9. a) Find a formula for the sum of the first n even positive integers.
b) Prove the formula that you conjectured in part (a).

10. a) Find a formula for

$$\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\cdots+\frac{1}{n(n+1)}$$

by examining the values of this expression for small values of n.
b) Prove the formula you conjectured in part (a).

11. a) Find a formula for

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^n}$$

by examining the values of this expression for small values of n.
b) Prove the formula you conjectured in part (a).

12. Prove that

$$\sum_{j=0}^{n}\left(-\frac{1}{2}\right)^j=\frac{2^{n+1}+(-1)^n}{3\cdot 2^n}$$

whenever n is a nonnegative integer.

13. Prove that $1^2-2^2+3^2-\cdots+(-1)^{n-1}n^2=(-1)^{n-1}n(n+1)/2$ whenever n is a positive integer.

14. Prove that for every positive integer n, $\sum_{k=1}^{n}k2^k=(n-1)2^{n+1}+2$.

15. Prove that for every positive integer n,

$$1\cdot 2+2\cdot 3+\cdots+n(n+1)=n(n+1)(n+2)/3.$$

16. Prove that for every positive integer n,

$$1\cdot 2\cdot 3+2\cdot 3\cdot 4+\cdots+n(n+1)(n+2)=n(n+1)(n+2)(n+3)/4.$$

17. Prove that $\sum_{j=1}^{n}j^4=n(n+1)(2n+1)(3n^2+3n-1)/30$ whenever n is a positive integer.

Use mathematical induction to prove the inequalities in Exercises 18–30.

18. Let $P(n)$ be the statement that $n!<n^n$, where n is an integer greater than 1.
a) What is the statement $P(2)$?
b) Show that $P(2)$ is true, completing the basis step of the proof.
c) What is the inductive hypothesis?
d) What do you need to prove in the inductive step?
e) Complete the inductive step.
f) Explain why these steps show that this inequality is true whenever n is an integer greater than 1.

19. Let $P(n)$ be the statement that

$$1+\frac{1}{4}+\frac{1}{9}+\cdots+\frac{1}{n^2}<2-\frac{1}{n},$$

where n is an integer greater than 1.
a) What is the statement $P(2)$?
b) Show that $P(2)$ is true, completing the basis step of the proof.
c) What is the inductive hypothesis?
d) What do you need to prove in the inductive step?
e) Complete the inductive step.
f) Explain why these steps show that this inequality is true whenever n is an integer greater than 1.

20. Prove that $3^n<n!$ if n is an integer greater than 6.

21. Prove that $2^n>n^2$ if n is an integer greater than 4.

22. For which nonnegative integers n is $n^2\le n!$? Prove your answer.

23. For which nonnegative integers n is $2n+3\le 2^n$? Prove your answer.

24. Prove that $1/(2n)\le[1\cdot 3\cdot 5\cdot\cdots\cdot(2n-1)]/(2\cdot 4\cdot\cdots\cdot 2n)$ whenever n is a positive integer.

***25.** Prove that if $h>-1$, then $1+nh\le(1+h)^n$ for all nonnegative integers n. This is called **Bernoulli's inequality**.

***26.** Suppose that a and b are real numbers with $0<b<a$. Prove that if n is a positive integer, then $a^n-b^n\le na^{n-1}(a-b)$.

***27.** Prove that for every positive integer n,

$$1+\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{3}}+\cdots+\frac{1}{\sqrt{n}}>2(\sqrt{n+1}-1).$$

28. Prove that $n^2-7n+12$ is nonnegative whenever n is an integer with $n\ge 3$.

In Exercises 29 and 30, H_n denotes the nth harmonic number.

***29.** Prove that $H_{2^n}\le 1+n$ whenever n is a nonnegative integer.

***30.** Prove that

$$H_1+H_2+\cdots+H_n=(n+1)H_n-n.$$

Use mathematical induction in Exercises 31–37 to prove divisibility facts.

31. Prove that 2 divides n^2+n whenever n is a positive integer.

32. Prove that 3 divides n^3+2n whenever n is a positive integer.

33. Prove that 5 divides n^5-n whenever n is a nonnegative integer.

34. Prove that 6 divides n^3-n whenever n is a nonnegative integer.

***35.** Prove that n^2-1 is divisible by 8 whenever n is an odd positive integer.

***36.** Prove that 21 divides $4^{n+1}+5^{2n-1}$ whenever n is a positive integer.

***37.** Prove that if n is a positive integer, then 133 divides $11^{n+1}+12^{2n-1}$.

Use mathematical induction in Exercises 38–46 to prove results about sets.

38. Prove that if $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_n$ are sets such that $A_j\subseteq B_j$ for $j=1,2,\ldots,n$, then

$$\bigcup_{j=1}^{n}A_j\subseteq\bigcup_{j=1}^{n}B_j.$$

39. Prove that if $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_n$ are sets such that $A_j \subseteq B_j$ for $j = 1, 2, \ldots, n$, then

$$\bigcap_{j=1}^{n} A_j \subseteq \bigcap_{j=1}^{n} B_j.$$

40. Prove that if $A_1, A_2, \ldots, A_n$ and B are sets, then

$$(A_1 \cap A_2 \cap \cdots \cap A_n) \cup B = (A_1 \cup B) \cap (A_2 \cup B) \cap \cdots \cap (A_n \cup B).$$

41. Prove that if $A_1, A_2, \ldots, A_n$ and B are sets, then

$$(A_1 \cup A_2 \cup \cdots \cup A_n) \cap B = (A_1 \cap B) \cup (A_2 \cap B) \cup \cdots \cup (A_n \cap B).$$

42. Prove that if $A_1, A_2, \ldots, A_n$ and B are sets, then

$$(A_1 - B) \cap (A_2 - B) \cap \cdots \cap (A_n - B) = (A_1 \cap A_2 \cap \cdots \cap A_n) - B.$$

43. Prove that if $A_1, A_2, \ldots, A_n$ are subsets of a universal set U, then

$$\overline{\bigcup_{k=1}^{n} A_k} = \bigcap_{k=1}^{n} \overline{A_k}.$$

44. Prove that if $A_1, A_2, \ldots, A_n$ and B are sets, then

$$(A_1 - B) \cup (A_2 - B) \cup \cdots \cup (A_n - B) = (A_1 \cup A_2 \cup \cdots \cup A_n) - B.$$

45. Prove that a set with n elements has $n(n-1)/2$ subsets containing exactly two elements whenever n is an integer greater than or equal to 2.

***46.** Prove that a set with n elements has $n(n-1)(n-2)/6$ subsets containing exactly three elements whenever n is an integer greater than or equal to 3.

In Exercises 47 and 48 we consider the problem of placing towers along a straight road, so that every building on the road receives cellular service. Assume that a building receives cellular service if it is within one mile of a tower.

47. Devise a greedy algorithm that uses the minimum number of towers possible to provide cell service to d buildings located at positions $x_1, x_2, \ldots, x_d$ from the start of the road. [*Hint:* At each step, go as far as possible along the road before adding a tower so as not to leave any buildings without coverage.]

***48.** Use mathematical induction to prove that the algorithm you devised in Exercise 47 produces an optimal solution, that is, that it uses the fewest towers possible to provide cellular service to all buildings.

Exercises 49–51 present incorrect proofs using mathematical induction. You will need to identify an error in reasoning in each exercise.

49. What is wrong with this "proof" that all horses are the same color?

Let $P(n)$ be the proposition that all the horses in a set of n horses are the same color.

Basis Step: Clearly, $P(1)$ is true.

Inductive Step: Assume that $P(k)$ is true, so that all the horses in any set of k horses are the same color. Consider any $k+1$ horses; number these as horses $1, 2, 3, \ldots, k, k+1$. Now the first k of these horses all must have the same color, and the last k of these must also have the same color. Because the set of the first k horses and the set of the last k horses overlap, all $k+1$ must be the same color. This shows that $P(k+1)$ is true and finishes the proof by induction.

50. What is wrong with this "proof"?
"Theorem" For every positive integer n, $\sum_{i=1}^{n} i = (n + \frac{1}{2})^2/2$.

Basis Step: The formula is true for $n = 1$.

Inductive Step: Suppose that $\sum_{i=1}^{n} i = (n + \frac{1}{2})^2/2$. Then $\sum_{i=1}^{n+1} i = (\sum_{i=1}^{n} i) + (n+1)$. By the inductive hypothesis, $\sum_{i=1}^{n+1} i = (n + \frac{1}{2})^2/2 + n + 1 = (n^2 + n + \frac{1}{4})/2 + n + 1 = (n^2 + 3n + \frac{9}{4})/2 = (n + \frac{3}{2})^2/2 = [(n+1) + \frac{1}{2}]^2/2$, completing the inductive step.

51. What is wrong with this "proof"?
"Theorem" For every positive integer n, if x and y are positive integers with $\max(x, y) = n$, then $x = y$.

Basis Step: Suppose that $n = 1$. If $\max(x, y) = 1$ and x and y are positive integers, we have $x = 1$ and $y = 1$.

Inductive Step: Let k be a positive integer. Assume that whenever $\max(x, y) = k$ and x and y are positive integers, then $x = y$. Now let $\max(x, y) = k + 1$, where x and y are positive integers. Then $\max(x - 1, y - 1) = k$, so by the inductive hypothesis, $x - 1 = y - 1$. It follows that $x = y$, completing the inductive step.

52. Suppose that m and n are positive integers with $m > n$ and f is a function from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, n\}$. Use mathematical induction on the variable n to show that f is not one-to-one.

***53.** Use mathematical induction to show that n people can divide a cake (where each person gets one or more separate pieces of the cake) so that the cake is divided fairly, that is, in the sense that each person thinks he or she got at least $(1/n)$th of the cake. [*Hint:* For the inductive step, take a fair division of the cake among the first k people, have each person divide their share into what this person thinks are $k+1$ equal portions, and then have the $(k+1)$st person select a portion from each of the k people. When showing this produces a fair division for $k+1$ people, suppose that person $k+1$ thinks that person i got p_i of the cake where $\sum_{i=1}^{k} p_i = 1$.]

54. Use mathematical induction to show that given a set of $n+1$ positive integers, none exceeding $2n$, there is at least one integer in this set that divides another integer in the set.

***55.** A knight on a chessboard can move one space horizontally (in either direction) and two spaces vertically (in either direction) or two spaces horizontally (in either direction) and one space vertically (in either direction). Suppose that we have an infinite chessboard, made up

of all squares (m, n) where m and n are nonnegative integers that denote the row number and the column number of the square, respectively. Use mathematical induction to show that a knight starting at $(0, 0)$ can visit every square using a finite sequence of moves. [*Hint:* Use induction on the variable $s = m + n$.]

56. Suppose that

$$\mathbf{A} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix},$$

where a and b are real numbers. Show that

$$\mathbf{A}^n = \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix}$$

for every positive integer n.

57. (*Requires calculus*) Use mathematical induction to prove that the derivative of $f(x) = x^n$ equals nx^{n-1} whenever n is a positive integer. (For the inductive step, use the product rule for derivatives.)

58. Suppose that $\mathbf{A}$ and $\mathbf{B}$ are square matrices with the property $\mathbf{AB} = \mathbf{BA}$. Show that $\mathbf{AB}^n = \mathbf{B}^n\mathbf{A}$ for every positive integer n.

59. Suppose that m is a positive integer. Use mathematical induction to prove that if a and b are integers with $a \equiv b \pmod{m}$, then $a^k \equiv b^k \pmod{m}$ whenever k is a nonnegative integer.

60. Use mathematical induction to show that $\neg(p_1 \vee p_2 \vee \cdots \vee p_n)$ is equivalent to $\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_n$ whenever $p_1, p_2, \ldots, p_n$ are propositions.

***61.** Show that

$$[(p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \wedge \cdots \wedge (p_{n-1} \rightarrow p_n)] \rightarrow [(p_1 \wedge p_2 \wedge \cdots \wedge p_{n-1}) \rightarrow p_n]$$

is a tautology whenever $p_1, p_2, \ldots, p_n$ are propositions, where $n \geq 2$.

***62.** Show that n lines separate the plane into $(n^2 + n + 2)/2$ regions if no two of these lines are parallel and no three pass through a common point.

****63.** Let $a_1, a_2, \ldots, a_n$ be positive real numbers. The **arithmetic mean** of these numbers is defined by

$$A = (a_1 + a_2 + \cdots + a_n)/n,$$

and the **geometric mean** of these numbers is defined by

$$G = (a_1 a_2 \cdots a_n)^{1/n}.$$

Use mathematical induction to prove that $A \geq G$.

64. Use mathematical induction to prove Lemma 3 of Section 4.3, which states that if p is a prime and $p \mid a_1 a_2 \cdots a_n$, where a_i is an integer for $i = 1, 2, 3, \ldots, n$, then $p \mid a_i$ for some integer i.

65. Show that if n is a positive integer, then

$$\sum_{\{a_1,\ldots,a_k\} \subseteq \{1,2,\ldots,n\}} \frac{1}{a_1 a_2 \cdots a_k} = n.$$

(Here the sum is over all nonempty subsets of the set of the n smallest positive integers.)

***66.** Use the well-ordering property to show that the following form of mathematical induction is a valid method to prove that $P(n)$ is true for all positive integers n.

Basis Step: $P(1)$ and $P(2)$ are true.

Inductive Step: For each positive integer k, if $P(k)$ and $P(k + 1)$ are both true, then $P(k + 2)$ is true.

67. Show that if $A_1, A_2, \ldots, A_n$ are sets where $n \geq 2$, and for all pairs of integers i and j with $1 \leq i < j \leq n$ either A_i is a subset of A_j or A_j is a subset of A_i, then there is an integer i, $1 \leq i \leq n$ such that A_i is a subset of A_j for all integers j with $1 \leq j \leq n$.

***68.** A guest at a party is a **celebrity** if this person is known by every other guest, but knows none of them. There is at most one celebrity at a party, for if there were two, they would know each other. A particular party may have no celebrity. Your assignment is to find the celebrity, if one exists, at a party, by asking only one type of question—asking a guest whether they know a second guest. Everyone must answer your questions truthfully. That is, if Alice and Bob are two people at the party, you can ask Alice whether she knows Bob; she must answer correctly. Use mathematical induction to show that if there are n people at the party, then you can find the celebrity, if there is one, with $3(n - 1)$ questions. [*Hint:* First ask a question to eliminate one person as a celebrity. Then use the inductive hypothesis to identify a potential celebrity. Finally, ask two more questions to determine whether that person is actually a celebrity.]

Suppose there are n people in a group, each aware of a scandal no one else in the group knows about. These people communicate by telephone; when two people in the group talk, they share information about all scandals each knows about. For example, on the first call, two people share information, so by the end of the call, each of these people knows about two scandals. The **gossip problem** asks for $G(n)$, the minimum number of telephone calls that are needed for all n people to learn about all the scandals. Exercises 69–71 deal with the gossip problem.

69. Find $G(1)$, $G(2)$, $G(3)$, and $G(4)$.

70. Use mathematical induction to prove that $G(n) \leq 2n - 4$ for $n \geq 4$. [*Hint:* In the inductive step, have a new person call a particular person at the start and at the end.]

****71.** Prove that $G(n) = 2n - 4$ for $n \geq 4$.

***72.** Show that it is possible to arrange the numbers $1, 2, \ldots, n$ in a row so that the average of any two of these numbers never appears between them. [*Hint:* Show that it suffices to prove this fact when n is a power of 2. Then use mathematical induction to prove the result when n is a power of 2.]

***73.** Show that if $I_1, I_2, \ldots, I_n$ is a collection of open intervals on the real number line, $n \geq 2$, and every pair of these intervals has a nonempty intersection, that is, $I_i \cap I_j \neq \emptyset$ whenever $1 \leq i \leq n$ and $1 \leq j \leq n$, then the intersection of all these sets is nonempty, that is, $I_1 \cap I_2 \cap \cdots \cap I_n \neq \emptyset$. (Recall that an **open interval** is

the set of real numbers x with $a < x < b$, where a and b are real numbers with $a < b$.)

Sometimes we cannot use mathematical induction to prove a result we believe to be true, but we can use mathematical induction to prove a stronger result. Because the inductive hypothesis of the stronger result provides more to work with, this process is called **inductive loading**. We use inductive loading in Exercise 74.

74. Suppose that we want to prove that

$$\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2n-1}{2n} < \frac{1}{\sqrt{3n}}$$

for all positive integers n.

a) Show that if we try to prove this inequality using mathematical induction, the basis step works, but the inductive step fails.

b) Show that mathematical induction can be used to prove the stronger inequality

$$\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2n-1}{2n} < \frac{1}{\sqrt{3n+1}}$$

for all integers greater than 1, which, together with a verification for the case where $n = 1$, establishes the weaker inequality we originally tried to prove using mathematical induction.

75. Let n be an even positive integer. Show that when n people stand in a yard at mutually distinct distances and each person throws a pie at their nearest neighbor, it is possible that everyone is hit by a pie.

76. Construct a tiling using right triominoes of the 4×4 checkerboard with the square in the upper left corner removed.

77. Construct a tiling using right triominoes of the 8×8 checkerboard with the square in the upper left corner removed.

78. Prove or disprove that all checkerboards of these shapes can be completely covered using right triominoes whenever n is a positive integer.

a) 3×2^n **b)** 6×2^n
c) $3^n \times 3^n$ **d)** $6^n \times 6^n$

***79.** Show that a three-dimensional $2^n \times 2^n \times 2^n$ checkerboard with one $1 \times 1 \times 1$ cube missing can be completely covered by $2 \times 2 \times 2$ cubes with one $1 \times 1 \times 1$ cube removed.

***80.** Show that an $n \times n$ checkerboard with one square removed can be completely covered using right triominoes if $n > 5$, n is odd, and $3 \nmid n$.

81. Show that a 5×5 checkerboard with a corner square removed can be tiled using right triominoes.

***82.** Find a 5×5 checkerboard with a square removed that cannot be tiled using right triominoes. Prove that such a tiling does not exist for this board.

83. Use the principle of mathematical induction to show that $P(n)$ is true for $n = b, b+1, b+2, \ldots$, where b is an integer, if $P(b)$ is true and the conditional statement $P(k) \rightarrow P(k+1)$ is true for all integers k with $k \geq b$.

5.2 Strong Induction and Well-Ordering

Introduction

In Section 5.1 we introduced mathematical induction and we showed how to use it to prove a variety of theorems. In this section we will introduce another form of mathematical induction, called **strong induction**, which can often be used when we cannot easily prove a result using mathematical induction. The basis step of a proof by strong induction is the same as a proof of the same result using mathematical induction. That is, in a strong induction proof that $P(n)$ is true for all positive integers n, the basis step shows that $P(1)$ is true. However, the inductive steps in these two proof methods are different. In a proof by mathematical induction, the inductive step shows that if the inductive hypothesis $P(k)$ is true, then $P(k+1)$ is also true. In a proof by strong induction, the inductive step shows that if $P(j)$ is true for all positive integers not exceeding k, then $P(k+1)$ is true. That is, for the inductive hypothesis we assume that $P(j)$ is true for $j = 1, 2, \ldots, k$.

The validity of both mathematical induction and strong induction follow from the well-ordering property in Appendix 1. In fact, mathematical induction, strong induction, and well-ordering are all equivalent principles (as shown in Exercises 41, 42, and 43). That is, the validity of each can be proved from either of the other two. This means that a proof using one of these two principles can be rewritten as a proof using either of the other two principles. Just as it is sometimes the case that it is much easier to see how to prove a result using strong induction rather than mathematical induction, it is sometimes easier to use well-ordering than one of the

two forms of mathematical induction. In this section we will give some examples of how the well-ordering property can be used to prove theorems.

Strong Induction

Before we illustrate how to use strong induction, we state this principle again.

STRONG INDUCTION To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:

BASIS STEP: We verify that the proposition $P(1)$ is true.

INDUCTIVE STEP: We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

Note that when we use strong induction to prove that $P(n)$ is true for all positive integers n, our inductive hypothesis is the assumption that $P(j)$ is true for $j = 1, 2, \ldots, k$. That is, the inductive hypothesis includes all k statements $P(1)$, $P(2)$, $\ldots$, $P(k)$. Because we can use all k statements $P(1)$, $P(2)$, $\ldots$, $P(k)$ to prove $P(k+1)$, rather than just the statement $P(k)$ as in a proof by mathematical induction, strong induction is a more flexible proof technique. Because of this, some mathematicians prefer to always use strong induction instead of mathematical induction, even when a proof by mathematical induction is easy to find.

You may be surprised that mathematical induction and strong induction are equivalent. That is, each can be shown to be a valid proof technique assuming that the other is valid. In particular, any proof using mathematical induction can also be considered to be a proof by strong induction because the inductive hypothesis of a proof by mathematical induction is part of the inductive hypothesis in a proof by strong induction. That is, if we can complete the inductive step of a proof using mathematical induction by showing that $P(k+1)$ follows from $P(k)$ for every positive integer k, then it also follows that $P(k+1)$ follows from all the statements $P(1)$, $P(2)$, $\ldots$, $P(k)$, because we are assuming that not only $P(k)$ is true, but also more, namely, that the $k-1$ statements $P(1)$, $P(2)$, $\ldots$, $P(k-1)$ are true. However, it is much more awkward to convert a proof by strong induction into a proof using the principle of mathematical induction. (See Exercise 42.)

Strong induction is sometimes called the **second principle of mathematical induction** or **complete induction**. When the terminology "complete induction" is used, the principle of mathematical induction is called **incomplete induction**, a technical term that is a somewhat unfortunate choice because there is nothing incomplete about the principle of mathematical induction; after all, it is a valid proof technique.

STRONG INDUCTION AND THE INFINITE LADDER To better understand strong induction, consider the infinite ladder in Section 5.1. Strong induction tells us that we can reach all rungs if

1. we can reach the first rung, and
2. for every integer k, if we can reach all the first k rungs, then we can reach the $(k+1)$st rung.

That is, if $P(n)$ is the statement that we can reach the nth rung of the ladder, by strong induction we know that $P(n)$ is true for all positive integers n, because (1) tells us $P(1)$ is true, completing the basis step and (2) tells us that $P(1) \wedge P(2) \wedge \cdots \wedge P(k)$ implies $P(k+1)$, completing the inductive step.

Example 1 illustrates how strong induction can help us prove a result that cannot easily be proved using the principle of mathematical induction.

EXAMPLE 1 Suppose we can reach the first and second rungs of an infinite ladder, and we know that if we can reach a rung, then we can reach two rungs higher. Can we prove that we can reach every rung using the principle of mathematical induction? Can we prove that we can reach every rung using strong induction?

Solution: We first try to prove this result using the principle of mathematical induction.

BASIS STEP: The basis step of such a proof holds; here it simply verifies that we can reach the first rung.

ATTEMPTED INDUCTIVE STEP: The inductive hypothesis is the statement that we can reach the kth rung of the ladder. To complete the inductive step, we need to show that if we assume the inductive hypothesis for the positive integer k, namely, if we assume that we can reach the kth rung of the ladder, then we can show that we can reach the $(k + 1)$st rung of the ladder. However, there is no obvious way to complete this inductive step because we do not know from the given information that we can reach the $(k + 1)$st rung from the kth rung. After all, we only know that if we can reach a rung we can reach the rung two higher.

Now consider a proof using strong induction.

BASIS STEP: The basis step is the same as before; it simply verifies that we can reach the first rung.

INDUCTIVE STEP: The inductive hypothesis states that we can reach each of the first k rungs. To complete the inductive step, we need to show that if we assume that the inductive hypothesis is true, that is, if we can reach each of the first k rungs, then we can reach the $(k + 1)$st rung. We already know that we can reach the second rung. We can complete the inductive step by noting that as long as $k \geq 2$, we can reach the $(k + 1)$st rung from the $(k - 1)$st rung because we know we can climb two rungs from a rung we can already reach, and because $k - 1 \leq k$, by the inductive hypothesis we can reach the $(k - 1)$st rung. This completes the inductive step and finishes the proof by strong induction.

We have proved that if we can reach the first two rungs of an infinite ladder and for every positive integer k if we can reach all the first k rungs then we can reach the $(k + 1)$st rung, then we can reach all rungs of the ladder. ◀

Examples of Proofs Using Strong Induction

Now that we have both mathematical induction and strong induction, how do we decide which method to apply in a particular situation? Although there is no cut-and-dried answer, we can supply some useful pointers. In practice, you should use mathematical induction when it is straightforward to prove that $P(k) \rightarrow P(k + 1)$ is true for all positive integers k. This is the case for all the proofs in the examples in Section 5.1. In general, you should restrict your use of the principle of mathematical induction to such scenarios. Unless you can clearly see that the inductive step of a proof by mathematical induction goes through, you should attempt a proof by strong induction. That is, use strong induction and not mathematical induction when you see how to prove that $P(k + 1)$ is true from the assumption that $P(j)$ is true for all positive integers j not exceeding k, but you cannot see how to prove that $P(k + 1)$ follows from just $P(k)$. Keep this in mind as you examine the proofs in this section. For each of these proofs, consider why strong induction works better than mathematical induction.

We will illustrate how strong induction is employed in Examples 2–4. In these examples, we will prove a diverse collection of results. Pay particular attention to the inductive step in each of these examples, where we show that a result $P(k + 1)$ follows under the assumption that $P(j)$ holds for all positive integers j not exceeding k, where $P(n)$ is a propositional function.

We begin with one of the most prominent uses of strong induction, the part of the fundamental theorem of arithmetic that tells us that every positive integer can be written as the product of primes.

EXAMPLE 2 Show that if n is an integer greater than 1, then n can be written as the product of primes.

Extra Examples

Solution: Let $P(n)$ be the proposition that n can be written as the product of primes.

BASIS STEP: $P(2)$ is true, because 2 can be written as the product of one prime, itself. (Note that $P(2)$ is the first case we need to establish.)

INDUCTIVE STEP: The inductive hypothesis is the assumption that $P(j)$ is true for all integers j with $2 \leq j \leq k$, that is, the assumption that j can be written as the product of primes whenever j is a positive integer at least 2 and not exceeding k. To complete the inductive step, it must be shown that $P(k+1)$ is true under this assumption, that is, that $k+1$ is the product of primes.

There are two cases to consider, namely, when $k+1$ is prime and when $k+1$ is composite. If $k+1$ is prime, we immediately see that $P(k+1)$ is true. Otherwise, $k+1$ is composite and can be written as the product of two positive integers a and b with $2 \leq a \leq b < k+1$. Because both a and b are integers at least 2 and not exceeding k, we can use the inductive hypothesis to write both a and b as the product of primes. Thus, if $k+1$ is composite, it can be written as the product of primes, namely, those primes in the factorization of a and those in the factorization of b. ◀

Remark: Because 1 can be thought of as the *empty* product of no primes, we could have started the proof in Example 2 with $P(1)$ as the basis step. We chose not to do so because many people find this confusing.

Example 2 completes the proof of the fundamental theorem of arithmetic, which asserts that every nonnegative integer can be written uniquely as the product of primes in nondecreasing order. We showed in Section 4.3 that an integer has at most one such factorization into primes. Example 2 shows there is at least one such factorization.

Next, we show how strong induction can be used to prove that a player has a winning strategy in a game.

EXAMPLE 3 Consider a game in which two players take turns removing any positive number of matches they want from one of two piles of matches. The player who removes the last match wins the game. Show that if the two piles contain the same number of matches initially, the second player can always guarantee a win.

Solution: Let n be the number of matches in each pile. We will use strong induction to prove $P(n)$, the statement that the second player can win when there are initially n matches in each pile.

BASIS STEP: When $n = 1$, the first player has only one choice, removing one match from one of the piles, leaving a single pile with a single match, which the second player can remove to win the game.

INDUCTIVE STEP: The inductive hypothesis is the statement that $P(j)$ is true for all j with $1 \leq j \leq k$, that is, the assumption that the second player can always win whenever there are j matches, where $1 \leq j \leq k$ in each of the two piles at the start of the game. We need to show that $P(k+1)$ is true, that is, that the second player can win when there are initially $k+1$ matches in each pile, under the assumption that $P(j)$ is true for $j = 1, 2, \ldots, k$. So suppose that there are $k+1$ matches in each of the two piles at the start of the game and suppose that the first player removes r matches ($1 \leq r \leq k$) from one of the piles, leaving $k+1-r$ matches in this pile. By removing the same number of matches from the other pile, the second player creates the

situation where there are two piles each with $k + 1 - r$ matches. Because $1 \leq k + 1 - r \leq k$, we can now use the inductive hypothesis to conclude that the second player can always win. We complete the proof by noting that if the first player removes all $k + 1$ matches from one of the piles, the second player can win by removing all the remaining matches. ◀

Using the principle of mathematical induction, instead of strong induction, to prove the results in Examples 2 and 3 is difficult. However, as Example 4 shows, some results can be readily proved using either the principle of mathematical induction or strong induction.

Before we present Example 4, note that we can slightly modify strong induction to handle a wider variety of situations. In particular, we can adapt strong induction to handle cases where the inductive step is valid only for integers greater than a particular integer. Let b be a fixed integer and j a fixed positive integer. The form of strong induction we need tells us that $P(n)$ is true for all integers n with $n \geq b$ if we can complete these two steps:

BASIS STEP: We verify that the propositions $P(b)$, $P(b + 1)$, ..., $P(b + j)$ are true.

INDUCTIVE STEP: We show that $[P(b) \wedge P(b + 1) \wedge \cdots \wedge P(k)] \rightarrow P(k + 1)$ is true for every integer $k \geq b + j$.

We will use this alternative form in the strong induction proof in Example 4. That this alternative form is equivalent to strong induction is left as Exercise 28.

EXAMPLE 4 Prove that every amount of postage of 12 cents or more can be formed using just 4-cent and 5-cent stamps.

Solution: We will prove this result using the principle of mathematical induction. Then we will present a proof using strong induction. Let $P(n)$ be the statement that postage of n cents can be formed using 4-cent and 5-cent stamps.

We begin by using the principle of mathematical induction.

BASIS STEP: Postage of 12 cents can be formed using three 4-cent stamps.

INDUCTIVE STEP: The inductive hypothesis is the statement that $P(k)$ is true. That is, under this hypothesis, postage of k cents can be formed using 4-cent and 5-cent stamps. To complete the inductive step, we need to show that when we assume $P(k)$ is true, then $P(k + 1)$ is also true where $k \geq 12$. That is, we need to show that if we can form postage of k cents, then we can form postage of $k + 1$ cents. So, assume the inductive hypothesis is true; that is, assume that we can form postage of k cents using 4-cent and 5-cent stamps. We consider two cases, when at least one 4-cent stamp has been used and when no 4-cent stamps have been used. First, suppose that at least one 4-cent stamp was used to form postage of k cents. Then we can replace this stamp with a 5-cent stamp to form postage of $k + 1$ cents. But if no 4-cent stamps were used, we can form postage of k cents using only 5-cent stamps. Moreover, because $k \geq 12$, we needed at least three 5-cent stamps to form postage of k cents. So, we can replace three 5-cent stamps with four 4-cent stamps to form postage of $k + 1$ cents. This completes the inductive step.

Because we have completed the basis step and the inductive step, we know that $P(n)$ is true for all $n \geq 12$. That is, we can form postage of n cents, where $n \geq 12$ using just 4-cent and 5-cent stamps. This completes the proof by mathematical induction.

Next, we will use strong induction to prove the same result. In this proof, in the basis step we show that $P(12)$, $P(13)$, $P(14)$, and $P(15)$ are true, that is, that postage of 12, 13, 14, or 15 cents can be formed using just 4-cent and 5-cent stamps. In the inductive step we show how to get postage of $k + 1$ cents for $k \geq 15$ from postage of $k - 3$ cents.

BASIS STEP: We can form postage of 12, 13, 14, and 15 cents using three 4-cent stamps, two 4-cent stamps and one 5-cent stamp, one 4-cent stamp and two 5-cent stamps, and three 5-cent stamps, respectively. This shows that $P(12)$, $P(13)$, $P(14)$, and $P(15)$ are true. This completes the basis step.

INDUCTIVE STEP: The inductive hypothesis is the statement that $P(j)$ is true for $12 \leq j \leq k$, where k is an integer with $k \geq 15$. To complete the inductive step, we assume that we can form postage of j cents, where $12 \leq j \leq k$. We need to show that under the assumption that $P(k+1)$ is true, we can also form postage of $k+1$ cents. Using the inductive hypothesis, we can assume that $P(k-3)$ is true because $k-3 \geq 12$, that is, we can form postage of $k-3$ cents using just 4-cent and 5-cent stamps. To form postage of $k+1$ cents, we need only add another 4-cent stamp to the stamps we used to form postage of $k-3$ cents. That is, we have shown that if the inductive hypothesis is true, then $P(k+1)$ is also true. This completes the inductive step.

Because we have completed the basis step and the inductive step of a strong induction proof, we know by strong induction that $P(n)$ is true for all integers n with $n \geq 12$. That is, we know that every postage of n cents, where n is at least 12, can be formed using 4-cent and 5-cent stamps. This finishes the proof by strong induction.

(There are other ways to approach this problem besides those described here. Can you find a solution that does not use mathematical induction?) ◀

Using Strong Induction in Computational Geometry

Our next example of strong induction will come from **computational geometry**, the part of discrete mathematics that studies computational problems involving geometric objects. Computational geometry is used extensively in computer graphics, computer games, robotics, scientific calculations, and a vast array of other areas. Before we can present this result, we introduce some terminology, possibly familiar from earlier studies in geometry.

A **polygon** is a closed geometric figure consisting of a sequence of line segments $s_1, s_2, \ldots, s_n$, called **sides**. Each pair of consecutive sides, s_i and s_{i+1}, $i = 1, 2, \ldots, n-1$, as well as the last side s_n and the first side s_1, of the polygon meet at a common endpoint, called a **vertex**. A polygon is called **simple** if no two nonconsecutive sides intersect. Every simple polygon divides the plane into two regions: its **interior**, consisting of the points inside the curve, and its **exterior**, consisting of the points outside the curve. This last fact is surprisingly complicated to prove. It is a special case of the famous Jordan curve theorem, which tells us that every simple curve divides the plane into two regions; see [Or00], for example.

A polygon is called **convex** if every line segment connecting two points in the interior of the polygon lies entirely inside the polygon. (A polygon that is not convex is said to be **nonconvex**.) Figure 1 displays some polygons; polygons (a) and (b) are convex, but polygons (c) and (d) are not. A **diagonal** of a simple polygon is a line segment connecting two nonconsecutive vertices of the polygon, and a diagonal is called an **interior diagonal** if it lies entirely inside the polygon, except for its endpoints. For example, in polygon (d), the line segment connecting a and f is an interior diagonal, but the line segment connecting a and d is a diagonal that is not an interior diagonal.

One of the most basic operations of computational geometry involves dividing a simple polygon into triangles by adding nonintersecting diagonals. This process is called **triangulation**. Note that a simple polygon can have many different triangulations, as shown in Figure 2. Perhaps the most basic fact in computational geometry is that it is possible to triangulate every simple

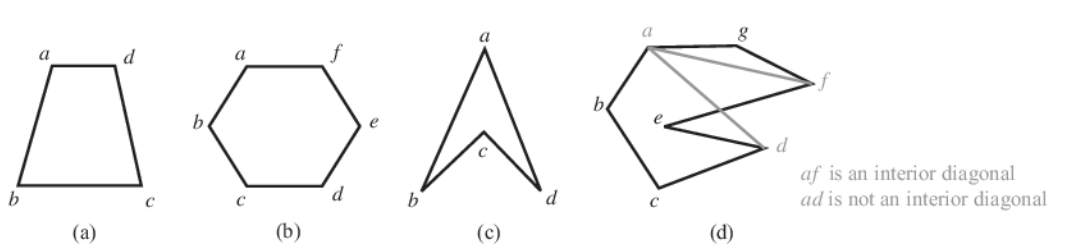
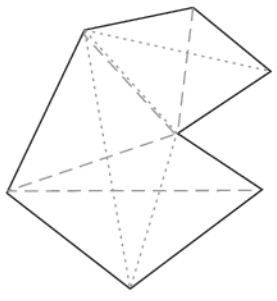


FIGURE 1 Convex and Nonconvex Polygons.



Two different triangulations of a simple polygon with seven sides into five triangles, shown with dotted lines and with dashed lines, respectively

FIGURE 2 Triangulations of a Polygon.

polygon, as we state in Theorem 1. Furthermore, this theorem tells us that every triangulation of a simple polygon with n sides includes $n - 2$ triangles.

THEOREM 1 A simple polygon with n sides, where n is an integer with $n \geq 3$, can be triangulated into $n - 2$ triangles.

It seems obvious that we should be able to triangulate a simple polygon by successively adding interior diagonals. Consequently, a proof by strong induction seems promising. However, such a proof requires this crucial lemma.

LEMMA 1 Every simple polygon with at least four sides has an interior diagonal.

Although Lemma 1 seems particularly simple, it is surprisingly tricky to prove. In fact, as recently as 30 years ago, a variety of incorrect proofs thought to be correct were commonly seen in books and articles. We defer the proof of Lemma 1 until after we prove Theorem 1. It is not uncommon to prove a theorem pending the later proof of an important lemma.

Proof (of Theorem 1): We will prove this result using strong induction. Let $T(n)$ be the statement that every simple polygon with n sides can be triangulated into $n - 2$ triangles.

BASIS STEP: $T(3)$ is true because a simple polygon with three sides is a triangle. We do not need to add any diagonals to triangulate a triangle; it is already triangulated into one triangle, itself. Consequently, every simple polygon with $n = 3$ has can be triangulated into $n - 2 = 3 - 2 = 1$ triangle.

INDUCTIVE STEP: For the inductive hypothesis, we assume that $T(j)$ is true for all integers j with $3 \leq j \leq k$. That is, we assume that we can triangulate a simple polygon with j sides into $j - 2$ triangles whenever $3 \leq j \leq k$. To complete the inductive step, we must show that when we assume the inductive hypothesis, $P(k + 1)$ is true, that is, that every simple polygon with $k + 1$ sides can be triangulated into $(k + 1) - 2 = k - 1$ triangles.

So, suppose that we have a simple polygon P with $k + 1$ sides. Because $k + 1 \geq 4$, Lemma 1 tells us that P has an interior diagonal ab. Now, ab splits P into two simple polygons Q, with s sides, and R, with t sides. The sides of Q and R are the sides of P, together with the side ab, which is a side of both Q and R. Note that $3 \leq s \leq k$ and $3 \leq t \leq k$ because both Q and R have at least one fewer side than P does (after all, each of these is formed from P by deleting at least two sides and replacing these sides by the diagonal ab). Furthermore, the number of sides of P is two less than the sum of the numbers of sides of Q and the number of

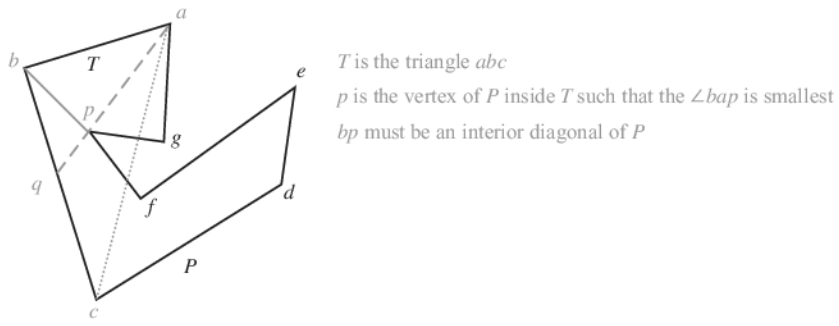


FIGURE 3 Constructing an Interior Diagonal of a Simple Polygon.

sides of R, because each side of P is a side of either Q or of R, but not both, and the diagonal ab is a side of both Q and R, but not P. That is, $k + 1 = s + t - 2$.

We now use the inductive hypothesis. Because both $3 \leq s \leq k$ and $3 \leq t \leq k$, by the inductive hypothesis we can triangulate Q and R into $s - 2$ and $t - 2$ triangles, respectively. Next, note that these triangulations together produce a triangulation of P. (Each diagonal added to triangulate one of these smaller polygons is also a diagonal of P.) Consequently, we can triangulate P into a total of $(s - 2) + (t - 2) = s + t - 4 = (k + 1) - 2$ triangles. This completes the proof by strong induction. That is, we have shown that every simple polygon with n sides, where $n \geq 3$, can be triangulated into $n - 2$ triangles. ◁

We now return to our proof of Lemma 1. We present a proof published by Chung-Wu Ho [Ho75]. Note that although this proof may be omitted without loss of continuity, it does provide a correct proof of a result proved incorrectly by many mathematicians.

Proof: Suppose that P is a simple polygon drawn in the plane. Furthermore, suppose that b is the point of P or in the interior of P with the least y-coordinate among the vertices with the smallest x-coordinate. Then b must be a vertex of P, for if it is an interior point, there would have to be a vertex of P with a smaller x-coordinate. Two other vertices each share an edge with b, say a and c. It follows that the angle in the interior of P formed by ab and bc must be less than 180 degrees (otherwise, there would be points of P with smaller x-coordinates than b).

Now let T be the triangle $\triangle abc$. If there are no vertices of P on or inside T, we can connect a and c to obtain an interior diagonal. On the other hand, if there are vertices of P inside T, we will find a vertex p of P on or inside T such that bp is an interior diagonal. (This is the tricky part. Ho noted that in many published proofs of this lemma a vertex p was found such that bp was not necessarily an interior diagonal of P. See Exercise 21.) The key is to select a vertex p such that the angle $\angle bap$ is smallest. To see this, note that the ray starting at a and passing through p hits the line segment bc at a point, say q. It then follows that the triangle $\triangle baq$ cannot contain any vertices of P in its interior. Hence, we can connect b and p to produce an interior diagonal of P. Locating this vertex p is illustrated in Figure 3. ◁

Proofs Using the Well-Ordering Property

The validity of both the principle of mathematical induction and strong induction follows from a fundamental axiom of the set of integers, the **well-ordering property** (see Appendix 1). The well-ordering property states that every nonempty set of nonnegative integers has a least element. We will show how the well-ordering property can be used directly in proofs. Furthermore, it can be shown (see Exercises 41, 42, and 43) that the well-ordering property, the principle of mathematical induction, and strong induction are all equivalent. That is, the validity of each of these three proof techniques implies the validity of the other two techniques. In Section 5.1 we

showed that the principle of mathematical induction follows from the well-ordering property. The other parts of this equivalence are left as Exercises 31, 42, and 43.

THE WELL-ORDERING PROPERTY Every nonempty set of nonnegative integers has a least element.

The well-ordering property can often be used directly in proofs.

EXAMPLE 5 Use the well-ordering property to prove the division algorithm. Recall that the division algorithm states that if a is an integer and d is a positive integer, then there are unique integers q and r with $0 \leq r < d$ and $a = dq + r$.

Extra Examples

Solution: Let S be the set of nonnegative integers of the form $a - dq$, where q is an integer. This set is nonempty because $-dq$ can be made as large as desired (taking q to be a negative integer with large absolute value). By the well-ordering property, S has a least element $r = a - dq_0$.

The integer r is nonnegative. It is also the case that $r < d$. If it were not, then there would be a smaller nonnegative element in S, namely, $a - d(q_0 + 1)$. To see this, suppose that $r \geq d$. Because $a = dq_0 + r$, it follows that $a - d(q_0 + 1) = (a - dq_0) - d = r - d \geq 0$. Consequently, there are integers q and r with $0 \leq r < d$. The proof that q and r are unique is left as Exercise 37. ◀

EXAMPLE 6 In a round-robin tournament every player plays every other player exactly once and each match has a winner and a loser. We say that the players $p_1, p_2, \ldots, p_m$ form a *cycle* if p_1 beats p_2, p_2 beats $p_3, \ldots, p_{m-1}$ beats p_m, and p_m beats p_1. Use the well-ordering principle to show that if there is a cycle of length m ($m \geq 3$) among the players in a round-robin tournament, there must be a cycle of three of these players.

Solution: We assume that there is no cycle of three players. Because there is at least one cycle in the round-robin tournament, the set of all positive integers n for which there is a cycle of length n is nonempty. By the well-ordering property, this set of positive integers has a least element k, which by assumption must be greater than three. Consequently, there exists a cycle of players $p_1, p_2, p_3, \ldots, p_k$ and no shorter cycle exists.

Because there is no cycle of three players, we know that $k > 3$. Consider the first three elements of this cycle, p_1, p_2, and p_3. There are two possible outcomes of the match between p_1 and p_3. If p_3 beats p_1, it follows that p_1, p_2, p_3 is a cycle of length three, contradicting our assumption that there is no cycle of three players. Consequently, it must be the case that p_1 beats p_3. This means that we can omit p_2 from the cycle $p_1, p_2, p_3, \ldots, p_k$ to obtain the cycle $p_1, p_3, p_4, \ldots, p_k$ of length $k - 1$, contradicting the assumption that the smallest cycle has length k. We conclude that there must be a cycle of length three. ◀

Exercises

1. Use strong induction to show that if you can run one mile or two miles, and if you can always run two more miles once you have run a specified number of miles, then you can run any number of miles.

2. Use strong induction to show that all dominoes fall in an infinite arrangement of dominoes if you know that the first three dominoes fall, and that when a domino falls, the domino three farther down in the arrangement also falls.

3. Let $P(n)$ be the statement that a postage of n cents can be formed using just 3-cent stamps and 5-cent stamps. The parts of this exercise outline a strong induction proof that $P(n)$ is true for $n \geq 8$.

a) Show that the statements $P(8)$, $P(9)$, and $P(10)$ are true, completing the basis step of the proof.

b) What is the inductive hypothesis of the proof?

c) What do you need to prove in the inductive step?

d) Complete the inductive step for $k \geq 10$.

e) Explain why these steps show that this statement is true whenever $n \geq 8$.

4. Let $P(n)$ be the statement that a postage of n cents can be formed using just 4-cent stamps and 7-cent stamps. The

parts of this exercise outline a strong induction proof that $P(n)$ is true for $n \geq 18$.

a) Show statements $P(18)$, $P(19)$, $P(20)$, and $P(21)$ are true, completing the basis step of the proof.

b) What is the inductive hypothesis of the proof?

c) What do you need to prove in the inductive step?

d) Complete the inductive step for $k \geq 21$.

e) Explain why these steps show that this statement is true whenever $n \geq 18$.

5. a) Determine which amounts of postage can be formed using just 4-cent and 11-cent stamps.

b) Prove your answer to (a) using the principle of mathematical induction. Be sure to state explicitly your inductive hypothesis in the inductive step.

c) Prove your answer to (a) using strong induction. How does the inductive hypothesis in this proof differ from that in the inductive hypothesis for a proof using mathematical induction?

6. a) Determine which amounts of postage can be formed using just 3-cent and 10-cent stamps.

b) Prove your answer to (a) using the principle of mathematical induction. Be sure to state explicitly your inductive hypothesis in the inductive step.

c) Prove your answer to (a) using strong induction. How does the inductive hypothesis in this proof differ from that in the inductive hypothesis for a proof using mathematical induction?

7. Which amounts of money can be formed using just two-dollar bills and five-dollar bills? Prove your answer using strong induction.

8. Suppose that a store offers gift certificates in denominations of 25 dollars and 40 dollars. Determine the possible total amounts you can form using these gift certificates. Prove your answer using strong induction.

***9.** Use strong induction to prove that $\sqrt{2}$ is irrational. [*Hint:* Let $P(n)$ be the statement that $\sqrt{2} \neq n/b$ for any positive integer b.]

10. Assume that a chocolate bar consists of n squares arranged in a rectangular pattern. The entire bar, a smaller rectangular piece of the bar, can be broken along a vertical or a horizontal line separating the squares. Assuming that only one piece can be broken at a time, determine how many breaks you must successively make to break the bar into n separate squares. Use strong induction to prove your answer.

11. Consider this variation of the game of Nim. The game begins with n matches. Two players take turns removing matches, one, two, or three at a time. The player removing the last match loses. Use strong induction to show that if each player plays the best strategy possible, the first player wins if $n = 4j$, $4j + 2$, or $4j + 3$ for some nonnegative integer j and the second player wins in the remaining case when $n = 4j + 1$ for some nonnegative integer j.

12. Use strong induction to show that every positive integer n can be written as a sum of distinct powers of two, that is, as a sum of a subset of the integers $2^0 = 1, 2^1 = 2, 2^2 = 4$, and so on. [*Hint:* For the inductive step, separately consider the case where $k + 1$ is even and where it is odd. When it is even, note that $(k + 1)/2$ is an integer.]

***13.** A jigsaw puzzle is put together by successively joining pieces that fit together into blocks. A move is made each time a piece is added to a block, or when two blocks are joined. Use strong induction to prove that no matter how the moves are carried out, exactly $n - 1$ moves are required to assemble a puzzle with n pieces.

14. Suppose you begin with a pile of n stones and split this pile into n piles of one stone each by successively splitting a pile of stones into two smaller piles. Each time you split a pile you multiply the number of stones in each of the two smaller piles you form, so that if these piles have r and s stones in them, respectively, you compute rs. Show that no matter how you split the piles, the sum of the products computed at each step equals $n(n - 1)/2$.

15. Prove that the first player has a winning strategy for the game of Chomp, introduced in Example 12 in Section 1.8, if the initial board is square. [*Hint:* Use strong induction to show that this strategy works. For the first move, the first player chomps all cookies except those in the left and top edges. On subsequent moves, after the second player has chomped cookies on either the top or left edge, the first player chomps cookies in the same relative positions in the left or top edge, respectively.]

***16.** Prove that the first player has a winning strategy for the game of Chomp, introduced in Example 12 in Section 1.8, if the initial board is two squares wide, that is, a $2 \times n$ board. [*Hint:* Use strong induction. The first move of the first player should be to chomp the cookie in the bottom row at the far right.]

17. Use strong induction to show that if a simple polygon with at least four sides is triangulated, then at least two of the triangles in the triangulation have two sides that border the exterior of the polygon.

***18.** Use strong induction to show that when a simple polygon P with consecutive vertices $v_1, v_2, \ldots, v_n$ is triangulated into $n - 2$ triangles, the $n - 2$ triangles can be numbered $1, 2, \ldots, n - 2$ so that v_i is a vertex of triangle i for $i = 1, 2, \ldots, n - 2$.

***19.** **Pick's theorem** says that the area of a simple polygon P in the plane with vertices that are all lattice points (that is, points with integer coordinates) equals $I(P) + B(P)/2 - 1$, where $I(P)$ and $B(P)$ are the number of lattice points in the interior of P and on the boundary of P, respectively. Use strong induction on the number of vertices of P to prove Pick's theorem. [*Hint:* For the basis step, first prove the theorem for rectangles, then for right triangles, and finally for all triangles by noting that the area of a triangle is the area of a larger rectangle containing it with the areas of at most three triangles subtracted. For the inductive step, take advantage of Lemma 1.]

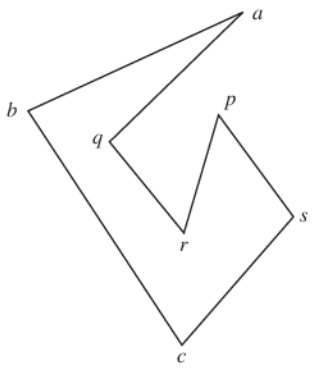
****20.** Suppose that P is a simple polygon with vertices $v_1, v_2, \ldots, v_n$ listed so that consecutive vertices are connected by an edge, and v_1 and v_n are connected by an edge. A vertex v_i is called an **ear** if the line segment connecting the two vertices adjacent to v_i is an interior diagonal of the simple polygon. Two ears v_i and v_j are called **nonoverlapping** if the interiors of the triangles with vertices v_i and its two adjacent vertices and v_j and its two adjacent vertices do not intersect. Prove that every simple polygon with at least four vertices has at least two nonoverlapping ears.

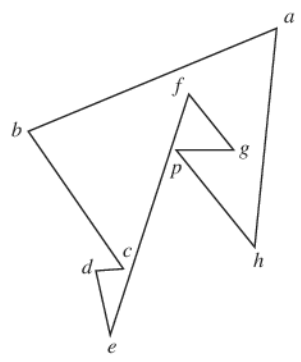
21. In the proof of Lemma 1 we mentioned that many incorrect methods for finding a vertex p such that the line segment bp is an interior diagonal of P have been published. This exercise presents some of the incorrect ways p has been chosen in these proofs. Show, by considering one of the polygons drawn here, that for each of these choices of p, the line segment bp is not necessarily an interior diagonal of P.

a) p is the vertex of P such that the angle $\angle abp$ is smallest.

b) p is the vertex of P with the least x-coordinate (other than b).

c) p is the vertex of P that is closest to b.





Exercises 22 and 23 present examples that show inductive loading can be used to prove results in computational geometry.

***22.** Let $P(n)$ be the statement that when nonintersecting diagonals are drawn inside a convex polygon with n sides, at least two vertices of the polygon are not endpoints of any of these diagonals.

a) Show that when we attempt to prove $P(n)$ for all integers n with $n \geq 3$ using strong induction, the inductive step does not go through.

b) Show that we can prove that $P(n)$ is true for all integers n with $n \geq 3$ by proving by strong induction the stronger assertion $Q(n)$, for $n \geq 4$, where $Q(n)$ states that whenever nonintersecting diagonals are drawn inside a convex polygon with n sides, at least two *nonadjacent* vertices are not endpoints of any of these diagonals.

23. Let $E(n)$ be the statement that in a triangulation of a simple polygon with n sides, at least one of the triangles in the triangulation has two sides bordering the exterior of the polygon.

a) Explain where a proof using strong induction that $E(n)$ is true for all integers $n \geq 4$ runs into difficulties.

b) Show that we can prove that $E(n)$ is true for all integers $n \geq 4$ by proving by strong induction the stronger statement $T(n)$ for all integers $n \geq 4$, which states that in every triangulation of a simple polygon, at least two of the triangles in the triangulation have two sides bordering the exterior of the polygon.

***24.** A stable assignment, defined in the preamble to Exercise 60 in Section 3.1, is called **optimal for suitors** if no stable assignment exists in which a suitor is paired with a suitee whom this suitor prefers to the person to whom this suitor is paired in this stable assignment. Use strong induction to show that the deferred acceptance algorithm produces a stable assignment that is optimal for suitors.

25. Suppose that $P(n)$ is a propositional function. Determine for which positive integers n the statement $P(n)$ must be true, and justify your answer, if

a) $P(1)$ is true; for all positive integers n, if $P(n)$ is true, then $P(n+2)$ is true.

b) $P(1)$ and $P(2)$ are true; for all positive integers n, if $P(n)$ and $P(n+1)$ are true, then $P(n+2)$ is true.

c) $P(1)$ is true; for all positive integers n, if $P(n)$ is true, then $P(2n)$ is true.

d) $P(1)$ is true; for all positive integers n, if $P(n)$ is true, then $P(n+1)$ is true.

26. Suppose that $P(n)$ is a propositional function. Determine for which nonnegative integers n the statement $P(n)$ must be true if

a) $P(0)$ is true; for all nonnegative integers n, if $P(n)$ is true, then $P(n+2)$ is true.

b) $P(0)$ is true; for all nonnegative integers n, if $P(n)$ is true, then $P(n+3)$ is true.

c) $P(0)$ and $P(1)$ are true; for all nonnegative integers n, if $P(n)$ and $P(n+1)$ are true, then $P(n+2)$ is true.

d) $P(0)$ is true; for all nonnegative integers n, if $P(n)$ is true, then $P(n+2)$ and $P(n+3)$ are true.

27. Show that if the statement $P(n)$ is true for infinitely many positive integers n and $P(n+1) \rightarrow P(n)$ is true for all positive integers n, then $P(n)$ is true for all positive integers n.

28. Let b be a fixed integer and j a fixed positive integer. Show that if $P(b), P(b+1), \ldots, P(b+j)$ are true and $[P(b) \wedge P(b+1) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for every integer $k \geq b+j$, then $P(n)$ is true for all integers n with $n \geq b$.

29. What is wrong with this "proof" by strong induction?

"Theorem" For every nonnegative integer n, $5n = 0$.

Basis Step: $5 \cdot 0 = 0$.

Inductive Step: Suppose that $5j = 0$ for all nonnegative integers j with $0 \leq j \leq k$. Write $k+1 = i+j$, where i and j are natural numbers less than $k+1$. By the inductive hypothesis, $5(k+1) = 5(i+j) = 5i + 5j = 0 + 0 = 0$.

***30.** Find the flaw with the following "proof" that $a^n = 1$ for all nonnegative integers n, whenever a is a nonzero real number.

Basis Step: $a^0 = 1$ is true by the definition of a^0.

Inductive Step: Assume that $a^j = 1$ for all nonnegative integers j with $j \leq k$. Then note that

$$a^{k+1} = \frac{a^k \cdot a^k}{a^{k-1}} = \frac{1 \cdot 1}{1} = 1.$$

***31.** Show that strong induction is a valid method of proof by showing that it follows from the well-ordering property.

32. Find the flaw with the following "proof" that every postage of three cents or more can be formed using just three-cent and four-cent stamps.

Basis Step: We can form postage of three cents with a single three-cent stamp and we can form postage of four cents using a single four-cent stamp.

Inductive Step: Assume that we can form postage of j cents for all nonnegative integers j with $j \leq k$ using just three-cent and four-cent stamps. We can then form postage of $k + 1$ cents by replacing one three-cent stamp with a four-cent stamp or by replacing two four-cent stamps by three three-cent stamps.

33. Show that we can prove that $P(n, k)$ is true for all pairs of positive integers n and k if we show

a) $P(1, 1)$ is true and $P(n, k) \rightarrow [P(n + 1, k) \wedge P(n, k + 1)]$ is true for all positive integers n and k.

b) $P(1, k)$ is true for all positive integers k, and $P(n, k) \rightarrow P(n + 1, k)$ is true for all positive integers n and k.

c) $P(n, 1)$ is true for all positive integers n, and $P(n, k) \rightarrow P(n, k + 1)$ is true for all positive integers n and k.

34. Prove that $\sum_{j=1}^{n} j(j + 1)(j + 2) \cdots (j + k - 1) = n(n + 1)(n + 2) \cdots (n + k)/(k + 1)$ for all positive integers k and n. [*Hint*: Use a technique from Exercise 33.]

***35.** Show that if $a_1, a_2, \ldots, a_n$ are n distinct real numbers, exactly $n - 1$ multiplications are used to compute the product of these n numbers no matter how parentheses are inserted into their product. [*Hint:* Use strong induction and consider the last multiplication.]

***36.** The well-ordering property can be used to show that there is a unique greatest common divisor of two positive integers. Let a and b be positive integers, and let S be the set of positive integers of the form $as + bt$, where s and t are integers.

a) Show that S is nonempty.

b) Use the well-ordering property to show that S has a smallest element c.

c) Show that if d is a common divisor of a and b, then d is a divisor of c.

d) Show that $c \mid a$ and $c \mid b$. [*Hint:* First, assume that $c \nmid a$. Then $a = qc + r$, where $0 < r < c$. Show that $r \in S$, contradicting the choice of c.]

e) Conclude from (c) and (d) that the greatest common divisor of a and b exists. Finish the proof by showing that this greatest common divisor is unique.

37. Let a be an integer and d be a positive integer. Show that the integers q and r with $a = dq + r$ and $0 \leq r < d$, which were shown to exist in Example 5, are unique.

38. Use mathematical induction to show that a rectangular checkerboard with an even number of cells and two squares missing, one white and one black, can be covered by dominoes.

****39.** Can you use the well-ordering property to prove the statement: "Every positive integer can be described using no more than fifteen English words"? Assume the words come from a particular dictionary of English. [*Hint:* Suppose that there are positive integers that cannot be described using no more than fifteen English words. By well ordering, *the smallest positive integer that cannot be described using no more than fifteen English words* would then exist.]

40. Use the well-ordering principle to show that if x and y are real numbers with $x < y$, then there is a rational number r with $x < r < y$. [*Hint:* Use the Archimedean property, given in Appendix 1, to find a positive integer A with $A > 1/(y - x)$. Then show that there is a rational number r with denominator A between x and y by looking at the numbers $\lfloor x \rfloor + j/A$, where j is a positive integer.]

***41.** Show that the well-ordering property can be proved when the principle of mathematical induction is taken as an axiom.

***42.** Show that the principle of mathematical induction and strong induction are equivalent; that is, each can be shown to be valid from the other.

***43.** Show that we can prove the well-ordering property when we take strong induction as an axiom instead of taking the well-ordering property as an axiom.

5.3 Recursive Definitions and Structural Induction

Introduction

Sometimes it is difficult to define an object explicitly. However, it may be easy to define this object in terms of itself. This process is called **recursion**. For instance, the picture shown in Figure 1 is produced recursively. First, an original picture is given. Then a process of successively superimposing centered smaller pictures on top of the previous pictures is carried out.

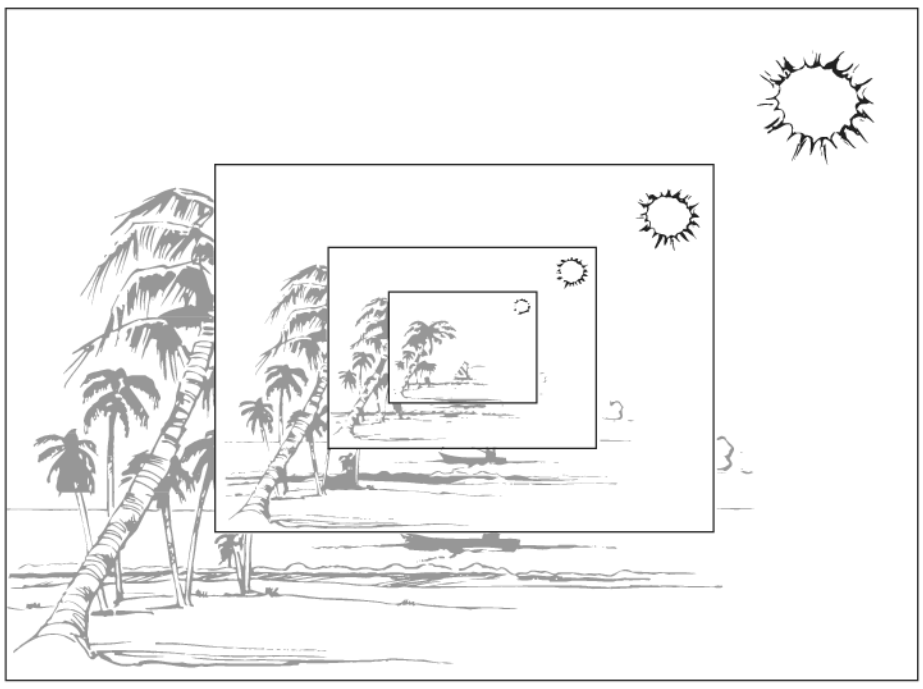


FIGURE 1 A Recursively Defined Picture.

We can use recursion to define sequences, functions, and sets. In Section 2.4, and in most beginning mathematics courses, the terms of a sequence are specified using an explicit formula. For instance, the sequence of powers of 2 is given by $a_n = 2^n$ for $n = 0, 1, 2, \ldots$. Recall from Section 2.4 that we can also define a sequence recursively by specifying how terms of the sequence are found from previous terms. The sequence of powers of 2 can also be defined by giving the first term of the sequence, namely, $a_0 = 1$, and a rule for finding a term of the sequence from the previous one, namely, $a_{n+1} = 2a_n$ for $n = 0, 1, 2, \ldots$. When we define a sequence recursively by specifying how terms of the sequence are found from previous terms, we can use induction to prove results about the sequence.

When we define a set recursively, we specify some initial elements in a basis step and provide a rule for constructing new elements from those we already have in the recursive step. To prove results about recursively defined sets we use a method called *structural induction*.

Recursively Defined Functions

We use two steps to define a function with the set of nonnegative integers as its domain:

BASIS STEP: Specify the value of the function at zero.

Assessment *RECURSIVE STEP:* Give a rule for finding its value at an integer from its values at smaller integers.

Such a definition is called a **recursive** or **inductive definition**. Note that a function $f(n)$ from the set of nonnegative integers to the set of a real numbers is the same as a sequence $a_0, a_1, \ldots$ where a_i is a real number for every nonnegative integer i. So, defining a real-valued sequence $a_0, a_1, \ldots$ using a recurrence relation, as was done in Section 2.4, is the same as defining a function from the set of nonnegative integers to the set of real numbers.

EXAMPLE 1 Suppose that f is defined recursively by

$$f(0) = 3,$$
$$f(n+1) = 2f(n) + 3.$$

Extra Examples

Find $f(1)$, $f(2)$, $f(3)$, and $f(4)$.

Solution: From the recursive definition it follows that

$$f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9,$$
$$f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21,$$
$$f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45,$$
$$f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93.$$

◀

Recursively defined functions are **well defined**. That is, for every positive integer, the value of the function at this integer is determined in an unambiguous way. This means that given any positive integer, we can use the two parts of the definition to find the value of the function at that integer, and that we obtain the same value no matter how we apply the two parts of the definition. This is a consequence of the principle of mathematical induction. (See Exercise 56.) Additional examples of recursive definitions are given in Examples 2 and 3.

EXAMPLE 2 Give a recursive definition of a^n, where a is a nonzero real number and n is a nonnegative integer.

Solution: The recursive definition contains two parts. First a^0 is specified, namely, $a^0 = 1$. Then the rule for finding a^{n+1} from a^n, namely, $a^{n+1} = a \cdot a^n$, for $n = 0, 1, 2, 3, \ldots$, is given. These two equations uniquely define a^n for all nonnegative integers n. ◀

EXAMPLE 3 Give a recursive definition of

$$\sum_{k=0}^{n} a_k.$$

Solution: The first part of the recursive definition is

$$\sum_{k=0}^{0} a_k = a_0.$$

The second part is

$$\sum_{k=0}^{n+1} a_k = \left(\sum_{k=0}^{n} a_k\right) + a_{n+1}.$$

◀

In some recursive definitions of functions, the values of the function at the first k positive integers are specified, and a rule is given for determining the value of the function at larger integers from its values at some or all of the preceding k integers. That recursive definitions defined in this way produce well-defined functions follows from strong induction (see Exercise 57).

Recall from Section 2.4 that the Fibonacci numbers, $f_0, f_1, f_2, \ldots$, are defined by the equations $f_0 = 0$, $f_1 = 1$, and

Links

$$f_n = f_{n-1} + f_{n-2}$$

for $n = 2, 3, 4, \ldots$. [We can think of the Fibonacci number f_n either as the nth term of the sequence of Fibonacci numbers $f_0, f_1, \ldots$ or as the value at the integer n of a function $f(n)$.]

We can use the recursive definition of the Fibonacci numbers to prove many properties of these numbers. We give one such property in Example 4.

EXAMPLE 4 Show that whenever $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = (1 + \sqrt{5})/2$.

Extra Examples

Solution: We can use strong induction to prove this inequality. Let $P(n)$ be the statement $f_n > \alpha^{n-2}$. We want to show that $P(n)$ is true whenever n is an integer greater than or equal to 3.

BASIS STEP: First, note that

$$\alpha < 2 = f_3, \qquad \alpha^2 = (3 + \sqrt{5})/2 < 3 = f_4,$$

so $P(3)$ and $P(4)$ are true.

INDUCTIVE STEP: Assume that $P(j)$ is true, namely, that $f_j > \alpha^{j-2}$, for all integers j with $3 \leq j \leq k$, where $k \geq 4$. We must show that $P(k + 1)$ is true, that is, that $f_{k+1} > \alpha^{k-1}$. Because α is a solution of $x^2 - x - 1 = 0$ (as the quadratic formula shows), it follows that $\alpha^2 = \alpha + 1$. Therefore,

$$\alpha^{k-1} = \alpha^2 \cdot \alpha^{k-3} = (\alpha + 1)\alpha^{k-3} = \alpha \cdot \alpha^{k-3} + 1 \cdot \alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3}.$$

By the inductive hypothesis, because $k \geq 4$, we have

$$f_{k-1} > \alpha^{k-3}, \qquad f_k > \alpha^{k-2}.$$

Therefore, it follows that

$$f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}.$$

Hence, $P(k + 1)$ is true. This completes the proof. ◀

Remark: The inductive step shows that whenever $k \geq 4$, $P(k + 1)$ follows from the assumption that $P(j)$ is true for $3 \leq j \leq k$. Hence, the inductive step does *not* show that $P(3) \rightarrow P(4)$. Therefore, we had to show that $P(4)$ is true separately.

We can now show that the Euclidean algorithm, introduced in Section 4.3, uses $O(\log b)$ divisions to find the greatest common divisor of the positive integers a and b, where $a \geq b$.

THEOREM 1 **LAMÉ'S THEOREM** Let a and b be positive integers with $a \geq b$. Then the number of divisions used by the Euclidean algorithm to find $\gcd(a, b)$ is less than or equal to five times the number of decimal digits in b.

Proof: Recall that when the Euclidean algorithm is applied to find $\gcd(a, b)$ with $a \geq b$, this sequence of equations (where $a = r_0$ and $b = r_1$) is obtained.

$$\begin{aligned}
r_0 &= r_1 q_1 + r_2 && 0 \leq r_2 < r_1, \\
r_1 &= r_2 q_2 + r_3 && 0 \leq r_3 < r_2, \\
&\;\;\vdots \\
r_{n-2} &= r_{n-1} q_{n-1} + r_n && 0 \leq r_n < r_{n-1}, \\
r_{n-1} &= r_n q_n.
\end{aligned}$$

Here n divisions have been used to find $r_n = \gcd(a, b)$. Note that the quotients $q_1, q_2, \ldots, q_{n-1}$ are all at least 1. Moreover, $q_n \geq 2$, because $r_n < r_{n-1}$. This implies that

$$\begin{aligned}
r_n &\geq 1 = f_2, \\
r_{n-1} &\geq 2r_n \geq 2f_2 = f_3, \\
r_{n-2} &\geq r_{n-1} + r_n \geq f_3 + f_2 = f_4, \\
&\;\;\vdots \\
r_2 &\geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n, \\
b &= r_1 \geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1}.
\end{aligned}$$

It follows that if n divisions are used by the Euclidean algorithm to find $\gcd(a, b)$ with $a \geq b$, then $b \geq f_{n+1}$. By Example 4 we know that $f_{n+1} > \alpha^{n-1}$ for $n > 2$, where $\alpha = (1 + \sqrt{5})/2$. Therefore, it follows that $b > \alpha^{n-1}$. Furthermore, because $\log_{10} \alpha \approx 0.208 > 1/5$, we see that

$$\log_{10} b > (n - 1) \log_{10} \alpha > (n - 1)/5.$$

Hence, $n - 1 < 5 \cdot \log_{10} b$. Now suppose that b has k decimal digits. Then $b < 10^k$ and $\log_{10} b < k$. It follows that $n - 1 < 5k$, and because k is an integer, it follows that $n \leq 5k$. This finishes the proof. ◁

Because the number of decimal digits in b, which equals $\lfloor \log_{10} b \rfloor + 1$, is less than or equal to $\log_{10} b + 1$, Theorem 1 tells us that the number of divisions required to find $\gcd(a, b)$ with

Links

FIBONACCI (1170–1250) Fibonacci (short for *filius Bonacci,* or "son of Bonacci") was also known as Leonardo of Pisa. He was born in the Italian commercial center of Pisa. Fibonacci was a merchant who traveled extensively throughout the Mideast, where he came into contact with Arabian mathematics. In his book *Liber Abaci,* Fibonacci introduced the European world to Arabic notation for numerals and algorithms for arithmetic. It was in this book that his famous rabbit problem (described in Section 8.1) appeared. Fibonacci also wrote books on geometry and trigonometry and on Diophantine equations, which involve finding integer solutions to equations.

$a > b$ is less than or equal to $5(\log_{10} b + 1)$. Because $5(\log_{10} b + 1)$ is $O(\log b)$, we see that $O(\log b)$ divisions are used by the Euclidean algorithm to find $\gcd(a, b)$ whenever $a > b$.

Recursively Defined Sets and Structures

Assessment

We have explored how functions can be defined recursively. We now turn our attention to how sets can be defined recursively. Just as in the recursive definition of functions, recursive definitions of sets have two parts, a **basis step** and a **recursive step**. In the basis step, an initial collection of elements is specified. In the recursive step, rules for forming new elements in the set from those already known to be in the set are provided. Recursive definitions may also include an **exclusion rule**, which specifies that a recursively defined set contains nothing other than those elements specified in the basis step or generated by applications of the recursive step. In our discussions, we will always tacitly assume that the exclusion rule holds and no element belongs to a recursively defined set unless it is in the initial collection specified in the basis step or can be generated using the recursive step one or more times. Later we will see how we can use a technique known as structural induction to prove results about recursively defined sets.

Examples 5, 6, 8, and 9 illustrate the recursive definition of sets. In each example, we show those elements generated by the first few applications of the recursive step.

EXAMPLE 5 Consider the subset S of the set of integers recursively defined by

BASIS STEP: $3 \in S$.

RECURSIVE STEP: If $x \in S$ and $y \in S$, then $x + y \in S$.

Extra Examples

The new elements found to be in S are 3 by the basis step, $3 + 3 = 6$ at the first application of the recursive step, $3 + 6 = 6 + 3 = 9$ and $6 + 6 = 12$ at the second application of the recursive step, and so on. We will show in Example 10 that S is the set of all positive multiples of 3. ◀

Recursive definitions play an important role in the study of strings. (See Chapter 13 for an introduction to the theory of formal languages, for example.) Recall from Section 2.4 that a string over an alphabet Σ is a finite sequence of symbols from Σ. We can define Σ^*, the set of strings over Σ, recursively, as Definition 1 shows.

DEFINITION 1 The set Σ^* of *strings* over the alphabet Σ is defined recursively by

BASIS STEP: $\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Links

GABRIEL LAMÉ (1795–1870) Gabriel Lamé entered the École Polytechnique in 1813, graduating in 1817. He continued his education at the École des Mines, graduating in 1820.

In 1820 Lamé went to Russia, where he was appointed director of the Schools of Highways and Transportation in St. Petersburg. Not only did he teach, but he also planned roads and bridges while in Russia. He returned to Paris in 1832, where he helped found an engineering firm. However, he soon left the firm, accepting the chair of physics at the École Polytechnique, which he held until 1844. While holding this position, he was active outside academia as an engineering consultant, serving as chief engineer of mines and participating in the building of railways.

Lamé contributed original work to number theory, applied mathematics, and thermodynamics. His best-known work involves the introduction of curvilinear coordinates. His work on number theory includes proving Fermat's last theorem for $n = 7$, as well as providing the upper bound for the number of divisions used by the Euclidean algorithm given in this text.

In the opinion of Gauss, one of the most important mathematicians of all time, Lamé was the foremost French mathematician of his time. However, French mathematicians considered him too practical, whereas French scientists considered him too theoretical.

The basis step of the recursive definition of strings says that the empty string belongs to Σ^*. The recursive step states that new strings are produced by adding a symbol from Σ to the end of strings in Σ^*. At each application of the recursive step, strings containing one additional symbol are generated.

EXAMPLE 6 If $\Sigma = \{0, 1\}$, the strings found to be in Σ^*, the set of all bit strings, are λ, specified to be in Σ^* in the basis step, 0 and 1 formed during the first application of the recursive step, 00, 01, 10, and 11 formed during the second application of the recursive step, and so on. ◀

Recursive definitions can be used to define operations or functions on the elements of recursively defined sets. This is illustrated in Definition 2 of the concatenation of two strings and Example 7 concerning the length of a string.

DEFINITION 2

Two strings can be combined via the operation of *concatenation*. Let Σ be a set of symbols and Σ^* the set of strings formed from symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

RECURSIVE STEP: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

The concatenation of the strings w_1 and w_2 is often written as $w_1 w_2$ rather than $w_1 \cdot w_2$. By repeated application of the recursive definition, it follows that the concatenation of two strings w_1 and w_2 consists of the symbols in w_1 followed by the symbols in w_2. For instance, the concatenation of $w_1 = abra$ and $w_2 = cadabra$ is $w_1 w_2 = abracadabra$.

EXAMPLE 7 **Length of a String** Give a recursive definition of $l(w)$, the length of the string w.

Solution: The length of a string can be recursively defined by

$$l(\lambda) = 0;$$
$$l(wx) = l(w) + 1 \text{ if } w \in \Sigma^* \text{ and } x \in \Sigma.$$

◀

Another important use of recursive definitions is to define **well-formed formulae** of various types. This is illustrated in Examples 8 and 9.

EXAMPLE 8 **Well-Formed Formulae in Propositional Logic** We can define the set of well-formed formulae in propositional logic involving **T, F,** propositional variables, and operators from the set $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

BASIS STEP: **T**, **F**, and s, where s is a propositional variable, are well-formed formulae.

RECURSIVE STEP: If E and F are well-formed formulae, then $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \rightarrow F)$, and $(E \leftrightarrow F)$ are well-formed formulae.

For example, by the basis step we know that **T**, **F**, p, and q are well-formed formulae, where p and q are propositional variables. From an initial application of the recursive step, we know that $(p \vee q)$, $(p \rightarrow \mathbf{F})$, $(\mathbf{F} \rightarrow q)$, and $(q \wedge \mathbf{F})$ are well-formed formulae. A second application of the recursive step shows that $((p \vee q) \rightarrow (q \wedge \mathbf{F}))$, $(q \vee (p \vee q))$, and $((p \rightarrow \mathbf{F}) \rightarrow \mathbf{T})$ are well-formed formulae. We leave it to the reader to show that $p\neg \wedge q$, $pq\wedge$, and $\neg \wedge pq$ are *not* well-formed formulae, by showing that none can be obtained using the basis step and one or more applications of the recursive step. ◀

EXAMPLE 9 **Well-Formed Formulae of Operators and Operands** We can define the set of well-formed formulae consisting of variables, numerals, and operators from the set $\{+, -, *, /, \uparrow\}$ (where $*$ denotes multiplication and $\uparrow$ denotes exponentiation) recursively.

BASIS STEP: x is a well-formed formula if x is a numeral or a variable.

RECURSIVE STEP: If F and G are well-formed formulae, then $(F + G)$, $(F - G)$, $(F * G)$, (F/G), and $(F \uparrow G)$ are well-formed formulae.

For example, by the basis step we see that x, y, 0, and 3 are well-formed formulae (as is any variable or numeral). Well-formed formulae generated by applying the recursive step once include $(x + 3)$, $(3 + y)$, $(x - y)$, $(3 - 0)$, $(x * 3)$, $(3 * y)$, $(3/0)$, (x/y), $(3 \uparrow x)$, and $(0 \uparrow 3)$. Applying the recursive step twice shows that formulae such as $((x + 3) + 3)$ and $(x - (3 * y))$ are well-formed formulae. [Note that $(3/0)$ is a well-formed formula because we are concerned only with syntax matters here.] We leave it to the reader to show that each of the formulae $x3+$, $y * +x$, and $* x/y$ is *not* a well-formed formula by showing that none of them can be obtained from the basis step and one or more applications of the recursive step. ◀

We will study trees extensively in Chapter 11. A tree is a special type of a graph; a graph is made up of vertices and edges connecting some pairs of vertices. We will study graphs in Chapter 10. We will briefly introduce them here to illustrate how they can be defined recursively.

DEFINITION 3

The set of *rooted trees,* where a rooted tree consists of a set of vertices containing a distinguished vertex called the *root,* and edges connecting these vertices, can be defined recursively by these steps:

BASIS STEP: A single vertex r is a rooted tree.

RECURSIVE STEP: Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees with roots $r_1, r_2, \ldots, r_n$, respectively. Then the graph formed by starting with a root r, which is not in any of the rooted trees $T_1, T_2, \ldots, T_n$, and adding an edge from r to each of the vertices $r_1, r_2, \ldots, r_n$, is also a rooted tree.

In Figure 2 we illustrate some of the rooted trees formed starting with the basis step and applying the recursive step one time and two times. Note that infinitely many rooted trees are formed at each application of the recursive definition.

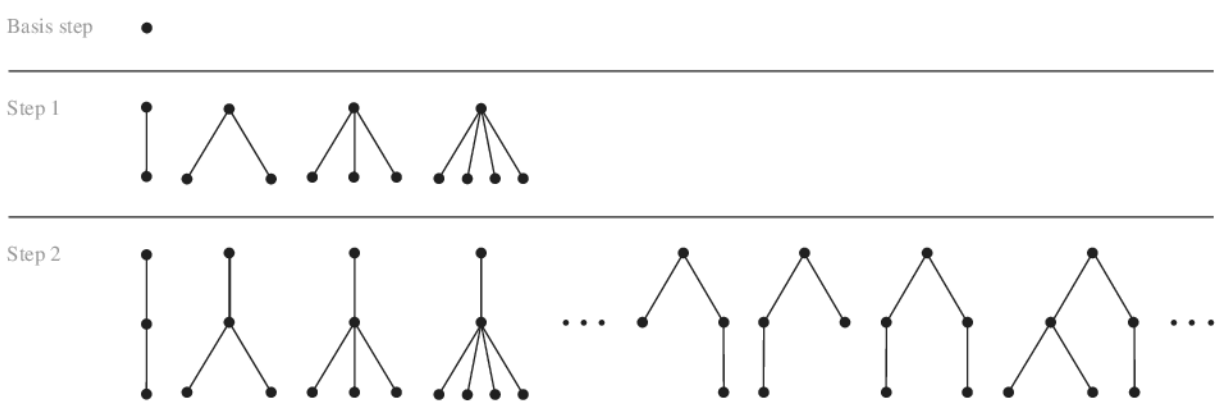


FIGURE 2 **Building Up Rooted Trees.**

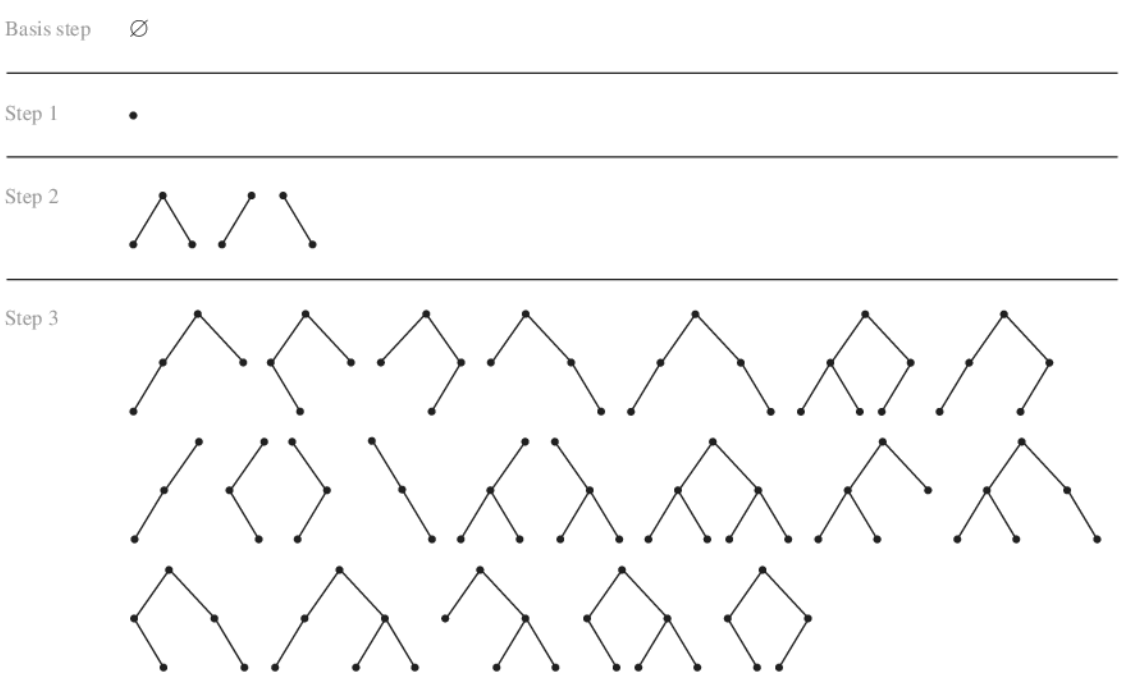


FIGURE 3 **Building Up Extended Binary Trees.**

Binary trees are a special type of rooted trees. We will provide recursive definitions of two types of binary trees—full binary trees and extended binary trees. In the recursive step of the definition of each type of binary tree, two binary trees are combined to form a new tree with one of these trees designated the left subtree and the other the right subtree. In extended binary trees, the left subtree or the right subtree can be empty, but in full binary trees this is not possible. Binary trees are one of the most important types of structures in computer science. In Chapter 11 we will see how they can be used in searching and sorting algorithms, in algorithms for compressing data, and in many other applications. We first define extended binary trees.

DEFINITION 4

The set of *extended binary trees* can be defined recursively by these steps:

BASIS STEP: The empty set is an extended binary tree.

RECURSIVE STEP: If T_1 and T_2 are disjoint extended binary trees, there is an extended binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2 when these trees are nonempty.

Figure 3 shows how extended binary trees are built up by applying the recursive step from one to three times.

We now show how to define the set of full binary trees. Note that the difference between this recursive definition and that of extended binary trees lies entirely in the basis step.

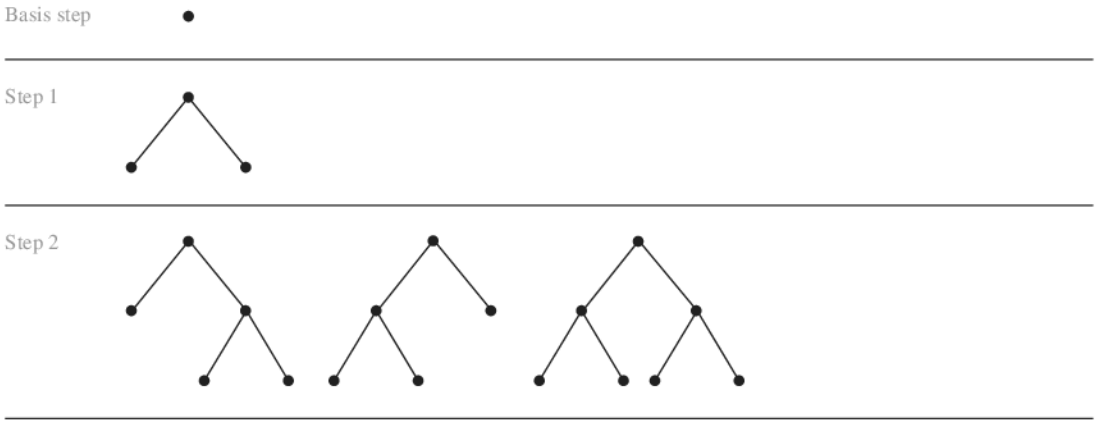


FIGURE 4 **Building Up Full Binary Trees.**

DEFINITION 5

The set of full binary trees can be defined recursively by these steps:

BASIS STEP: There is a full binary tree consisting only of a single vertex r.

RECURSIVE STEP: If T_1 and T_2 are disjoint full binary trees, there is a full binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2.

Figure 4 shows how full binary trees are built up by applying the recursive step one and two times.

Structural Induction

To prove results about recursively defined sets, we generally use some form of mathematical induction. Example 10 illustrates the connection between recursively defined sets and mathematical induction.

EXAMPLE 10 Show that the set S defined in Example 5 by specifying that $3 \in S$ and that if $x \in S$ and $y \in S$, then $x + y \in S$, is the set of all positive integers that are multiples of 3.

Solution: Let A be the set of all positive integers divisible by 3. To prove that $A = S$, we must show that A is a subset of S and that S is a subset of A. To prove that A is a subset of S, we must show that every positive integer divisible by 3 is in S. We will use mathematical induction to prove this.

Let $P(n)$ be the statement that $3n$ belongs to S. The basis step holds because by the first part of the recursive definition of S, $3 \cdot 1 = 3$ is in S. To establish the inductive step, assume that $P(k)$ is true, namely, that $3k$ is in S. Because $3k$ is in S and because 3 is in S, it follows from the second part of the recursive definition of S that $3k + 3 = 3(k + 1)$ is also in S.

To prove that S is a subset of A, we use the recursive definition of S. First, the basis step of the definition specifies that 3 is in S. Because $3 = 3 \cdot 1$, all elements specified to be in S in this step are divisible by 3 and are therefore in A. To finish the proof, we must show that all integers in S generated using the second part of the recursive definition are in A. This consists of showing that $x + y$ is in A whenever x and y are elements of S also assumed to be in A. Now if x and y are both in A, it follows that $3 \mid x$ and $3 \mid y$. By part (i) of Theorem 1 of Section 4.1, it follows that $3 \mid x + y$, completing the proof. ◀

In Example 10 we used mathematical induction over the set of positive integers and a recursive definition to prove a result about a recursively defined set. However, instead of using mathematical induction directly to prove results about recursively defined sets, we can use a more convenient form of induction known as **structural induction**. A proof by structural induction consists of two parts. These parts are

BASIS STEP: Show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.

RECURSIVE STEP: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

The validity of structural induction follows from the principle of mathematical induction for the nonnegative integers. To see this, let $P(n)$ state that the claim is true for all elements of the set that are generated by n or fewer applications of the rules in the recursive step of a recursive definition. We will have established that the principle of mathematical induction implies the principle of structural induction if we can show that $P(n)$ is true whenever n is a positive integer. In the basis step of a proof by structural induction we show that $P(0)$ is true. That is, we show that the result is true of all elements specified to be in the set in the basis step of the definition. A consequence of the recursive step is that if we assume $P(k)$ is true, it follows that $P(k + 1)$ is true. When we have completed a proof using structural induction, we have shown that $P(0)$ is true and that $P(k)$ implies $P(k + 1)$. By mathematical induction it follows that $P(n)$ is true for all nonnegative integers n. This also shows that the result is true for all elements generated by the recursive definition, and shows that structural induction is a valid proof technique.

EXAMPLES OF PROOFS USING STRUCTURAL INDUCTION Structural induction can be used to prove that all members of a set constructed recursively have a particular property. We will illustrate this idea by using structural induction to prove results about well-formed formulae, strings, and binary trees. For each proof, we have to carry out the appropriate basis step and the appropriate recursive step. For example, to use structural induction to prove a result about the set of well-formed formulae defined in Example 8, where we specify that **T**, **F**, and every propositional variable s are well-formed formulae and where we specify that if E and F are well-formed formulae, then $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \rightarrow F)$, and $(E \leftrightarrow F)$ are well-formed formulae, we need to complete this basis step and this recursive step.

BASIS STEP: Show that the result is true for **T**, **F**, and s whenever s is a propositional variable.

RECURSIVE STEP: Show that if the result is true for the compound propositions p and q, it is also true for $(\neg p)$, $(p \vee q)$, $(p \wedge q)$, $(p \rightarrow q)$, and $(p \leftrightarrow q)$.

Example 11 illustrates how we can prove results about well-formed formulae using structural induction.

EXAMPLE 11 Show that every well-formed formula for compound propositions, as defined in Example 8, contains an equal number of left and right parentheses.

Solution:

BASIS STEP: Each of the formula **T**, **F**, and s contains no parentheses, so clearly they contain an equal number of left and right parentheses.

RECURSIVE STEP: Assume p and q are well-formed formulae each containing an equal number of left and right parentheses. That is, if l_p and l_q are the number of left parentheses in p and q, respectively, and r_p and r_q are the number of right parentheses in p and q, respectively, then $l_p = r_p$ and $l_q = r_q$. To complete the inductive step, we need to show that each of

$(\neg p)$, $(p \vee q)$, $(p \wedge q)$, $(p \rightarrow q)$, and $(p \leftrightarrow q)$ also contains an equal number of left and right parentheses. The number of left parentheses in the first of these compound propositions equals $l_p + 1$ and in each of the other compound propositions equals $l_p + l_q + 1$. Similarly, the number of right parentheses in the first of these compound propositions equals $r_p + 1$ and in each of the other compound propositions equals $r_p + r_q + 1$. Because $l_p = r_p$ and $l_q = r_q$, it follows that each of these compound expressions contains the same number of left and right parentheses. This completes the proof by structural induction. ◀

Suppose that $P(w)$ is a propositional function over the set of strings $w \in \Sigma^*$. To use structural induction to prove that $P(w)$ holds for all strings $w \in \Sigma^*$, we need to complete both a basis step and a recursive step. These steps are:

BASIS STEP: Show that $P(\lambda)$ is true.

RECURSIVE STEP: Assume that $P(w)$ is true, where $w \in \Sigma^*$. Show that if $x \in \Sigma$, then $P(wx)$ must also be true.

Example 12 illustrates how structural induction can be used in proofs about strings.

EXAMPLE 12 Use structural induction to prove that $l(xy) = l(x) + l(y)$, where x and y belong to Σ^*, the set of strings over the alphabet Σ.

Solution: We will base our proof on the recursive definition of the set Σ^* given in Definition 1 and the definition of the length of a string in Example 7, which specifies that $l(\lambda) = 0$ and $l(wx) = l(w) + 1$ when $w \in \Sigma^*$ and $x \in \Sigma$. Let $P(y)$ be the statement that $l(xy) = l(x) + l(y)$ whenever x belongs to Σ^*.

BASIS STEP: To complete the basis step, we must show that $P(\lambda)$ is true. That is, we must show that $l(x\lambda) = l(x) + l(\lambda)$ for all $x \in \Sigma^*$. Because $l(x\lambda) = l(x) = l(x) + 0 = l(x) + l(\lambda)$ for every string x, it follows that $P(\lambda)$ is true.

RECURSIVE STEP: To complete the inductive step, we assume that $P(y)$ is true and show that this implies that $P(ya)$ is true whenever $a \in \Sigma$. What we need to show is that $l(xya) = l(x) + l(ya)$ for every $a \in \Sigma$. To show this, note that by the recursive definition of $l(w)$ (given in Example 7), we have $l(xya) = l(xy) + 1$ and $l(ya) = l(y) + 1$. And, by the inductive hypothesis, $l(xy) = l(x) + l(y)$. We conclude that $l(xya) = l(x) + l(y) + 1 = l(x) + l(ya)$. ◀

We can prove results about trees or special classes of trees using structural induction. For example, to prove a result about full binary trees using structural induction we need to complete this basis step and this recursive step.

BASIS STEP: Show that the result is true for the tree consisting of a single vertex.

RECURSIVE STEP: Show that if the result is true for the trees T_1 and T_2, then it is true for tree $T_1 \cdot T_2$ consisting of a root r, which has T_1 as its left subtree and T_2 as its right subtree.

Before we provide an example showing how structural induction can be used to prove a result about full binary trees, we need some definitions. We will recursively define the height $h(T)$ and the number of vertices $n(T)$ of a full binary tree T. We begin by defining the height of a full binary tree.

DEFINITION 6

We define the height $h(T)$ of a full binary tree T recursively.

BASIS STEP: The height of the full binary tree T consisting of only a root r is $h(T) = 0$.

RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has height $h(T) = 1 + \max(h(T_1), h(T_2))$.

If we let $n(T)$ denote the number of vertices in a full binary tree, we observe that $n(T)$ satisfies the following recursive formula:

BASIS STEP: The number of vertices $n(T)$ of the full binary tree T consisting of only a root r is $n(T) = 1$.

RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the number of vertices of the full binary tree $T = T_1 \cdot T_2$ is $n(T) = 1 + n(T_1) + n(T_2)$.

We now show how structural induction can be used to prove a result about full binary trees.

THEOREM 2 If T is a full binary tree T, then $n(T) \leq 2^{h(T)+1} - 1$.

Proof: We prove this inequality using structural induction.

BASIS STEP: For the full binary tree consisting of just the root r the result is true because $n(T) = 1$ and $h(T) = 0$, so that $n(T) = 1 \leq 2^{0+1} - 1 = 1$.

RECURSIVE STEP: For the inductive hypothesis we assume that $n(T_1) \leq 2^{h(T_1)+1} - 1$ and $n(T_2) \leq 2^{h(T_2)+1} - 1$ whenever T_1 and T_2 are full binary trees. By the recursive formulae for $n(T)$ and $h(T)$ we have $n(T) = 1 + n(T_1) + n(T_2)$ and $h(T) = 1 + \max(h(T_1), h(T_2))$.

We find that

$$
\begin{aligned}
n(T) &= 1 + n(T_1) + n(T_2) && \text{by the recursive formula for } n(T) \\
&\leq 1 + (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1) && \text{by the inductive hypothesis} \\
&\leq 2 \cdot \max(2^{h(T_1)+1}, 2^{h(T_2)+1}) - 1 && \text{because the sum of two terms is at most 2 times the larger} \\
&= 2 \cdot 2^{\max(h(T_1),h(T_2))+1} - 1 && \text{because } \max(2^x, 2^y) = 2^{\max(x,y)} \\
&= 2 \cdot 2^{h(T)} - 1 && \text{by the recursive definition of } h(T) \\
&= 2^{h(T)+1} - 1.
\end{aligned}
$$

This completes the recursive step. ◁

Generalized Induction

We can extend mathematical induction to prove results about other sets that have the well-ordering property besides the set of integers. Although we will discuss this concept in detail in Section 9.6, we provide an example here to illustrate the usefulness of such an approach.

As an example, note that we can define an ordering on $\mathbf{N} \times \mathbf{N}$, the ordered pairs of non-negative integers, by specifying that (x_1, y_1) is less than or equal to (x_2, y_2) if either $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$; this is called the **lexicographic ordering**. The set $\mathbf{N} \times \mathbf{N}$ with this ordering has the property that every subset of $\mathbf{N} \times \mathbf{N}$ has a least element (see Exercise 53 in Section 9.6). This implies that we can recursively define the terms $a_{m,n}$, with $m \in \mathbf{N}$ and $n \in \mathbf{N}$, and prove results about them using a variant of mathematical induction, as illustrated in Example 13.

EXAMPLE 13 Suppose that $a_{m,n}$ is defined recursively for $(m, n) \in \mathbf{N} \times \mathbf{N}$ by $a_{0,0} = 0$ and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 1 & \text{if } n = 0 \text{ and } m > 0 \\ a_{m,n-1} + n & \text{if } n > 0. \end{cases}$$

Show that $a_{m,n} = m + n(n + 1)/2$ for all $(m, n) \in \mathbf{N} \times \mathbf{N}$, that is, for all pairs of nonnegative integers.

Solution: We can prove that $a_{m,n} = m + n(n + 1)/2$ using a generalized version of mathematical induction. The basis step requires that we show that this formula is valid when $(m, n) = (0, 0)$. The induction step requires that we show that if the formula holds for all pairs smaller than (m, n) in the lexicographic ordering of $\mathbf{N} \times \mathbf{N}$, then it also holds for (m, n).

BASIS STEP: Let $(m, n) = (0, 0)$. Then by the basis case of the recursive definition of $a_{m,n}$ we have $a_{0,0} = 0$. Furthermore, when $m = n = 0$, $m + n(n + 1)/2 = 0 + (0 \cdot 1)/2 = 0$. This completes the basis step.

INDUCTIVE STEP: Suppose that $a_{m',n'} = m' + n'(n' + 1)/2$ whenever (m', n') is less than (m, n) in the lexicographic ordering of $\mathbf{N} \times \mathbf{N}$. By the recursive definition, if $n = 0$, then $a_{m,n} = a_{m-1,n} + 1$. Because $(m - 1, n)$ is smaller than (m, n), the inductive hypothesis tells us that $a_{m-1,n} = m - 1 + n(n + 1)/2$, so that $a_{m,n} = m - 1 + n(n + 1)/2 + 1 = m + n(n + 1)/2$, giving us the desired equality. Now suppose that $n > 0$, so $a_{m,n} = a_{m,n-1} + n$. Because $(m, n - 1)$ is smaller than (m, n), the inductive hypothesis tells us that $a_{m,n-1} = m + (n - 1)n/2$, so $a_{m,n} = m + (n - 1)n/2 + n = m + (n^2 - n + 2n)/2 = m + n(n + 1)/2$. This finishes the inductive step. ◀

As mentioned, we will justify this proof technique in Section 9.6.

Exercises

1. Find $f(1)$, $f(2)$, $f(3)$, and $f(4)$ if $f(n)$ is defined recursively by $f(0) = 1$ and for $n = 0, 1, 2, \ldots$
   a) $f(n + 1) = f(n) + 2$.
   b) $f(n + 1) = 3f(n)$.
   c) $f(n + 1) = 2^{f(n)}$.
   d) $f(n + 1) = f(n)^2 + f(n) + 1$.

2. Find $f(1)$, $f(2)$, $f(3)$, $f(4)$, and $f(5)$ if $f(n)$ is defined recursively by $f(0) = 3$ and for $n = 0, 1, 2, \ldots$
   a) $f(n + 1) = -2f(n)$.
   b) $f(n + 1) = 3f(n) + 7$.
   c) $f(n + 1) = f(n)^2 - 2f(n) - 2$.
   d) $f(n + 1) = 3^{f(n)/3}$.

3. Find $f(2)$, $f(3)$, $f(4)$, and $f(5)$ if f is defined recursively by $f(0) = -1$, $f(1) = 2$, and for $n = 1, 2, \ldots$
   a) $f(n + 1) = f(n) + 3f(n - 1)$.
   b) $f(n + 1) = f(n)^2 f(n - 1)$.
   c) $f(n + 1) = 3f(n)^2 - 4f(n - 1)^2$.
   d) $f(n + 1) = f(n - 1)/f(n)$.

4. Find $f(2)$, $f(3)$, $f(4)$, and $f(5)$ if f is defined recursively by $f(0) = f(1) = 1$ and for $n = 1, 2, \ldots$
   a) $f(n + 1) = f(n) - f(n - 1)$.
   b) $f(n + 1) = f(n)f(n - 1)$.
   c) $f(n + 1) = f(n)^2 + f(n - 1)^3$.
   d) $f(n + 1) = f(n)/f(n - 1)$.

5. Determine whether each of these proposed definitions is a valid recursive definition of a function f from the set of nonnegative integers to the set of integers. If f is well defined, find a formula for $f(n)$ when n is a nonnegative integer and prove that your formula is valid.
   a) $f(0) = 0$, $f(n) = 2f(n - 2)$ for $n \geq 1$
   b) $f(0) = 1$, $f(n) = f(n - 1) - 1$ for $n \geq 1$
   c) $f(0) = 2$, $f(1) = 3$, $f(n) = f(n - 1) - 1$ for $n \geq 2$
   d) $f(0) = 1$, $f(1) = 2$, $f(n) = 2f(n - 2)$ for $n \geq 2$
   e) $f(0) = 1$, $f(n) = 3f(n - 1)$ if n is odd and $n \geq 1$ and $f(n) = 9f(n - 2)$ if n is even and $n \geq 2$

6. Determine whether each of these proposed definitions is a valid recursive definition of a function f from the set of nonnegative integers to the set of integers. If f is well defined, find a formula for $f(n)$ when n is a nonnegative integer and prove that your formula is valid.
   a) $f(0) = 1$, $f(n) = -f(n - 1)$ for $n \geq 1$
   b) $f(0) = 1$, $f(1) = 0$, $f(2) = 2$, $f(n) = 2f(n - 3)$ for $n \geq 3$
   c) $f(0) = 0$, $f(1) = 1$, $f(n) = 2f(n + 1)$ for $n \geq 2$
   d) $f(0) = 0$, $f(1) = 1$, $f(n) = 2f(n - 1)$ for $n \geq 1$
   e) $f(0) = 2$, $f(n) = f(n - 1)$ if n is odd and $n \geq 1$ and $f(n) = 2f(n - 2)$ if $n \geq 2$

7. Give a recursive definition of the sequence $\{a_n\}$, $n = 1, 2, 3, \ldots$ if

a) $a_n = 6n$. **b)** $a_n = 2n + 1$.
c) $a_n = 10^n$. **d)** $a_n = 5$.

8. Give a recursive definition of the sequence $\{a_n\}$, $n = 1, 2, 3, \ldots$ if

a) $a_n = 4n - 2$. **b)** $a_n = 1 + (-1)^n$.
c) $a_n = n(n + 1)$. **d)** $a_n = n^2$.

9. Let F be the function such that $F(n)$ is the sum of the first n positive integers. Give a recursive definition of $F(n)$.

10. Give a recursive definition of $S_m(n)$, the sum of the integer m and the nonnegative integer n.

11. Give a recursive definition of $P_m(n)$, the product of the integer m and the nonnegative integer n.

In Exercises 12–19 f_n is the nth Fibonacci number.

12. Prove that $f_1^2 + f_2^2 + \cdots + f_n^2 = f_n f_{n+1}$ when n is a positive integer.

13. Prove that $f_1 + f_3 + \cdots + f_{2n-1} = f_{2n}$ when n is a positive integer.

***14.** Show that $f_{n+1} f_{n-1} - f_n^2 = (-1)^n$ when n is a positive integer.

***15.** Show that $f_0 f_1 + f_1 f_2 + \cdots + f_{2n-1} f_{2n} = f_{2n}^2$ when n is a positive integer.

***16.** Show that $f_0 - f_1 + f_2 - \cdots - f_{2n-1} + f_{2n} = f_{2n-1} - 1$ when n is a positive integer.

17. Determine the number of divisions used by the Euclidean algorithm to find the greatest common divisor of the Fibonacci numbers f_n and f_{n+1}, where n is a nonnegative integer. Verify your answer using mathematical induction.

18. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

Show that

$$\mathbf{A}^n = \begin{bmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{bmatrix}$$

when n is a positive integer.

19. By taking determinants of both sides of the equation in Exercise 18, prove the identity given in Exercise 14. (Recall that the determinant of the matrix $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ is $ad - bc$.)

***20.** Give a recursive definition of the functions max and min so that $\max(a_1, a_2, \ldots, a_n)$ and $\min(a_1, a_2, \ldots, a_n)$ are the maximum and minimum of the n numbers $a_1, a_2, \ldots, a_n$, respectively.

***21.** Let $a_1, a_2, \ldots, a_n$, and $b_1, b_2, \ldots, b_n$ be real numbers. Use the recursive definitions that you gave in Exercise 20 to prove these.

a) $\max(-a_1, -a_2, \ldots, -a_n) = -\min(a_1, a_2, \ldots, a_n)$
b) $\max(a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n) \leq \max(a_1, a_2, \ldots, a_n) + \max(b_1, b_2, \ldots, b_n)$
c) $\min(a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n) \geq \min(a_1, a_2, \ldots, a_n) + \min(b_1, b_2, \ldots, b_n)$

22. Show that the set S defined by $1 \in S$ and $s + t \in S$ whenever $s \in S$ and $t \in S$ is the set of positive integers.

23. Give a recursive definition of the set of positive integers that are multiples of 5.

24. Give a recursive definition of

a) the set of odd positive integers.
b) the set of positive integer powers of 3.
c) the set of polynomials with integer coefficients.

25. Give a recursive definition of

a) the set of even integers.
b) the set of positive integers congruent to 2 modulo 3.
c) the set of positive integers not divisible by 5.

26. Let S be the subset of the set of ordered pairs of integers defined recursively by

Basis step: $(0, 0) \in S$.

Recursive step: If $(a, b) \in S$, then $(a + 2, b + 3) \in S$ and $(a + 3, b + 2) \in S$.

a) List the elements of S produced by the first five applications of the recursive definition.
b) Use strong induction on the number of applications of the recursive step of the definition to show that $5 \mid a + b$ when $(a, b) \in S$.
c) Use structural induction to show that $5 \mid a + b$ when $(a, b) \in S$.

27. Let S be the subset of the set of ordered pairs of integers defined recursively by

Basis step: $(0, 0) \in S$.

Recursive step: If $(a, b) \in S$, then $(a, b + 1) \in S$, $(a + 1, b + 1) \in S$, and $(a + 2, b + 1) \in S$.

a) List the elements of S produced by the first four applications of the recursive definition.
b) Use strong induction on the number of applications of the recursive step of the definition to show that $a \leq 2b$ whenever $(a, b) \in S$.
c) Use structural induction to show that $a \leq 2b$ whenever $(a, b) \in S$.

28. Give a recursive definition of each of these sets of ordered pairs of positive integers. [*Hint:* Plot the points in the set in the plane and look for lines containing points in the set.]

a) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, \text{and } a + b \text{ is odd}\}$
b) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, \text{and } a \mid b\}$
c) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, \text{and } 3 \mid a + b\}$

29. Give a recursive definition of each of these sets of ordered pairs of positive integers. Use structural induction to prove that the recursive definition you found is correct. [*Hint:* To find a recursive definition, plot the points in the set in the plane and look for patterns.]

a) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, \text{and } a + b \text{ is even}\}$
b) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, \text{and } a \text{ or } b \text{ is odd}\}$
c) $S = \{(a, b) \mid a \in \mathbf{Z}^+, b \in \mathbf{Z}^+, a + b \text{ is odd, and } 3 \mid b\}$

30. Prove that in a bit string, the string 01 occurs at most one more time than the string 10.

31. Define well-formed formulae of sets, variables representing sets, and operators from $\{\overline{\ }, \cup, \cap, -\}$.

32. a) Give a recursive definition of the function $ones(s)$, which counts the number of ones in a bit string s.

b) Use structural induction to prove that $ones(st) = ones(s) + ones(t)$.

33. a) Give a recursive definition of the function $m(s)$, which equals the smallest digit in a nonempty string of decimal digits.

b) Use structural induction to prove that $m(st) = \min(m(s), m(t))$.

The **reversal** of a string is the string consisting of the symbols of the string in reverse order. The reversal of the string w is denoted by w^R.

34. Find the reversal of the following bit strings.

a) 0101 **b)** 1 1011 **c)** 1000 1001 0111

35. Give a recursive definition of the reversal of a string. [*Hint:* First define the reversal of the empty string. Then write a string w of length $n + 1$ as xy, where x is a string of length n, and express the reversal of w in terms of x^R and y.]

***36.** Use structural induction to prove that $(w_1 w_2)^R = w_2^R w_1^R$.

37. Give a recursive definition of w^i, where w is a string and i is a nonnegative integer. (Here w^i represents the concatenation of i copies of the string w.)

***38.** Give a recursive definition of the set of bit strings that are palindromes.

39. When does a string belong to the set A of bit strings defined recursively by

$\lambda \in A$

$0x1 \in A$ if $x \in A$,

where λ is the empty string?

***40.** Recursively define the set of bit strings that have more zeros than ones.

41. Use Exercise 37 and mathematical induction to show that $l(w^i) = i \cdot l(w)$, where w is a string and i is a nonnegative integer.

***42.** Show that $(w^R)^i = (w^i)^R$ whenever w is a string and i is a nonnegative integer; that is, show that the ith power of the reversal of a string is the reversal of the ith power of the string.

43. Use structural induction to show that $n(T) \geq 2h(T) + 1$, where T is a full binary tree, $n(T)$ equals the number of vertices of T, and $h(T)$ is the height of T.

The set of leaves and the set of internal vertices of a full binary tree can be defined recursively.

Basis step: The root r is a leaf of the full binary tree with exactly one vertex r. This tree has no internal vertices.

Recursive step: The set of leaves of the tree $T = T_1 \cdot T_2$ is the union of the sets of leaves of T_1 and of T_2. The internal vertices of T are the root r of T and the union of the set of internal vertices of T_1 and the set of internal vertices of T_2.

44. Use structural induction to show that $l(T)$, the number of leaves of a full binary tree T, is 1 more than $i(T)$, the number of internal vertices of T.

45. Use generalized induction as was done in Example 13 to show that if $a_{m,n}$ is defined recursively by $a_{0,0} = 0$ and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 1 & \text{if } n = 0 \text{ and } m > 0 \\ a_{m,n-1} + 1 & \text{if } n > 0, \end{cases}$$

then $a_{m,n} = m + n$ for all $(m, n) \in \mathbf{N} \times \mathbf{N}$.

46. Use generalized induction as was done in Example 13 to show that if $a_{m,n}$ is defined recursively by $a_{1,1} = 5$ and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 2 & \text{if } n = 1 \text{ and } m > 1 \\ a_{m,n-1} + 2 & \text{if } n > 1, \end{cases}$$

then $a_{m,n} = 2(m + n) + 1$ for all $(m, n) \in \mathbf{Z}^+ \times \mathbf{Z}^+$.

***47.** A **partition** of a positive integer n is a way to write n as a sum of positive integers where the order of terms in the sum does not matter. For instance, $7 = 3 + 2 + 1 + 1$ is a partition of 7. Let P_m equal the number of different partitions of m, and let $P_{m,n}$ be the number of different ways to express m as the sum of positive integers not exceeding n.

a) Show that $P_{m,m} = P_m$.

b) Show that the following recursive definition for $P_{m,n}$ is correct:

$$P_{m,n} = \begin{cases} 1 & \text{if } m = 1 \\ 1 & \text{if } n = 1 \\ P_{m,m} & \text{if } m < n \\ 1 + P_{m,m-1} & \text{if } m = n > 1 \\ P_{m,n-1} + P_{m-n,n} & \text{if } m > n > 1. \end{cases}$$

c) Find the number of partitions of 5 and of 6 using this recursive definition.

Consider an inductive definition of a version of **Ackermann's function**. This function was named after Wilhelm Ackermann, a German mathematician who was a student of the great mathematician David Hilbert. Ackermann's function plays an important role in the theory of recursive functions and in the study of the complexity of certain algorithms involving set unions. (There are several different variants of this function. All are called Ackermann's function and have similar properties even though their values do not always agree.)

$$A(m, n) = \begin{cases} 2n & \text{if } m = 0 \\ 0 & \text{if } m \geq 1 \text{ and } n = 0 \\ 2 & \text{if } m \geq 1 \text{ and } n = 1 \\ A(m-1, A(m, n-1)) & \text{if } m \geq 1 \text{ and } n \geq 2 \end{cases}$$

Exercises 48–55 involve this version of Ackermann's function.

48. Find these values of Ackermann's function.

a) $A(1, 0)$ **b)** $A(0, 1)$

c) $A(1, 1)$ **d)** $A(2, 2)$

49. Show that $A(m, 2) = 4$ whenever $m \geq 1$.

50. Show that $A(1, n) = 2^n$ whenever $n \geq 1$.

51. Find these values of Ackermann's function.

a) $A(2, 3)$ ***b)** $A(3, 3)$

***52.** Find $A(3, 4)$.

****53.** Prove that $A(m, n+1) > A(m, n)$ whenever m and n are nonnegative integers.

***54.** Prove that $A(m+1, n) \geq A(m, n)$ whenever m and n are nonnegative integers.

55. Prove that $A(i, j) \geq j$ whenever i and j are nonnegative integers.

56. Use mathematical induction to prove that a function F defined by specifying $F(0)$ and a rule for obtaining $F(n+1)$ from $F(n)$ is well defined.

57. Use strong induction to prove that a function F defined by specifying $F(0)$ and a rule for obtaining $F(n+1)$ from the values $F(k)$ for $k = 0, 1, 2, \ldots, n$ is well defined.

58. Show that each of these proposed recursive definitions of a function on the set of positive integers does not produce a well-defined function.

a) $F(n) = 1 + F(\lfloor n/2 \rfloor)$ for $n \geq 1$ and $F(1) = 1$.

b) $F(n) = 1 + F(n-3)$ for $n \geq 2$, $F(1) = 2$, and $F(2) = 3$.

c) $F(n) = 1 + F(n/2)$ for $n \geq 2$, $F(1) = 1$, and $F(2) = 2$.

d) $F(n) = 1 + F(n/2)$ if n is even and $n \geq 2$, $F(n) = 1 - F(n-1)$ if n is odd, and $F(1) = 1$.

e) $F(n) = 1 + F(n/2)$ if n is even and $n \geq 2$, $F(n) = F(3n-1)$ if n is odd and $n \geq 3$, and $F(1) = 1$.

59. Show that each of these proposed recursive definitions of a function on the set of positive integers does not produce a well-defined function.

a) $F(n) = 1 + F(\lfloor (n+1)/2 \rfloor)$ for $n \geq 1$ and $F(1) = 1$.

b) $F(n) = 1 + F(n-2)$ for $n \geq 2$ and $F(1) = 0$.

c) $F(n) = 1 + F(n/3)$ for $n \geq 3$, $F(1) = 1$, $F(2) = 2$, and $F(3) = 3$.

d) $F(n) = 1 + F(n/2)$ if n is even and $n \geq 2$, $F(n) = 1 + F(n-2)$ if n is odd, and $F(1) = 1$.

e) $F(n) = 1 + F(F(n-1))$ if $n \geq 2$ and $F(1) = 2$.

Exercises 60–62 deal with iterations of the logarithm function. Let $\log n$ denote the logarithm of n to the base 2, as usual. The function $\log^{(k)} n$ is defined recursively by

$$\log^{(k)} n = \begin{cases} n & \text{if } k = 0 \\ \log(\log^{(k-1)} n) & \text{if } \log^{(k-1)} n \text{ is defined and positive} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The **iterated logarithm** is the function $\log^* n$ whose value at n is the smallest nonnegative integer k such that $\log^{(k)} n \leq 1$.

60. Find these values.

a) $\log^{(2)} 16$ **b)** $\log^{(3)} 256$

c) $\log^{(3)} 2^{65536}$ **d)** $\log^{(4)} 2^{2^{65536}}$

61. Find the value of $\log^* n$ for these values of n.

a) 2 **b)** 4 **c)** 8 **d)** 16

e) 256 **f)** 65536 **g)** 2^{2048}

62. Find the largest integer n such that $\log^* n = 5$. Determine the number of decimal digits in this number.

Exercises 63–65 deal with values of iterated functions. Suppose that $f(n)$ is a function from the set of real numbers, or positive real numbers, or some other set of real numbers, to the set of real numbers such that $f(n)$ is monotonically increasing [that is, $f(n) < f(m)$ when $n < m$) and $f(n) < n$ for all n in the domain of f.] The function $f^{(k)}(n)$ is defined recursively by

$$f^{(k)}(n) = \begin{cases} n & \text{if } k = 0 \\ f(f^{(k-1)}(n)) & \text{if } k > 0. \end{cases}$$

Furthermore, let c be a positive real number. The **iterated function** f_c^* is the number of iterations of f required to reduce its argument to c or less, so $f_c^*(n)$ is the smallest nonnegative integer k such that $f^k(n) \leq c$.

63. Let $f(n) = n - a$, where a is a positive integer. Find a formula for $f^{(k)}(n)$. What is the value of $f_0^*(n)$ when n is a positive integer?

64. Let $f(n) = n/2$. Find a formula for $f^{(k)}(n)$. What is the value of $f_1^*(n)$ when n is a positive integer?

65. Let $f(n) = \sqrt{n}$. Find a formula for $f^{(k)}(n)$. What is the value of $f_2^*(n)$ when n is a positive integer?

5.4 Recursive Algorithms

Introduction

Here's a famous humorous quote: "To understand recursion, you must first understand recursion."

Sometimes we can reduce the solution to a problem with a particular set of input values to the solution of the same problem with smaller input values. For instance, the problem of finding the greatest common divisor of two positive integers a and b, where $b > a$, can be reduced to finding the greatest common divisor of a pair of smaller integers, namely, $b \bmod a$ and a, because $\gcd(b \bmod a, a) = \gcd(a, b)$. When such a reduction can be done, the solution to the original problem can be found with a sequence of reductions, until the problem has been reduced to some initial case for which the solution is known. For instance, for finding the greatest common divisor, the reduction continues until the smaller of the two numbers is zero, because $\gcd(a, 0) = a$ when $a > 0$.

We will see that algorithms that successively reduce a problem to the same problem with smaller input are used to solve a wide variety of problems.

DEFINITION 1 An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.

Links We will describe a variety of different recursive algorithms in this section.

EXAMPLE 1 Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

Extra Examples *Solution:* We can build a recursive algorithm that finds $n!$, where n is a nonnegative integer, based on the recursive definition of $n!$, which specifies that $n! = n \cdot (n-1)!$ when n is a positive integer, and that $0! = 1$. To find $n!$ for a particular integer, we use the recursive step n times, each time replacing a value of the factorial function with the value of the factorial function at the next smaller integer. At this last step, we insert the value of $0!$. The recursive algorithm we obtain is displayed as Algorithm 1.

To help understand how this algorithm works, we trace the steps used by the algorithm to compute $4!$. First, we use the recursive step to write $4! = 4 \cdot 3!$. We then use the recursive step repeatedly to write $3! = 3 \cdot 2!$, $2! = 2 \cdot 1!$, and $1! = 1 \cdot 0!$. Inserting the value of $0! = 1$, and working back through the steps, we see that $1! = 1 \cdot 1 = 1$, $2! = 2 \cdot 1! = 2$, $3! = 3 \cdot 2! = 3 \cdot 2 = 6$, and $4! = 4 \cdot 3! = 4 \cdot 6 = 24$. ◀

ALGORITHM 1 A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n-1)$
{output is $n!$}

Example 2 shows how a recursive algorithm can be constructed to evaluate a function from its recursive definition.

EXAMPLE 2 Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

Solution: We can base a recursive algorithm on the recursive definition of a^n. This definition states that $a^{n+1} = a \cdot a^n$ for $n > 0$ and the initial condition $a^0 = 1$. To find a^n, successively use the recursive step to reduce the exponent until it becomes zero. We give this procedure in Algorithm 2. ◀

ALGORITHM 2 A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n-1)$
{output is a^n}

Next we give a recursive algorithm for finding greatest common divisors.

EXAMPLE 3 Give a recursive algorithm for computing the greatest common divisor of two nonnegative integers a and b with $a < b$.

Solution: We can base a recursive algorithm on the reduction $\gcd(a, b) = \gcd(b \bmod a, a)$ and the condition $\gcd(0, b) = b$ when $b > 0$. This produces the procedure in Algorithm 3, which is a recursive version of the Euclidean algorithm.

We illustrate the workings of Algorithm 3 with a trace when the input is $a = 5, b = 8$. With this input, the algorithm uses the "else" clause to find that $\gcd(5, 8) = \gcd(8 \bmod 5, 5) = \gcd(3, 5)$. It uses this clause again to find that $\gcd(3, 5) = \gcd(5 \bmod 3, 3) = \gcd(2, 3)$, then to get $\gcd(2, 3) = \gcd(3 \bmod 2, 2) = \gcd(1, 2)$, then to get $\gcd(1, 2) = \gcd(2 \bmod 1, 1) = \gcd(0, 1)$. Finally, to find $\gcd(0, 1)$ it uses the first step with $a = 0$ to find that $\gcd(0, 1) = 1$. Consequently, the algorithm finds that $\gcd(5, 8) = 1$. ◀

ALGORITHM 3 A Recursive Algorithm for Computing $\gcd(a, b)$.

procedure $gcd(a, b$: nonnegative integers with $a < b)$
if $a = 0$ **then return** b
else return $gcd(b \bmod a, a)$
{output is $\gcd(a, b)$}

EXAMPLE 4 Devise a recursive algorithm for computing $b^n \bmod m$, where b, n, and m are integers with $m \geq 2$, $n \geq 0$, and $1 \leq b < m$.

Solution: We can base a recursive algorithm on the fact that

$$b^n \bmod m = (b \cdot (b^{n-1} \bmod m)) \bmod m,$$

which follows by Corollary 2 in Section 4.1, and the initial condition $b^0 \bmod m = 1$. We leave this as Exercise 12 for the reader.

However, we can devise a much more efficient recursive algorithm based on the observation that

$$b^n \bmod m = (b^{n/2} \bmod m)^2 \bmod m$$

when n is even and

$$b^n \bmod m = \left((b^{\lfloor n/2 \rfloor} \bmod m)^2 \bmod m \cdot b \bmod m\right) \bmod m$$

when n is odd, which we describe in pseudocode as Algorithm 4.

We trace the execution of Algorithm 4 with input $b = 2, n = 5$, and $m = 3$ to illustrate how it works. First, because $n = 5$ is odd we use the "else" clause to see that $mpower(2, 5, 3) = (mpower(2, 2, 3)^2 \bmod 3 \cdot 2 \bmod 3) \bmod 3$. We next use the "else if" clause to see that $mpower(2, 2, 3) = mpower(2, 1, 3)^2 \bmod 3$. Using the "else" clause again, we see that $mpower(2, 1, 3) = (mpower(2, 0, 3)^2 \bmod 3 \cdot 2 \bmod 3) \bmod 3$. Finally, using the "if" clause, we see that $mpower(2, 0, 3) = 1$. Working backwards, it follows that $mpower(2, 1, 3) = (1^2 \bmod 3 \cdot 2 \bmod 3) \bmod 3 = 2$, so $mpower(2, 2, 3) = 2^2 \bmod 3 = 1$, and finally $mpower(2, 5, 3) = (1^2 \bmod 3 \cdot 2 \bmod 3) \bmod 3 = 2$. ◀

ALGORITHM 4 Recursive Modular Exponentiation.

procedure *mpower*$(b, n, m$: integers with $b > 0$ and $m \geq 2, n \geq 0)$
if $n = 0$ **then**
    return 1
else if n is even **then**
    return $mpower(b, n/2, m)^2$ **mod** m
else
    return $(mpower(b, \lfloor n/2 \rfloor, m)^2$ **mod** $m \cdot b$ **mod** $m)$ **mod** m
{output is b^n **mod** m}

We will now give recursive versions of searching algorithms that were introduced in Section 3.1.

EXAMPLE 5 Express the linear search algorithm as a recursive procedure.

Solution: To *search* for the first occurrence of x in the sequence $a_1, a_2, \ldots, a_n$, at the ith step of the algorithm, x and a_i are compared. If x equals a_i, then the algorithm returns i, the location of x in the sequence. Otherwise, the search for the first occurrence of x is reduced to a search in a sequence with one fewer element, namely, the sequence $a_{i+1}, \ldots, a_n$. The algorithm returns 0 when x is never found in the sequence after all terms have been examined. We can now give a recursive procedure, which is displayed as pseudocode in Algorithm 5.

Let *search* (i, j, x) be the procedure that searches for the first occurrence of x in the sequence $a_i, a_{i+1}, \ldots, a_j$. The input to the procedure consists of the triple $(1, n, x)$. The algorithm terminates at a step if the first term of the remaining sequence is x or if there is only one term of the sequence and this is not x. If x is not the first term and there are additional terms, the same procedure is carried out but with a search sequence of one fewer term, obtained by deleting the first term of the search sequence. If the algorithm terminates without x having been found, the algorithm returns the value 0. ◀

ALGORITHM 5 A Recursive Linear Search Algorithm.

procedure *search*$(i, j, x$: i, j, x integers, $1 \leq i \leq j \leq n)$
if $a_i = x$ **then**
    return i
else if $i = j$ **then**
    return 0
else
    return $search(i + 1, j, x)$
{output is the location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

EXAMPLE 6 Construct a recursive version of a binary search algorithm.

Solution: Suppose we want to locate x in the sequence $a_1, a_2, \ldots, a_n$ of integers in increasing order. To perform a binary search, we begin by comparing x with the middle term, $a_{\lfloor (n+1)/2 \rfloor}$. Our algorithm will terminate if x equals this term and return the location of this term in the sequence. Otherwise, we reduce the search to a smaller search sequence, namely, the first half of the sequence if x is smaller than the middle term of the original sequence, and the second half otherwise. We have reduced the solution of the search problem to the solution of the same

problem with a sequence at most half as long. If have we never encountered the search term x, our algorithm returns the value 0. We express this recursive version of a binary search algorithm as Algorithm 6. ◀

ALGORITHM 6 A Recursive Binary Search Algorithm.

procedure *binary search*$(i, j, x: i, j, x$ integers, $1 \leq i \leq j \leq n)$
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if $(x < a_m$ and $i < m)$ **then**
    return *binary search*$(i, m - 1, x)$
else if $(x > a_m$ and $j > m)$ **then**
    return *binary search*$(m + 1, j, x)$
else return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

Proving Recursive Algorithms Correct

Mathematical induction, and its variant strong induction, can be used to prove that a recursive algorithm is correct, that is, that it produces the desired output for all possible input values. Examples 7 and 8 illustrate how mathematical induction or strong induction can be used to prove that recursive algorithms are correct. First, we will show that Algorithm 2 is correct.

EXAMPLE 7 Prove that Algorithm 2, which computes powers of real numbers, is correct.

Solution: We use mathematical induction on the exponent n.

BASIS STEP: If $n = 0$, the first step of the algorithm tells us that $power\,(a, 0) = 1$. This is correct because $a^0 = 1$ for every nonzero real number a. This completes the basis step.

INDUCTIVE STEP: The inductive hypothesis is the statement that $power\,(a, k) = a^k$ for all $a \neq 0$ for an arbitrary nonnegative integer k. That is, the inductive hypothesis is the statement that the algorithm correctly computes a^k. To complete the inductive step, we show that if the inductive hypothesis is true, then the algorithm correctly computes a^{k+1}. Because $k + 1$ is a positive integer, when the algorithm computes a^{k+1}, the algorithm sets $power\,(a,\ k + 1) = a \cdot power\,(a,\ k)$. By the inductive hypothesis, we have $power\,(a,\ k) = a^k$, so $power\,(a,\ k + 1) = a \cdot power\,(a,\ k) = a \cdot a^k = a^{k+1}$. This completes the inductive step.

We have completed the basis step and the inductive step, so we can conclude that Algorithm 2 always computes a^n correctly when $a \neq 0$ and n is a nonnegative integer. ◀

Generally, we need to use strong induction to prove that recursive algorithms are correct, rather than just mathematical induction. Example 8 illustrates this; it shows how strong induction can be used to prove that Algorithm 4 is correct.

EXAMPLE 8 Prove that Algorithm 4, which computes modular powers, is correct.

Solution: We use strong induction on the exponent n.

BASIS STEP: Let b be an integer and m an integer with $m \geq 2$. When $n = 0$, the algorithm sets $mpower(b, n, m)$ equal to 1. This is correct because b^0 **mod** $m = 1$. The basis step is complete.

INDUCTIVE STEP: For the inductive hypothesis we assume that $mpower(b, j, m) = b^j \bmod m$ for all integers $0 \leq j < k$ whenever b is a positive integer and m is an integer with $m \geq 2$. To complete the inductive step, we show that if the inductive hypothesis is correct, then $mpower(b, k, m) = b^k \bmod m$. Because the recursive algorithm handles odd and even values of k differently, we split the inductive step into two cases.

When k is even, we have

$$mpower(b, k, m) = (mpower(b, k/2, m))^2 \bmod m = (b^{k/2} \bmod m)^2 \bmod m = b^k \bmod m,$$

where we have used the inductive hypothesis to replace $mpower(b, k/2, m)$ by $b^{k/2} \bmod m$.

When k is odd, we have

$$\begin{aligned} mpower(b, k, m) &= ((mpower(b, \lfloor k/2 \rfloor, m))^2 \bmod m \cdot b \bmod m) \bmod m \\ &= ((b^{\lfloor k/2 \rfloor} \bmod m)^2 \bmod m \cdot b \bmod m) \bmod m \\ &= b^{2\lfloor k/2 \rfloor + 1} \bmod m = b^k \bmod m, \end{aligned}$$

using Corollary 2 in Section 4.1, because $2\lfloor k/2 \rfloor + 1 = 2(k-1)/2 + 1 = k$ when k is odd. Here we have used the inductive hypothesis to replace $mpower(b, \lfloor k/2 \rfloor, m)$ by $b^{\lfloor k/2 \rfloor} \bmod m$. This completes the inductive step.

We have completed the basis step and the inductive step, so by strong induction we know that Algorithm 4 is correct. ◀

Recursion and Iteration

A recursive definition expresses the value of a function at a positive integer in terms of the values of the function at smaller integers. This means that we can devise a recursive algorithm to evaluate a recursively defined function at a positive integer. Instead of successively reducing the computation to the evaluation of the function at smaller integers, we can start with the value of the function at one or more integers, the base cases, and successively apply the recursive definition to find the values of the function at successive larger integers. Such a procedure is called **iterative**. Often an iterative approach for the evaluation of a recursively defined sequence requires much less computation than a procedure using recursion (unless special-purpose recursive machines are used). This is illustrated by the iterative and recursive procedures for finding the nth Fibonacci number. The recursive procedure is given first.

ALGORITHM 7 A Recursive Algorithm for Fibonacci Numbers.

procedure *fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else if $n = 1$ **then return** 1
else return $fibonacci(n-1) + fibonacci(n-2)$
{output is $fibonacci(n)$}

When we use a recursive procedure to find f_n, we first express f_n as $f_{n-1} + f_{n-2}$. Then we replace both of these Fibonacci numbers by the sum of two previous Fibonacci numbers, and so on. When f_1 or f_0 arises, it is replaced by its value.

Note that at each stage of the recursion, until f_1 or f_0 is obtained, the number of Fibonacci numbers to be evaluated has doubled. For instance, when we find f_4 using this recursive algorithm, we must carry out all the computations illustrated in the tree diagram in Figure 1. This

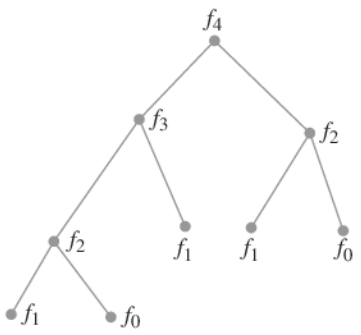


FIGURE 1 **Evaluating f_4 Recursively.**

tree consists of a root labeled with f_4, and branches from the root to vertices labeled with the two Fibonacci numbers f_3 and f_2 that occur in the reduction of the computation of f_4. Each subsequent reduction produces two branches in the tree. This branching ends when f_0 and f_1 are reached. The reader can verify that this algorithm requires $f_{n+1} - 1$ additions to find f_n.

Now consider the amount of computation required to find f_n using the iterative approach in Algorithm 8.

ALGORITHM 8 **An Iterative Algorithm for Computing Fibonacci Numbers.**

procedure *iterative fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else
    $x := 0$
    $y := 1$
    for $i := 1$ to $n - 1$
        $z := x + y$
        $x := y$
        $y := z$
    return y
{output is the nth Fibonacci number}

This procedure initializes x as $f_0 = 0$ and y as $f_1 = 1$. When the loop is traversed, the sum of x and y is assigned to the auxiliary variable z. Then x is assigned the value of y and y is assigned the value of the auxiliary variable z. Therefore, after going through the loop the first time, it follows that x equals f_1 and y equals $f_0 + f_1 = f_2$. Furthermore, after going through the loop $n - 1$ times, x equals f_{n-1} and y equals f_n (the reader should verify this statement). Only $n - 1$ additions have been used to find f_n with this iterative approach when $n > 1$. Consequently, this algorithm requires far less computation than does the recursive algorithm.

We have shown that a recursive algorithm may require far more computation than an iterative one when a recursively defined function is evaluated. It is sometimes preferable to use a recursive procedure even if it is less efficient than the iterative procedure. In particular, this is true when the recursive approach is easily implemented and the iterative approach is not. (Also, machines designed to handle recursion may be available that eliminate the advantage of using iteration.)

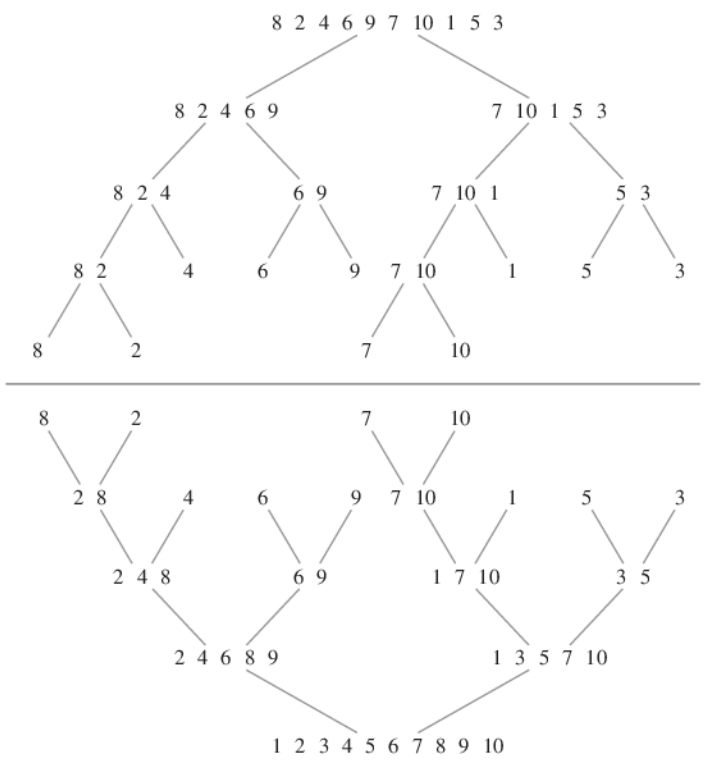


FIGURE 2 **The Merge Sort of 8, 2, 4, 6, 9, 7, 10, 1, 5, 3.**

The Merge Sort

Links

We now describe a recursive sorting algorithm called the **merge sort** algorithm. We will demonstrate how the merge sort algorithm works with an example before describing it in generality.

EXAMPLE 9 Use the merge sort to put the terms of the list 8, 2, 4, 6, 9, 7, 10, 1, 5, 3 in increasing order.

Solution: A merge sort begins by splitting the list into individual elements by successively splitting lists in two. The progression of sublists for this example is represented with the balanced binary tree of height 4 shown in the upper half of Figure 2.

Sorting is done by successively merging pairs of lists. At the first stage, pairs of individual elements are merged into lists of length two in increasing order. Then successive merges of pairs of lists are performed until the entire list is put into increasing order. The succession of merged lists in increasing order is represented by the balanced binary tree of height 4 shown in the lower half of Figure 2 (note that this tree is displayed "upside down"). ◀

In general, a merge sort proceeds by iteratively splitting lists into two sublists of equal length (or where one sublist has one more element than the other) until each sublist contains one element. This succession of sublists can be represented by a balanced binary tree. The procedure continues by successively merging pairs of lists, where both lists are in increasing order, into a larger list with elements in increasing order, until the original list is put into increasing order. The succession of merged lists can be represented by a balanced binary tree.

Demo

We can also describe the merge sort recursively. To do a merge sort, we split a list into two sublists of equal, or approximately equal, size, sorting each sublist using the merge sort

algorithm, and then merging the two lists. The recursive version of the merge sort is given in Algorithm 9. This algorithm uses the subroutine *merge*, which is described in Algorithm 10.

ALGORITHM 9 A Recursive Merge Sort.

procedure *mergesort*($L = a_1, \ldots, a_n$)
if $n > 1$ **then**
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, \ldots, a_m$
    $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
    $L :=$ *merge*(*mergesort*(L_1), *mergesort*(L_2))
{L is now sorted into elements in nondecreasing order}

An efficient algorithm for merging two ordered lists into a larger ordered list is needed to implement the merge sort. We will now describe such a procedure.

EXAMPLE 10 Merge the two lists 2, 3, 5, 6 and 1, 4.

Solution: Table 1 illustrates the steps we use. First, compare the smallest elements in the two lists, 2 and 1, respectively. Because 1 is the smaller, put it at the beginning of the merged list and remove it from the second list. At this stage, the first list is 2, 3, 5, 6, the second is 4, and the combined list is 1.

Next, compare 2 and 4, the smallest elements of the two lists. Because 2 is the smaller, add it to the combined list and remove it from the first list. At this stage the first list is 3, 5, 6, the second is 4, and the combined list is 1, 2.

Continue by comparing 3 and 4, the smallest elements of their respective lists. Because 3 is the smaller of these two elements, add it to the combined list and remove it from the first list. At this stage the first list is 5, 6, and the second is 4. The combined list is 1, 2, 3.

Then compare 5 and 4, the smallest elements in the two lists. Because 4 is the smaller of these two elements, add it to the combined list and remove it from the second list. At this stage the first list is 5, 6, the second list is empty, and the combined list is 1, 2, 3, 4.

Finally, because the second list is empty, all elements of the first list can be appended to the end of the combined list in the order they occur in the first list. This produces the ordered list 1, 2, 3, 4, 5, 6. ◀

We will now consider the general problem of merging two ordered lists L_1 and L_2 into an ordered list L. We will describe an algorithm for solving this problem. Start with an empty list L. Compare the smallest elements of the two lists. Put the smaller of these two elements at the right end of L, and remove it from the list it was in. Next, if one of L_1 and L_2 is empty, append the other (nonempty) list to L, which completes the merging. If neither L_1 nor L_2 is empty, repeat this process. Algorithm 10 gives a pseudocode description of this procedure.

TABLE 1 Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4.

First List	*Second List*	*Merged List*	*Comparison*
2 3 5 6	1 4		1 < 2
2 3 5 6	4	1	2 < 4
3 5 6	4	1 2	3 < 4
5 6	4	1 2 3	4 < 5
5 6		1 2 3 4	
		1 2 3 4 5 6	

We will need estimates for the number of comparisons used to merge two ordered lists in the analysis of the merge sort. We can easily obtain such an estimate for Algorithm 10. Each time a comparison of an element from L_1 and an element from L_2 is made, an additional element is added to the merged list L. However, when either L_1 or L_2 is empty, no more comparisons are needed. Hence, Algorithm 10 is least efficient when $m + n - 2$ comparisons are carried out, where m and n are the number of elements in L_1 and L_2, respectively, leaving one element in each of L_1 and L_2. The next comparison will be the last one needed, because it will make one of these lists empty. Hence, Algorithm 10 uses no more than $m + n - 1$ comparisons. Lemma 1 summarizes this estimate.

ALGORITHM 10 **Merging Two Lists.**

procedure *merge*(L_1, L_2: sorted lists)
L := empty list
while L_1 and L_2 are both nonempty
  remove smaller of first elements of L_1 and L_2 from its list; put it at the right end of L
  if this removal makes one list empty **then** remove all elements from the other list and append them to L
return L {L is the merged list with elements in increasing order}

LEMMA 1 Two sorted lists with m elements and n elements can be merged into a sorted list using no more than $m + n - 1$ comparisons.

Sometimes two sorted lists of length m and n can be merged using far fewer than $m + n - 1$ comparisons. For instance, when $m = 1$, a binary search procedure can be applied to put the one element in the first list into the second list. This requires only $\lceil \log n \rceil$ comparisons, which is much smaller than $m + n - 1 = n$, for $m = 1$. On the other hand, for some values of m and n, Lemma 1 gives the best possible bound. That is, there are lists with m and n elements that cannot be merged using fewer than $m + n - 1$ comparisons. (See Exercise 47.)

We can now analyze the complexity of the merge sort. Instead of studying the general problem, we will assume that n, the number of elements in the list, is a power of 2, say 2^m. This will make the analysis less complicated, but when this is not the case, various modifications can be applied that will yield the same estimate.

At the first stage of the splitting procedure, the list is split into two sublists, of 2^{m-1} elements each, at level 1 of the tree generated by the splitting. This process continues, splitting the two sublists with 2^{m-1} elements into four sublists of 2^{m-2} elements each at level 2, and so on. In general, there are 2^{k-1} lists at level $k - 1$, each with 2^{m-k+1} elements. These lists at level $k - 1$ are split into 2^k lists at level k, each with 2^{m-k} elements. At the end of this process, we have 2^m lists each with one element at level m.

We start merging by combining pairs of the 2^m lists of one element into 2^{m-1} lists, at level $m - 1$, each with two elements. To do this, 2^{m-1} pairs of lists with one element each are merged. The merger of each pair requires exactly one comparison.

The procedure continues, so that at level k ($k = m, m - 1, m - 2, \ldots, 3, 2, 1$), 2^k lists each with 2^{m-k} elements are merged into 2^{k-1} lists, each with 2^{m-k+1} elements, at level $k - 1$. To do this a total of 2^{k-1} mergers of two lists, each with 2^{m-k} elements, are needed. But,

by Lemma 1, each of these mergers can be carried out using at most $2^{m-k} + 2^{m-k} - 1 = 2^{m-k+1} - 1$ comparisons. Hence, going from level k to $k-1$ can be accomplished using at most $2^{k-1}(2^{m-k+1} - 1)$ comparisons.

Summing all these estimates shows that the number of comparisons required for the merge sort is at most

$$\sum_{k=1}^{m} 2^{k-1}(2^{m-k+1} - 1) = \sum_{k=1}^{m} 2^m - \sum_{k=1}^{m} 2^{k-1} = m2^m - (2^m - 1) = n \log n - n + 1,$$

because $m = \log n$ and $n = 2^m$. (We evaluated $\sum_{k=1}^{m} 2^m$ by noting that it is the sum of m identical terms, each equal to 2^m. We evaluated $\sum_{k=1}^{m} 2^{k-1}$ using the formula for the sum of the terms of a geometric progression from Theorem 1 of Section 2.4.)

Theorem 1 summarizes what we have discovered about the worst-case complexity of the merge sort algorithm.

THEOREM 1 The number of comparisons needed to merge sort a list with n elements is $O(n \log n)$.

In Chapter 11 we will show that the fastest comparison-based sorting algorithm have $O(n \log n)$ time complexity. (A comparison-based sorting algorithm has the comparison of two elements as its basic operation.) Theorem 1 tells us that the merge sort achieves this best possible big-O estimate for the complexity of a sorting algorithm. We describe another efficient algorithm, the quick sort, in the preamble to Exercise 50.

Exercises

1. Trace Algorithm 1 when it is given $n = 5$ as input. That is, show all steps used by Algorithm 1 to find 5!, as is done in Example 1 to find 4!.
2. Trace Algorithm 1 when it is given $n = 6$ as input. That is, show all steps used by Algorithm 1 to find 6!, as is done in Example 1 to find 4!.
3. Trace Algorithm 3 when it finds gcd(8, 13). That is, show all the steps used by Algorithm 3 to find gcd(8, 13).
4. Trace Algorithm 3 when it finds gcd(12, 17). That is, show all the steps used by Algorithm 3 to find gcd(12, 17).
5. Trace Algorithm 4 when it is given $m = 5$, $n = 11$, and $b = 3$ as input. That is, show all the steps Algorithm 4 uses to find $3^{11} \bmod 5$.
6. Trace Algorithm 4 when it is given $m = 7$, $n = 10$, and $b = 2$ as input. That is, show all the steps Algorithm 4 uses to find $2^{10} \bmod 7$.
7. Give a recursive algorithm for computing nx whenever n is a positive integer and x is an integer, using just addition.
8. Give a recursive algorithm for finding the sum of the first n positive integers.
9. Give a recursive algorithm for finding the sum of the first n odd positive integers.
10. Give a recursive algorithm for finding the maximum of a finite set of integers, making use of the fact that the maximum of n integers is the larger of the last integer in the list and the maximum of the first $n-1$ integers in the list.
11. Give a recursive algorithm for finding the minimum of a finite set of integers, making use of the fact that the minimum of n integers is the smaller of the last integer in the list and the minimum of the first $n-1$ integers in the list.
12. Devise a recursive algorithm for finding $x^n \bmod m$ whenever n, x, and m are positive integers based on the fact that $x^n \bmod m = (x^{n-1} \bmod m \cdot x \bmod m) \bmod m$.
13. Give a recursive algorithm for finding $n! \bmod m$ whenever n and m are positive integers.
14. Give a recursive algorithm for finding a **mode** of a list of integers. (A **mode** is an element in the list that occurs at least as often as every other element.)
15. Devise a recursive algorithm for computing the greatest common divisor of two nonnegative integers a and b with $a < b$ using the fact that $\gcd(a, b) = \gcd(a, b - a)$.
16. Prove that the recursive algorithm for finding the sum of the first n positive integers you found in Exercise 8 is correct.

17. Describe a recursive algorithm for multiplying two nonnegative integers x and y based on the fact that $xy = 2(x \cdot (y/2))$ when y is even and $xy = 2(x \cdot \lfloor y/2 \rfloor) + x$ when y is odd, together with the initial condition $xy = 0$ when $y = 0$.

18. Prove that Algorithm 1 for computing $n!$ when n is a nonnegative integer is correct.

19. Prove that Algorithm 3 for computing $\gcd(a, b)$ when a and b are positive integers with $a < b$ is correct.

20. Prove that the algorithm you devised in Exercise 17 is correct.

21. Prove that the recursive algorithm that you found in Exercise 7 is correct.

22. Prove that the recursive algorithm that you found in Exercise 10 is correct.

23. Devise a recursive algorithm for computing n^2 where n is a nonnegative integer, using the fact that $(n+1)^2 = n^2 + 2n + 1$. Then prove that this algorithm is correct.

24. Devise a recursive algorithm to find a^{2^n}, where a is a real number and n is a positive integer. [*Hint:* Use the equality $a^{2^{n+1}} = (a^{2^n})^2$.]

25. How does the number of multiplications used by the algorithm in Exercise 24 compare to the number of multiplications used by Algorithm 2 to evaluate a^{2^n}?

***26.** Use the algorithm in Exercise 24 to devise an algorithm for evaluating a^n when n is a nonnegative integer. [*Hint:* Use the binary expansion of n.]

***27.** How does the number of multiplications used by the algorithm in Exercise 26 compare to the number of multiplications used by Algorithm 2 to evaluate a^n?

28. How many additions are used by the recursive and iterative algorithms given in Algorithms 7 and 8, respectively, to find the Fibonacci number f_7?

29. Devise a recursive algorithm to find the nth term of the sequence defined by $a_0 = 1, a_1 = 2$, and $a_n = a_{n-1} \cdot a_{n-2}$, for $n = 2, 3, 4, \ldots$.

30. Devise an iterative algorithm to find the nth term of the sequence defined in Exercise 29.

31. Is the recursive or the iterative algorithm for finding the sequence in Exercise 29 more efficient?

32. Devise a recursive algorithm to find the nth term of the sequence defined by $a_0 = 1$, $a_1 = 2$, $a_2 = 3$, and $a_n = a_{n-1} + a_{n-2} + a_{n-3}$, for $n = 3, 4, 5, \ldots$.

33. Devise an iterative algorithm to find the nth term of the sequence defined in Exercise 32.

34. Is the recursive or the iterative algorithm for finding the sequence in Exercise 32 more efficient?

35. Give iterative and recursive algorithms for finding the nth term of the sequence defined by $a_0 = 1, a_1 = 3, a_2 = 5$, and $a_n = a_{n-1} \cdot a_{n-2}^2 \cdot a_{n-3}^3$. Which is more efficient?

36. Give a recursive algorithm to find the number of partitions of a positive integer based on the recursive definition given in Exercise 47 in Section 5.3.

37. Give a recursive algorithm for finding the reversal of a bit string. (See the definition of the reversal of a bit string in the preamble of Exercise 34 in Section 5.3.)

38. Give a recursive algorithm for finding the string w^i, the concatenation of i copies of w, when w is a bit string.

39. Prove that the recursive algorithm for finding the reversal of a bit string that you gave in Exercise 37 is correct.

40. Prove that the recursive algorithm for finding the concatenation of i copies of a bit string that you gave in Exercise 38 is correct.

***41.** Give a recursive algorithm for tiling a $2^n \times 2^n$ checkerboard with one square missing using right trominoes.

42. Give a recursive algorithm for triangulating a simple polygon with n sides, using Lemma 1 in Section 5.2.

43. Give a recursive algorithm for computing values of the Ackermann function. [*Hint:* See the preamble to Exercise 48 in Section 5.3.]

44. Use a merge sort to sort 4, 3, 2, 5, 1, 8, 7, 6 into increasing order. Show all the steps used by the algorithm.

45. Use a merge sort to sort *b, d, a, f, g, h, z, p, o, k* into alphabetic order. Show all the steps used by the algorithm.

46. How many comparisons are required to merge these pairs of lists using Algorithm 10?

a) 1, 3, 5, 7, 9; 2, 4, 6, 8, 10
b) 1, 2, 3, 4, 5; 6, 7, 8, 9, 10
c) 1, 5, 6, 7, 8; 2, 3, 4, 9, 10

47. Show that for all positive integers m and n there are sorted lists with m elements and n elements, respectively, such that Algorithm 10 uses $m + n - 1$ comparisons to merge them into one sorted list.

***48.** What is the least number of comparisons needed to merge any two lists in increasing order into one list in increasing order when the number of elements in the two lists are

a) 1, 4? **b)** 2, 4? **c)** 3, 4? **d)** 4, 4?

***49.** Prove that the merge sort algorithm is correct.

The **quick sort** is an efficient algorithm. To sort $a_1, a_2, \ldots, a_n$, this algorithm begins by taking the first element a_1 and forming two sublists, the first containing those elements that are less than a_1, in the order they arise, and the second containing those elements greater than a_1, in the order they arise. Then a_1 is put at the end of the first sublist. This procedure is repeated recursively for each sublist, until all sublists contain one item. The ordered list of n items is obtained by combining the sublists of one item in the order they occur.

50. Sort 3, 5, 7, 8, 1, 9, 2, 4, 6 using the quick sort.

51. Let $a_1, a_2, \ldots, a_n$ be a list of n distinct real numbers. How many comparisons are needed to form two sublists from this list, the first containing elements less than a_1 and the second containing elements greater than a_1?

52. Describe the quick sort algorithm using pseudocode.

53. What is the largest number of comparisons needed to order a list of four elements using the quick sort algorithm?

54. What is the least number of comparisons needed to order a list of four elements using the quick sort algorithm?

55. Determine the worst-case complexity of the quick sort algorithm in terms of the number of comparisons used.

5.5 Program Correctness

Introduction

Suppose that we have designed an algorithm to solve a problem and have written a program to implement it. How can we be sure that the program always produces the correct answer? After all the bugs have been removed so that the syntax is correct, we can test the program with sample input. It is not correct if an incorrect result is produced for any sample input. But even if the program gives the correct answer for all sample input, it may not always produce the correct answer (unless all possible input has been tested). We need a proof to show that the program *always* gives the correct output.

Program verification, the proof of correctness of programs, uses the rules of inference and proof techniques described in this chapter, including mathematical induction. Because an incorrect program can lead to disastrous results, a large amount of methodology has been constructed for verifying programs. Efforts have been devoted to automating program verification so that it can be carried out using a computer. However, only limited progress has been made toward this goal. Indeed, some mathematicians and theoretical computer scientists argue that it will never be realistic to mechanize the proof of correctness of complex programs.

Some of the concepts and methods used to prove that programs are correct will be introduced in this section. Many different methods have been devised for proving that programs are correct. We will discuss a widely used method for program verification introduced by Tony Hoare in this section; several other methods are also commonly used. Furthermore, we will not develop a complete methodology for program verification in this book. This section is meant to be a brief introduction to the area of program verification, which ties together the rules of logic, proof techniques, and the concept of an algorithm.

Program Verification

A program is said to be **correct** if it produces the correct output for every possible input. A proof that a program is correct consists of two parts. The first part shows that the correct answer is obtained if the program terminates. This part of the proof establishes the **partial correctness** of the program. The second part of the proof shows that the program always terminates.

To specify what it means for a program to produce the correct output, two propositions are used. The first is the **initial assertion**, which gives the properties that the input values must have. The second is the **final assertion**, which gives the properties that the output of the program should have, if the program did what was intended. The appropriate initial and final assertions must be provided when a program is checked.

DEFINITION 1

A program, or program segment, S is said to be *partially correct with respect to* the initial assertion p and the final assertion q if whenever p is true for the input values of S and S terminates, then q is true for the output values of S. The notation $p\{S\}q$ indicates that the program, or program segment, S is partially correct with respect to the initial assertion p and the final assertion q.

Note: The notation $p\{S\}q$ is known as a *Hoare triple*. Tony Hoare introduced the concept of partial correctness.

Links

Note that the notion of partial correctness has nothing to do with whether a program terminates; it focuses only on whether the program does what it is expected to do if it terminates.

A simple example illustrates the concepts of initial and final assertions.

EXAMPLE 1 Show that the program segment

$$y := 2$$
$$z := x + y$$

Extra Examples

is correct with respect to the initial assertion p: $x = 1$ and the final assertion q: $z = 3$.

Solution: Suppose that p is true, so that $x = 1$ as the program begins. Then y is assigned the value 2, and z is assigned the sum of the values of x and y, which is 3. Hence, S is correct with respect to the initial assertion p and the final assertion q. Thus, $p\{S\}q$ is true. ◀

Rules of Inference

A useful rule of inference proves that a program is correct by splitting the program into a sequence of subprograms and then showing that each subprogram is correct.

Suppose that the program S is split into subprograms S_1 and S_2. Write $S = S_1; S_2$ to indicate that S is made up of S_1 followed by S_2. Suppose that the correctness of S_1 with respect to the initial assertion p and final assertion q, and the correctness of S_2 with respect to the initial assertion q and the final assertion r, have been established. It follows that if p is true and S_1 is executed and terminates, then q is true; and if q is true, and S_2 executes and terminates, then r is true. Thus, if p is true and $S = S_1; S_2$ is executed and terminates, then r is true. This rule of inference, called the **composition rule**, can be stated as

$$\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \\ \hline \therefore\ p\{S_1; S_2\}r. \end{array}$$

This rule of inference will be used later in this section.

Next, some rules of inference for program segments involving conditional statements and loops will be given. Because programs can be split into segments for proofs of correctness, this will let us verify many different programs.

Conditional Statements

First, rules of inference for conditional statements will be given. Suppose that a program segment has the form

if *condition* **then**
  S

where S is a block of statements. Then S is executed if *condition* is true, and it is not executed when *condition* is false. To verify that this segment is correct with respect to the initial assertion p and final assertion q, two things must be done. First, it must be shown that when p is true and *condition* is also true, then q is true after S terminates. Second, it must be shown that when p is true and *condition* is false, then q is true (because in this case S does not execute).

This leads to the following rule of inference:

$$\begin{array}{l} (p \wedge condition)\{S\}q \\ (p \wedge \neg condition) \rightarrow q \\ \hline \therefore\ p\{\textbf{if } condition \textbf{ then } S\}q. \end{array}$$

Example 2 illustrates how this rule of inference is used.

EXAMPLE 2 Verify that the program segment

if $x > y$ **then**
$y := x$

is correct with respect to the initial assertion **T** and the final assertion $y \geq x$.

Solution: When the initial assertion is true and $x > y$, the assignment $y := x$ is carried out. Hence, the final assertion, which asserts that $y \geq x$, is true in this case. Moreover, when the initial assertion is true and $x > y$ is false, so that $x \leq y$, the final assertion is again true. Hence, using the rule of inference for program segments of this type, this program is correct with respect to the given initial and final assertions. ◀

Similarly, suppose that a program has a statement of the form

if *condition* **then**
S_1
else
S_2

If *condition* is true, then S_1 executes; if *condition* is false, then S_2 executes. To verify that this program segment is correct with respect to the initial assertion p and the final assertion q, two things must be done. First, it must be shown that when p is true and *condition* is true, then q is true after S_1 terminates. Second, it must be shown that when p is true and *condition* is false, then q is true after S_2 terminates. This leads to the following rule of inference:

$$\frac{(p \wedge condition)\{S_1\}q \qquad (p \wedge \neg condition)\{S_2\}q}{\therefore\ p\{\textbf{if } condition \textbf{ then } S_1 \textbf{ else } S_2\}q.}$$

Links

C. ANTHONY R. HOARE (BORN 1934) Tony Hoare was born in Colombo, Ceylon (now known as Sri Lanka), where his father was a civil servant of the British Empire and his mother's father owned a plantation. He spent his early childhood in Ceylon, moving to England in 1945. Hoare studied philosophy, together with the classics, at the University of Oxford, where he became interested in computing as a result of his fascination with the power of mathematical logic and the certainty of mathematical truth. He received his bachelors degree from Oxford in 1956.

Hoare learned Russian during his service in the Royal Navy, and latter studied the computer translation of natural languages at Moscow State University. He returned to England in 1960, taking a job at a small computer manufacturer, where he wrote a compiler for the programming language Algol. In 1968, he became Professor of Computing Science at the Queen's University, Belfast; in 1977, he moved to the University of Oxford as Professor of Computing; he is now Professor Emeritus. He is a Fellow of the Royal Society and also holds a position at Microsoft Research in Cambridge.

Hoare has made many contributions to the theory of programming languages and to programming methodology. He was first to define a programming language based on how programs could be proved correct with respect to their specifications. Hoare also invented quick sort, one of the most commonly used sorting algorithms (see the preamble to Exercise 50 in Section 5.4). He received the ACM Turing Award in 1980 and in 2000 he was knighted for services to education and computer science. Hoare is a noted writer in the technical and social aspects of computer science.

Example 3 illustrates how this rule of inference is used.

EXAMPLE 3 Verify that the program segment

```
if x < 0 then
    abs := −x
else
    abs := x
```

is correct with respect to the initial assertion **T** and the final assertion $abs = |x|$.

Solution: Two things must be demonstrated. First, it must be shown that if the initial assertion is true and $x < 0$, then $abs = |x|$. This is correct, because when $x < 0$ the assignment statement $abs := -x$ sets $abs = -x$, which is $|x|$ by definition when $x < 0$. Second, it must be shown that if the initial assertion is true and $x < 0$ is false, so that $x \geq 0$, then $abs = |x|$. This is also correct, because in this case the program uses the assignment statement $abs := x$, and x is $|x|$ by definition when $x \geq 0$, so $abs := x$. Hence, using the rule of inference for program segments of this type, this segment is correct with respect to the given initial and final assertions. ◀

Loop Invariants

Links

Next, proofs of correctness of **while** loops will be described. To develop a rule of inference for program segments of the type

```
while condition
    S
```

note that S is repeatedly executed until *condition* becomes false. An assertion that remains true each time S is executed must be chosen. Such an assertion is called a **loop invariant**. In other words, p is a loop invariant if $(p \wedge condition)\{S\}p$ is true.

Suppose that p is a loop invariant. It follows that if p is true before the program segment is executed, p and $\neg condition$ are true after termination, if it occurs. This rule of inference is

$$\frac{(p \wedge condition)\{S\}p}{\therefore\ p\{\textbf{while}\ condition\ \ S\}(\neg\ condition\ \wedge\ p).}$$

The use of a loop invariant is illustrated in Example 4.

EXAMPLE 4 A loop invariant is needed to verify that the program segment

Extra Examples

```
i := 1
factorial := 1
while i < n
    i := i + 1
    factorial := factorial · i
```

terminates with $factorial = n!$ when n is a positive integer.

Let p be the assertion "$factorial = i!$ and $i \leq n$." We first prove that p is a loop invariant. Suppose that, at the beginning of one execution of the **while** loop, p is true and the condition of the **while** loop holds; in other words, assume that $factorial = i!$ and that $i < n$. The new values i_{new} and $factorial_{new}$ of i and $factorial$ are $i_{new} = i + 1$ and $factorial_{new} = factorial \cdot (i + 1) = (i + 1)! = i_{new}!$. Because $i < n$, we also have $i_{new} = i + 1 \leq n$. Thus, p is true at the end of the execution of the loop. This shows that p is a loop invariant.

Now we consider the program segment. Just before entering the loop, $i = 1 \leq n$ and $factorial = 1 = 1! = i!$ both hold, so p is true. Because p is a loop invariant, the rule of inference just introduced implied that if the **while** loop terminates, it terminates with p true and with $i < n$ false. In this case, at the end, $factorial = i!$ and $i \leq n$ are true, but $i < n$ is false; in other words, $i = n$ and $factorial = i! = n!$, as desired.

Finally, we need to check that the **while** loop actually terminates. At the beginning of the program i is assigned the value 1, so after $n - 1$ traversals of the loop, the new value of i will be n, and the loop terminates at that point. ◀

A final example will be given to show how the various rules of inference can be used to verify the correctness of a longer program.

EXAMPLE 5 We will outline how to verify the correctness of the program S for computing the product of two integers.

procedure *multiply*(m, n: integers)

S_1:
if $n < 0$ **then** $a := -n$
else $a := n$

S_2:
$k := 0$
$x := 0$

S_3:
while $k < a$
  $x := x + m$
  $k := k + 1$

S_4:
if $n < 0$ **then** $product := -x$
else $product := x$

return *product*
{*product* equals mn}

The goal is to prove that after S is executed, *product* has the value mn. The proof of correctness can be carried out by splitting S into four segments, with $S = S_1; S_2; S_3; S_4$, as shown in the listing of S. The rule of composition can be used to build the correctness proof. Here is how the argument proceeds. The details will be left as an exercise for the reader.

Let p be the initial assertion "m and n are integers." Then, it can be shown that $p\{S_1\}q$ is true, when q is the proposition $p \wedge (a = |n|)$. Next, let r be the proposition $q \wedge (k = 0) \wedge (x = 0)$. It is easily verified that $q\{S_2\}r$ is true. It can be shown that "$x = mk$ and $k \leq a$" is an invariant for the loop in S_3. Furthermore, it is easy to see that the loop terminates after a iterations, with $k = a$, so $x = ma$ at this point. Because r implies that $x = m \cdot 0$ and $0 \leq a$, the loop invariant is true before the loop is entered. Because the loop terminates with $k = a$, it follows that $r\{S_3\}s$ is true where s is the proposition "$x = ma$ and $a = |n|$." Finally, it can be shown that S_4 is correct with respect to the initial assertion s and final assertion t, where t is the proposition "$product = mn$."

Putting all this together, because $p\{S_1\}q$, $q\{S_2\}r$, $r\{S_3\}s$, and $s\{S_4\}t$ are all true, it follows from the rule of composition that $p\{S\}t$ is true. Furthermore, because all four segments terminate, S does terminate. This verifies the correctness of the program. ◀

Exercises

1. Prove that the program segment

$y := 1$
$z := x + y$

is correct with respect to the initial assertion $x = 0$ and the final assertion $z = 1$.

2. Verify that the program segment

if $x < 0$ **then** $x := 0$

is correct with respect to the initial assertion **T** and the final assertion $x \geq 0$.

3. Verify that the program segment

$x := 2$
$z := x + y$
if $y > 0$ **then**
  $z := z + 1$
else
  $z := 0$

is correct with respect to the initial assertion $y = 3$ and the final assertion $z = 6$.

4. Verify that the program segment

if $x < y$ **then**
  $min := x$
else
  $min := y$

is correct with respect to the initial assertion **T** and the final assertion $(x \leq y \wedge min = x) \vee (x > y \wedge min = y)$.

***5.** Devise a rule of inference for verification of partial correctness of statements of the form

if *condition* 1 **then**
  S_1
else if *condition* 2 **then**
  S_2
  ⋮
else
  S_n

where $S_1, S_2, \ldots, S_n$ are blocks.

6. Use the rule of inference developed in Exercise 5 to verify that the program

if $x < 0$ **then**
  $y := -2|x|/x$
else if $x > 0$ **then**
  $y := 2|x|/x$
else if $x = 0$ **then**
  $y := 2$

is correct with respect to the initial assertion **T** and the final assertion $y = 2$.

7. Use a loop invariant to prove that the following program segment for computing the nth power, where n is a positive integer, of a real number x is correct.

$power := 1$
$i := 1$
while $i \leq n$
  $power := power * x$
  $i := i + 1$

***8.** Prove that the iterative program for finding f_n given in Section 5.4 is correct.

9. Provide all the details in the proof of correctness given in Example 5.

10. Suppose that both the conditional statement $p_0 \rightarrow p_1$ and the program assertion $p_1\{S\}q$ are true. Show that $p_0\{S\}q$ also must be true.

11. Suppose that both the program assertion $p\{S\}q_0$ and the conditional statement $q_0 \rightarrow q_1$ are true. Show that $p\{S\}q_1$ also must be true.

12. This program computes quotients and remainders.

$r := a$
$q := 0$
while $r \geq d$
  $r := r - d$
  $q := q + 1$

Verify that it is partially correct with respect to the initial assertion "a and d are positive integers" and the final assertion "q and r are integers such that $a = dq + r$ and $0 \leq r < d$."

13. Use a loop invariant to verify that the Euclidean algorithm (Algorithm 1 in Section 4.3) is partially correct with respect to the initial assertion "a and b are positive integers" and the final assertion "$x = \gcd(a, b)$."

Key Terms and Results

TERMS

sequence: a function with domain that is a subset of the set of integers

geometric progression: a sequence of the form $a, ar, ar^2, \ldots$, where a and r are real numbers

arithmetic progression: a sequence of the form a, $a + d$, $a + 2d, \ldots$, where a and d are real numbers

the principle of mathematical induction: the statement $\forall n\, P(n)$ is true if $P(1)$ is true and $\forall k[P(k) \rightarrow P(k + 1)]$ is true.

basis step: the proof of $P(1)$ in a proof by mathematical induction of $\forall n\, P(n)$

inductive step: the proof of $P(k) \rightarrow P(k + 1)$ for all positive integers k in a proof by mathematical induction of $\forall n\, P(n)$

strong induction: the statement $\forall n\, P(n)$ is true if $P(1)$ is true and $\forall k[(P(1) \wedge \cdots \wedge P(k)) \rightarrow P(k+1)]$ is true

well-ordering property: Every nonempty set of nonnegative integers has a least element.

recursive definition of a function: a definition of a function that specifies an initial set of values and a rule for obtaining values of this function at integers from its values at smaller integers

recursive definition of a set: a definition of a set that specifies an initial set of elements in the set and a rule for obtaining other elements from those in the set

structural induction: a technique for proving results about recursively defined sets

recursive algorithm: an algorithm that proceeds by reducing a problem to the same problem with smaller input

merge sort: a sorting algorithm that sorts a list by splitting it in two, sorting each of the two resulting lists, and merging the results into a sorted list

iteration: a procedure based on the repeated use of operations in a loop

program correctness: verification that a procedure always produces the correct result

loop invariant: a property that remains true during every traversal of a loop

initial assertion: the statement specifying the properties of the input values of a program

final assertion: the statement specifying the properties the output values should have if the program worked correctly

Review Questions

1. a) Can you use the principle of mathematical induction to find a formula for the sum of the first n terms of a sequence?
b) Can you use the principle of mathematical induction to determine whether a given formula for the sum of the first n terms of a sequence is correct?
c) Find a formula for the sum of the first n even positive integers, and prove it using mathematical induction.

2. a) For which positive integers n is $11n + 17 \leq 2^n$?
b) Prove the conjecture you made in part (a) using mathematical induction.

3. a) Which amounts of postage can be formed using only 5-cent and 9-cent stamps?
b) Prove the conjecture you made using mathematical induction.
c) Prove the conjecture you made using strong induction.
d) Find a proof of your conjecture different from the ones you gave in (b) and (c).

4. Give two different examples of proofs that use strong induction.

5. a) State the well-ordering property for the set of positive integers.
b) Use this property to show that every positive integer greater than one can be written as the product of primes.

6. a) Explain why a function f from the set of positive integers to the set of real numbers is well-defined if it is defined recursively by specifying $f(1)$ and a rule for finding $f(n)$ from $f(n-1)$.
b) Provide a recursive definition of the function $f(n) = (n+1)!$.

7. a) Give a recursive definition of the Fibonacci numbers.
b) Show that $f_n > \alpha^{n-2}$ whenever $n \geq 3$, where f_n is the nth term of the Fibonacci sequence and $\alpha = (1+\sqrt{5})/2$.

8. a) Explain why a sequence a_n is well defined if it is defined recursively by specifying a_1 and a_2 and a rule for finding a_n from $a_1, a_2, \ldots, a_{n-1}$ for $n = 3, 4, 5, \ldots$.
b) Find the value of a_n if $a_1 = 1$, $a_2 = 2$, and $a_n = a_{n-1} + a_{n-2} + \cdots + a_1$, for $n = 3, 4, 5, \ldots$.

9. Give two examples of how well-formed formulae are defined recursively for different sets of elements and operators.

10. a) Give a recursive definition of the length of a string.
b) Use the recursive definition from part (a) and structural induction to prove that $l(xy) = l(x) + l(y)$.

11. a) What is a recursive algorithm?
b) Describe a recursive algorithm for computing the sum of n numbers in a sequence.

12. Describe a recursive algorithm for computing the greatest common divisor of two positive integers.

13. a) Describe the merge sort algorithm.
b) Use the merge sort algorithm to put the list 4, 10, 1, 5, 3, 8, 7, 2, 6, 9 in increasing order.
c) Give a big-O estimate for the number of comparisons used by the merge sort.

14. a) Does testing a computer program to see whether it produces the correct output for certain input values verify that the program always produces the correct output?
b) Does showing that a computer program is partially correct with respect to an initial assertion and a final assertion verify that the program always produces the correct output? If not, what else is needed?

15. What techniques can you use to show that a long computer program is partially correct with respect to an initial assertion and a final assertion?

16. What is a loop invariant? How is a loop invariant used?

Supplementary Exercises

1. Use mathematical induction to show that $\frac{2}{3} + \frac{2}{9} + \frac{2}{27} + \cdots + \frac{2}{3^n} = 1 - \frac{1}{3^n}$ whenever n is a positive integer.

2. Use mathematical induction to show that $1^3 + 3^3 + 5^3 + \cdots + (2n+1)^3 = (n+1)^2(2n^2+4n+1)$ whenever n is a positive integer.

3. Use mathematical induction to show that $1 \cdot 2^0 + 2 \cdot 2^1 + 3 \cdot 2^2 + \cdots + n \cdot 2^{n-1} = (n-1) \cdot 2^n + 1$ whenever n is a positive integer.

4. Use mathematical induction to show that

$$\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \cdots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$$

whenever n is a positive integer.

5. Show that

$$\frac{1}{1 \cdot 4} + \frac{1}{4 \cdot 7} + \cdots + \frac{1}{(3n-2)(3n+1)} = \frac{n}{3n+1}$$

whenever n is a positive integer.

6. Use mathematical induction to show that $2^n > n^2 + n$ whenever n is an integer greater than 4.

7. Use mathematical induction to show that $2^n > n^3$ whenever n is an integer greater than 9.

8. Find an integer N such that $2^n > n^4$ whenever n is greater than N. Prove that your result is correct using mathematical induction.

9. Use mathematical induction to prove that $a - b$ is a factor of $a^n - b^n$ whenever n is a positive integer.

10. Use mathematical induction to prove that 9 divides $n^3 + (n+1)^3 + (n+2)^3$ whenever n is a nonnegative integer.

11. Use mathematical induction to prove that 43 divides $6^{n+1} + 7^{2n-1}$ for every positive integer n.

12. Use mathematical induction to prove that 64 divides $3^{2n+2} + 56n + 55$ for every positive integer n.

13. Use mathematical induction to prove this formula for the sum of the terms of an arithmetic progression.

$$a + (a+d) + \cdots + (a+nd) = (n+1)(2a+nd)/2$$

14. Suppose that $a_j \equiv b_j \pmod{m}$ for $j = 1, 2, \ldots, n$. Use mathematical induction to prove that

a) $\sum_{j=1}^{n} a_j \equiv \sum_{j=1}^{n} b_j \pmod{m}$.

b) $\prod_{j=1}^{n} a_j \equiv \prod_{j=1}^{n} b_j \pmod{m}$.

15. Show that if n is a positive integer, then

$$\sum_{k=1}^{n} \frac{k+4}{k(k+1)(k+2)} = \frac{n(3n+7)}{2(n+1)(n+2)}.$$

16. For which positive integers n is $n + 6 < (n^2 - 8n)/16$? Prove your answer using mathematical induction.

17. (*Requires calculus*) Suppose that $f(x) = e^x$ and $g(x) = xe^x$. Use mathematical induction together with the product rule and the fact that $f'(x) = e^x$ to prove that $g^{(n)}(x) = (x+n)e^x$ whenever n is a positive integer.

18. (*Requires calculus*) Suppose that $f(x) = e^x$ and $g(x) = e^{cx}$, where c is a constant. Use mathematical induction together with the chain rule and the fact that $f'(x) = e^x$ to prove that $g^{(n)} = c^n e^{cx}$ whenever n is a positive integer.

***19.** Formulate a conjecture about which Fibonacci numbers are even, and use a form of mathematical induction to prove your conjecture.

***20.** Determine which Fibonacci numbers are divisible by 3. Use a form of mathematical induction to prove your conjecture.

***21.** Prove that $f_k f_n + f_{k+1} f_{n+1} = f_{n+k+1}$ for all nonnegative integers n and k, where f_i denotes the ith Fibonacci number.

Recall from Example 15 of Section 2.4 that the sequence of **Lucas numbers** is defined by $l_0 = 2$, $l_1 = 1$, and $l_n = l_{n-1} + l_{n-2}$ for $n = 2, 3, 4, \ldots$.

22. Show that $f_n + f_{n+2} = l_{n+1}$ whenever n is a positive integer, where f_i and l_i are the ith Fibonacci number and ith Lucas number, respectively.

23. Show that $l_0^2 + l_1^2 + \cdots + l_n^2 = l_n l_{n+1} + 2$ whenever n is a nonnegative integer and l_i is the ith Lucas number.

***24.** Use mathematical induction to show that the product of any n consecutive positive integers is divisible by $n!$. [*Hint:* Use the identity $m(m+1)\cdots(m+n-1)/n! = (m-1)m(m+1)\cdots(m+n-2)/n! + m(m+1)\cdots(m+n-2)/(n-1)!$.]

25. Use mathematical induction to show that $(\cos x + i \sin x)^n = \cos nx + i \sin nx$ whenever n is a positive integer. (Here i is the square root of -1.) [*Hint:* Use the identities $\cos(a+b) = \cos a \cos b - \sin a \sin b$ and $\sin(a+b) = \sin a \cos b + \cos a \sin b$.]

***26.** Use mathematical induction to show that $\sum_{j=1}^{n} \cos jx = \cos[(n+1)x/2] \sin(nx/2) / \sin(x/2)$ whenever n is a positive integer and $\sin(x/2) \neq 0$.

27. Use mathematical induction to prove that $\sum_{j=1}^{n} j^2 2^j = n^2 2^{n+1} - n2^{n+2} + 3 \cdot 2^{n+1} - 6$ for every positive integer n.

28. (*Requires calculus*) Suppose that the sequence $x_1, x_2, \ldots, x_n, \ldots$ is recursively defined by $x_1 = 0$ and $x_{n+1} = \sqrt{x_n + 6}$.

a) Use mathematical induction to show that $x_1 < x_2 < \cdots < x_n < \cdots$, that is, the sequence $\{x_n\}$ is monotonically increasing.

b) Use mathematical induction to prove that $x_n < 3$ for $n = 1, 2, \ldots$.

c) Show that $\lim_{n \to \infty} x_n = 3$.

29. Show if n is a positive integer with $n \geq 2$, then

$$\sum_{j=2}^{n} \frac{1}{j^2 - 1} = \frac{(n-1)(3n+2)}{4n(n+1).}$$