangle \rangle_{n-1}$$

and so we can write

$$z = \langle y_1, \ldots, y_{n-2}, y_{n-1} \rangle_{n-1}$$

where we have set $y_k = x_k$ for $1 \le k \le n - 2$ and $y_{n-1} = \langle x_{n-1}, x_n \rangle$. By definition of our projection functions, we have (for $n > 2$)

$$\Pi_i^n(z) = \begin{cases} \Pi_i^{n-1}(z) & i < n - 1 \\ \Pi_1(\Pi_{n-1}^{n-1}(z)) & i = n - 1 \\ \Pi_2(\Pi_{n-1}^{n-1}(z)) & i > n - 1 \end{cases}$$

By inductive hypothesis, all of the Pi_k^{n-1} return the kth of $n - 1$ paired values, so we have $\Pi_k^{n-1}(z) = y_k$ for $1 \le k \le n - 1$.

In the third case, we must have $i = n$ and so we can write

$$\Pi_n^n(z) = \Pi_2(\Pi_{n-1}^{n-1}(z)) = \Pi_2(y_{n-1}) = \Pi_2(\langle x_{n-1}, x_n\rangle) = x_n$$

as desired. In the second case, we have

$$\Pi_{n-1}^n(z) = \Pi_1(\Pi_{n-1}^{n-1}(z)) = \Pi_1(y_{n-1}) = \Pi_1(\langle x_{n-1}, x_n\rangle) = x_{n-1}$$

as desired. Finally, in the first case ($i < n - 1$), we have

$$\Pi_i^n(z) = \Pi_i^{n-1}(z) = y_i = x_i$$

(by inductive hypothesis) as desired.

Exercise 2.7

1. By induction on n. By definition $\langle 0, 0\rangle$ must be the smallest value produced by a pairing function (because of monotonicity), thus it must be 0. Assume then that $\langle 0, 0, \ldots, 0\rangle_{n-1}$ equals 0; by definition, we have

   $$\langle 0, 0, \ldots, 0\rangle_n = \langle 0, 0, \ldots, \langle 0, 0\rangle\rangle_{n-1} = \langle 0, 0, \ldots, 0\rangle_{n-1} = 0$$

   The pairing of x with zeros is quite similar. As basis, we have

   $$\langle x, 0, 0\rangle_3 = \langle a, \langle 0, 0\rangle\rangle = \langle x, 0\rangle$$

   (the last from the result above) and the step gives us

   $$\langle x, 0, 0, \ldots, 0\rangle_n = \langle x, 0, \ldots, \langle 0, 0\rangle\rangle_{n-1} = \langle x, 0, \ldots, 0\rangle_{n-1} = \langle x, 0\rangle$$

   as desired.
2. By definition, we have $\Pi_i^{n+1}(z) = \Pi_i^n(z)$ for $i < n$; thus we have $\Pi_0^n(z) = \Pi_0^{n-1}(z) = \ldots = \Pi_0^1(z)$. Again by definition we have $\Pi_i^1(z) = z$ for all i. Thus we have $\Pi_0^n(z) = \Pi_0^1(z) = z = \Pi_1^1(z)$..
   By definition, we have $\Pi_i^{n+1}(z) = \Pi_2(\Pi_n^n(z))$ for all $i > n$; thus, in particular, we obtain the same value for $i = n + 1$ and for any larger i, so we can write $\Pi_{n+1}^{n+1}(z) = \Pi_i^{n+1}(z)$ for all $i \geq n + 1$, or, reducing n by 1, $\Pi_n^n(z) = \Pi_i^n(z)$ for all $i \geq n$.
3. We have, by definition $\Pi_1^0(z) = 0$, which is obviously no larger than any z. Similarly. we have $\Pi_1^1(z) = z$, again no larger than z. These two form the basis of an inductive proof. Assume then that we have $\Pi_1^j(z) \leq z$ for all j up to $n - 1$ and consider $\Pi_1^n(z)$. By definition, we have $\Pi_1^n(z) = \Pi_1^{n-1}(z)$; but the second term is no larger than z by inductive hypothesis, so our conclusion holds.

Exercise 2.8

1. We need a collection of items; let that be a set S. Each item has a size which is a natural number; thus we have an assignment of sizes to the elements of S, i.e., a function from S to $\mathbb{N}$, call it $s\colon S \longrightarrow \mathbb{N}$. Finally, all bins have the same size, but that size must be specified, say $B \in natural$. These items define an instance of the problem.
   Placing an item x in the ith bin is equivalent to assigning to item x an index i; let that be the assignment function $b\colon S \longrightarrow \{1, 2, \ldots, k\}$ when a total of k bins are used. Then we want all items assigned to bin i to fit within the bin, that is, we want to have $\sum_{b(x)=i} s(x) \le B$. These are the conditions that a solution to the problem must fulfill.
   The objective is then to find the smallest number of bins in which it is possible to pack all items, that is, to identify the smallest value of k for which an assignment function b exists that fulfills the conditions.
   Thus an instance of our problem is given by a finite set S of items, a size function $s\colon S \longrightarrow \mathbb{N}$ that assigns a size to each item, and a bin size $B \in \mathbb{N}$. The objective is to minimize k subject to finding a function $b\colon S \longrightarrow \{1, 2, \ldots, k\}$ obeying $\sum_{b(x)=i} s(x) \le B$ for each $1 \le i \le k$.
2. We need a Boolean formula; this requires a certain number of variables, call them x_1, dots, x_n. Specifying the formula can be done with a suitable alphabet (with symbols for the Boolean operators and for parentheses as well as for the variables) or by encoding everything into a simpler alphabet. A solution is a truth assignment, that is, a function from the set of variables to the set {true,false}, that causes the formula to evaluate to 'true."
   Further details could be given about the evaluation of the formula, but such tend to be messy and uninformative—and we all know how to evaluate a Boolean formula. Most *Satisfiability* problems are restricted to instances that use only certain types of Boolean connectives (e.g., just $\vee$, $\wedge$ and)/and often certain types of formula, in which cases more formal descriptions of legal instances and of the constraint on the solution are quite reasonable.
3. The input is a sequence of numbers, presumably natural integers; we need a notion of the length of that sequence. So we can formalize the input as an ordered sequence of n natural integers, say $(x_1, x_2, \ldots, x_n)$ with $x_i \in \mathbb{N}$. A subsequence can be viewed as a selection from the set $\{1, 2, \ldots, n\}$ of indices, i.e., a function $s\colon \{1, 2, \ldots, n\} \longrightarrow \{0, 1\}$, where $s(i) = 1$ denotes the fact that the x_i is part of the chosen subsequence. The condition that must be met by a solution may then

be written as

$$i < j \Longrightarrow (s(i) = 0 \vee s(j) = 0 \vee x_i < x_j)$$

The objective of the problem is to identify a selection function s that selects as many items as possible, that is, that maximizes the sum $\sum_{i=1}^{n} s(i)$.

Formally, then, our problem is given by an ordered sequence of n natural integers, $(x_1, x_2, \ldots, x_n)$ with $x_i \in \mathbb{N}$. The objective is to find a function $s\colon \{1, 2, \ldots, n\} \longrightarrow \{0, 1\}$ that maximizes the sum $\sum_{i=1}^{n} s(i)$ subject to

$$i < j \Longrightarrow (s(i) = 0 \vee s(j) = 0 \vee x_i < x_j)$$

for all $1 \leq i, j \leq n$.

Exercise 2.9

The relationships are: (2) is a subproblem of (1) and (4) is (a version of) a subproblem of each of the other three.

That (2) is a subproblem of (1) is clear: the two problems have exactly the same question, but the set of legal instances of (2) is a subset of the set of legal instances of (1).

Problem (3) asks a different question and it is not hard to find instances common to (1) and (3) or to (2) and (3) that yield one answer for (1) or (2) and a different answer for (3). For instance, consider the set $\{a, b, c, d, e\}$ with subsets $\{a, b, c\}$, $\{a, b, e\}$, $\{a, c, d\}$, and $\{c, d, e\}$. This is a legal instance of (1), (2), and (3) (but not of (4)); in both (1) and (2), a solution is the pair of subsets $\{a, b, c\}$ and $\{c, d, e\}$, but in (3) there is no solution at all! Thus (3) is not a subproblem of either (1) or (2), nor is either (1) or (2) a subset of (3).

Finally, consider (4): we claim that it is a subproblem of (1), (2), and (3). Note that its legal set of instances is a restricted subset of the legal set of instances of any of the other three—we further restrict the instances of (2) by requiring a certain size for the set itself (a multiple of 3). Now note that any cover for a set of $3n$ elements made entirely of subsets of 3 elements each must have at least n subsets; hence asking for n subsets is asking for the smallest possible cover. In that sense, the problem is a subproblem of (1) and (2), although there remains a difference: if no cover of size n exists, (1) and (2) will still return a (larger) cover, but (4) will simply return "no solution." Requiring that the cover consists of n subsets of 3 elements each means that each of the $3n$ elements of the set appears in exactly one of the n

subsets—otherwise we would not have a cover with just n subsets. Thus the requirement that the cover have size n is equivalent to a requirement that the subsets forming the cover be disjoint; thus (4) effectively asks exactly the same question as (3), so that it is a subproblem of (3).

Exercise 2.10

The requirement on a minimum test set is to separate every pair of classes that is separated by the full collection of tests. If we replace "separate" by "cover," we have the solution to our questions! All we need to do is to set up the set S of all pairs separated by the full collection of tests (S is at most quadratic in the number of classes), then, for each test, couch it as a subset of pairs—all the pairs that this test separates by itself. Finding a minimum test set is then the same problem as finding a minimum cover for S.

Exercise 2.11

We are to show that the relation "is big Oh of" is transitive.

If f is $O(g)$, then there exist constants c_1 and N_1 such that, for all $n > N_1$, we have $f(n) \le c_1 g(n)$; similarly, if g is $O(h)$, then there exist constants c_2 and N_2 such that, for all $n > N_2$, we have $g(n) \le c_2 h(n)$.

Now simply set $N = \max\{N_1, N_2\}$ and $c = c_1 \cdot c_2$. Then, for all $n > N$, we have $f(n) \le c_1 g(n) \le c_1 c_2 h(n) = ch(n)$. Thus we have shown that f is $O(h)$.

Exercise 2.12

If 3^n were $O(2^n)$, there would exist $c, N > 0$ such that, for all $n > N$, $3^n \le c \cdot 2^n$, or, equivalently, $\frac{3^n}{c \cdot 2^n} \le 1$. But, for any constant $c > 0$,

$$\lim_{n\to\infty} \frac{3^n}{c \cdot 2^n} = c \cdot \lim_{n\to\infty} \left(\frac{3}{2}\right)^n$$

diverges, so that no such c and N can exist.

On the other hand, we can write $3^n = 2^{\alpha n}$ and solve for α, obtaining $\alpha = \log_2 3$. Hence, for any $\alpha \ge \log_2 3$, it is indeed the case that 3^n is $O(2^{\alpha n}$.

Our first line of reasoning is not directly affected if we replace n by n^k for $k > 1$—the ratio simply diverges a little faster, since it now becomes $\frac{3}{2}^{n^k}$. Thus we can state that 3^{n^k} is not $O(2^{n^k}$ for $k > 1$.

Our second line of reasoning is equally unaffected; we have

$$2^{\log_3 2 \cdot n^k} = \left(2^{\log_2 3}\right)^{n^k} = 3^{n^k}$$

so that the analogous statement holds: for any $\alpha \ge \log_2 3$, it is indeed the case that 3^{n^k} is $O(2^{\alpha n^k}$.

Exercise 2.13

- $n!$ is difficult to manipulate. The easiest way to solve this problem is to use Stirling's approximation, which, in asymptotic terms, gives $n! = \Theta(\sqrt{n}(\frac{n}{e})^n)$. Using this result, it is then easy to prove both points. n^n grows faster than $n!$, since the limit of their ratio,

$$\lim_{n\to\infty} \frac{n^n}{n!} = \lim_{n\to\infty} \frac{n^n}{\sqrt{n}(\frac{n}{e})^n} = \lim_{n\to\infty} \frac{e^n}{\sqrt{n}}$$

diverges. But $\log n^n = n \log n$, while $\log n! = \Theta(\frac{1}{2}n + n(\log n - \log e)) = \Theta(n \log n)$, so that the two functions grow at exactly the same rate.
Without Stirling's approximation, we need more work, but it is also more interesting. We begin by writing the inequality that would have to be respected if n^n were $O(n!)$: for some c, N, we would have to have, for all $n > N$, $n^n \leq c \cdot n!$, or $\frac{n^n}{n!} \leq c$. Consider the ratio: if we can show that it diverges, then no suitable constant c can exist. Unfortunately, applying L'Hospital's rule right away would require differentiating $n!$. So let us instead take logs on both sides of the inequality:

$$n \log n - \sum_{i=1}^{n} \log i \leq \log c = c'$$

We can rewrite this inequality as

$$\sum_{i=1}^{n} (\log n - \log i) \leq c'$$

The sum is not easy to manipulate, but note that the first $n/2$ terms are each at least as large as $\log n - \log(n/2)$, which is in turn just 1, so that we can write (for even n)

$$\sum_{i=1}^{n} (\log n - \log i) \geq \sum_{i=1}^{n/2} (\log n - \log(n/2)) = \sum_{i=1}^{n/2} 1 = n/2$$

But that difference grows with n and thus can never be bounded by a constant c'.
For the logs, we can use ratios directly and use L'Hospital's rule to determine the convergence of these ratios. (Recall that the rule states that the limit of the ratio can be obtained by taking the limit of the ratio of the derivative of the numerator to the derivative of the

denominator.) The limit of the ratio of the logarithms can be derived as follows.

$$\lim_{n\to\infty} \frac{\log n^n}{\log n!} = \lim_{n\to\infty} \frac{n \log n}{\sum_{i=1}^{n} \log i}$$

Now apply L'Hospital's rule (the derivative of $\log n$ is $1/n$ times a constant, which drops out of the ratio—we could as easily have used logarithms in base e):

$$= \lim_{n\to\infty} \frac{\log n}{\sum_{i=1}^{n} \frac{1}{i}}$$

Now note that the denominator, $\sum_{i=1}^{n} \frac{1}{i}$, is an approximation of the integral $\int_1^n \frac{1}{x} dx$; in fact, we can easily verify (e.g., visually, by looking at the rectangles given by the sum and at the curve $\frac{1}{x}$) that we have

$$\int_1^{n+1} \frac{1}{x} dx < \sum_{i=1}^{n} \frac{1}{i} < 1 + \int_1^n \frac{1}{x} dx$$

Since we have $\int_1^n \frac{1}{x} dx = \ln n$, it follows that we have $\sum_{i=1}^{n} \frac{1}{i}$ is $\Theta(\log n)$ and we are done.

- For $n^{\alpha \log n}$ to be $O(n^{\log n})$, we would have to find $c, N > 0$ such that, for all $n > N$, we have $n^{\alpha \log n} \leq c \cdot (n^{\log n})$. But the limit of the ratio $\frac{n^{\alpha \log n}}{c \cdot (n^{\log n})}$ diverges, as we prove below:

$$\lim_{n\to\infty} \frac{n^{\alpha \log n}}{c \cdot (n^{\log n})} = \lim_{n\to\infty} \frac{n^{\alpha \log n - \log n}}{c} = \lim_{n\to\infty} \frac{n^{(\alpha-1) \log n}}{c}$$

Since $\alpha > 1$, the numerator is n raised to a power equal to some positive multiple of $\log n$ and thus grows as n grows, whereas the denominator is a constant; therefore, the ratio diverges in the limit and no c and N can be found that meet the requirements.
For the logs, we have $\log n^{\alpha \log n} = \alpha \log n \log n = \alpha (\log n)^2$ and $\log n^{\log n} = \log n \log n = (\log n)^2$; thus the two are within a constant ratio (α) of each other, so the result is clearly true.

Exercise 2.14

Here are a few results about $O(\,)$ notation.

1. [Transitivity] If $f(n)$ is $O\big(g(n)\big)$ and $g(n)$ is $O\big(h(n)\big)$, then $f(n)$ is $O\big(h(n)\big)$.

2. [Scaling] If $f(n)$ is $O\big(g(n)\big)$, then, for any $k > 0$, $f(n)$ is $O\big(k \cdot g(n)\big)$.
3. [Rule of sums] If $f_1(n)$ is $O\big(g_1(n)\big)$ and $f_2(n)$ is $O\big(g_2(n)\big)$, then $\big(f_1 + f_2\big)(n)$ is $O\big(\max\big(g_1(n), g_2(n)\big)\big)$.
4. [Rule of products] If $f_1(n)$ is $O\big(g_1(n)\big)$ and $f_2(n)$ is $O\big(g_2(n)\big)$, then $(f_1 \cdot f_2)(n)$ is $O\big((g_1 \cdot g_2)(n)\big)$. Note that $(f_1 \cdot f_2)(n)$ refers to the product of the values of the two functions at n and not to the value at n of the composition of the two functions.

Proof.

1. $f(n)$ is $O\big(g(n)\big)$ says that there exist c_1 and N_1 such that $f(n) \leq c_1 \cdot g(n)$ holds for all $n \geq N_1$; similarly, $g(n)$ is $O\big(h(n)\big)$ says that there exist c_2 and N_2 such that $g(n) \leq c_2 \cdot h(n)$ holds for all $n \geq N_2$. Take $c = c_1 \cdot c_2$ and $N = \max(N_1, N_2)$. Now for any $n \geq n_0$, we have

$$f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot \big(c_2 \cdot h(n)\big) = c \cdot h(n),$$

   where the first inequality follows from $n \geq N_1$ and the second from $n \geq N_2$. Thus, for any $n \geq N$, we have $f(n) \leq c \cdot h(n)$, so that $f(n)$ is $O\big(h(n)\big)$.
2. Again, $f(n)$ is $O\big(g(n)\big)$ means that there exist c_1 and N_1 such that $f(n) \leq c_1 \cdot g(n)$ holds for all $n \geq N_1$. Take $c = c_1/k$ and $N = N_1$. Then, for $n \geq N$, we have $f(n) \leq c_1 \cdot g(n) = (ck)g(n) = c\big(k \cdot g(n)\big)$, so that $f(n)$ is $O\big(k \cdot g(n)\big)$.
3. Before proving this result, note that $\max\big(g_1(n), g_2(n)\big)$ is a function defined *pointwise*, i.e., independently for each n; for some n it takes on the value of g_1, and for others it takes on the value of g_2. Take $c = 2 \cdot \max(c_1, c_2)$ and $N = \max(N_1, N_2)$. The rest follows easily.
4. Here take $c = c_1 \cdot c_2$ and $N = \max(N_1, N_2)$. The rest follows easily.

Q.E.D.

Exercise 2.15

To prove that the two define the same concept, we need to prove two implications. Actually, as we see below, one implication is true (the second one) and one is only "almost" true—i.e., false...

- If f is $\Theta(g)$, then the limit of the ratio $\frac{f}{g}$ is a positive constant.
  Our definition is a symmetric $O(\,)$. Thus there exist positive constants c_1, c_2, N_1, and N_2 such that, for all $n > N_1$, $f(n) \leq c_1 g(n)$ and, for all $n > N_2$, $g(n) \leq c_2 f(n)$. Let $N = \max(N_1, N_2)$; then the two inequalities

hold for all $n > N$. By dividing both inequalities by $g(n)$ and isolating the ratio $\frac{f}{g}$, we get the inequalities

$$\frac{1}{c_2} \leq \frac{f(n)}{g(n)} \leq c_1 \text{ for all } n > N$$

Thus the ratio is bounded above and below by two positive constants This is not exactly the same as postulating that the limit of the ratio is a fixed positive constant; conceivably, we could have strange functions that make the ratio oscillate forever between $\frac{1}{c_2}$ and c_1, so that the limit does not exist. However, the difference is negligible in practice—and is easy to fix by replacing the ratio $\frac{f}{g}$ by $\sup \frac{f}{g}$ (or by $\inf \frac{f}{g}$), the supremum (infimum) of the ratio, thereby ensuring that the limit exists.

- If the limit of the ratio $\frac{f}{g}$ is a positive constant, then f is $\Theta(g)$ by our definition.
  We need to return to calculus and the definition of limits. If we know that the limit of the ratio is a fixed constant c, it means that, for each positive constant ϵ, there exists a positive constant N such that

$$\left| \frac{f(n)}{g(n)} - c \right| \leq \epsilon, \text{ for all } n > N$$

or, rewriting as two inequalities,

$$c - \epsilon \leq \frac{f(n)}{g(n)} \leq c + \epsilon, \text{ for all } n > N$$

Comparing with our derivation for the other implication, we see that this is equivalent to our definition of $\Theta()$, with $\frac{1}{c_2} = c - \epsilon$ and $c_1 = c + \epsilon$.

Exercise 2.16

We prove the result indirectly, by proving that the sum of all node degrees is even. It then follows immediately that the number of nodes of odd degree must be even.

The sum of the degrees of the nodes is even because it is exactly twice the number of edges of the graph: each edge contributes a value of 1 to exactly two nodes' degrees and thus contributes exactly 2 to the sum of all node degrees.

Exercise 2.17

To prove Euler's theorem about the existence Eulerian circuits, we need two parts, since the result is a set of necessary and sufficient conditions.

That the conditions are necessary is clear: any Eulerian circuit has to enter and leave each vertex exactly the same number of times, so that the indegree of each vertex must equal its outdegree—or, for an undirected graph, the degree of each vertex must be even.

The difficult part of the proof is the sufficiency. Assume that we have a connected undirected graph where all vertices have even degree. We need to show that this graph has an Eulerian circuit. We use induction on the number of edges. The basis is vacuously true for $m = 0$, since such a graph has no edges. (If you are uneasy with vacuous bases, consider the case $m = 3$—each vertex has degree 2 and so the graph is just K_3 which clearly has an Eulerian circuit, which, in this case only, also happens to be a Hamiltonian circuit. The cases $m = 1$ and $m = 2$ cannot arise, since vertices could not all have even degree.)

Assume then that the statement holds for all graphs of up to $m - 1$ edges and consider a connected undirected graph of m edges ($m > 3$) where all vertices have even degree. By an earlier result (Exercise 2.1), such a graph must have a cycle. Remove the edges of this cycle from the graph: what is left is an undirected graph on fewer edges where all vertices still have even degree. If that graph is connected, we are done, since the inductive hypothesis applies and it thus has an Eulerian circuit, to which we can add the cycle just removed. The trouble is that the resulting graph need not be connected. However, each connected component must obey the inductive hypothesis (all of its vertices have even degree and it has fewer than n edges), so that each has an Eulerian circuit; moreover, the cycle that we originally removed intersects each of these circuits (otherwise the graph would not have been connected in the first place), so that we can construct an Eulerian circuit for the entire graph by moving along the cycle, entering an Eulerian circuit at the first intersection with a connected component, completing that circuit, and then following the cycle until it first intersects the next connected component, etc. (We take no special action when the cycle intersects a connected component for the second, third, or kth time, but just continue on the cycle.) The correctness of this step is clear, so that our inductive step is proved.

Exercise 2.18

By definition, a strongly connected graph has a (directed) path between any pair of vertices. If our graph has n vertices, call them $\{v_1, v_2, \ldots, v_n\}$, then in particular, we have a directed path from v_1 to v_2, from v_2 to v_3, and so on, going to a directed path from v_{n-1} to v_n, and finally one from v_n to

v_1. Concatenating all of these paths yields a directed cycle from v_1 back to itself that passes through all vertices of the graph (although, more than likely, more than once through many of them).

Exercise 2.19

Together, a graph and its complement (on n vertices) have $n(n-1)/2$ edges. In order for the graph and its complement to be isomorphic, each must have the same number of edges, so that $n(n-1)/2$ must be divisible by 2—or $n(n-1)/4$ must be an integer.

Write $n = 4i + j$; then we must have $(4i + j)(4i + j - 1)/4 = 4i^2 + 2ij - i + \frac{1}{4}(j^2 - j)$ an integer,that is, $j^2 - j$ must be a multiple of 4. Since j can only assume the values 0, 1, 2, or 3, we can check each of the 4 and verify that $j^2 - j = 0$ (a multiple of 4) for $j = 0$ and $j = 1$ and that $j^2 - j$ is not a multiple of 4 for $j = 2$ or $j = 3$.

Exercise 2.20

We want to prove the equality $|R| = |E| - |V| + 2$. A non-trivial region must have at least 3 vertices and 3 edges on its boundary; each edge can be shared by at most two regions.

We exclude a graph with no edge and no vertex. The three smallest nonempty graphs otherwise all obey the equality. If the graph has 1 vertex, it has 0 edges, and 1 region, so that the equality holds. If the graph has 1 edge, then it has 1 regions; it also has 2 vertices, so the equality holds. For 2 edges, we still have 1 region, but 3 vertices; again, the equality holds.

Let us then assume that the equality holds for graphs of up to m edges, that is, we have $r = m - n + 2$ for all such graphs, where r is the number of regions and n the number of vertices. Since our basis covers cases up to $m = 2$ and $n = 3$, assume our graph already has at least 2 edges and 3 vertices. If we add an edge, we can add it in just three ways:

1. We add one vertex and one edge by placing the vertex within a region and connecting a vertex on the boundary of the region to the new vertex; the number of regions does not change, so the equality is respected.
2. We split an existing edge into two edges by placing a vertex in the "middle." The number of regions is unaffected, while the number of vertices and the number of edges both go up by 1; hence the equality is respected.
3. We add an edge between two existing vertices; in that case, these vertices must both be on or in the same region (otherwise planarity would be destroyed), so that the new edge will split that region into

two. The number of edges goes up by 1, the number of regions goes up by 1, and the number of vertices remains unchanged, so that the equality is respected.

No other way is possible: adding an edge and two vertices would imply that the new edge is unattached—but we must build a connected graph; adding an edge between two existing vertices without splitting a region is not possible once the graph has at least 3 vertices—as ours does by inductive hypothesis.

Exercise 2.21

By Exercise 2.20, we have the equality $|R| = |E| - |V| + 2$. A non-trivial region must have at least 3 vertices and 3 edges on its boundary; each edge can be shared by at most two regions. Thus the number of regions is at most 2/3 of the number of edges, $|R| \leq \frac{2}{3}|E|$. Putting the two inequalities together, we get

$$|E| - |V| + 2 \leq \frac{2}{3}|E|$$

or

$$|E| \leq 3|V| - 6$$

as desired.

Exercise 2.22

By Exercise 2.21, a graph with 11 vertices cannot have more than $3 \cdot 11 - 6 = 27$ edges; thus its complement must have at least $(11 \cdot 10)/2 - 27 = 28$ edges. But no planar graph on 11 vertices may have more than 27 edges, so that the complement must be nonplanar.

The calculation is easy: let the number of vertices be n. Then the largest number of edges that the graph may have (and remain planar) is $3n - 6$; the complement of a graph of n vertices and $3n - 6$ edges has $n(n-1)/2 - (3n - 6) = \frac{1}{2}(n^2 - 7n + 12)$ edges. In order for the complement to remain planar, it also must have at most $3n - 6$ edges, yielding the inequality $\frac{1}{2}(n^2 - 7n + 12) \leq 3n - 6$, or $n^2 - 13n + 24 \leq 0$, with roots $\frac{1}{2}(13 + \sqrt{73})$ and $\frac{1}{2}(13 - \sqrt{73})$ Since the square root of 73 is slightly larger than 8.5, our n must be less than $\frac{1}{2}(13 + 8.5)$ and an integer, that is, must be at most 10. Thus 11 is indeed the smallest value with that property, as could also have been deduced from the fact that the the number of edges of the complement was the smallest value that would violate the limit and that the number of edges of the complement grows quadratically while that of the planar graph grows only linearly.

Exercise 2.23

In Exercise 2.1, we proved several useful statements about a tree, ending with the conclusion that a tree on n vertices has exactly $n - 1$ edges. That is by itself sufficient to prove both statements made in the exercise at hand: a tree is maximally connected acyclic graph because adding any edge to it induces a cycle, as was proved in Exercise 2.1; and a tree is a minimally connected graph, because any graph with fewer edges is disconnected, as we also proved in Exercise 2.1

Exercise 2.24

We can combine our result of Exercise 2.16—that the sum of the degrees of the vertices of undirected graph (or the sum of the outdegrees and indegrees of the vertices of a directed graph) is exactly twice the number of edges (or arcs) of the graph—with that of Exercise 2.1—that a tree on n vertices has $n - 1$ edges—to obtain directly our first result: the sum of the degree of the vertices of a tree of n vertices is $2(n - 1)$.

The converse is harder. We prove it by induction on n, the number of nodes. The basis (for $n = 1$) is vacuously true, since the degree is 0 and since there exists a single tree with one vertex and no edge. Assume then that the result holds for all values of i up to $n - 1$ and consider a collection of n natural numbers, $\{d_1, d_2, \ldots, d_n\}$s (where we have $n > 1$) obeying $\sum_{i=1}^{n} d_i = 2(n - 1)$. At least one of the d_i must equal 1, since otherwise (all d_i are at least 2), we would have a sum at least as large as $2n$. Let j be the index of one such value, i.e., we have $d_j = 1$. Thus vertex j is a leaf of the putative tree we want to build; but note that a leaf can be attached to *any* non-leaf node of the tree—that is, it has no "structural" effect on the tree. Thus we remove d_j from our collection and decrease one of the other values by 1, making sure to pick one that is at least 2 (i.e., that is not another leaf). (Again, there must be at least one d_i with a value of at least 2, since otherwise the sum would only equal n.) Now the resulting collection has $n - 1$ numbers and its sum is the old sum, $2(n - 1)$, minus 2 (the removed d_j and the one decremented value), that is, its sum is $2(n - 1) - 2 = 2(n - 2)$. Thus the resulting collection obeys the inductive hypothesis and can indeed be mapped into a tree; once that is done, we attach to (any node of) this tree one more leaf and we have our desired tree.

Exercise 2.25

We can provide a purely constructive solution to this problem, with a linear-time recursive algorithm based on depth-first search of the tree.

```
/* n is the total number of nodes of the tree;
   it may already be known; if not, it can be computed in linear time
   with any tree traversal */
/* a is any node of the tree -- it need not be a root or a leaf */
/* the main procedure just calls the recursive subroutine with some
   initialization; the recursive subroutine basically computes the size
   of each subtree, but keeps checking whether the collection of subtrees
   of a node number among them at least n/2 nodes, in which case
   that node is the desired answer and the recursive routine aborts */

main procedure
  begin
    check(a)
  99:
  end

function check(t: tree)
  /* on normal return, the value of the function is the number of nodes
     in the subtree rooted at t */
  begin
    count := 0
    for all (non-nil) children c of t do
        count := count + check(c)
    if count > n/2 go to 99 /* abort the recursion: t is the answer */
    check := count + 1   /* to account for t itself */
  end
```

Correctness is fairly obvious. Termination by the `goto` must occur, since the value of `count` keeps increasing. At termination, the node `t` being explored by the dfs has a collection of dfs children that together include at least `n/2` nodes; but none of its children has that property, because none caused termination (we had a normal return from each of them). Thus none of the dfs subtrees of `t` has more than `n/2` nodes; but the dfs subtree rooted at `t` has at least `n/2` nodes, so that the rest of the tree (which can be viewed as a subtree of `t`, but hanging "up" from it, since its first member, if any, is the parent of `t` in the dfs search), if any, cannot have more than `n/2` nodes. Hence `t` is the desired answer. The dfs search artificially ordered the tree, by assigning it a root `a` and ordering all nodes "downwards" from there; but the ordering does not matter, since any node can cause termination and since the termination condition ensures that any subtree connected to the termination node has at most `n/2` nodes in it—whether that subtree is one of the dfs subtrees or the subtree that would be created by removing the node and its dfs descendants. In particular, note that, if `a` is the answer, it will correctly be selected by the routine, since none of its subtrees will be large enough to cause termination.

Exercise 2.26

The key insight is to realize that there cannot be any advantage to not matching a leaf. Of course we may not be able to match all leaves—whenever two or more leaves share the same "parent" (neighbor), at most one of these leaves can be matched, but there is no gain in leaving them all unmatched. This gives rise to a simple recursive strategy: recursively traverse the tree (depth-first search); on backing out of a leaf, if its parent is not matched, then match it with the leaf; in general, in backing our of an unmatched node, if its parent is not matched, then match it with the node. This algorithm clearly runs in linear time. It is optimal because the principle of labeling leaves whenever possible (which is easily seen never to hurt) is applied recursively by the algorithm: having matched what leaves is could, it then removes from consideration all edges touching matched vertices and effectively starts again matching what newly created leaves it can.

Exercise 2.27

This is the proof of Hall's theorem. Since the theorem states necessary and sufficient conditions, our proof has two parts. Let the bipartite graph be $G = (\{U, V\}, E)$, with $|V| < |U|$.

The necessary condition is easy. If G has a matching of size $|V|$, denote that matching by the assignment of each vertex of V of some vertex of U, $m\colon V \longrightarrow U$, where m is injective—in this somewhat unusual notation, a matched edge is of the form $\{v, m(v)\}$ for $v \in V$. Then note that, for any subset $X \subseteq V$, the set $U_X = \{u \in U \mid \exists v \in X,\ \{u, v\} \in E\}$ contains at least all of the vertices $m(x)$ for $x \in X$. Since m is injective, we must have $|m(X)| = |X|$ and so the set U_X has cardinality at least as large as that of X, as desired.

The sufficient condition is harder. Assume that, for each subset X of V, we have $|U_X| \geq |X|$. We need to prove that G then has a matching of size $|V|$. We proceed by induction on the size of V. The statement holds vacuously for $|V| = 0$. If V contains only one vertex, a matching of size $|V| = 1$ exists if there is at least one edge from that vertex to some vertex in U, which is exactly what the condition states. Assume then that the statement holds for all graphs with vertex sets V no larger than $n - 1$ and consider a graph G with vertex set V of $n > 1$ vertices. We consider two cases:

- For each proper, nonempty subset X of V, $\emptyset \subset X \subset V$, we have $|U_X| > |X|$. Then pick any vertex v in V and use any edge incident upon v to match v, say to vertex $u \in U$. Then consider the induced bipartite graph on $U - \{u\}$ and $V - \{v\}$. This new graph obeys the inductive hypothesis (we had strict inequalities and now we have reduced the size of the U_X for any X by at most 1, so we have, at worst, equalities) and thus it has a matching of size $|V| - 1$; adding in the edge $\{v, u\}$, which cannot, by construction, share any endpoint with the inductively defined matching, yields a matching of size $|V|$ for the original graph.

- There is a nonempty proper subset X of V with $|U_X| = |X|$. The induced bipartite graph formed by X and U_X obeys the inductive hypothesis, so that it has a matching of size $|X|$. Consider then the induced bipartite graph formed by $V - X$ and $U - U_X$; it too obeys the inductive hypothesis! (Otherwise, there would be a set $A \subseteq V - X$ with $|A| > |U_A|$ and thus such that $|A \cup X| > |U_{A \cup X}|$ in the original graph, a contradiction.) Thus this induced graph has a matching of size $|V| - |X|$, one that is entirely disjoint (no shared endpoint) from the matching obtained for X. The union of the two matchings is thus a matching of size $|V|$ for the original graph, as desired.

Exercise 2.28

Since we need to prove an equality, we have two parts to the proof. Let $G = (\{U, V\}, E)$ be our bipartite graph—we need not have $|U| = |V|$.

One part is very easy. Let $M \subseteq E$ be a matching of maximal size for G. We want to show that a vertex cover of minimum size for G must have size at least $|M|$. Since no two edges in M share a vertex, a vertex cover for just M will require at least one vertex per edge and thus at least $|M|$ vertices.

Let $C \subset U \cup V$ be a vertex cover of minimum size for G. We want to show that G has a maximum matching of size at least $|C|$. If no edge of G has both endpoints in the cover, the problem is trivial: for each cover vertex, just pick one of the edges that it covers. Otherwise there must be at least one cover vertex that covers only edges that have no other endpoint in the minimum cover. (The only graph for which that is not true is the complete graph on$n$ vertices, which is not bipartite.) For that vertex, we pick an arbitrary edge among those it covers for the matching, then remove the matched edge and all edges that share an endpoint with it from the graph. The result is a subgraph with two fewer vertices and at least one less edge, which has a minimum cover of size $|C| - 1$. By induction, that subgraph has a matching of size at least $|C| - 1$; to this matching we can add the matched edge we had picked, since it cannot share any endpoint with edges from the subgraph. Thus we have obtained a matching of size $|C|$ for G. (The basis for the induction is simple: the statement is true for graphs with no vertex on one side of the bipartite graph, and thus no edge at all.)

Taken together, our two results show that the size of a maximum matching is bounded above and below by that of a minimum cover, so that the two must be equal.

Exercise 2.29

The idea here is to map a bipartite graph onto a suitable 0/1 matrix and then show that the result of Exercise 2.28 is equivalent to the statement of this exercise. Fortunately, the mapping is quite simple: given the bipartite graph $G = (\{U, V\}, E)$, we define an $|U| \times |V|$ 0/1 matrix where the ijth entry is 1 if and only if there is an edge between the ith vertex of U and the jth vertex of V. This mapping is clearly a bijection between 0/1 matrices and bipartite graphs. We show that the term rank of this matrix equals the size of the maximum matching. Then we show that the minimum number of rows and columns that contain all nonzero entries equals the size of the minimum cover.

The term rank is the largest number of nonzero entries, no two in the same row or column. Each nonzero entry corresponds to an edge; having no two entries in the same row or column is thus exactly equivalent to requiring a matching—i.e., to requiring that no two edges (nonzero entries) share an endpoint (a row or a column). Thus the term rank is exactly the size of the largest matching.

Containing all nonzero entries is equivalent to covering all edges; rows and columns correspond exactly to vertices. Thus the smallest collection of rows and columns that contain all nonzero entries is exactly the smallest collection of vertices that covers all edges, as we needed to show.

Thus the two theorems (this exercise and Exercise 2.28) are stating the same result in different terms.

Exercise 2.30

The prefix sum at any given point is simply the difference between the number of left (opening) parentheses and the number of right (closing) parentheses. That number must be nonnegative everywhere because every closing parenthesis must match an opening one that preceded it. That number must be 0 at the last parenthesis, because by then we must have seen exactly the same number of each. Thus it is clear that a string of properly nested parentheses will yield the proper prefix sum—the conditions are necessary.

We need to prove that the conditions are sufficient—that any string of parentheses with a prefix sum that has these two properties must be properly nested. It is simplest to proceed by induction.

Clearly, there is only one string of length 2 that is nested, namely (); its prefix sum is as indicated. Assume then that all strings of length $2n$ that obey the properties of the prefix sum are properly nested and consider a string of length $2(n+1)$ that also obeys these properties. (There is of course no point in considering strings of odd length: no such string can ever have a last value of 0—nor could any such string be balanced.) The first entry is nonnegative, so the first character must be an opening parenthesis; the last entry is 0 and the one before non-negative, so the last character must be a closing parenthesis; however, we cannot just remove the first and last character and proceed inductively, because the first and last parenthesis need not form a matched pair—we could have the string ()(), for instance. So we scan the prefix sum for any appearance of a value of 0 (we know there is at least one, at the last entry). The first such value (scanning from the first entry) is the one we want because it is the closing parenthesis that matches the first symbol: that it is a closing parenthesis follows from its value and the fact that the previous entry cannot be negative, while the fact that it matches the first symbol is simply due to the fact that it is the first place where the number of opening parentheses exactly matches the number of closing ones. Thus we can in fact treat any substring ending in 0 independently of the others. We remove this entry and the first entry from the string and decrease all intervening values by 1 (to reflect that fact that we removed an opening parenthesis). Since all intervening values are larger than 0, the resulting string of prefix sum values still obeys the first condition. What about the second condition—after all, we just removed the last entry with its 0 value! But note that the entry preceding the 0 must have been a 1 (no other choice is possible), so that it is now, after the decrease by 1, a 0, as needed to obey the second condition. Thus the inductive hypothesis applies, the string between the opening and closing pair must be balanced and the opening and closing pair obviously maintains that property, so that the whole string (or each substring, if there were several 0 values) is balanced.

Exercise 2.31

Let S have m elements and T have n elements and consider functions from S to T.

- Assume that we have $m \geq n$. How many distinct surjective functions are there? Because we must map n of the elements of S to the n elements of T, we get $\binom{m}{n} \cdot n! = m!/(m-n)!$ choices there (how to choose the n elements and then the obvious $n!$ permutations of mapping). The remaining $m - n$ elements of S can each be mapped onto anything at all in T, and so appear to give another factor of n^{m-n}, for a total of $n^{m-n}m!/(m-n)!$ choices. However, this expression overcounts the number of choices: the n^{m-n} remaining choices in fact will cause repetitions, even when they are all mapped to the same element of T, because one or more of these $n-m$ mappings could be used in place of the n mappings chosen earlier. For example, consider mapping the set $\{a, b, c, d\}$ to the set $\{0, 1\}$. We can pick the subset $\{a, b\}$ to map injectively to $\{0, 1\}$, say with a mapped to 0 and b mapped to 1, then pick any mapping for c and d, say by mapping c to 0 and d to 1; but that is indistinguishable from picking the subset b, c to map injectively to $\{0, 1\}$ and then mapping a to 0 and d to 1.

To come up with an exact formula, we need to use the inclusion/exclusion principle: start with the $m!/(m-n)!$ choices for the injection on a subset of n elements, then painstakingly count the number of choices for the remaining elements. We will not obtain a nice and concise formula from this process: instead, it will remain in a sum form.

- Assume that we have $m \leq n$. How many distinct injective functions are there? That one is quite simple: we have n choices for mapping the first element of S, then $n-1$ for the second, and so on; thus we have $n!/(n-m)!$ choices in all.

Exercise 2.32

To obtain a derangement of the set $\{1, 2, \ldots, n\}$, we begin by mapping the first element, 1, to some other element of the set, say i, for which we have $n-1$ choices. The key at this point is to realize that it does not matter too much that we now have to map the remaining set $\{2, 3, \ldots, n\}$ to a different set of the same size, namely, $\{1, \ldots, i-1, i+1 \ldots, n\}$; the number of choices for derangements that we now have is easily related to the number of derangements of the set $\{2, 3, \ldots, n\}$, because any derangement of that set is a valid completion of our initial map, where we replace whatever was mapped to i (which, note, could not have been i) by a map to 1; in addition, we get the possibility of mapping i to 1 and any derangement of the set $\{2, \ldots, i-1, i+1, \ldots, n\}$. Thus our recurrence for the number of derangements of a set of size n is

$$d(n) = (n-1) \cdot (d(n-1) + d(n-2))$$

with $d(1) = 0$ and $d(2) = 1$. This is a very fast-growing function—it is clearly a double exponential, since the right-hand side is $n-1$ times the familiar

Fibonacci right-hand side. This is also not a recurrence with a clean closed form, so we stop here.

Exercise 2.33

1. Assume we have two inverses for f, call them g and h. Consider the composition $f \cdot f^{-1}$, which is the identity on Ts: we would have, with $f^{-1} = g$, $f(f^{-1}(y)) = f(g(y))$ and with $f^{-1} = h$, $f(f^{-1}(y)) = f(h(y))$, for any $y \in T$. Since each is the identity, we would then have $f(g(y)) = f(h(y)) = y$, for any $y \in T$. Now apply f^{-1} to the first equality: we have $f^{-1}(f(g(y))) = f^{-1}(f(h(y)))$, or $(f^{-1} \cdot f)(g(y)) = (f^{-1} \cdot f)(h(y))$, for all $y \in T$; but $f^{-1} \cdot f$ is the identity on S, so that we have $g(y) = h(y)$, for all $y \in T$. Hence we have $g = h$, as desired.
2. Much the same manipulations as under (1) prove the result.
   We begin with the only if part. Let x and y be two elements of S. If we have $f(x) = f(y)$, then we can apply f^{-1} to each side to yield $f^{-1}(f(x)) = f^{-1}(f(y))$, or $(f^{-1} \cdot f)(x) = (f^{-1} \cdot f)(y)$; but $f^{-1} \cdot f$ is the identity on S, so that we have $x = y$, as desired. Hence f is injective. Now let z be any element of T and consider $y = f^{-1}(z)$. By definition, we must have $y \in S$. If we apply f to the equality above, we get $f(y) = f(f^{-1}(z))$ or $f(y) = (f \cdot f^{-1})(z)$; but $f \cdot f^{-1}$ is the identity on T, so that we have $f(y) = z$ and thus, for any $z \in T$, there exists a $y \in S$ (namely, $f^{-1}(z)$) with $f(y) = z$. Hence f is surjective. Being both injective and surjective, it is a bijection.
   Now for the if part. If f is a bijection, it is both injective and surjective. Thus, given any $z \in T$, there exists a $y \in S$ with $f(y) = z$. Moreover, we claim that this y is unique: if we have some $x \in S$ also with $f(x) = z$, it would violate the injectivity of f. Since the y is unique, we can legitimately define $f^{-1}(z) = y$. By construction, we then have $(f \cdot f^{-1})(z) = f(f^{-1}(z)) = f(y) = z$, so that $f \cdot f^{-1}$ is the identity on T, as desired. Consider now $(f^{-1} \cdot f)(x)$ for some $x \in S$; since we have $z = f(x) \in T$, our construction of f^{-1} ensures that there exists a unique $y \in S$ such that $f(y) = z$; but this y must then be our x, since our x has that property and since it must be unique. Thus we can write $(f^{-1} \cdot f)(x) = f^{-1}(f(x)) - f^{-1}(z) = x$, showing that $f^{-1} \cdot f$ is the identity on S, as desired.
3. First we verify that $g^{-1} \cdot f^{-1}$ has the desired properties. Then we appeal to part (1) of this exercise, where we showed that the inverse must be unique, so that $g^{-1} \cdot f^{-1}$, which is *an* inverse, must be *the* inverse.

We verify that $h \cdot h^{-1}$ is an identity: $h \cdot h^{-1} = (f \cdot g) \cdot (g^{-1} \cdot h^{-1})$, but composition of functions is associative, so that we can write

$$h \cdot h^{-1} = (f \cdot g) \cdot (g^{-1} \cdot h^{-1}) = f \cdot (g \cdot g^{-1}) \cdot f^{-1} = f \cdot I \cdot f^{-1}$$

where we used I to denote an identity function. But identity composed with any function is that function, so that we can further write

$$h \cdot h^{-1} = f \cdot I \cdot f^{-1} = f \cdot f^{-1} = I$$

as desired. We now verify that $h^{-1} \cdot h$ is also an identity:

$$h^{-1} \cdot h = (g^{-1} \cdot f^{-1}) \cdot (f \cdot g) = g^{-1} \cdot (f^{-1} \cdot f) \cdot g = g^{-1} \cdot g = I$$

as desired.

Exercise 2.34

Another way to phrase this exercise is to say that any undirected graph with at least two vertices has at least a pair of vertices of the same degree—because we can represent people with vertices and the relationship "i knows j" by an edge $\{i, j\}$.

A graph on n vertices has a choice of degrees for its vertices from a minimum of 0 to a maximum of $n - 1$; thus, if no two vertices have the same degree, then the graph must have exactly one vertex of each degree. But that is impossible: since one vertex has degree $n - 1$, it is connected to all other vertices, so that there cannot be a vertex of degree 0. Hence there must be at least one pair of vertices with the same degree.

Exercise 2.35

From a purely mathematical point, the answer is quite simple: as we dovetail, we check whether the fraction represented by the two indices can be reduced. If it cannot, we are encountering this rational number for the first time and so do enumerate it; if it can, however, the rational number it represents has been seen before (because of the monotonicity of the pairing function used in the dovetailing process) and so we simply skip to the next fraction. The enumeration algorithm is clear and remains quite efficient, since a check for common factors is not too expensive; however, the mathematical expressions we derived for the pairing and unpairing are now invalid and cannot easily be fixed: the new pairing function defined implicitly by our procedure does not have a simple closed form, nor do its inverses.

Exercise 2.36

This question has two parts, the second not well defined. We shall use only positive rationals—adding the negative rationals does not really change the problem. It is rather easy to establish a bijection between $\mathbb{Q} \times \mathbb{Q}$ and $\mathbb{Q}$, at least if we view $\mathbb{Q}$ as formal fractions (i.e., if we consider two fractions of the same value, but with different numerators and denominators, to be distinct). For instance, we can simply pair up the the two numerators and pair up the two denominators, considering the two numbers thus produced to be the numerator and denominator of the new fraction. This mapping can clearly be inverted using the projection functions corresponding to the pairings used, so we have a bijection.

The interesting part comes when considering the requirement of monotonicity. Obviously, we can no longer define monotonicity in quite the same way, but we could ask that an increase in either argument cause an increase in the result. The construction used above, i.e.,

$$f : (\frac{a}{b}, \frac{c}{d}) \to \frac{\langle a, c\rangle}{\langle b, d\rangle}$$

may no longer be monotonic in each argument: it is possible to increase the fraction $\frac{a}{b}$ by increasing *both* a and b, in which case we do not really know how much each of $\langle a, c\rangle$ and $\langle b, d\rangle$ increases, so we cannot say much about the behavior of the resulting fraction.

On the other hand, note that "increasing in value" may not be the right criterion when we consider fractions to be formal symbols. In that case, however, what is a valid definition of monotonicity? We need an ordering on $\mathbb{Q}$ to define it; with an ordering, the definition can be the same as for $\mathbb{N}$: as we move from one element of $\mathbb{Q}$ to the next in the argument pair, the result of the bijection must itself move from an element of $\mathbb{Q}$ to one farther down in the ordering. Any pairing from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ gives us a possible ordering for $\mathbb{Q}$. Once we have this ordering, think of replacing each fraction in $\mathbb{Q}$ by its index in the ordering: thus fraction $\frac{a}{b}$ has some index i and fraction $\frac{c}{d}$ has some index j; a pairing function applied to i and j will give us the new natural number $k = \langle i, j\rangle$, which is larger than either i or j; but k also corresponds to a fraction and, by definition, to a fraction that is farther in the ordering than either $\frac{a}{b}$ or $\frac{c}{d}$. If we now replace $\frac{a}{b}$ by $\frac{a'}{b'}$ which is farther in the ordering—say it has index $i + n$ for some n—, then our bijection now produces the fraction corresponding to the index $\langle i + n, j\rangle$, which is larger than k. Thus increasing the first argument in the bijection increases the result. The same is obviously true of the second argument. Thus we have a bijection that is also monotonic in each argument—a pairing function between $\mathbb{Q} \times \mathbb{Q}$ and $\mathbb{Q}$.

Exercise 2.37

Of the three, surely the easiest to reverse is the second: factoring into a power of two and an odd factor is simply shifting down until all trailing zeros have disappeared, something that can be done very efficiently on most arithmetic and logic units. The other two require the computation of a square root for reversal and are thus rather more expensive. The pairing part is a simple second-degree polynomial for our first function and our third, both easily and efficiently computed; our third function requires exponentiation, but only as a power of two, again just a shift (this time up) and thus something easily and efficiently done. Thus, of the three, the second is likely to be the fastest, while the first and third remain quite reasonable.

A formal analysis of the running time is difficult, because it depends very much on the detailed model of computation—in particular, on the cost of arithmetic and logic operations, on what is representable in one word, and on what operations are supplied by the ALU.

Exercise 2.38

Actually, it is fairly easy to design new pairing functions: all we need to do is to picture sweeping the upper quadrant in some systematic way. The interesting parts come in making the function monotone in both arguments and in devising a way to compute it and invert it.

Here is a fairly simple example, which is really a variation on the basic dovetailing function defined in the text. Imagine sweeping the upper quadrant by running right along a horizontal sweep up to a point, then running up along a vertical sweep at a coordinate just one larger than that reached in the horizontal sweep. Formally, we sweep in order the pairs

$$(i, 1), (i, 2), \ldots, (i, i), (i+1, 1), (i+1, 2), \ldots, (i+1, i),$$

then start again with the pairs

$$(i+1, 1), (i+1, 2), \ldots, (i+1, i+1), (i+2, 1), (i+2, 2), \ldots, \\ (i+2, i+1),$$

and so on, starting with the sequence

$$(0, 0), (1, 0), (0, 1), (1, 1), (2, 0), (2, 1), \ldots$$

It is clear that this sequence goes through all pairs; it is also clear that it is monotone in both arguments. Computing it is not too hard, nor is inverting it: the key is to note that the switching point from a horizontal sweep to a vertical one occurs on the diagonal and so after enumerating i^2 pairs. Thus we can write $\langle i, i\rangle = i^2$ and hence also $\langle i - k, i\rangle = i^2 - k$, for $0 \le k \le i$; finally, we can write $\langle i + 1, k\rangle = i^2 + k$ for $0 \le k \le i$. Inverting the pairing thus also relies on the switching point and starts by computing $n - \lfloor\sqrt{n}\rfloor^2$ and checking it against $\lfloor\sqrt{n}\rfloor$.

Another example is even more trivial. Given numbers i and j, write the new number (in binary) as $100\ldots0j$, with i 0s between the initial 1 and the copy of j, then subtract 2. This function is clearly monotone in both arguments; it is injective, because j does not have leading 0s (unless, that is, $j = 0$, in which case things clearly work); and it is surjective thanks to the subtraction of 2—the smallest number that can be written before subtraction is $\langle 0, 0\rangle = 10$. To decode, we start by adding 2, then distinguish two cases: if the result is of the form $100\ldots00$, then we have $j = 0$ and i is one less than the number of 0; otherwise, j starts with the first one past the first digit. Thus decoding is very easily and efficiently implemented with a single scan of the bitstring (although writing formulas for it is rather more painful!) The only problem with this function is that it is computationally very expensive in the first argument and very cheap in the second; a more balanced function might be preferable.

Exercise 2.39

Any bijection that is easy to compute and that can be decoded can be used for dovetailing. The requirement of monotonicity that we imposed on pairing functions is strictly a matter of convenience—it simplifies a number of arguments in Chapter 5—and simplicity—the dovetailing is more intuitive when it proceeds in a monotone manner.

Exercise 2.40

Diagonalization is normally used to prove that some set cannot be enumerated, but of course every finite set is trivially enumerable. Thus diagonalization cannot be used for that purpose.

Diagonalization, however, is a bit more general: it is really a process whereby one uses an enumeration (real or assumed) to construct a new object that cannot be in the enumeration. In that sense, there is no reason why it could not be used with a finite set to construct an object not in that finite set; however, there is little incentive to use it for such a purpose, since finite sets are generally simple enough that producing an element not in a finite set is an easy task that does not require such an elaborate tool.

Exercise 2.41

See Section A.3.4, p. 437.

Exercise 2.42

Let S and T be two countable sets; that is, we have a bijection $f : \mathbb{N} \longleftrightarrow S$ and another $g : \mathbb{N} \longleftrightarrow T$.

Union The union $S \cup T$ can be enumerated as follows: define $h : S \cup T \longrightarrow \mathbb{N}$ by

$$h(z) = \begin{cases} 2f(z) & z \in S \\ 2g(z) + 1 & \text{otherwise} \end{cases}$$

It is clear that h is injective: both f and g are injective and we use f and g on entirely disjoint sets to produce entirely disjoint sets of values. Thus the cardinality of $S \cup T$ cannot exceed that of $\mathbb{N}$ and thus $S \cup T$ is countable. (Producing a bijection is rather harder, unless, of course, S and T are disjoint; but an injection suffices to prove the point.)

Intersection The intersection $S \cap T$ can be enumerated as follows: define $h : S \cap T \longrightarrow \mathbb{N}$ by $h(z) = f(z)$. Since f is injective, so is h and we are done. (Again, defining a bijection—when $S \cap T$ is not finite, that is—is harder.)

Cartesian Product The Cartesian product $S \times T$ can be enumerated as follows: define $h : S \times T \longrightarrow \mathbb{N}$ by dovetailing on $S \cup T$ exactly as we did for the rational numbers, using $f(x)$ and $g(y)$ (for $x \in S$ and $y \in T$) as indices:

$$h(x, y) = \frac{1}{2}(f^2(x) + g^2(y) + 2f(x)g(y) - f(x) - 3g(y) + 2)$$

Because both f and g are bijections and because our dovetailing process induces a bijection, h is also a bijection.

Exercise 2.43

Yes, such a set is countable. In fact, the rational numbers are such a set: think of the finite set as having two elements, call them n and d; then each function f from that set to $\mathbb{N}$ (minus the 0) defines the positive rational number $f(n)/f(d)$. The set of functions from S to T with S finite and T countable is, in effect, the set $T \times T \times \ldots \times T$, with $|S|$ copies of T in the Cartesian product; by dovetailing along $|S| - 1$ "dimensions" we will correctly enumerate that set.

Exercise 2.44

The hint pretty much solves the problem. Basically, we dovetail on n, the degree of the polynomial, and on $n + 1$ copies of the natural numbers to assign the $n + 1$ coefficients.

Formally, we proceed by induction. For $n = 0$, the set of polynomials of degree 0 is just $\mathbb{Z}$, which is known to be countable. Assume that the set of all polynomials of degree not exceeding $n - 1$ is countable and consider the set of all polynomials of degree not exceeding n. This set consists of all polynomials of degree not exceeding $n - 1$, a countable set by inductive hypothesis, and of all polynomials of degree n. Thus, by Exercise 2.42, we need only show that the set of polynomials of degree n is countable, since the union of two countable sets is itself countable. Any polynomial of degree n can be written as $a_n x^n + p_{n-1}(x)$, where $p_{n-1}(x)$ is a polynomial of degree not exceeding $n - 1$. by inductive hypothesis, we have a procedure to enumerate the latter type of polynomial; the first term is simply a matter of choosing $a_n \in \mathbb{Z}$. Thus we can dovetail on the value of a_n (all integers) and on all polynomials of degree not exceeding $n - 1$ and enumerate all polynomials of degree n.

Exercise 2.45

The argument used in Exercise 2.44 for polynomials in one variable can be used again (with a bit more dovetailing) to show that polynomials in two variables (or indeed in k variables for any fixed k) are countable. In fact polynomials in at most k variables (for any fixed k) are countable (just the sum of k countable sets). What, however, if we do not have a bound on k? It remains a countable set: we can dovetail once more, on all numbers of variables, to obtain the desired enumeration. This enumeration ends up dovetailing on all numbers of variables, on all degrees, and (for what is now a polynomial with a fixed number of terms) on all coefficient values for each coefficient.

Solution Manual
Chapter 2, Additional Problems

Additional Problem 1

1. True. We know that $\log n$ grows more slowly than $\sqrt{n}$; hence $n/\log n$ grows faster than $n/\sqrt{n} = \sqrt{n}$.
2. False. $3^{2\log_2 n} = 9^{\log_2 n} = 9^{\log_2 9 \log_9 n} = n^{\log_2 9}$ $\log_2 9$ is larger than 3, since we have $2^3 = 8 < 9$; hence $n^{\log_2 9}$ grows faster than n^3.
3. False. $3^{2n} = 9^n$, which grows faster than 6^n.

Additional Problem 2

1. If $(k+1)^n$ was $O(k^n)$, we would have constants c and N such that, for all $n > N$, $(k+1)^n \leq c \cdot k^n$; but, after dividing both sides by k^n (a positive, non-zero value), this becomes $((k+1)/k)^n \leq c$ or $c \geq (1 + \frac{1}{k})^n$, which cannot be true for all $n > N$, since the right-hand side of the inequality grows without bounds as n grows.
   The second statement can be rewritten, after simplification, as "$n \log(k+1)$ is $O(n \log k)$." That is pretty obviously true, since both terms are $\Theta(n)$—both are just n times a constant factor. Formally, we would write $n \log(k+1) \leq c \cdot n \log k$ for all $n \geq N$, yielding $c \geq \log(k+1)/\log k$, certainly an inequality that can be met for all values of n. So we can set $c = \log(k+1)/\log k$ and $N = 1$.
2. While it is fastest to solve this problem by using Sterling's approximation for factorials, it is most instructive to solve it without. Let us start with Sterling's approximation:

$$(\log n)! \approx \sqrt{2\pi \log n} \left(\frac{\log n}{e}\right)^{\log n}$$

   Using the approximation and setting up the inequality, we get, if the first function is to be $O(\,)$ of the second,

$$n^{\log\log\log n} \leq c \cdot \sqrt{2\pi \log n} \left(\frac{\log n}{e}\right)^{\log n}$$

   Taking logs on both sides, we get

$$\log\log\log n \log n \leq \log c + 0.5 \log 2\pi + 0.5 \log\log n + (\log\log n - \log e) \log n$$

or

$$\log c \geq \log\log\log n \log n - 0.5 \log 2\pi - 0.5 \log\log n - (\log\log n - \log e) \log n$$

$$= (-\log\log n + \log\log\log n + \log e) \log n - 0.5 \log\log n - 0.5 \log 2\pi$$

Asymptotically, the right-hand side is dominated by the $-\log\log n \log n$ term and so decreases without bound, so that the inequality is respected for all n beyond some N. We conclude that $n^{\log\log\log n}$ is $O\big((\log n)!\big)$. By reversing the inequality, we can also immediately verify that $n^{\log\log\log n}$ is not $\Omega\big((\log n)!\big)$.

Without Sterling's approximation, we can prove the $O(\,)$ relationship as follows. We want to show that there exist positive constants c and N such that, for all $n \geq N$, $n^{\log\log\log n} \leq c \cdot (\log n)!$. Taking logs on both sides, we get

$$\log\log\log n \log n \leq \log c + \sum_{i=1}^{\log n} \log i$$

Notice that the last $\frac{1}{2}\log n$ terms of the sum are all at least as large as $\log(\frac{1}{2}\log n)$, so that the sum is at least as large as $\frac{1}{2}\log n \log(\frac{1}{2}\log n)$. Conversely, none of the terms in the sum exceeds $\log\log n$, so that the sum does not exceed $\log n \log\log n$; therefore the sum is $\Theta(\log\log n \log n)$. Now, that means that the right-hand side of the inequality grows faster than the left-hand side, so that, for each choice of $c > 0$, the inequality is respected for all n beyond some suitable N_c. Again, inverting the inequality shows that there is no constant c that can work in the reverse direction.

Additional Problem 3

Additional Problem 4

The key is that every question "does i know j?" eliminates either i (if the answer is "yes") or j (if the answer is "no"). Thus asking $n-1$ questions will eliminate all but one candidate (there cannot be more than one star in any meeting, by definition—the two stars could not both know and not know each other); the choice of question is arbitrary, as long as it does not involve any of the eliminated persons. The following pseudocode does the job:

```
tester = 1; candidate = 2;
for i=1 to n-2 do
    if tester knows candidate
       then tester = candidate (* tester is eliminated *)
       (* else candidate is eliminated--nothing to do *)
    candidate = candidate+1
(* last test: which of the two is the candidate? *)
if candidate knows tester
    then candidate = tester
```

Once we have that one candidate, we need to ascertain that (s)he is a star; since we may not yet have asked any question in which that person appeared, we need to check that everyone else knows that person ($n - 1$ questions at most) and also that that person knows no one (another $n - 1$ questions at most).

Additional Problem 5

The key to the solution is to realize that what is needed here is to keep the undirected graph with as many edges as possible while making it acyclic; in other words, we want the graph to be acyclic and maximally connected, exactly our characterization of a tree (or forest, if the original graph is not connected) of Exercise 2.23. Since running a depth-first search on an undirected graph imposes a tree (or forest) on the graph, our algorithm is very simple: we place in the feedback edge set every edge encountered in the depth-first search that connects the current vertex to a vertex already visited (every non-tree edge). The resulting collection is a feedback edge set, because depth-first search imposes a tree (or forest), and is a smallest such set, because trees (forests) are maximally connected acyclic graphs.