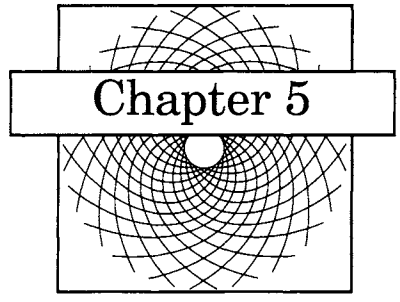


Recursion

It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all, try something.

—FRANKLIN ROOSEVELT

Recursion is an elegant and powerful problem-solving technique, used extensively in both discrete mathematics and computer science. Many programming languages, such as ALGOL, FORTRAN 90, C++, and Java, support recursion. This chapter investigates this powerful method in detail.

In addition, we will study three simple methods for solving recurrence relations: iteration, characteristic equations, and generating functions.

We also will establish the validity of recursive algorithms using induction and analyze their complexities using the big-oh and big-theta notations.

Some of the interesting problems we pursue in this chapter are:

- There are three pegs X, Y, and Z on a platform and 64 disks of increasing sizes at X. We would like to move them from X to Z using Y as an auxiliary peg subject to the following conditions:

  Only one disk can be moved at a time.

  No disk can be placed on the top of a smaller disk.

  If it takes one second to transfer a disk from one peg to another, how long will it take to solve the puzzle?

- Is there a formula for the number of n-bit words containing no two consecutive 1's?
- Suppose we introduce a mixed pair (male and female) of 1-month-old rabbits into a large enclosure on January 1. By the end of each month, the rabbits become mature, and each pair produces $k - 1$ mixed pairs of offspring at the beginning of the following month. Find the average age of the rabbit pairs at the beginning of the nth month.
- Can we estimate the number of divisions required to compute $\gcd\{a, b\}$ by the euclidean algorithm?
- What is a divide-and-conquer algorithm? If $f(n)$ denotes the number of operations required by such an algorithm, what can you say about its order of complexity?

5.1 Recursively Defined Functions

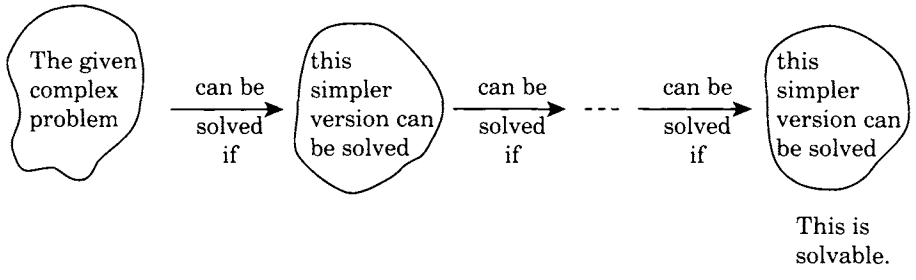
Recall that in Section 2.5 we employed recursion to define sets; we invoked the recursive clause to construct new elements from known elements. The same idea can be applied to define functions, and hence sequences as well.

This section illustrates how powerful a problem-solving technique recursion is. We begin with a simple problem:

> There are n guests at a sesquicentennial ball. Each person shakes hands with everybody else exactly once. How many handshakes are made?

Suppose you would like to solve a problem such as this. (See Example 5.3.) The solution may not be obvious. However, it may turn out that the problem could be defined in terms of a simpler version of itself. Such a definition is a **recursive definition**. Consequently, the given problem can be solved provided the simpler version can be solved. This idea is pictorially represented in Figure 5.1.

Figure 5.1



Recursive Definition of a Function

Let $a \in \mathbf{W}$ and $X = \{a, a+1, a+2, \ldots\}$. The **recursive definition** of a function f with domain X consists of three parts, where $k \geq 1$:

- **Basis clause** A few initial values of the function $f(a), f(a+1), \ldots, f(a+k-1)$ are specified. An equation that specifies such initial values is an **initial condition**.
- **Recursive clause** A formula to compute $f(n)$ from the k preceding functional values $f(n-1), f(n-2), \ldots, f(n-k)$ is made. Such a formula is a **recurrence relation** (or **recursion formula**).
- **Terminal clause** Only values thus obtained are valid functional values. (For convenience, we drop this clause from our recursive definition.)

Thus the recursive definition of f consists of one or more (a finite number of) initial conditions, and a recurrence relation.

Is the recursive definition of f a valid definition? In other words, if the k initial values $f(a), f(a+1), \ldots, f(a+k-1)$ are known and $f(n)$ is defined in terms of k of its predecessors $f(n-1), f(n-2), \ldots, f(n-k)$, where $n \geq a+k$, is $f(n)$ defined for $n \geq a$? Fortunately, the next theorem comes to our rescue. Its proof uses strong induction and is complicated, so we omit it.

THEOREM 5.1 Let $a \in \mathbf{W}$, $X = \{a, a+1, a+2, \ldots\}$, and $k \in \mathbb{N}$. Let $f : X \to \mathbb{R}$ such that $f(a), f(a+1), \ldots, f(a+k-1)$ are known. Let n be any positive integer $\geq a+k$ such that $f(n)$ is defined in terms of $f(n-1), f(n-2), \ldots$ and $f(n-k)$. Then $f(n)$ is defined for every $n \geq a$. ■

By virtue of this theorem, recursive definitions are also known as **inductive definitions**.

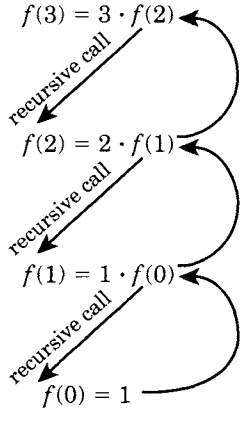
The following examples illustrate the recursive definition of a function.

EXAMPLE 5.1 Define recursively the factorial function f.

SOLUTION:
Recall that the factorial function f is defined by $f(n) = n!$, where $f(0) = 1$. Since $n! = n(n-1)!$, f can be defined recursively as follows:

$$f(0) = 1 \qquad \leftarrow \text{initial condition}$$
$$f(n) = n \cdot f(n-1), \quad n \geq 1 \qquad \leftarrow \text{recurrence relation}$$ ■

Suppose we would like to compute $f(3)$ using this recursive definition. We then continue to apply the recurrence relation until the initial condition is reached, as shown below:



(5.1)

(5.2)

(5.3)

(5.4)

Since $f(0) = 1$, 1 is substituted for $f(0)$ in Equation (5.3) and $f(1)$ is computed: $f(1) = 1 \cdot f(0) = 1 \cdot 1 = 1$. This value is substituted for $f(1)$ in Equation (5.2) and $f(2)$ is computed: $f(2) = 2 \cdot f(1) = 2 \cdot 1 = 2$. This value is now returned to Equation (5.1) to compute $f(3)$: $f(3) = 3 \cdot f(2) = 3 \cdot 2 = 6$, as expected.

EXAMPLE 5.2 Judy deposits \$1000 in a local savings bank at an annual interest rate of 8% compounded annually. Define recursively the compound amount $A(n)$ she will have in her account at the end of n years.

SOLUTION:
Clearly, $A(0)$ = initial deposit = \$1000. Let $n \geq 1$. Then:

$$\begin{aligned} A(n) &= \left(\begin{array}{l}\text{compound amount}\\ \text{at the end of the}\\ (n-1)\text{st year}\end{array}\right) + \left(\begin{array}{l}\text{interest earned}\\ \text{during the}\\ n\text{th year}\end{array}\right) \\ &= A(n-1) + (0.08)A(n-1) \\ &= 1.08A(n-1) \end{aligned}$$

Thus $A(n)$ can be defined recursively as follows:

$$\begin{aligned} A(0) &= 1000 && \leftarrow \text{initial condition} \\ A(n) &= 1.08A(n-1), \quad n \geq 1 && \leftarrow \text{recurrence relation} \end{aligned}$$

■

For instance, the compound amount Judy will have at the end of three years is

$$\begin{aligned} A(3) &= 1.08A(2) \\ &= 1.08[1.08A(1)] = 1.08^2A(1) \\ &= 1.08^2[1.08A(0)] = 1.08^3(1000) \\ &\approx \$1259.71^* \end{aligned}$$

The next two examples illustrate an extremely useful problem-solving technique, used often in discrete mathematics and computer science.

EXAMPLE 5.3 **(The handshake problem)** There are n guests at a sesquicentennial ball. Each person shakes hands with everybody else exactly once. Define recursively the number of handshakes $h(n)$ that occur.

SOLUTION:
Clearly, $h(1) = 0$, so let $n \geq 2$. Let x be one of the guests. By definition, the number of handshakes made by the remaining $n - 1$ guests among themselves is $h(n - 1)$. Now person x shakes hands with each of these

*The symbol $\approx$ means is *approximately equal to.*

$n - 1$ guests, yielding $n - 1$ additional handshakes. So the total number of handshakes made equals $h(n - 1) + (n - 1)$, where $n \geq 2$.

Thus $h(n)$ can be defined recursively as follows:

$$h(1) = 0 \qquad \leftarrow \text{initial condition}$$

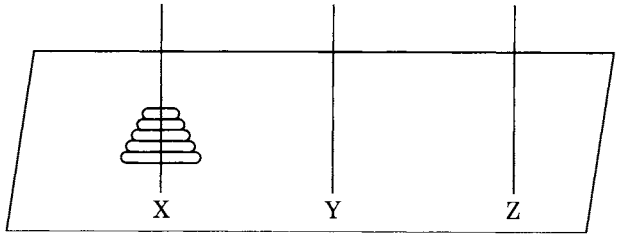
$$h(n) = h(n - 1) + (n - 1), \quad n \geq 2 \qquad \leftarrow \text{recurrence relation}$$ ■

EXAMPLE 5.4 **(Tower of Brahma*)** According to a legend of India, at the beginning of creation, God stacked 64 golden disks on one of three diamond pegs on a brass platform in the temple of Brahma at Benares† (see Figure 5.2). The priests on duty were asked to move the disks from peg X to peg Z using Y as an auxiliary peg under the following conditions:

- Only one disk can be moved at a time.
- No disk can be placed on the top of a smaller disk.

The priests were told that the world would end when the job was completed.

Figure 5.2



Suppose there are n disks on peg X. Let b_n denote the number of moves needed to move them from peg X to peg Z, using peg Y as an intermediary. Define b_n recursively.

SOLUTION:

Clearly $b_1 = 1$. Assume $n \geq 2$. Consider the top $n - 1$ disks on peg X. By definition, it takes b_{n-1} moves to transfer them from X to Y using Z as an auxiliary. That leaves the largest disk at peg X; it takes one move to transfer it from X to Z. See Figure 5.3.

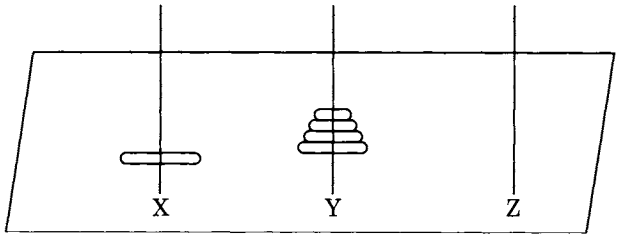
Now the $n - 1$ disks at Y can be moved from Y to Z using X as an intermediary in b_{n-1} moves, so the total number of moves needed is $b_{n-1} + 1 + b_{n-1} = 2b_{n-1} + 1$. Thus b_n can be defined recursively as follows:

$$b_n = \begin{cases} 1 & \text{if } n = 1 \quad \leftarrow \text{initial condition} \\ 2b_{n-1} + 1 & \text{otherwise} \quad \leftarrow \text{recurrence relation} \end{cases}$$ ■

*A puzzle based on the Tower of Brahma was marketed in 1883 under the name **Tower of Hanoi**.

†Benares is now known as Varanasi.

Figure 5.3



For example,

$$\begin{aligned} b_4 &= 2b_3 + 1 & &= 2[2b_2 + 1] + 1 \\ &= 4b_2 + 2 + 1 & &= 4[2b_1 + 1] + 2 + 1 \\ &= 8b_1 + 4 + 2 + 1 & &= 8(1) + 4 + 2 + 1 \\ &= 15 \end{aligned}$$

so it takes 15 moves to transfer 4 disks from X to Z, by this strategy.

The next example also illustrates the same technique. We will take it a step further in Chapter 6.

EXAMPLE 5.5 Imagine n lines in a plane such that no two lines are parallel, and no three are concurrent.* Let f_n denote the number of distinct regions into which the plane is divided by them. Define f_n recursively.

SOLUTION:

If there is just one line ℓ_1 in the plane, then $f_1 = 2$ (see Figure 5.4). Now consider a second line ℓ_2; it is intersected at exactly one point by ℓ_1. Each half of ℓ_2 divides an original region into two, adding two more regions (see Figure 5.5). Thus $f_2 = f_1 + 2 = 4$. Suppose we add a third line ℓ_3. It is

Figure 5.4

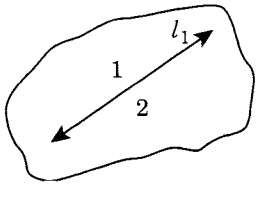
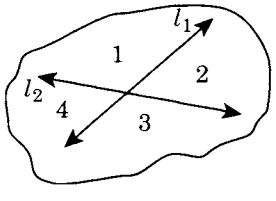


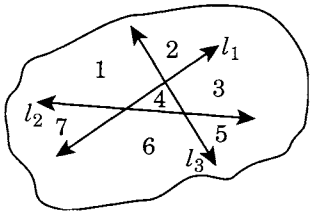
Figure 5.5



*Three or more lines in a plane are **concurrent** if they intersect at a point.

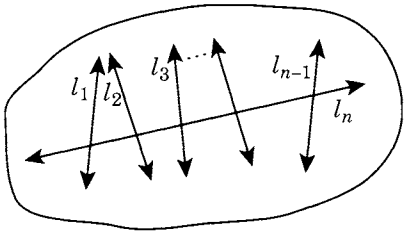
intersected by ℓ_1 and ℓ_2 in two points; in other words, line ℓ_3 is divided by ℓ_1 and ℓ_2 into three parts. Each portion divides an existing region into two, yielding three new regions (see Figure 5.6). So $f_3 = f_2 + 3 = 7$.

Figure 5.6



More generally, suppose there are $n-1$ lines $\ell_1, \ell_2, \ldots, \ell_{n-1}$ in the plane. They divide the plane into f_{n-1} disjoint regions, by definition. Now add one more line ℓ_n (see Figure 5.7). Since no three lines are concurrent, line ℓ_n must intersect lines $\ell_1, \ell_2, \ldots, \ell_{n-1}$ at new points and hence is divided by

Figure 5.7



them into n segments. Each segment divides an existing region into two subregions, contributing n more regions, so $f_n = f_{n-1} + n$. Thus f_n can be defined recursively as follows:

$$f_n = \begin{cases} 1 & \text{if } n = 0 \\ f_{n-1} + n & \text{otherwise} \end{cases}$$

■

The next example illustrates how to define recursively the number of times an assignment is executed by nested **for** loops.

EXAMPLE 5.6 Let a_n denote the number of times the assignment statement $x \leftarrow x + 1$ is executed by the following nested **for** loops. Define a_n recursively.

```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to j do
      x ← x + 1
```

SOLUTION:

- First, we must find the initial condition satisfied by a_n. When $n = 1$, $i = j = k = 1$, so the assignment statement is executed exactly once. Thus $a_1 = 1$.

- *To find the recurrence relation satisfied by a_n:*
  Let $n \geq 2$. As i runs from 1 through $n - 1$, by definition, the statement is executed a_{n-1} times.
  When $i = n$, the inner loops become:

```
for j = 1 to n do
  for k = 1 to j do
    x ← x + 1
```

  For each value of j, where $1 \leq j \leq n$, the innermost loop executes the statement j times. So these nested loops execute it $\sum_{j=1}^{n} j = \dfrac{n(n+1)}{2}$ times. Therefore,

$$a_n = \left(\begin{array}{l} \text{no. of times the statement} \\ \text{is executed as } i \text{ runs from} \\ 1 \text{ through } n-1 \end{array} \right) + \left(\begin{array}{l} \text{no. of times the} \\ \text{statement is executed} \\ \text{when } i = n \end{array} \right)$$

$$= a_{n-1} + \frac{n(n+1)}{2}$$

  Thus a_n can be defined as follows:

$$a_1 = 1$$

$$a_n = a_{n-1} + \frac{n(n+1)}{2}, \quad n \geq 2$$

(We shall pursue this definition in Example 5. 11.) ■

The next example provides a recursive definition with two initial conditions. We shall use it often in the following sections and in the next chapter.

EXAMPLE 5.7 **(Fibonacci)** Leonardo Fibonacci, the most outstanding Italian mathematician of the Middle Ages, proposed the following problem around 1202:

Suppose there are two newborn rabbits, one male and the other female. Find the number of rabbits produced in a year if:

- Each pair takes one month to become mature.
- Each pair produces a mixed pair every month, from the second month.
- No rabbits die.

Suppose, for convenience, that the original pair of rabbits was born on January 1. They take a month to become mature. So there is still only one pair on February 1. On March 1, they are 2 months old and produce a new mixed pair, a total of two pairs. Continuing like this, there will be three pairs on April 1, five pairs on May 1, and so on. See the last row of Table 5.1.

Table 5.1

No. of pairs	Jan	Feb	March	April	May	June	July	Aug
Adults	0	1	1	2	3	5	8	13
Babies	1	0	1	1	2	3	5	8
Total	1	1	2	3	5	8	13	21

←

Leonardo Fibonacci *(1170?–1250?), also known as Leonardo of Pisa, was born in the commercial center of Pisa, Italy, into the Bonacci family. His father, a customs manager, expected the son to become a merchant and took him to Bougie, Algeria, to receive good training in arithmetic with Indian numerals. Leonardo's subsequent business trips to Egypt, Syria, Greece, and Sicily brought him closer to Indian mathematics.*

In 1202, shortly after his return, convinced of the elegance of the Indian methods of computation, Fibonacci published his famous work, Liber Abaci. *(The word* abaci *in the title does not refer to the old abacus, but to computation in general.) This book, devoted to arithmetic and elementary algebra, introduced the Indian notation and arithmetic algorithms to Europe.*

Fibonacci wrote three additional books: Practica Geometriae, *a collection of results in geometry and trigonometry;* Liber Quadratorum, *a major work on number theory; and* Flos, *also on number theory.*

Fibonacci's importance and usefulness to Pisa and its citizenry through his teaching and services were honored by Emperor Frederick II of Pisa.

The numbers 1, 1, 2, 3, 5, 8, ... are **Fibonacci numbers.*** They have a fascinating property: Any Fibonacci number, except the first two, is the sum of the two immediately preceding Fibonacci numbers. (At the given rate, there will be 144 pairs of rabbits on December 1.)

This yields the following recursive definition of the nth Fibonacci number F_n:

$$F_1 = F_2 = 1 \qquad \leftarrow \text{initial conditions}$$

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 3 \qquad \leftarrow \text{recurrence relation}$$ ■

The next example illustrates recursion and also shows that Fibonacci numbers occur in quite unexpected places.

EXAMPLE 5.8 Let a_n denote the number of n-bit words containing no two consecutive 1's. Define a_n recursively.

*See author's *Fibonacci and Lucas Numbers with Applications* for a thorough discussion of Fibonacci numbers.

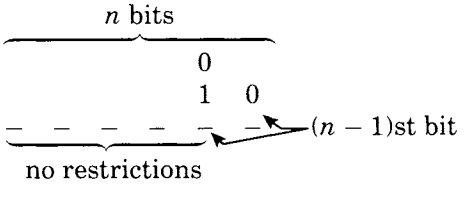
SOLUTION:
First, let us find the n-bit words containing no two consecutive 1's corresponding to $n = 1, 2, 3$, and 4 (see Table 5.2). It follows from the table that $a_1 = 2$, $a_2 = 3$, $a_3 = 5$, and $a_4 = 8$.

Table 5.2

$n = 1$	$n = 2$	$n = 3$	$n = 4$
0	00	000	0000
1	01	010	0100
	10	100	1000
		001	0010
		101	1010
			0001
			0101
			1001

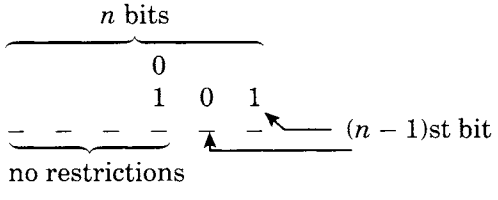
Now, consider an arbitrary n-bit word. It may end in 0 or 1.

Case 1 Suppose the n-bit word ends in 0. Then the $(n - 1)$st bit can be a 0 or a 1, so there are no restrictions on the $(n - 1)$st bit:



Therefore, a_{n-1} n-bit words end in 0 and contain no two consecutive 1's.

Case 2 Suppose the n-bit word ends in 1. Then the $(n - 1)$st bit must be a zero. Further, there are no restrictions on the $(n - 2)$nd bit:



Thus a_{n-2} n-bit words end in 1 and contain no two consecutive 1's.

Since the two cases are mutually exclusive, by the addition principle, we have:

$$a_1 = 2, \quad a_2 = 3 \qquad \leftarrow \text{initial conditions}$$

$$a_n = a_{n-1} + a_{n-2}, \quad n \geq 3 \qquad \leftarrow \text{recurrence relation}$$

Notice that the above recurrence relation is exactly the same as the Fibonacci recurrence relation, but with different initial conditions! The resulting numbers are the Fibonacci numbers 2, 3, 5, 8, 13, ... ■

Notice that this example does *not* provide a constructive method for systematically listing all n-bit words with the required property. It is given in Exercise 19.

Interestingly enough, the delightful Fibonacci numbers occur in numerous totally unexpected places. For instance, the numbers of spiral arrays of seeds in mature sunflowers in the clockwise and counterclockwise directions are often consecutive Fibonacci numbers, usually 34 and 55, or 55 and 89. See Figures 5.8 and 5.9.

Figure 5.8

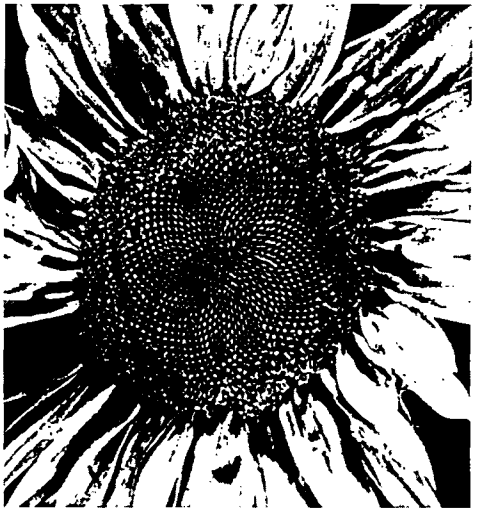
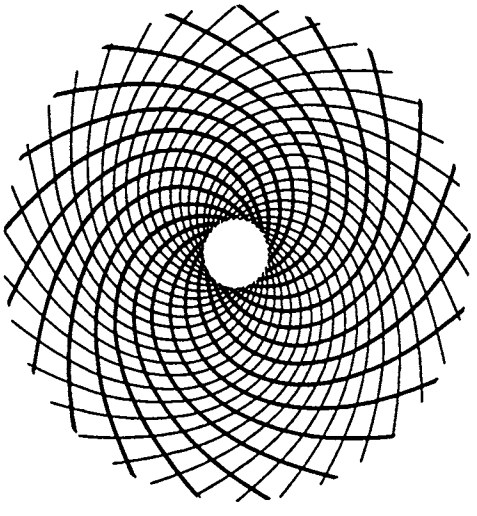


Figure 5.9



Before closing this section, we establish an important result from the theory of formal languages. First, recall that Σ^* denotes the set of words over an alphabet Σ. Also Σ^* can be defined recursively as follows (see Exercise 35 in Section 2.6):

- $\lambda \in \Sigma^*$.
- If $w \in \Sigma^*$ and $s \in \Sigma$, then $ws \in \Sigma^*$.

Furthermore, the length $\|w\|$ of a word w over Σ can be defined recursively as follows:

- $\|\lambda\| = 0$.
- If $w \in \Sigma^*$ and $s \in \Sigma$, then $\|ws\| = \|w\| + 1$.

Using these definitions and induction, we prove below that $\|xy\| = \|x\| + \|y\|$ for any two words x and y in Σ^*.

EXAMPLE 5.9 Let x and y be any two words over an alphabet Σ. Prove that $\|xy\| = \|x\| + \|y\|$.

PROOF (by induction):
Let x be any element in Σ^*. Let $P(y)$ denote the predicate that $\|xy\| = \|x\| + \|y\|$, where $y \in \Sigma^*$. Since $y \in \Sigma^*$, y can be the null word λ or a nonempty word.

Basis step *To show that* $P(\lambda)$ *is true;* that is, $\|x\lambda\| = \|x\| + \|\lambda\|$:

Since $x\lambda = x$, $\|x\lambda\| = \|x\| = \|x\| + 0 = \|x\| + \|\lambda\|$. So $P(\lambda)$ is true.

Induction step Assume $P(y)$ is true, that is, $\|xy\| = \|x\| + \|y\|$ (inductive hypothesis). We must show that $P(ys)$ is true, that is, $\|xys\| = \|x\| + \|ys\|$. Notice that:

$$xys = (xy)s \qquad \text{assoc. prop. of concatenation}$$

Then

$$\begin{aligned} \|xys\| &= \|(xy)s\| && \text{length is a function} \\ &= \|xy\| + 1 && \text{recursive def. of length} \\ &= (\|x\| + \|y\|) + 1 && \text{inductive hypothesis} \\ &= \|x\| + (\|y\| + 1) && \text{assoc. prop. of addition} \\ &= \|x\| + \|ys\| && \text{recursive def. of length} \end{aligned}$$

Therefore, $P(ys)$ is true. Thus $P(y)$ implies $P(ys)$.

Therefore, by induction, $P(y)$ is true for every $y \in \Sigma^*$; that is, $\|xy\| = \|x\| + \|y\|$ for every $x, y \in \Sigma^*$. ■

Finally, we emphasize that the immediate predecessor f_{n-1} need not appear in the recursive definition of a function f at n. For example, consider the function f: $\mathbf{W} \to \mathbf{W}$ defined by

$$f_0 = 1, \quad f_1 = 0, \quad f_2 = 1$$
$$f_n = f_{n-2} + 2f_{n-3}, \quad n \geq 3$$

Clearly, f_{n-1} is not needed to compute f_n, when $n \geq 3$. Try f_6 as an exercise.

Exercises 5.1

In Exercises 1–6, a_n denotes the nth term of a number sequence satisfying the given initial condition(s) and the recurrence relation. Compute the first four terms of the sequence.

1. $a_1 = 1$
$a_n = a_{n-1} + 3, n \geq 2$

2. $a_0 = 1$
$a_n = a_{n-1} + n, n \geq 1$

3. $a_1 = 1$
$a_n = \dfrac{n}{n-1} a_{n-1}, n \geq 2$

4. $a_1 = 1, a_2 = 2$
$a_n = a_{n-1} + a_{n-2}, n \geq 3$

5. $a_1 = 1, a_2 = 1, a_3 = 2$
$a_n = a_{n-1} + a_{n-2} + a_{n-3}, n \geq 4$

6. $a_1 = 1, a_2 = 2, a_3 = 3$
$a_n = a_{n-1} + a_{n-2} + a_{n-3}, n \geq 4$

7. The nth **Lucas number** L_n, named after the French mathematician François-Edouard-Anatole Lucas, is defined recursively as follows:

$$L_1 = 1, \quad L_2 = 3$$
$$L_n = L_{n-1} + L_{n-2}, n \geq 3$$

(The Lucas sequence and the Fibonacci sequence satisfy the same recurrence relation, but have different initial conditions.) Compute the first six Lucas numbers.

The gcd of two integers x (> 0) and y (≥ 0) can be defined recursively as follows:

$$\gcd\{x,y\} = \begin{cases} \gcd\{y,x\} & \text{if } y > x \\ x & \text{if } y \leq x \text{ and } y = 0 \\ \gcd\{y, x \bmod y\} & \text{if } y \leq x \text{ and } y > 0 \end{cases}$$

Using this definition, compute the gcd of each pair of integers.

8. 28, 18

9. 24, 75

François-Edouard-Anatole Lucas *(1842–1891) was born in Amiens, France. After completing his studies at the École Normale in Amiens, he worked as an assistant at the Paris Observatory. He served as an artillery officer in the Franco-Prussian war and then became professor of mathematics at the Lycée Saint-Louis and Lycée Charlemagne, both in Paris. A gifted and entertaining teacher, Lucas died of a freak accident at a banquet: His cheek was gashed by a piece of a plate that was accidentally dropped, and he died from infection within a few days.*

Lucas loved computing and developed plans for a computer that never materialized. Besides his contributions to number theory, he is known for his four-volume classic on recreational mathematics. Best known among the problems he developed is the Tower of Brahma.

A person deposits \$1000 in a bank at an annual interest rate of 6%. Let $A(n)$ denote the compound amount she will receive at the end of n interest periods. Define $A(n)$ recursively if interest is compounded:

10. Semiannually **11.** Quarterly **12.** Monthly

Ned deposits a certain amount A_0 in a bank at an annual interest rate of 12% compounded annually. The compound amount he would receive at the end of n years is given by $A_n = 1.12A_{n-1}$, where $n \geq 1$. Determine the initial deposit A_0 if he would receive:

13. \$1804.64 at the end of 5 years. **14.** \$3507.00 at the end of 6 years.

Define recursively each sequence of numbers. (*Hint*: Look for a pattern and define the nth term a_n recursively.)

15. 1, 4, 7, 10, 13 ...

16. 3, 8, 13, 18, 23 ...

17. 0, 3, 9, 21, 45 ...

18. 1, 2, 5, 26, 677 ...

19. An n-bit word containing no two consecutive ones can be constructed recursively as follows: Append a 0 to such $(n-1)$-bit words or append a 01 to such $(n-2)$-bit words. Using this procedure construct all 5-bit words containing no two consecutive ones. There are 13 such words.

Define each recursively, where $n \geq 0$.

20. The nth power of a positive real number x.

21. The union of n sets.

22. The intersection of n sets.

23. The number S_n of subsets of a set with n elements.

24. The nth term a_n of an arithmetic sequence with first term a and common difference d.

John McCarthy *(1927–), one of the fathers of artificial intelligence (AI), was born in Boston. He graduated in mathematics from the California Institute of Technology, receiving his Ph.D. from Princeton in 1951. After teaching at Princeton, Stanford, Dartmouth, and MIT, he returned to Stanford as a full professor. While at Princeton, he was named a Proctor Fellow and later the Higgins Research Instructor in mathematics. At Stanford, he headed the Artificial Intelligence Laboratory.*

During his tenure at Dartmouth, McCarthy coined the term artificial intelligence *(AI). He developed LISP (*LISt Programming*), one of the most widely used programming languages in AI. He also helped develop ALGOL 58 and ALGOL 60. In 1971 he received the prestigious Alan M. Turing award for his outstanding contributions to data processing.*

25. The nth term a_n of a geometric sequence with first term a and common ratio r.

26. Let $f : X \to X$ be bijective. Define f^n recursively, where $f^2 = f \circ f$.

The **91-function** f, invented by John McCarthy, is defined recursively on **W** as follows.

$$f(x) = \begin{cases} x - 10 & \text{if } x > 100 \\ f(f(x + 11)) & \text{if } 0 \leq x \leq 100 \end{cases}$$

Compute each

27. $f(99)$ **28.** $f(98)$ **29.** $f(f(99))$ **30.** $f(f(91))$

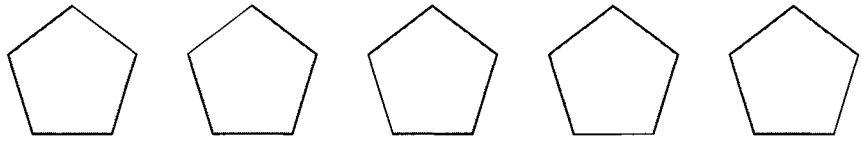
31. Show that $f(99) = 91$.

32. Prove that $f(x) = 91$ for $90 \leq x \leq 100$.

33. Prove that $f(x) = 91$ for $0 \leq x < 90$.

(Triangulation of convex polygons) The nth Catalan number C_n denotes the number of ways to divide a convex $(n + 2)$-gon into triangles by drawing nonintersecting diagonals. For instance, there are five ways of triangulating a convex pentagon, as shown in Figure 5.10; therefore, $C_3 = 5$. C_n is given recursively by $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$, where $C_0 = 1$. Compute each.

Figure 5.10



34. C_6 **35.** C_7

36. The sequence defined by $a_{n+1} = \frac{1}{2}(a_n + \frac{N}{a_n})$ can be used to approximate $\sqrt{N}$ to any desired degree of accuracy, where a_1 is an estimate of $\sqrt{N}$. Use this fact to compute $\sqrt{19}$ correct to six decimal places. Use $a_1 = 4$.

37. Let F_n denote the nth Fibonacci number. Compute $\frac{F_{n+1}}{F_n}$ correct to eight decimal places for $1 \le n \le 10$. Compare each value to $(1 + \sqrt{5})/2$ correct to eight decimal places.

38. (For those familiar with the concept of limits) Use Exercise 37 to predict $\lim_{n\to\infty} \frac{F_{n+1}}{F_n}$.

Prove each, where F_n is the nth Fibonacci number, L_n the nth Lucas number, and $\alpha = (1 + \sqrt{5})/2$, the **golden ratio**.

39. $F_n = 2F_{n-2} + F_{n-3}, n \ge 4$

40. $F_n^2 - F_{n-1}F_{n+1} = (-1)^{n-1}, n \ge 2$

41. F_{5n} is divisible by 5, $n \ge 1$.

42. $F_n < \alpha^{n-1}$, $n \ge 3$

43. $F_n \le 2^n$, $n \ge 1$

44. Let $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. Then $A^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$, $n \ge 1$. Assume $F_0 = 0$.

45. Using Exercise 44, deduce that $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$.

(*Hint*: Let A be a square matrix. Then $|A^n| = |A|^n$, where $|A|$ denotes the determinant of A.)

46. $L_n = F_{n+1} + F_{n-1}, n \ge 2$ **47.** $L_{2n} = 3 + \sum_{k=1}^{2n-2} L_k$

The nth term b_n of a number sequence is defined by $b_n = \frac{\alpha^n - \beta^n}{\alpha - \beta}$, where $\alpha = (1 + \sqrt{5})/2$ and $\beta = (1 - \sqrt{5})/2$ are solutions of the equation $x^2 = x + 1$. Verify each.

48. $b_1 = 1$ **49.** $b_2 = 1$ **50.** $b_n = b_{n-1} + b_{n-2}, n \ge 3$

(It follows from Exercises 48–50 that $b_n = F_n$. It is called the **Binet form** of the nth Fibonacci number, after the French mathematician Jacques-Phillipe-Marie Binet.)

With α and β as above, let $u_n = \alpha^n + \beta^n$, $n \ge 1$. Verify each.

51. $u_1 = 1$ **52.** $u_2 = 3$ **53.** $u_n = u_{n-1} + u_{n-2}, n \ge 3$

[These exercises indicate that $u_n = L_n$, the nth Lucas number. Accordingly, $u_n = \alpha^n + \beta^n$ is the Binet form of L_n.]

Jacques Phillippe Marie Binet *(1786–1865), a French mathematician and astronomer, was born at Rennes, Brittany. In 1804, he entered the Ecole Polytechnique in Paris, graduated 2 years later, and took a job in the Department of Bridges and Roads of the French government. In 1807, Binet became a teacher at the École Polytechnique, and the following year became assistant to the professor of applied analysis and descriptive geometry. In 1814, he was appointed examiner of descriptive geometry, then professor of mechanics (1815), and then inspector general of studies (1816). In 1821, he was awarded the Chevalier de la Légion d'Honneur. Two years later, Binet was appointed chair of astronomy at the Collège de France.*

But the July 1830 revolution was not kind to him. A strong supporter of Charles X, Binet became a victim of Charles' abdication; he was dismissed from École Polytechnique by King Louis-Phillipe in November, 1830.

Binet made many contributions to mathematics, physics, and astronomy. In 1812, he discovered the rule for matrix multiplication and, in 1840, discovered the explicit formula for the nth Fibonacci number. In 1843, he was elected to the Academy of Sciences and later became its president. A devout Catholic, Binet died in Paris.

54. Let $a_1, a_2, \ldots, a_n \in \mathbb{N}$, where $n \geq 2$. Prove that $\gcd\{a_1, a_2, \ldots, a_n\} = \gcd\{\gcd\{a_1, a_2, \ldots, a_{n-1}\}, a_n\}$.

Using Exercise 54 compute the gcd of each set of numbers.

55. 6, 12, 20, 38

56. 12, 28, 48, 104, 252

Let a_n denote the number of times the assignment statement $x \leftarrow x + 1$ is executed by each nested **for** loop. Define a_n recursively.

57.
```
for i = 1 to n do
  for j = 1 to i do
    x ← x + 1
```

58.
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to i do
      x ← x + 1
```

59. Let a_n denote the number of rectangles that can be formed on a $1 \times n$ rectangular board. Find the recurrence relation satisfied by a_n. (*Hint*: Look for a pattern. Every square is also a rectangle.)

A subset of the set $S = \{1, 2, \ldots, n\}$ is **alternating** if its elements, when arranged in increasing order, follow the pattern odd, even, odd, even, etc. For example, $\{3\}$, $\{1, 2, 5\}$, and $\{3, 4\}$ are alternating subsets of $\{1, 2, 3, 4, 5\}$, whereas $\{1, 3, 4\}$ and $\{2, 3, 4, 5\}$ are not; $\emptyset$ is considered alternating.* Let a_n denote the number of alternating subsets of S.

60. Define a_n recursively.

61. Prove that $a_n = F_{n+2}$, where F_n denotes the nth Fibonacci number.

*Proposed by Olry Terquem (1782–1862).

Stirling numbers of the second kind, denoted by $S(n, r)$ and used in combinatorics, are defined recursively as follows, where $n, r \in \mathbb{N}$:

$$S(n,r) = \begin{cases} 1 & \text{if } r = 1 \text{ or } r = n \\ S(n-1, r-1) + rS(n-1, r) & \text{if } 1 < r < n \\ 0 & \text{if } r > n \end{cases}$$

They are named after the English mathematician James Stirling (1692–1770). Compute each Stirling number.

62. $S(2, 2)$ **63.** $S(5, 2)$

A function of theoretical importance in the study of algorithms is the **Ackermann's function**, named after the German mathematician and logician Wilhelm Ackermann (1896–1962). It is defined recursively as follows, where $m, n \in \mathbf{W}$:

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

Compute each.

64. $A(0, 7)$ **65.** $A(1, 1)$

66. $A(4, 0)$ **67.** $A(2, 2)$

Prove each for $n \geq 0$.

68. $A(1, n) = n + 2$ **69.** $A(2, n) = 2n + 3$

***70.** Predict a formula for $A(3, n)$.

***71.** Prove the formula in Exercise 70, where $n \geq 0$.

5.2 Solving Recurrence Relations

The recursive definition of a function f does not provide us with an explicit formula for $f(n)$, but establishes a systematic procedure for finding it. This section illustrates the iterative method of finding a formula for $f(n)$ for a simple class of recurrence relations.

Solving the recurrence relation for a function f means finding an explicit formula for $f(n)$. The **iterative method** of solving it involves two steps:

- Apply the recurrence formula iteratively and look for a pattern to predict an explicit formula.
- Use induction to prove that the formula does indeed hold for every possible value of the integer n.

The next example illustrates this method.

EXAMPLE 5.10 (The handshake problem continued) By Example 5.3, the number of handshakes made by n guests at a dinner party is given by

$$h(1) = 0$$
$$h(n) = h(n-1) + (n-1), n \geq 2$$

Solve this recurrence relation.

SOLUTION:

Step 1 To predict a formula for $h(n)$:

Using iteration,
$$\begin{aligned} h(n) &= h(n-1) + (n-1) \\ &= h(n-2) + (n-2) + (n-1) \\ &= h(n-3) + (n-3) + (n-2) + (n-1) \\ &\vdots \\ &= h(1) + 1 + 2 + 3 + \cdots + (n-2) + (n-1) \\ &= 0 + 1 + 2 + 3 + \cdots + (n-1) \\ &= \frac{n(n-1)}{2} \end{aligned}$$

Step 2 To prove, by induction, that $h(n) = \dfrac{n(n-1)}{2}$, where $n \geq 1$:

Basis step When $n = 1, h(1) = \dfrac{1 \cdot 0}{2} = 0$, which agrees with the initial condition. So the formula holds when $n = 1$.

Induction step Assume $h(k) = \dfrac{k(k-1)}{2}$ for any $k \geq 1$. Then:

$h(k+1) = h(k) + k$, by the recurrence relation

$$= \frac{k(k-1)}{2} + k, \qquad \text{by the induction hypothesis}$$
$$= \frac{k(k+1)}{2}$$

Therefore, if the formula holds for $n = k$, it also holds for $n = k + 1$. Thus, by PMI, the result holds for $n \geq 1$. ■

More generally, using iteration we can solve the recurrence relation

$$a_n = a_{n-1} + f(n) \tag{5.5}$$

as follows:

$$\begin{aligned} a_n &= a_{n-1} + f(n) \\ &= [a_{n-2} + f(n-1)] + f(n) = a_{n-2} + f(n-1) + f(n) \\ &= [a_{n-3} + f(n-2)] + f(n-1) + f(n) \\ &= a_{n-3} + f(n-2) + f(n-1) + f(n) \\ &\vdots \\ &= a_0 + \sum_{i=1}^{n} f(i) \end{aligned} \tag{5.6}$$

You can verify that this is the actual solution of the recurrence relation (5.5).

For example, in the handshake problem $f(n) = n - 1$ and $h(0) = 0$, so the solution of the recurrence relation is

$$\begin{aligned} h(n) &= h(0) + \sum_{i=1}^{n} f(i) = 0 + \sum_{i=1}^{n} (i-1) \\ &= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}, \quad n \geq 1 \end{aligned}$$

which is exactly the solution obtained in the example.

EXAMPLE 5.11 Solve the recurrence relation in Example 5.6.

SOLUTION:
Notice that a_n can be redefined as

$$a_n = a_{n-1} + \frac{n(n+1)}{2}, \quad n \geq 1$$

where $a_0 = 0$. Comparing this with recurrence relation (5.5), we have $f(n) = \dfrac{n(n+1)}{2}$. Therefore, by Equation (5.6),

$$\begin{aligned}
a_n &= a_0 + \sum_{i=1}^{n} f(i) \\
&= a_0 + \sum_{i=1}^{n} \frac{i(i+1)}{2} = 0 + \frac{1}{2}\sum_{i=1}^{n}(i^2 + i) \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i\right) \\
&= \frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right] \\
&= \frac{n(n+1)}{2}\left(\frac{2n+1}{6} + \frac{1}{2}\right) = \frac{n(n+1)}{2} \cdot \frac{2n+4}{6} \\
&= \frac{n(n+1)(n+2)}{6}, \quad n \geq 0
\end{aligned}$$

■

The following illustration of the iterative method brings us again to the Tower of Brahma puzzle.

EXAMPLE 5.12 Recall from Example 5.4 that the number of moves needed to transfer n disks from peg X to peg Z is given by

$$\begin{aligned}
b_1 &= 1 \\
b_n &= 2b_{n-1} + 1, n \geq 2
\end{aligned}$$

Solve this recurrence relation.

SOLUTION:

Step 1 To predict a formula for b_n:
Using iteration,

$$\begin{aligned}
b_n &= 2b_{n-1} + 1 \\
&= 2[2b_{n-2} + 1] + 1 = 2^2 b_{n-2} + 2 + 1 \\
&= 2^2[2b_{n-3} + 1] + 2 + 1 = 2^3 b_{n-3} + 2^2 + 2 + 1 \\
&\vdots \\
&= 2^{n-1} b_1 + 2^{n-2} + \cdots + 2^2 + 2 + 1 \\
&= 2^{n-1} + 2^{n-2} + \cdots + 2 + 1 \\
&= 2^n - 1, \quad \text{by Exercise 8 in Section 4.4.}
\end{aligned}$$

Step 2 You may prove by induction that $b_n = 2^n - 1$, where $n \geq 1$. ■

More generally, you may verify that the solution of the recurrence relation $a_n = ca_{n-1} + 1$, where c is a constant ($\neq 1$), is

$$a_n = c^n a_0 + \frac{c^n - 1}{c - 1}$$

For instance, in Example 5.12, $b_0 = 0$ and $c = 2$, so

$$b_n = 2^n \cdot 0 + \frac{2^n - 1}{2 - 1} = 2^n - 1$$

as expected.

Let us pursue Example 5.12 a bit further. Suppose there are 64 disks at peg X, as in the original puzzle, and it takes 1 second to move a disk from one peg to another. Then it takes a total of $2^{64} - 1$ seconds to solve the puzzle.

To get an idea how incredibly large this total is, notice that there are about $365 \cdot 24 \cdot 60 \cdot 60 = 31{,}536{,}000$ seconds in a year. Therefore,

$$\begin{aligned} \text{Total time taken} &= 2^{64} - 1 \text{ seconds} \\ &\approx 1.844674407 \times 10^{19} \text{ seconds} \\ &\approx 5.84942417 \times 10^{11} \text{ years} \\ &\approx 600 \text{ billion years!} \end{aligned}$$

Intriguingly, according to some estimates, the universe is only about 18 billion years old.

Exercises 5.2

Using the iterative method, predict a solution to each recurrence relation satisfying the given initial condition.

1. $s_0 = 1$
$s_n = 2s_{n-1}, n \geq 1$

2. $a_1 = 1$
$a_n = a_{n-1} + n, n \geq 2$

3. $a_0 = 1$
$a_n = a_{n-1} + n, n \geq 1$

4. $a_1 = 1$
$a_n = a_{n-1} + (2n - 1), n \geq 2$

5. $a_0 = 0$
$a_n = a_{n-1} + 4n, n \geq 1$

6. $s_1 = 1$
$s_n = s_{n-1} + n^3, n \geq 2$

7. $s_1 = 1$

$s_n = s_{n-1} + n^2, n \geq 2$

8. $a_1 = 1$

$a_n = 2a_{n-1} + (2^n - 1), n \geq 2$

9–16. Using induction, verify the solutions to Exercises 1–8.

17. Using the data in Example 5.2, show that the compound amount Judy will receive at the end of n years is given by $A(n) = 1000(1.08)^n$, where $n \geq 0$.

Use the recursive definition of f_n in Example 5.5 to answer Exercises 18 and 19.

18. Predict a formula for f_n.

19. Prove that the formula holds for $n \geq 1$.

20. Using induction, establish the explicit formula for b_n in Example 5.12.

Using induction, prove that each is a solution to the corresponding recurrence relation, where c is a constant and $f(n)$ a function of n.

21. $a_n = a_0 + \sum_{i=1}^{n} f(i), a_n = a_{n-1} + f(n)$

22. $a_n = c^n a_0 + \dfrac{c^n - 1}{c - 1}, a_n = ca_{n-1} + 1$ (assume $c \neq 1$)

23. $a_n = c^n a_0 + \sum_{i=1}^{n} c^{n-i} f(i), a_n = ca_{n-1} + f(n)$

Let a_n denote the number of times the statement $x \leftarrow x + 1$ is executed by the following loops.

```
for i = 1 to n do
  for j = 1 to ⌊i/2⌋ do
    x ← x + 1
```

24. Define a_n recursively.

25. Show that $a_n = \begin{cases} 0 & \text{if } n = 1 \\ a_{n-1} + n/2 & \text{if } n > 1 \text{ and even} \\ a_{n-1} + (n-1)/2 & \text{if } n > 1 \text{ and odd} \end{cases}$

26. Solve the recurrence relation satisfied by a_n.

Let a_n denote the number of times the statement $x \leftarrow x + 1$ is executed by the following **for** loops:

```
for i = 1 to n do
  for j = 1 to ⌈i/2⌉ do
    x ← x + 1
```

27. Define a_n recursively.

28. Show that $a_n = \begin{cases} 1 & \text{if } n = 1 \\ a_{n-1} + n/2 & \text{if } n > 1 \text{ and even} \\ a_{n-1} + (n+1)/2 & \text{if } n > 1 \text{ and odd} \end{cases}$

29. Solve the recurrence relation satisfied by a_n.

Let a_n denote the number of times the statement $x \leftarrow x + 1$ is executed by the nested **for** loops in Exercise 35 in Section 4.4.

30. Define a_n recursively.

31. Solve the recurrence relation satisfied by a_n.

32–33. Redo Exercises 30 and 31 using the loops in Exercise 36 in Section 4.4.

34–35. Redo Exercises 30 and 31 using the loops in Exercise 37 in Section 4.4.

36–37. Redo Exercises 30 and 31 using the loops in Exercise 38 in Section 4.4.

Let t_n denote the nth triangular number.

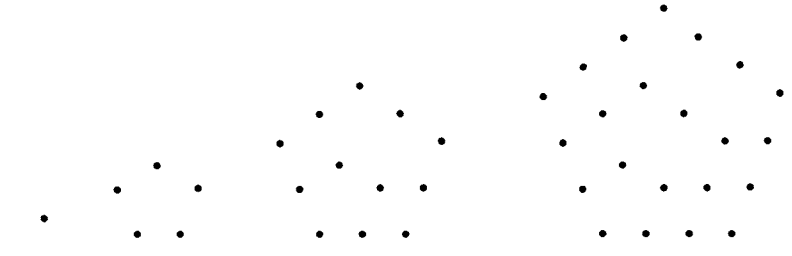
38. Define t_n recursively.

39. Find an explicit formula for t_n.

40. Prove that $8t_n + 1$ is a perfect square.

The nth **pentagonal number** p_n is obtained from its predecessor by adding three rows of dots plus one. The first four pentagonal numbers are represented pictorially in Figure 5.11.

Figure 5.11



41. Represent p_5 pictorially.

42–43. Redo Exercises 38 and 39 using p_n.

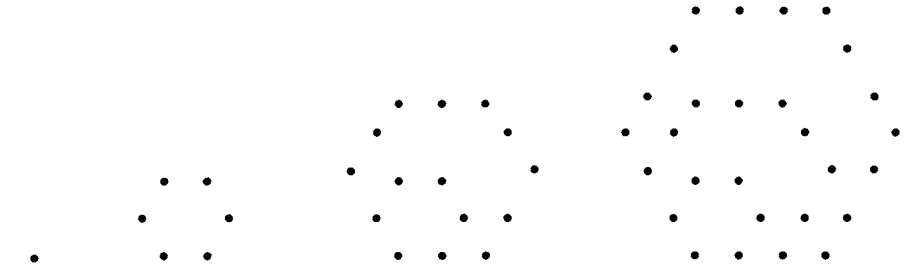
The nth **hexagonal number** h_n is obtained from its predecessor by adding four rows of dots plus one dot. The first four hexagonal numbers are shown pictorially in Figure 5.12.

44–46. Redo Exercises 41–43 using h_n.

47. Prove that $h_n = p_n + t_n - n$, using the explicit formulas for p_n and t_n.

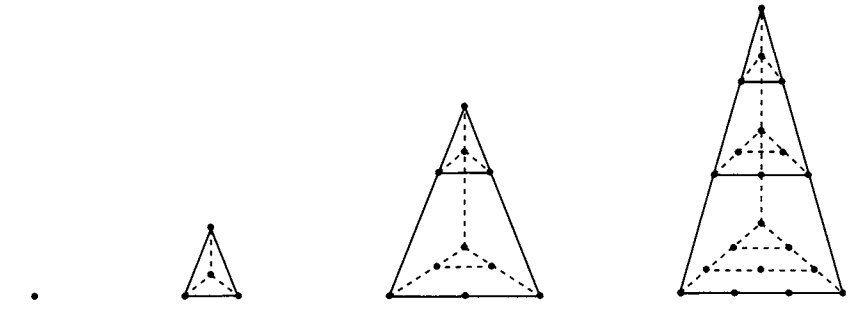
48. Prove that $h_n = p_n + t_n - n$, using the recurrence relations for p_n and t_n.

Figure 5.12



Triangular pyramidal numbers T_n (or **tetrahedral numbers**) are positive integers that can be represented by triangular pyramidal shapes. The first four tetrahedral numbers are 1, 4, 10, and 20; see Figure 5.13.

Figure 5.13



49. Define T_n recursively.

50. Conjecture an explicit formula for T_n.

51. Establish the formula in Exercise 50.

Square pyramidal numbers S_n are positive integers that can be represented by pyramidal shapes, where the base is a square. The first four square pyramidal numbers are 1, 5, 14, and 30; see Figure 5.14.

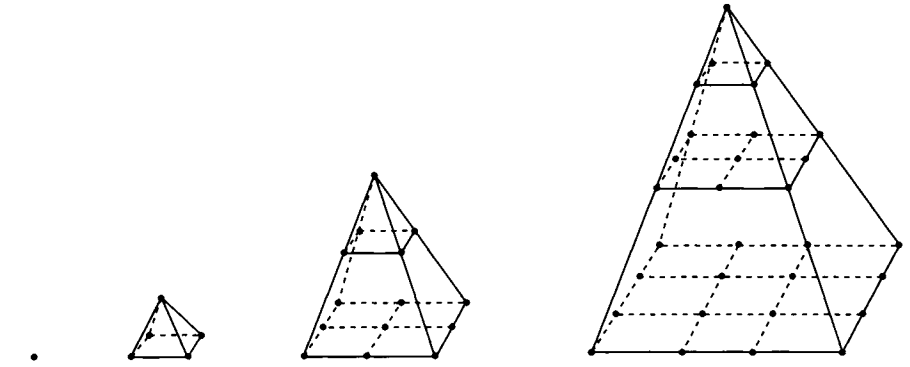
52–54. Redo Exercises 49–51 with S_n.

Let a_n denote the number of subsets of the set $S = \{1, 2, \ldots, n\}$ that contain no consecutive integers, where $n \geq 0$. When $n = 0$, $S = \emptyset$.[†] Compute each.

55. a_0 **56.** a_1 **57.** a_2 **58.** a_3

[†]Proposed by Irving Kaplansky of The University of Chicago.

Figure 5.14



59. Define a_n recursively.

60. Solve the recurrence relation satisfied by a_n.

Suppose we introduce a mixed pair of 1-month-old rabbits into a large enclosure on the first day of a certain month. By the end of each month, the rabbits become mature and each pair produces $k-1$ mixed pairs of offspring at the beginning of the following month. (*Note*: $k \geq 2$.) For instance, at the beginning of the second month, there is one pair of 2-month-old rabbits and $k-1$ pairs of 0-month-olds; at the beginning of the third month, there is one pair of 3-month-olds, $k-1$ pairs of 1-month-olds, and $k(k-1)$ pairs of 0-month-olds. Assume the rabbits are immortal. Let a_n denote the average age of the rabbit pairs at the beginning of the nth month. (P. Filipponi, 1990)

****61.** Define a_n recursively.

****62.** Predict an explicit formula for a_n.

****63.** Prove the formula in Exercise 64.

64. (For those familiar with the concept of limits) Find $\lim_{n\to\infty} a_n$.

5.3 Solving Recurrence Relations Revisited

Unfortunately, the iterative method illustrated in the preceding section can be applied to only a small and simple class of recurrence relations. The present section develops a method for solving two large, important classes of recurrence relations.

Linear Homogeneous Recurrence Relations with Constant Coefficients (LHRRWCCs)

A *k*th-order linear homogeneous recurrence relation with constant coefficients is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} \tag{5.7}$$

where $c_1, c_2, \ldots, c_k \in \mathbb{R}$ and $c_k \neq 0$.

First, a few words of explanation: The term *linear* means that every term on the RHS of Equation (5.7) contains at most the first power of any predecessor a_i. A recurrence relation is *homogeneous* if every term on the RHS is a multiple of some a_i; in other words, the relation is satisfied by the sequence $\{0\}$; that is, $a_n = 0$ for every n. All coefficients c_i are constants. Since a_n depends on its k immediate predecessors, the *order* of the recurrence relation is k. Accordingly, to solve a kth-order LHRRWCC, we will need k initial conditions, say, $a_0 = C_0, a_1 = C_1, \ldots, a_{k-1} = C_{k-1}$.

The next example illustrates in detail the various terms in this definition.

EXAMPLE 5.13

- The recurrence relation $s_n = 2s_{n-1}$ is a LHRRWCC. Its order is one.
- The recurrence relation $a_n = na_{n-1}$ is linear and homogeneous. But the coefficient on the RHS is not a constant. Therefore, it is not a LHRRWCC.
- $h_n = h_{n-1} + (n - 1)$ is a linear recurrence relation. But it is not homogeneous because of the term $n - 1$.
- The recurrence relation $a_n = a_{n-1}^2 + 3a_{n-2}$ is homogeneous. But it is not linear since the power of a_{n-1} is 2.
- $a_n = a_{n-1} + 2a_{n-2} + 3a_{n-6}$ is a LHRRWCC of order six. ■

Before we discuss solving second-order LHRRWCCs, notice that the solution of the recurrence relation $s_n = 2s_{n-1}$, where $s_0 = 1$, is $s_n = 2^n$, $n \geq 0$ (see Exercise 1 in Section 5.2). More generally, you may verify that the solution of the recurrence relation $a_n = \alpha a_{n-1}$, where $a_0 = c$, is $a_n = c\alpha^n$, $n \geq 0$.

We now turn our attention to the second-order LHRRWCC

$$a_n = aa_{n-1} + ba_{n-2} \tag{5.8}$$

where a and b are nonzero constants. If it has a nonzero solution of the form $c\alpha^n$, then $c\alpha^n = ac\alpha^{n-1} + bc\alpha^{n-2}$. Since $c\alpha \neq 0$, this yields $\alpha^2 = a\alpha + b$; that is, $\alpha^2 - a\alpha - b = 0$, so α must be a solution of the **characteristic equation**

$$x^2 - ax - b = 0 \tag{5.9}$$

of the recurrence relation (5.8). The roots of Equation (5.9) are the **characteristic roots** of recurrence relation (5.8).

Theorems 5.2 through 5.4 show how characteristic roots help solve LHRRWCCs.

THEOREM 5.2 Let α and β be the distinct (real or complex) solutions of the equation $x^2 - ax - b = 0$, where $a, b \in \mathbb{R}$ and $b \neq 0$. Then every solution of the LHRRWCC $a_n = aa_{n-1} + ba_{n-2}$, where $a_0 = C_0$ and $a_1 = C_1$, is of the form $a_n = A\alpha^n + B\beta^n$ for some constants A and B.

PROOF:
The proof consists of two parts:

- First, we will show that $a_n = A\alpha^n + B\beta^n$ is a solution of the recurrence relation for any constants A and B.
- We will then find the values of A and B satisfying the given initial conditions.

First, notice that since α and β are solutions of equation (5.9), $\alpha^2 = a\alpha + b$ and $\beta^2 = a\beta + b$.

- *To show that $a_n = A\alpha^n + B\beta^n$ is a solution of the recurrence relation:*

$$\begin{aligned} aa_{n-1} + ba_{n-2} &= a(A\alpha^{n-1} + B\beta^{n-1}) + b(A\alpha^{n-2} + B\beta^{n-2}) \\ &= A\alpha^{n-2}(a\alpha + b) + B\beta^{n-2}(a\beta + b) \\ &= A\alpha^{n-2} \cdot \alpha^2 + B\beta^{n-2} \cdot \beta^2 \\ &= A\alpha^n + B\beta^n \\ &= a_n \end{aligned}$$

  Thus $a_n = A\alpha^n + B\beta^n$ is a solution of the recurrence relation (5.8).

- Secondly, let $a_n = A\alpha^n + B\beta^n$ be a solution of (5.8). To find the values of A and B, notice that the conditions $a_0 = C_0$ and $a_1 = C_1$ yield the following linear system:

$$C_0 = A + B \tag{5.10}$$

$$C_1 = A\alpha + B\beta \tag{5.11}$$

  Solving this system, we get (Verify.)

$$A = \frac{C_1 - C_0\beta}{\alpha - \beta} \text{ and } \frac{C_0\alpha - C_1}{\alpha - \beta} \text{ (Remember, } \alpha \neq \beta.)$$

With these values for A and B, a_n satisfies the initial conditions and the recurrence relation. Since the recurrence relation and the initial conditions determine a unique sequence, $\{a_n\}$, $a_n = A\alpha^n + B\beta^n$ is indeed the unique solution of the recurrence relation. ■

A few interesting observations:

- The solutions α and β are nonzero, since $\alpha = 0$, for instance, would imply that $b = 0$.
- Theorem 5.2 *cannot* be applied if $\alpha = \beta$. However, it works even if α and β are complex numbers.
- The solutions α^n and β^n are the **basic solutions** of the recurrence relation. In general, the number of basic solutions equals the order of the recurrence relation. The **general solution** $a_n = A\alpha^n + B\beta^n$ is a **linear combination** of the basic solutions. The particular solution is obtained by selecting A and B in such a way that the initial conditions are satisfied, as in Theorem 5.2.

The next three examples illustrate how to solve second-order LHRRWCCs using their characteristic equations.

EXAMPLE 5.14 Solve the recurrence relation $a_n = 5a_{n-1} - 6a_{n-2}$, where $a_0 = 4$ and $a_1 = 7$.

SOLUTION:

- *To find the general solution of the recurrence relation*:
  The characteristic equation of the recurrence relation is $x^2 - 5x + 6 = 0$; the characteristic roots are 2 and 3. Therefore, by Theorem 5.2, the general solution of the recurrence relation is $a_n = A \cdot 2^n + B \cdot 3^n$. (This solution is used in Examples 5.19 and 5.20.)
- *To find the values of A and B*:
  Using the initial conditions we find:

$$a_0 = A + B = 4$$
$$a_1 = 2A + 3B = 7$$

  Solving this linear system yields $A = 5$ and $B = -1$ (Verify this.).

Thus the solution of the recurrence relation satisfying the given conditions is $a_n = 5 \cdot 2^n - 3^n$, $n \geq 0$. ■

The next example finds an explicit formula for the nth Fibonacci number F_n, which we have been waiting for.

EXAMPLE 5.15 Solve the Fibonacci recurrence relation $F_n = F_{n-1} + F_{n-2}$, where $F_1 = 1 = F_2$.

SOLUTION:

The characteristic equation of the recurrence relation is $x^2 - x - 1 = 0$, and its solutions are $\alpha = \dfrac{1+\sqrt{5}}{2}$ and $\beta = \dfrac{1-\sqrt{5}}{2}$. You may verify $\alpha + \beta = 1$ and $\alpha\beta = -1$.

The general solution is $F_n = A\alpha^n + B\beta^n$. To find A and B, we have:

$$F_1 = A\alpha + B\beta = 1$$
$$F_2 = A\alpha^2 + B\beta^2 = 1$$

Solving these two equations, we get (Verify):

$$A = \frac{\alpha}{1+\alpha^2} = \frac{(1+\sqrt{5})/2}{(5+\sqrt{5})/2} = \frac{1+\sqrt{5}}{5+\sqrt{5}}$$
$$= \frac{(1+\sqrt{5})(5-\sqrt{5})}{(5+\sqrt{5})(5-\sqrt{5})} = \frac{5+5\sqrt{5}-\sqrt{5}-5}{25-5} = \frac{1}{\sqrt{5}}$$

and similarly $B = \dfrac{\beta}{1+\beta^2} = -\dfrac{1}{\sqrt{5}}$ (Verify this.).

Thus the solution of the recurrence relation satisfying the given conditions is

$$a_n = \frac{\alpha^n - \beta^n}{\sqrt{5}} = \frac{\alpha^n - \beta^n}{\alpha - \beta}$$

which is the *Binet form* for the nth Fibonacci number F_n. (See Example 5.26 for a different method.) ■

The next example, proposed by Irving Kaplansky of The University of Chicago, also illustrates solving second order LHRRWCCs and is closely related to Example 5.15.

EXAMPLE 5.16 Let a_n denote the number of subsets of the set $S = \{1, 2, \ldots, n\}$ that do not contain consecutive integers, where $n \geq 0$. When $n = 0$, $S = \emptyset$. Find an explicit formula for a_n.

SOLUTION:

To get an idea about a_n, let us find its value for $n = 0$, 1, 2, 3, and 4 by constructing a table, as in Table 5.3. It appears from the table that a_n is a Fibonacci number and $a_n = F_{n+2}$.

Table 5.3

n	Subsets of S that do not contain consecutive integers	a_n
0	$\emptyset$,	1
1	$\emptyset$, {1}	2
2	$\emptyset$, {1}, {2}	3
3	$\emptyset$, {1}, {2}, {3}, {1,3}	5
4	$\emptyset$, {1}, {2}, {3}, {4}, {1,3}, {1,4}, {2,4}	8
		↑ F_{n+2}

We shall, in fact, prove that $a_n = F_{n+2}$ in two steps: First we shall define a_n recursively and then solve the recurrence relation to obtain this explicit formula.

- *To define a_n recursively*:
  From Table 5.3, $a_0 = 1$ and $a_1 = 2$. So let $n \geq 2$. Let A be a subset of S that does not contain two consecutive integers. Then either $n \in A$ or $n \notin A$.

Case 1 Suppose $n \in A$. Then $n - 1 \notin A$. By definition, $S^* = \{1, 2, \ldots, n-2\}$ has a_{n-2} subsets not containing two consecutive integers. Add n to each of the subsets. The resulting sets are subsets of S satisfying the desired property, so S has a_{n-2} such subsets.

Case 2 Suppose $n \notin A$. By definition, there are a_{n-1} such subsets of S having the required property.
Since these two cases are mutually exclusive, by the addition principle, $a_n = a_{n-1} + a_{n-2}$.

Thus a_n can be defined recursively as

$$a_0 = 1, a_1 = 2$$
$$a_n = a_{n-1} + a_{n-2}, \quad n \geq 2.$$

- *To solve the recurrence relation*:

  This recurrence relation is exactly the same as the Fibonacci one with the initial conditions $a_0 = 1$, $a_1 = 2$. So instead of going through a complete solution, as in Example 5.15, notice that this definition yields the Fibonacci numbers 1, 2, 3, 5, 8, ... It follows that $a_n = F_{n+2}$, $n \geq 0$.

Using the values of α and β from Example 5.15,

$$a_n = F_{n+2} = \frac{\alpha^{n+2} - \beta^{n+2}}{\alpha - \beta}, \quad n \geq 0$$

(Verify this. See Exercise 13.) ■

Theorem 5.2 does not work if the characteristic roots α and β are equal, that is, if α is a root with degree of multiplicity two. The following theorem, however, comes to our rescue. It shows that, in addition to α^n, $n\alpha^n$ is a basic solution.

THEOREM 5.3 Let $a, b \in \mathbb{R}$ and $b \neq 0$. Let α be a real or complex solution of the equation $x^2 - ax - b = 0$ with degree of multiplicity two. Then $a_n = A\alpha^n + Bn\alpha^n$ is the general solution of the LHRRWCC $a_n = aa_{n-1} + ba_{n-2}$.

PROOF:
Since α is a root of the equation $x^2 - ax - b = 0$ with degree of multiplicity two,

$$x^2 - ax - b = (x - \alpha)^2$$
$$= x^2 - 2\alpha x + \alpha^2$$

Therefore,

$$a = 2\alpha \quad \text{and} \quad b = -\alpha^2 \tag{5.12}$$

- *To show that $a_n = n\alpha^n$ satisfies the recurrence relation*:
  Notice that

$$\begin{aligned} aa_{n-1} + ba_{n-2} &= a[(n-1)\alpha^{n-1}] + b[(n-2)\alpha^{n-2}] \\ &= 2\alpha[(n-1)\alpha^{n-1}] + (-\alpha^2)[(n-2)\alpha^{n-2}] \quad \text{by (5.12)} \\ &= \alpha^n[2(n-1) - (n-2)] \\ &= n\alpha^n = a_n \end{aligned}$$

  Therefore, $n\alpha^n$ is a solution of the recurrence relation.

Then $a_n = A\alpha^n + Bn\beta^n$ is the general solution of the given recurrence relation, where A and B are selected in such a way that the initial conditions are satisfied. (The values of A and B can be found using initial conditions, as in Theorem 5.2.) ■

The next example illustrates Theorem 5.3.

EXAMPLE 5.17 Solve the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$, where $a_0 = 2$ and $a_1 = 3$.

SOLUTION:
The characteristic equation of the recurrence relation is $x^2 - 6x + 9 = 0$; its solution is 3 with degree of multiplicity two. Therefore, by Theorem 5.3, the general solution of the recurrence relation is $a_n = A \cdot 3^n + B \cdot n3^n$. (We use this in Example 5.21.)

The initial conditions $a_0 = 2$ and $a_1 = 3$ yield the equations

$$A \cdot 3^0 + B \cdot 0 \cdot 3^0 = 2$$

and

$$A \cdot 3 + B \cdot 1 \cdot 3 = 3.$$

Solving these equations, we get $A = 2$ and $B = -1$. (Verify).

Thus the solution of the recurrence relation satisfying the given conditions is $a_n = 2 \cdot 3^n - n \cdot 3^n$, $n \geq 0$. ■

Theorems 5.2 and 5.3 can be combined to yield the following general result.

THEOREM 5.4 Let α be a characteristic root of the LHRRWCC (5.7).

- If the degree of multiplicity of α is 1, then α^n is a basic solution of the LHRRWCC.
- If the degree of multiplicity of α is m, then $\alpha^n, n\alpha^n, \ldots, n^{m-1}\alpha^n$ are basic solutions of the LHRRWCC. (*Note*: A kth-order LHRRWCC has k basic solutions.)
- The general solution of the LHRRWCC is a linear combination of all basic solutions. ■

The following example illustrates this general theorem.

EXAMPLE 5.18 Solve the recurrence relation $a_n = 7a_{n-1} - 13a_{n-2} - 3a_{n-3} + 18a_{n-4}$, where $a_0 = 5, a_1 = 3, a_2 = 6$, and $a_3 = -21$.

SOLUTION:
The characteristic equation of the LHRRWCC is $x^4 - 7x^3 + 13x^2 + 3x - 18 = 0$. Since $x^4 - 7x^3 + 13x^2 + 3x - 18 = (x + 1)(x - 2)(x - 3)^2$, the characteristic roots are:

$$-1 \text{ and } 2 \text{ with degree of multiplicity one each}$$

and

$$3 \text{ with degree of multiplicity two}$$

Since 3 is a root with degree of multiplicity two, it yields two basic solutions, 3^n and $n3^n$. Thus the general solution of the LHRRWCC is a linear combination of the basic solutions $(-1)^n, 2^n, 3^n$, and $n3^n$; that is, $a_n = A(-1)^n + B2^n + C3^n + Dn3^n$.

To find the values of A, B, C, and D:
Since $a_0 = 5, a_1 = 3, a_2 = 6$, and $a_3 = -21$, we have

$$\begin{aligned} A + B + C &= 5 \\ -A + 2B + 3C + 3D &= 3 \\ A + 4B + 9C + 18D &= 6 \\ \text{and} \quad -A + 8B + 27C + 81D &= -21 \end{aligned}$$

Solving this linear system, we get $A = 2 = C, B = 1$, and $D = -1$ (Verify this.). Thus the solution of the LHRRWCC satisfying the initial conditions is $a_n = 2(-1)^n + 2^n + 2 \cdot 3^n - n3^n, n \geq 0$. ■

The technique of solving LHRRWCCs cannot be applied to the seemingly simple recurrence relations $f_n = f_{n-1} + n$ (Example 5.5) and $b_n = 2b_{n-1} + 1$ (Example 5.4), which are linear, but nonhomogeneous. So we now turn to solving **linear nonhomogeneous recurrence relations with constant coefficients** (LNHRRWCCs).

LNHRRWCCs

The *general form* of a LNHRRWCC is

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} + f(n) \tag{5.13}$$

where $c_1, c_2, \ldots, c_k \in \mathbb{R}, c_k \neq 0$, and $f(n)$ is *not* identically zero. Its solution depends on that of the **associated linear homogeneous recurrence relation with constant coefficients** (ALHRRWCCs)

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} \tag{5.14}$$

we studied earlier.

Solving LNHRRWCCs

To solve the LNHRRWCCs (5.13), let $a_n^{(h)}$ denote the general solution of the ALHRRWCCs (5.14). Suppose we know some solution $a_n^{(p)}$ of the recurrence relation (5.13); $a_n^{(p)}$ is a **particular solution** of the LNHRRWCCs (5.13). Then the **general solution** of (5.13) is given by

$$a_n = a_n^{(h)} + a_n^{(p)}$$

This fact is confirmed by the following theorem; we leave its proof as an exercise (see Exercise 44).

THEOREM 5.5 Let $a_n^{(h)}$ denote the general solution of the ALHRRWCCs (5.14) and $a_n^{(p)}$ a particular solution of the LNHRRWCC (5.13). Then $a_n = a_n^{(h)} + a_n^{(p)}$ is the general solution of the LNHRRWCCs (5.13). ■

It follows from this theorem that solving the LNHRRWCCs (5.13) depends on finding a particular solution $a_n^{(p)}$. Although no general algorithm exists for solving an arbitrary LNHRRWCCs, two special cases can be handled fairly easily. When $f(n)$ is a polynomial in n or is of the form $C\alpha^n$, a particular solution can be extracted with ease, as the next two examples demonstrate, where C and α are constants. The techniques we employ are similar to those used to solve linear nonhomogeneous differential equations.

EXAMPLE 5.19 Solve the LNHRRWCCs $a_n = 5a_{n-1} - 6a_{n-2} + 8n^2$, where $a_0 = 4$ and $a_1 = 7$.

SOLUTION:

It follows from Example 5.14 that the general solution of the ALHRRWCCs $a_n = 5a_{n-1} - 6a_{n-2}$ is given by $a_n^{(h)} = A \cdot 2^n + B \cdot 3^n$. Since $f(n) = 8n^2$ is a quadratic polynomial in n, it seems reasonable to look for a particular solution of the same form, say, $a_n = an^2 + bn + c$. Then the given recurrence relation yields

$$\begin{aligned} an^2 + bn + c &= 5[a(n-1)^2 + b(n-1) + c] - 6[a(n-2)^2 + b(n-2) + c] + 8n^2 \\ &= (8-a)n^2 + (14a - b)n - 19a + 7b - c \end{aligned}$$

Equating the coefficients of like terms, we get the linear system:

$$\begin{aligned} a &= 8 - a \\ b &= 14a - b \\ c &= -19a + 7b - c \end{aligned}$$

Solving the system, we get $a = 4, b = 28$, and $c = 60$ (Verify). We now claim that $a_n^{(p)} = 4n^2 + 28n + 60$ is a particular solution (Verify).

Thus, by Theorem 5.5, the general solution of the given recurrence relation is

$$\begin{aligned} a_n &= a_n^{(h)} + a_n^{(p)} \\ &= A \cdot 2^n + B \cdot 3^n + 4n^2 + 28n + 60 \end{aligned}$$

Using the two given initial conditions, this yields the linear system:

$$\begin{aligned} A + B &= -56 \\ 2A + 3B &= -85 \end{aligned}$$

This yields $A = -83$ and $B = 27$ (Verify this also.).

Thus the desired solution is

$$a_n = (-83) \cdot 2^n + 27 \cdot 3^n + 4n^2 + 28n + 60, \quad n \geq 0$$ ■

The next example illustrates how to solve the LNHRRWCCs (5.13) when $f(n)$ is of the form $C\alpha^n$, where C and α are constants.

EXAMPLE 5.20 Solve the LNHRRWCCs $a_n = 5a_{n-1} - 6a_{n-2} + 3 \cdot 5^n$, where $a_0 = 4$ and $a_1 = 7$.

SOLUTION:

As in Example 5.19, the general solution of the ALHRRWCCs $a_n = 5a_{n-1} - 6a_{n-2}$ is given by $a_n^{(h)} = A \cdot 2^n + B \cdot 3^n$. Since $f(n) = 3 \cdot 5^n$, we search for a particular solution of the form $a_n = c \cdot 5^n$. Then we must have

$$c \cdot 5^n = 5(c \cdot 5^{n-1}) - 6(c \cdot 5^{n-2}) + 3 \cdot 5^n$$

Canceling 5^{n-2} from both sides, the resulting equation yields $c = 25/2$. We now claim that $a_n = (25/2)5^n$ is a particular solution of the recurrence relation (Verify this.).

Thus the general solution of the LNHRRWCCs is

$$a_n = A \cdot 2^n + B \cdot 3^n + (25/2)5^n$$

Using the initial conditions, we get the linear system:

$$\begin{aligned} A + B &= -17/2 \\ 2A + 3B &= -111/2 \end{aligned}$$

Solving this system, we get $A = 30$ and $B = -77/2$ (Verify this.).

Thus the solutions of the given recurrence relation are given by

$$a_n = (30) \cdot 2^n - (77/2) \cdot 3^n + (25/2) \cdot 5^n, \quad n \geq 0$$

(Verify this also.) ■

An important observation: In this example, notice that the 5 in $f(n)$ is *not* a characteristic root of the ALHRRWCCs. If it were, we would have needed to make adjustments in our search for a particular solution, as in Theorem 5.3. We shall pursue this case shortly.

The following theorem justifies the techniques demonstrated in these two examples; we omit its proof in the interest of brevity.

THEOREM 5.6 In the LNHRRWCCs (5.13), suppose $f(n) = (b_k n^k + b_{k-1}n^{k-1} + \cdots + b_1 n + b_0)\alpha^n$. If α is *not* a characteristic root of the ALHRRWCCs (5.14), then a particular solution is of the form $(d_k n^k + d_{k-1}n^{k-1} + \cdots + d_1 n + d_0)\alpha^n$. If α is a characteristic root with multiplicity m, then a particular solution is of the form $n^m(e_k n^k + e_{k-1}n^{k-1} + \cdots + e_1 n + e_0)\alpha^n$. ■

We conclude this section with the following example, which illustrates this theorem when α is a characteristic root of the ALHRRWCCs.

EXAMPLE 5.21 Solve the LNHRRWCCs $a_n = 6a_{n-1} - 9a_{n-2} + 4(n+1)3^n$, where $a_0 = 2$ and $a_1 = 3$.

SOLUTION:
From Example 5.17, the general solution of the ALHRRWCCs is $a_n^{(h)} = A \cdot 3^n + B \cdot n3^n$, where $n \geq 0$. Since 3 is a characteristic root with multiplicity 2, we search for a particular solution of the form $n^2(cn + d)3^n$, where the constants c and d are to be determined. Then we must have

$$\begin{aligned} n^2(cn+d)3^n &= 6\{(n-1)^2[c(n-1)+d]3^{n-1}\} \\ &\quad - 9\{(n-2)^2[c(n-2)+d]3^{n-2}\} + 4(n+1)3^n \end{aligned}$$

Equating the coefficients of like terms, this yields $c = 2/3$ and $d = 4$ (Verify); so $a_n^{(p)} = 2n^2(n+6)3^{n-1}$.

Thus the general solution of the recurrence relation is

$$a_n = A \cdot 3^n + B \cdot n3^n + 2n^2(n+6)3^{n-1}, \quad n \geq 0$$

Using the initial conditions, this yields

$$a_n = (6 - 19n) \cdot 3^{n-1} + 2n^2(n+6)3^{n-1}, \quad n \geq 0$$ ■

(You can confirm this.)

Exercises 5.3

Determine if each recurrence relation is a LHRRWCC.

1. $L_n = L_{n-1} + L_{n-2}$
2. $D_n = nD_{n-1} + (-1)^n$
3. $a_n = 1.08a_{n-1}$
4. $b_n = 2b_{n-1} + 1$

5. $a_n = a_{n-1} + n$

6. $a_n = 2a_{n-1} + (2^n - 1)$

7. $a_n = a_{n-1} + 2a_{n-2} + 3a_{n-5}$

8. $a_n = a_{n-1} + 2a_{n-3} + n^2$

Solve each LHRRWCC.

9. $a_n = a_{n-1} + 2a_{n-2}, a_0 = 3, a_1 = 0$

10. $a_n = 5a_{n-1} - 6a_{n-2}, a_0 = 4, a_1 = 7$

11. $a_n = a_{n-1} + 6a_{n-2}, a_0 = 5, a_1 = 0$

12. $a_n = 4a_{n-2}, a_0 = 2, a_1 = -8$

13. $a_n = a_{n-1} + a_{n-2}, a_0 = 1, a_1 = 2$

14. $a_n = a_{n-1} + a_{n-2}, a_0 = 2, a_1 = 3$

15. $L_n = L_{n-1} + L_{n-2}, L_1 = 1, L_2 = 3$

16. $a_n = 4a_{n-1} - 4a_{n-2}, a_0 = 3, a_1 = 10$

17. $a_n = 6a_{n-1} - 9a_{n-2}, a_0 = 2, a_1 = 3$

18. $a_n = 3a_{n-1} + 4a_{n-2} - 12a_{n-3}, a_0 = 3, a_1 = -7, a_2 = 7$

19. $a_n = 8a_{n-1} - 21a_{n-2} + 18a_{n-3}, a_0 = 0, a_1 = 2, a_2 = 13$

20. $a_n = 7a_{n-1} - 16a_{n-2} + 12a_{n-3}, a_0 = 0, a_1 = 5, a_2 = 19$

21. $a_n = -a_{n-1} + 16a_{n-2} + 4a_{n-3} - 48a_{n-4}, a_0 = 0, a_1 = 16, a_2 = -2, a_3 = 142$

22. $a_n = 13a_{n-2} - 36a_{n-4}, a_0 = 7, a_1 = -6, a_2 = 38, a_3 = -84$

23. $a_n = 9a_{n-1} - 30a_{n-2} + 44a_{n-3} - 24a_{n-4}, a_0 = 5, a_1 = 12, a_2 = 38, a_3 = 126$

24. $a_n = 8a_{n-1} - 24a_{n-2} + 32a_{n-3} - 16a_{n-4}, a_0 = 1, a_1 = 4, a_2 = 44, a_3 = 272$

Find the general form of a particular solution of the LNHRRWCCs (5.13) corresponding to each function $f(n)$.

25. $f(n) = n$

26. $f(n) = 1$

27. $f(n) = 3n^2$

28. $f(n) = 3^n$

29. $f(n) = n2^n$

30. $f(n) = 43n^2 5^n$

Find the general form of a particular solution of the LNHRRWCCs $a_n = 4a_{n-1} - 4a_{n-2} + f(n)$ corresponding to each function $f(n)$.

31. $f(n) = 3 \cdot 2^n$

32. $f(n) = n2^n$

33. $f(n) = 23n^2 2^n$

34. $(17n^3 - 1)2^n$

Solve each LNHRRWCCs.

35. $a_n = 2a_{n-1} + 1, a_0 = 1$

36. $a_n = 7a_{n-1} - 10a_{n-2} + n^2, a_0 = 0, a_1 = 1$

37. $a_n = 7a_{n-1} - 12a_{n-2} + 3^n, a_0 = 0, a_1 = 2$

38. $a_n = 7a_{n-1} - 12a_{n-2} + 3n4^n, a_0 = 0, a_1 = 2$

***39.** $a_n = a_{n-1} + n, a_0 = 1$

***40.** $a_n = a_{n-1} + n - 1, a_1 = 0$

41. Let r_n and s_n be two solutions of the recurrence relation (5.8). Prove that $a_n = r_n + s_n$ is also a solution.

42. Let α be a solution of the equation $x^k - c_1x^{k-1} - \cdots - c_k = 0$. Show that α^n is a solution of LHRRWCC (5.7).

43. Let α be a characteristic root of the LHRRWCC $a_n = aa_{n-1} + ba_{n-2} + ca_{n-3}$ with degree of multiplicity three. Show that $\alpha^n, n\alpha^n, n^2\alpha^n$ are solutions of LHRRWCC.

44. Let $a_n^{(h)}$ denote the general solution of the ALHRRWCCs (5.14) and $a_n^{(p)}$ a particular solution of the LNHRRWCCs (5.13). Prove that $a_n = a_n^{(h)} + a_n^{(p)}$ is the general solution of the LNHRRWCCs (5.13).

5.4 Generating Functions

Generating functions provide a powerful tool for solving LHRRWCCs, as will be seen shortly. They were invented in 1718 by the French mathematician Abraham De Moivre, when he used them to solve the Fibonacci recurrence relation (see Example 5.26). Generating functions can also solve combinatorial problems, as the next chapter shows.

To begin with, notice that the polynomial $1 + x + x^2 + x^3 + x^4 + x^5$ can be written as $\dfrac{x^6 - 1}{x - 1}$. You may verify this by either cross-multiplication, the familiar long division method, or Exercise 8 in Section 4.4. Accordingly, $f(x) = \dfrac{x^6 - 1}{x - 1}$ is called the **generating function** of the sequence of coefficients 1, 1, 1, 1, 1, 1 in the polynomial.

More generally, we make the following definition.

Generating Function

Let $a_0, a_1, a_2, \ldots$ be a sequence of real numbers. Then the function

$$g(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n + \cdots \tag{5.15}$$

is the **generating function** for the sequence $\{a_n\}$. Generating functions for the finite sequence $a_0, a_1, \ldots, a_n$ can also be defined by letting $a_i = 0$ for $i > n$; thus $g(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ is the generating function for the finite sequence $a_0, a_1, \ldots, a_n$.

Abraham De Moivre *(1667–1754), son of a surgeon, was born in Vitry-le-Francois, France. His formal education began at the Catholic village school, and then continued at the Protestant Academy at Sedan and later at Saumur. He did not receive good training in mathematics until he moved to Paris in 1684, where he studied Euclid's later books and other texts.*

Around 1686, De Moivre emigrated to England, where he began his life-long profession, tutoring in mathematics, and mastered Newton's Principia Mathematica. *In 1695 he presented a paper, his first, on Newton's theory of fluxions to the Royal Society of London and 2 years later he was elected a member of the Society. Unfortunately, despite his influential friends, he could not find an academic position. He had to earn a living as a tutor, author, and expert on applications of probability to gambling and annuities.*

He dedicated his first book, a masterpiece, The Doctrine of Chances, *to Newton. His most notable discovery concerns probability theory:* The binomial probability distribution can be approximated by the normal distribution.

De Moivre died in London.

For example,

$$g(x) = 1 + 2x + 3x^2 + \cdots + (n+1)x^n + \cdots$$

is the generating function for the sequence of positive integers and

$$f(x) = 1 + 3x + 6x^2 + \cdots + \frac{n(n+1)}{2}x^2 + \cdots$$

is the generating function for the sequence of triangular numbers. Since

$$\frac{x^n - 1}{x - 1} = 1 + x + x^2 + \cdots + x^{n-1}$$

$g(x) = \dfrac{x^n - 1}{x - 1}$ is the generating function for the sequence of n ones.

A word of caution: The RHS of Equation (5.15) is a **formal power series** in x. The letter x does not represent anything. The various powers x^n of x are simply used to keep track of the corresponding terms a_n of the sequence. In other words, think of the powers x^n as placeholders. Consequently, unlike in calculus, the convergence of the series is of no interest to us.

Equality of Generating Functions

Two generating functions $f(x) = \sum_{n=0}^{\infty} a_n x^n$ and $g(x) = \sum_{n=0}^{\infty} b_n x^n$ are **equal** if $a_n = b_n$ for every $n \geq 0$.

For example, let $f(x) = 1 + 3x + 6x^2 + 10x^3 + \cdots$ and $g(x) = 1 + \frac{2 \cdot 3}{2}x + \frac{3 \cdot 4}{2}x^2 + \frac{4 \cdot 5}{2}x^3 + \cdots$. Then $f(x) = g(x)$.

A generating function we will use frequently is

$$\frac{1}{1 - ax} = 1 + ax + a^2x^2 + \cdots + a^nx^n + \cdots \tag{5.16}$$

Then

$$\frac{1}{1 - x} = 1 + x + x^2 + \cdots + x^n + \cdots \tag{5.17}$$

Can we add and multiply generating functions? Yes! Such operations are performed exactly the same way as polynomials are combined.

Addition and Multiplication of Generating Functions

Let $f(x) = \sum_{n=0}^{\infty} a_n x^n$ and $g(x) = \sum_{n=0}^{\infty} b_n x^n$ be two generating functions. Then

$$f(x) + g(x) = \sum_{n=0}^{\infty} (a_n + b_n)x^n \text{ and } f(x)g(x) = \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n} a_i b_{n-i} \right) x^n$$

For example,

$$\begin{aligned} \frac{1}{(1-x)^2} &= \frac{1}{1-x} \cdot \frac{1}{1-x} \\ &= \left(\sum_{i=0}^{\infty} x^i \right) \left(\sum_{i=0}^{\infty} x^i \right) = \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n} 1 \cdot 1 \right) x^n \\ &= \sum_{n=0}^{\infty} (n+1)x^n \\ &= 1 + 2x + 3x^2 + \cdots + (n+1)x^n + \cdots \end{aligned} \tag{5.18}$$

and

$$\begin{aligned} \frac{1}{(1-x)^3} &= \frac{1}{1-x} \cdot \frac{1}{(1-x)^2} \\ &= \left(\sum_{n=0}^{\infty} x^n \right) \left[\sum_{n=0}^{\infty} (n+1)x^n \right] \\ &= \sum_{n=0}^{\infty} \left[\sum_{i=0}^{n} 1 \cdot (n+1-i) \right] x^n \\ &= \sum_{n=0}^{\infty} [(n+1) + n + \cdots + 1]x^n \end{aligned}$$

$$= \sum_{n=0}^{\infty} \frac{(n+1)(n+2)}{2} x^n$$

$$= 1 + 3x + 6x^2 + 10x^3 + \cdots \tag{5.19}$$

Before exploring how valuable generating functions are in solving LHRRWCCs, we illustrate how the technique of **partial fraction decomposition**, used in integral calculus, enables us to express the quotient $\frac{p(x)}{q(x)}$ of two polynomials $p(x)$ and $q(x)$ as a sum of proper fractions, where $\deg p(x) < \deg q(x)$.[†]

For example,

$$\frac{6x+1}{(2x-1)(2x+3)} = \frac{1}{2x-1} + \frac{2}{2x+3}$$

Partial Fraction Decomposition Rule for $\frac{p(x)}{q(x)}$, where $\deg p(x) < \deg q(x)$

If $q(x)$ has a factor of the form $(ax+b)^m$, then the decomposition contains a sum of the form

$$\frac{A_1}{ax+b} + \frac{A_2}{(ax+b)^2} + \cdots + \frac{A_m}{(ax+b)^m}$$

where A_i is a rational number.

Examples 5.22–5.24 illustrate the partial fraction decomposition technique. We use their results to solve the recurrence relations in Examples 5.25–5.27.

EXAMPLE 5.22 Express $\frac{x}{(1-x)(1-2x)}$ as a sum of partial fractions.

SOLUTION:

Since the denominator contains two linear factors, we let

$$\frac{x}{(1-x)(1-2x)} = \frac{A}{1-x} + \frac{B}{1-2x}$$

To find the constants A and B, multiply both sides by $(1-x)(1-2x)$:

$$x = A(1-2x) + B(1-x)$$

Now give convenient values to x. Setting $x = 1$ yields $A = -1$ and setting $x = 1/2$ yields $B = 1$. (The values of A and B can also be found by equating

[†]$\deg f(x)$ denotes the degree of the polynomial $f(x)$.

coefficients of like terms from either side of the equation and solving the resulting linear system.)

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

(You may verify this by combining the sum on the RHS into a single fraction.) We use this result in Example 5.25. ■

EXAMPLE 5.23 Express $\frac{x}{1-x-x^2}$ as a sum of partial fractions.

SOLUTION:
First, factor $1-x-x^2$:

$$1-x-x^2 = (1-\alpha x)(1-\beta x)$$

where $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1-\sqrt{5}}{2}$. (Notice that $\alpha+\beta = 1, \alpha\beta = -1$, and $\alpha - \beta = \sqrt{5}$.)

Let

$$\frac{x}{1-x-x^2} = \frac{A}{1-\alpha x} + \frac{B}{1-\beta x}$$

Then

$$x = A(1-\beta x) + B(1-\alpha x)$$

Equating coefficients of like terms, we get:

$$\begin{aligned} A + B &= 0 \\ -\beta A - \alpha B &= 1 \end{aligned}$$

Solving this linear system yields $A = \frac{1}{\sqrt{5}} = -B$ (Verify this.).

Thus

$$\frac{x}{(1-x-x^2)} = \frac{1}{\sqrt{5}}\left[\frac{1}{1-\alpha x} - \frac{1}{1-\beta x}\right]$$

We use this result in Example 5.26. ■

EXAMPLE 5.24 Express $\frac{2-9x}{1-6x+9x^2}$ as a sum of partial fractions.

SOLUTION:
Again, factor the denominator:

$$1-6x+9x^2 = (1-3x)^2$$

By the decomposition rule, let

$$\frac{2-9x}{1-6x+9x^2} = \frac{A}{1-3x} + \frac{B}{(1-3x)^2}$$

Then

$$2-9x = A(1-3x)+B$$

This yields $A = 3$ and $B = -1$ (Verify this.).

Thus

$$\frac{2-9x}{1-6x+9x^2} = \frac{3}{1-3x} - \frac{1}{(1-3x)^2}$$

We use this result in Example 5.27. ■

Now we are ready to use partial fraction decompositions and generating functions to solve recurrence relations in the next three examples.

EXAMPLE 5.25 Use generating functions to solve the recurrence relation $b_n = 2b_{n-1} + 1$, where $b_1 = 1$.

SOLUTION:
First, notice that the condition $b_1 = 1$ yields $b_0 = 0$. To find the sequence $\{b_n\}$ that satisfies the recurrence relation, consider the corresponding generating function

$$g(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + \cdots + b_nx^n + \cdots$$

Then

$$2xg(x) = 2b_1x^2 + 2b_2x^3 + \cdots + 2b_{n-1}x^n + \cdots$$

Also,

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots + x^n + \cdots$$

Then

$$\begin{aligned} g(x) - 2xg(x) - \frac{1}{1-x} &= -1 + (b_1 - 1)x + (b_2 - 2b_1 - 1)x^2 + \cdots \\ &\quad + (b_n - 2b_{n-1} - 1)x^n + \cdots \\ &= -1 \end{aligned}$$

since $b_1 = 1$ and $b_n = 2b_{n-1} + 1$ for $n \geq 2$. That is,

$$(1-2x)g(x) = \frac{1}{1-x} - 1 = \frac{x}{1-x}$$

Then

$$g(x) = \frac{x}{(1-x)(1-2x)}$$

$$= -\frac{1}{1-x} + \frac{1}{1-2x}, \text{ by Example 5.22}$$
$$= -\left(\sum_{n=0}^{\infty} x^n\right) + \left(\sum_{n=0}^{\infty} 2^n x^n\right), \text{ by (5.16)}$$
$$= \sum_{n=0}^{\infty} (2^n - 1)x^n$$

But $g(x) = \sum_{n=0}^{\infty} b_n x^n$, so $b_n = 2^n - 1$, $n \geq 1$. (Notice that this is the same solution obtained in Example 5.12.) ■

EXAMPLE 5.26 Using generating functions, solve the Fibonacci recurrence relation $F_n = F_{n-1} + F_{n-2}$, where $F_1 = 1 = F_2$.

SOLUTION:
Notice that the two initial conditions yield $F_0 = 0$. Let

$$g(x) = F_0 + F_1 x + F_2 x^2 + \cdots + F_n x^n + \cdots$$

be the generating function of the Fibonacci sequence. Since the orders of F_{n-1} and F_{n-2} are 1 and 2 less than the order of F_n, respectively, we find $xg(x)$ and $x^2 g(x)$:

$$xg(x) = F_1 x^2 + F_2 x^3 + F_3 x^4 + \cdots + F_{n-1} x^n + \cdots$$
$$x^2 g(x) = F_1 x^3 + F_2 x^4 + F_3 x^5 + \cdots + F_{n-2} x^n + \cdots$$

Then

$$\begin{aligned} g(x) - xg(x) - x^2 g(x) &= F_1 x + (F_2 - F_1)x^2 + (F_3 - F_2 - F_1)x^3 + \cdots \\ &\quad + (F_n - F_{n-1} - F_{n-2})x^n + \cdots \\ &= x \end{aligned}$$

since $F_2 = F_1$ and $F_n = F_{n-1} + F_{n-2}$.

That is,

$$(1 - x - x^2)g(x) = x$$
$$g(x) = \frac{x}{1 - x - x^2}$$
$$= \frac{1}{\sqrt{5}}\left[\frac{1}{1-\alpha x} - \frac{1}{1-\beta x}\right], \text{ by Example 5.23}$$

where $\alpha = \dfrac{1+\sqrt{5}}{2}$ and $\beta = \dfrac{1-\sqrt{5}}{2}$

Then

$$\sqrt{5}g(x) = \frac{1}{1-\alpha x} - \frac{1}{1-\beta x}$$
$$= \sum_{n=0}^{\infty} \alpha^n x^n - \sum_{n=0}^{\infty} \beta^n x^n = \sum_{n=0}^{\infty} (\alpha^n - \beta^n) x^n$$

So

$$g(x) = \sum_{n=0}^{\infty} \frac{(\alpha^n - \beta^n)}{\sqrt{5}} x^n$$

Therefore, by the equality of generating functions,

$$F_n = \frac{\alpha^n - \beta^n}{\sqrt{5}} = \frac{\alpha^n - \beta^n}{\alpha - \beta}$$

(Recall that this is the **Binet form** of F_n.) ■

We close this section with the following example.

EXAMPLE 5.27 Using generating functions, solve the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$, where $a_0 = 2$ and $a_1 = 3$.

SOLUTION:
Let

$$g(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots$$

Then

$$6xg(x) = 6a_0 x + 6a_1 x^2 + 6a_2 x^3 + \cdots + 6a_{n-1} x^n + \cdots$$
$$9x^2 g(x) = 9a_0 x^2 + 9a_1 x^3 + 9a_2 x^4 + \cdots + 9a_{n-2} x^n + \cdots$$

Then

$$g(x) - 6xg(x) + 9x^2 g(x) = a_0 + (a_1 - 6a_0)x + (a_2 - 6a_1 + 9a_0)x^2 + \cdots$$
$$+ (a_n - 6a_{n-1} + 9a_{n-2})x^n + \cdots$$
$$= 2 - 9x$$

using the given conditions. Thus

$$(1 - 6x + 9x^2)g(x) = 2 - 9x$$

Therefore,

$$g(x) = \frac{2 - 9x}{1 - 6x + 9x^2}$$

$$= \frac{3}{1-3x} - \frac{1}{(1-3x)^2}, \text{ by Example 5.24}$$

$$= 3\left(\sum_{n=0}^{\infty} 3^n x^n\right) - \sum_{n=0}^{\infty}(n+1)3^n x^n$$

$$= \sum_{n=0}^{\infty}[3^{n+1} - (n+1)3^n]x^n$$

$$= \sum_{n=0}^{\infty} 3^n(2-n)x^n$$

Thus

$$a_n = (2-n)3^n, \quad n \geq 0$$ ■

The following exercises provide ample practice in this problem-solving technique.

Exercises 5.4

Express each quotient as a sum of partial fractions.

1. $\dfrac{x+7}{(x-1)(x+3)}$

2. $\dfrac{4x^2-3x-25}{(x+1)(x-2)(x+3)}$

3. $\dfrac{5}{1-x-6x^2}$

4. $\dfrac{2+4x}{1+8x+15x^2}$

5. $\dfrac{x(x+2)}{(2+3x)(x^2+1)}$

6. $\dfrac{-2x^2-2x+2}{(x-1)(x^2+2x)}$

7. $\dfrac{x^3+x^2+x+3}{x^4+5x^2+6}$

8. $\dfrac{-x^3+2x^2+x}{x^4+x^3+x+1}$

9. $\dfrac{3x^3-x^2+4x}{x^4-x^3+2x^2-x+1}$

***10.** $\dfrac{x^3+x^2+5x-2}{x^4-x^2+x-1}$

Using generating functions, solve each LHRRWCC.

11. $a_n = 2a_{n-1}, a_0 = 1$

12. $a_n = a_{n-1} + 1, a_1 = 1$

13. $a_n = a_{n-1} + 2, a_1 = 1$

14. $a_n = a_{n-1} + 2a_{n-2}, a_0 = 3, a_1 = 0$

15. $a_n = 4a_{n-2}, a_0 = 2, a_1 = -8$

16. $a_n = a_{n-1} + 6a_{n-2}, a_0 = 5, a_1 = 0$

17. $a_n = 5a_{n-1} - 6a_{n-2}, a_0 = 4, a_1 = 7$

18. $a_n = a_{n-1} + a_{n-2}, a_0 = 1, a_1 = 2$

19. $a_n = a_{n-1} + a_{n-2}, a_0 = 2, a_1 = 3$

20. $L_n = L_{n-1} + L_{n-2}, L_1 = 1, L_2 = 3$

21. $a_n = 4a_{n-1} - 4a_{n-2}, a_0 = 3, a_1 = 10$

22. $a_n = 6a_{n-1} - 9a_{n-2}, a_0 = 2, a_1 = 3$

23. $a_n = 3a_{n-1} + 4a_{n-2} - 12a_{n-3}, a_0 = 3, a_1 = -7, a_2 = 7$

24. $a_n = 8a_{n-1} - 21a_{n-2} + 18a_{n-3}, a_0 = 0, a_1 = 2, a_2 = 13$

25. $a_n = 7a_{n-1} - 16a_{n-2} + 12a_{n-3}, a_0 = 0, a_1 = 5, a_2 = 19$

26. $a_n = 3a_{n-1} + 4a_{n-2} - 12a_{n-3}, a_0 = 3, a_1 = -7, a_2 = 7$

27. $a_n = 6a_{n-1} - 12a_{n-2} + 8a_{n-3}, a_0 = 0, a_1 = 2, a_2 = -2$

28. $a_n = 13a_{n-2} - 36a_{n-4}, a_0 = 7, a_1 = -6, a_2 = 38, a_3 = -84$

29. $a_n = -a_{n-1} + 3a_{n-2} + 5a_{n-3} + 2a_{n-4}, a_0 = 0, a_1 = -8, a_2 = 4, a_3 = -42$

5.5 Recursive Algorithms

Recall that the recursive definition of the factorial function f expresses $f(n)$ in terms of itself with a smaller argument $n - 1$. Accordingly, it can be employed to write a simple algorithm to compute $n!$ This algorithm has the interesting property that it invokes itself with a smaller argument. Such an algorithm is a recursive algorithm.

Recursive Algorithm

An algorithm is **recursive** if it invokes itself with a smaller argument; that is, if it invokes a reduced version of itself. (See Figure 5.1.)

Recursive definitions invariably lead to recursive algorithms. This section translates some of the examples discussed in Section 5.1 into recursive algorithms and presents a few new ones — gcd, binary search, and merge sort.

EXAMPLE 5.28 Write a recursive algorithm to compute $n!$, where $n \geq 0$.

SOLUTION:

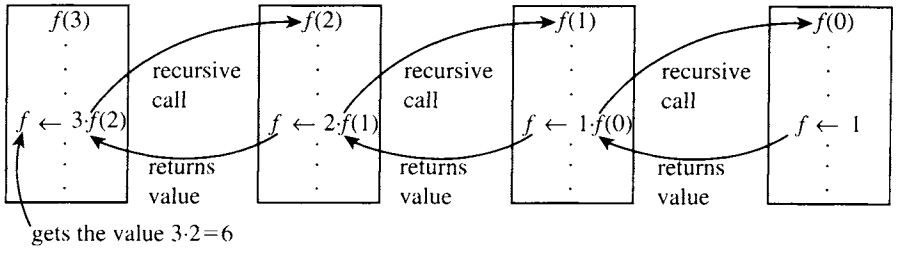
When $n = 0$, the algorithm must terminate and yield the value 1. When $n > 0$, the recurrence relation $f(n) = n \cdot f(n - 1)$ must be applied: the algorithm must invoke itself with $n - 1$ as the new argument. The recursive algorithm is given in Algorithm 5.1.

```
Algorithm factorial(n)
(* This algorithm computes n! using recursion *)
0.  Begin (* algorithm *)
1.    if n = 0 then     (* base case *)
2.      factorial ← 1
3.    else              (* invoke the algorithm *)
4.      factorial ← n · factorial(n - 1)
5.  End (* algorithm *)
```

Algorithm 5.1 ■

Figure 5.15 shows the result of invoking the factorial algorithm with $n = 3$, where f means *factorial*.

Figure 5.15



Every recursive algorithm has two important characteristics, or cases:

- The **base case** ensures the sequence of recursive calls will terminate after a finite number of steps. This case corresponds to the initial condition(s) of a recursive definition.
- The **general case** continues to call itself so long as the base case is not satisfied.

The next example presents an algorithm for computing the number of handshakes made by n guests, discussed in Example 5.3.

EXAMPLE 5.29 Using Example 5.3 write a recursive algorithm to compute the number of handshakes made by n guests.

SOLUTION:

Base case The algorithm terminates when $n = 1$, in which case the number of handshakes made is zero.

General case When $n \geq 2$, the algorithm invokes itself using the recurrence relation $h(n) = h(n-1) + (n-1)$.

These two cases lead to Algorithm 5.2.

```
Algorithm handshake(n)
(* This algorithm computes the number of handshakes made
   by n guests at a party by recursion. *)
0.  Begin (* algorithm *)
```

```
1.    if n = 1 then     (* basis case *)
2.      handshake ← 0
3.    else              (* general case *)
4.      handshake ← handshake(n - 1) + (n - 1)
5.  End (* algorithm *)
```

Algorithm 5.2 ■

EXAMPLE 5.30 Write a recursive algorithm to print the moves and the total number of moves needed to transfer the n disks from peg X to peg Z in the Tower of Brahma puzzle in Example 5.4.

SOLUTION:
Recall that solving the puzzle involves three steps:

- Move the top $n-1$ disks from X to Y using Z as an auxiliary peg;
- Move disk n from X to Z; and
- Move the $n-1$ disks from Y to Z using X as an auxiliary.

We also must count the moves made. The resulting Algorithm 5.3 follows.

```
Algorithm tower (X,Z,Y,n,count)
(* This algorithm, using recursion, prints the various moves
   needed to solve the Tower of Brahma puzzle and returns
   the total number of moves needed in the global variable count.
   Count must be initialized to 0 in the calling module. *)
 0.  Begin (* algorithm *)
 1.    if n = 1 then (* base case *)
 2.      begin (* if *)
 3.        move disk 1 from X to Z
 4.        count ← count + 1
 5.      endif
 6.    else (* general case *)
 7.      begin (* else *)
 8.        tower(X,Y,Z,n - 1,count) (* move the top n - 1 disks *)
 9.        move disk n from X to Z
10.        count ← count + 1
11.        tower(Y,Z,X,n - 1,count)
12.      endelse
13.  End (* algorithm *)
```

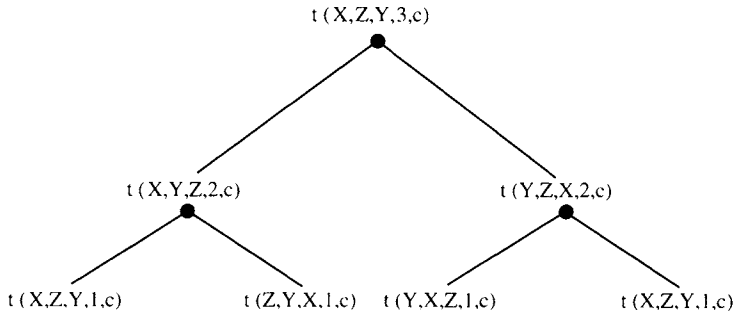
Algorithm 5.3 ■

Suppose we invoke this algorithm by tower (X,Y,Z,3,count). The tree diagram in Figure 5.16 illustrates the various recursive calls, where t stands for *tower* and c for *count*. Seven moves are needed:

> move 1 from X to Z; move 2 from X to Y; move 1 from Z to Y; move 3 from X to Z; move 1 from Y to X; move 2 from Y to Z; move 1 from X to Z.

You may verify this.

Figure 5.16



The next example displays a Fibonacci algorithm.

EXAMPLE 5.31 Write a recursive algorithm to compute the nth Fibonacci number F_n.

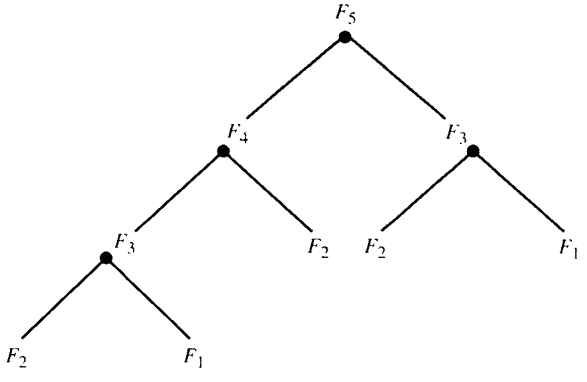
SOLUTION:
Recall from Example 5.7 that the recursive definition of F_n involves two initial conditions $F_1 = 1 = F_2$, and the recurrence relation $F_n = F_{n-1} + F_{n-2}$, where $n \geq 3$. These two cases can be combined into straightforward Algorithm 5.4.

```
Algorithm Fibonacci(n)
(* This algorithm computes the nth Fibonacci number
   using recursion. *)
0.  Begin (* algorithm *)
1.    if n = 1 or n = 2 then     (* base cases *)
2.      Fibonacci ← 1
3.    else                       (* general case *)
4.      Fibonacci ← Fibonacci(n - 1) + Fibonacci(n - 2)
5.  End (* algorithm *)
```

Algorithm 5.4 ■

The tree diagram in Figure 5.17 illustrates the recursive computing of F_5, where each dot represents an addition.

Figure 5.17



The next example shows how we can use recursion to compute the gcd of two positive integers x and y.

EXAMPLE 5.32 Write a recursive algorithm to compute the gcd of two positive integers x and y.

SOLUTION:
If $x > y$, $\gcd\{x,y\} = \gcd\{x - y,y\}$. (See Exercise 34 in Section 4.2.) We use this fact to write Algorithm 5.5.

```
Algorithm gcd(x,y)
(* This algorithm computes the gcd of two positive
   integers x and y using recursion. *)
0.  Begin (* algorithm *)
1.    if x > y then
2.       gcd ← gcd{x - y, y}
3.    else if x < y then
4.            gcd ← gcd{y, x}
5.         else
6.            gcd ← x
7.  End (* algorithm *)
```

Algorithm 5.5

(As an exercise, use this algorithm to compute $\gcd\{x,y\}$ with $x = 28$ and $y = 12$, $x = 13$ and $y = 20$, and $x = 17$ and $y = y$.) ■

We now turn our attention to the recursive version of the binary search algorithm, presented in Example 4.28 in Section 4.5. Recall that binary search, a divide-and-conquer technique, is an efficient method for searching an ordered list for a *key* (say, for example, a certain name in your local telephone directory).

EXAMPLE 5.33 **(Binary Search Algorithm)** Write a recursive algorithm to search an ordered list X of n items and determine if a certain item (*key*) occurs in the list. Return the location of *key* if the search is successful.

SOLUTION:
Because the algorithm is extremely useful, we first outline it:

```
compute the middle index.
if key = middle value then
  we are done and exit
else if key < middle value then
        search the lower half
     else
        search the upper half.
```

The algorithm is given in Algorithm 5.6.

```
Algorithm binary search(X,low,high,key,found,mid)
(* The algorithm returns the location of key in the
   variable mid in the list X if the search is successful.
```

```
   Low, mid, and high denote the lowest, middle, and highest
   indices of the list. Found is a boolean variable;
   it is true if key is found and false otherwise. *)
0. Begin (* algorithm *)
1.   if low ≤ high then (* list is nonempty *)
2.     begin (* if *)
3.       found ← false (* boolean flag *)
4.       mid ← ⌊(low + high)/2⌋
5.       if key = x_mid then
6.         found ← true (* we are done. *)
7.       else
8.         if key < x_mid then (* search the lower half *)
9.           binary search(X,low,mid - 1,key,found,mid)
10.        else (* search the upper half *)
11.          binary search(X,mid + 1,high,key,found,mid)
12.  endif
13. End (* algorithm *)
```

Algorithm 5.6

(As an exercise, use this algorithm to search the list $[3, 5, 8, 13, 21, 34, 55, 89]$ with $key = 5$ and $key = 23$.) ■

The Merge Algorithm

Before presenting the merge sort algorithm that sorts a list into ascending order, we show how the **merge algorithm** works. It combines two ordered lists A and B into an ordered list C, eliminating all duplicate elements.

Consider the two lists A and B:

	1	2	3
A	2	3	5

	1	2	3	4	5
B	1	3	5	8	13

Clearly, the combined sorted list contains at most 8 elements.

Let a_i denote the ith element of A, b_j the jth element of B, and c_k the kth element of C, where $1 \le i \le 3, 1 \le j \le 5$, and $1 \le k \le 8$.

Step 1 Initially, compare a_1 and b_1. Since $b_1 < a_1$, store b_1 in c_1. This yields the following

	1	2	3	4	5	6	7	8
C	1							

Step 2 Compare a_1 and b_2. $a_1 < b_2$. So store a_1 in c_2:

	1	2	3	4	5	6	7	8
C	1	2						

Step 3 Compare a_2 and b_2. Since they are equal, store a_2 in c_3:

	1	2	3	4	5	6	7	8
C	1	2	3					

Step 4 Since $a_3 = b_3$, store a_3 in c_4:

	1	2	3	4	5	6	7	8
C	1	2	3	5				

Step 5 There are no more elements left in A, so copy the remaining elements of B into C. This yields the following sorted list:

	1	2	3	4	5	6	7	8
C	1	2	3	5	8	13		

We now explore the merge sort algorithm, which uses both recursion and the merge algorithm.

The Merge Sort Algorithm

The **merge sort algorithm** sorts a list X of n elements into increasing order. First, partition the list into one-element sublists by successively dividing lists in two. Then invoke the merge algorithm successively to merge the sublists, a pair at a time, into increasing order until the entire list is sorted.

For instance, suppose the one-element sublists after successive division are $x_1, x_2, \ldots$, and x_n; then merge the sublists x_1 and x_2, x_3 and x_4, etc., to form new sublists x_{12}, x_{34}, etc.; now merge the sublists $x_{12}, x_{34}, \ldots$ pair by pair; continue like this until there is a single ordered list.

The following example illustrates this method.

EXAMPLE 5.34 Using the merge sort algorithm, sort the list 13, 8, 3, 5, 2 into ascending order.

SOLUTION:

Divide the given list into two sublists of equal or about the same size: [13, 8, 3] and [5, 2]. Split each sublist into two sublists, resulting in four sublists: [13, 8], [3], [5], [2]. Now divide the first sublist into two sublists, resulting in five one-element sublists: [13], [8], [3], [5], [2].

The tree diagram in Figure 5.18 illustrates this splitting process.

Now the merge algorithm combines them successively in pairs into sorted sublists until the original list is sorted, as shown by the upside-down tree in Figure 5.19.

The recursive merge sort algorithm is given in Algorithm 5.7. Use it to sort the list [13, 55, 3, 8, 34, 5, 2, 31, 29, 6].

Figure 5.18

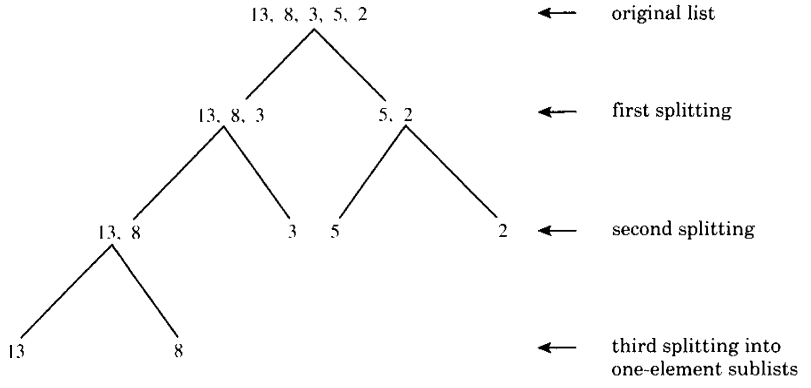
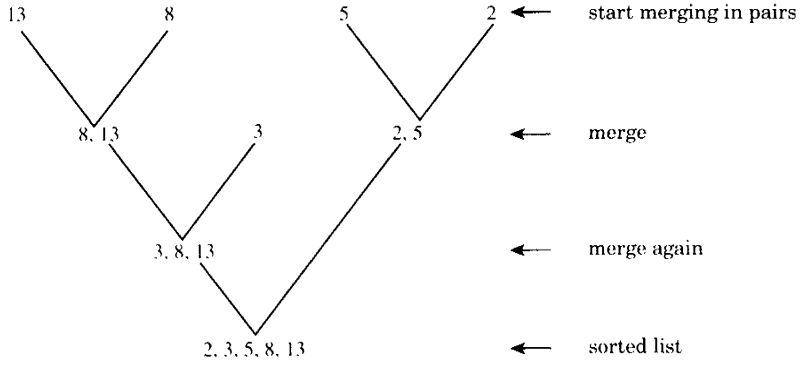


Figure 5.19



■

```
Algorithm merge sort(X,low,high)
(* This recursive algorithm    successively divides a list X of
   high - low + 1 elements into sublists of one element. Then it
   continues to merge sublists in pairs into ordered sublists by
   invoking the merge algorithm until the whole list is ordered. *)
0.  Begin (* algorithm *)
1.    if low < high then(*list contains more than one element* )
2.    begin (* if *)
3.      middle ← ⌊(low + high)/2⌋
4.      merge sort(X,low,middle) (* sort the lower sublist *)
5.      merge sort(X,middle + 1,high) (* sort the upper list*)
6.      merge the two sublists
7.    endif
8.  End (* algorithm *)
```

Algorithm 5.7

Exercises 5.5

Using Algorithm 5.4, compute the nth Fibonacci number for each value of n.

1. 3 **2.** 6 **3.** 7 **4.** 10

Using Algorithm 5.4, find the number of computations needed to compute the nth Fibonacci number F_n for each value of n. (*Hint*: Draw a tree diagram.)

5. 4 **6.** 5 **7.** 6 **8.** 7

9. Let a_n denote the number of additions needed to compute F_n using recursion. Use Exercises 5–8 to predict a formula for a_n.

10. Using induction, prove the formula in Exercise 9 for every $n \geq 1$.

11. Write an iterative algorithm to compute the nth Fibonacci number.

12. Mrs. Zee deposits A dollars at a bank at an annual interest rate of $r\%$ compounded semiannually. Write a recursive algorithm to compute the compound amount she will receive at the end of n years.

Using the recursive binary search algorithm in Example 5.33, determine if the given key occurs in the corresponding list. Show the successive values of *low*, *high*, and *mid*.

13. 2, 3, 5, 8, 13, 21; $key = 13$ **14.** 3, 5, 7, 8, 10; $key = 9$

Using the merge sort algorithm, arrange each list into ascending order.

15. 9, 5, 2, 7, 19, 17, 3, 11 **16.** 9, 11, 6, 2, 12, 3, 8, 5, 31, 13

17. Write an algorithm to compute the nth Lucas number L_n using recursion.

18. Let x be a positive real number and n a nonnegative integer. Write a recursive algorithm to compute x^n.

Let $X = [x_1, x_2, \ldots, x_n]$ and $Y = [y_1, y_2, \ldots, y_n]$ be two lists of numbers. Write a recursive algorithm to accomplish the tasks in Exercises 19–31.

19. Find the sum of the numbers from left to right.

20. Find the sum of the numbers from right to left.

21. Compute the product of the numbers from left to right.

22. Compute the product of the numbers from right to left.

23. Find the maximum of the numbers in the list.

24. Find the minimum of the numbers in the list.

25. Print the numbers in the given order $x_1, x_2, \ldots, x_n$.

26. Print the numbers in the reverse order $x_n, x_{n-1}, \ldots, x_2, x_1$.

27. (**Linear search**) Search the list for a specific item (*key*). Return the location of *key* if the search is successful.

28. Determine if two lists X and Y of n items of the same type are identical.

29. Determine if a word of n alphanumeric characters is a palindrome.

30. Evaluate Ackermann's function $A(x,y)$, where x and y are nonnegative integers. See Exercises 5.1 for a definition of $A(x,y)$.

31. Sort the list X using bubble sort.

32. Use the recursive bubble sort algorithm to sort the list 13, 5, 2, 8, 3.

Quicksort, invented in 1962 by C. Anthony R. Hoare of Oxford University, is an extremely efficient technique for sorting a large list X of n items $x_1, x_2, \ldots, x_n$. It is based on the fact that it is easier to sort two small lists than one large list. Choose the first element x_1 as the *pivot*. To place the pivot in its final *resting place*, compare it to each element in the list. Move the elements less than x_1 to the front of the list and those greater than x_1 to the rear. Now place pivot in its final position. Partition the list X into two sublists such that the elements in the first sublist are less than x_1 and the elements in the second sublist are greater than x_1. Continue this procedure recursively with the two sublists.

***33.** Use quicksort to sort the list 7, 8, 13, 11, 5, 6, 4.

***34.** Use quicksort to write a recursive algorithm to sort a list X of n elements.

5.6 Correctness of Recursive Algorithms

We now use induction to establish the correctness of two well-known recursive algorithms, linear search and bubble sort. We begin with the linear search algorithm.

Recall that the linear search algorithm searches a list X of n elements for a given *key*. If the search is successful, the algorithm returns the location of *key*; otherwise, it returns zero. A recursive version is given in Algorithm 5.8. Again, as an exercise, use it to search the list $X = [13, 5, 47, 7, 11, 8, 3]$ for $key = 11$.

```
Algorithm linear search (X,n,key,location)
(* This algorithm returns the position of key in the
   variable location. If location = 0, then key does
   not exist in the list. *)
0.  Begin (* algorithm *)
1.     if n = 0 then (* unsuccessful search *)
2.       location ← 0
3.     else if x_n = key then
4.             location ← n
5.          else
6.             linear search(X,n - 1,key,location)
7.  End (* algorithm *)
```

Algorithm 5.8

EXAMPLE 5.35 Establish the correctness of Algorithm 5.8.

PROOF (by PMI):
To prove the correctness of the algorithm, we must show that it works correctly for $n \geq 0$. Let P(n): The algorithm returns the correct value of location for every list of size n.

Basis step When $n = 0$, lines 3 through 6 in the algorithm are skipped and the algorithm returns the value 0 from line 2. So the algorithm works correctly when $n = 0$.

Induction step Let k be an arbitrary integer $k \geq 0$ such that P(k) is true; that is, assume the algorithm works correctly for a list of arbitrary size $k \geq 0$. To prove that P($k + 1$) is true, invoke the algorithm for a list X of size $k + 1$. Note that $k + 1 \geq 1$.

Case 1 If $x_{k+1} = key$, the algorithm returns the value $k + 1$ from line 4.

Case 2 If $x_{k+1} \neq key$, line 6 is executed; so the algorithm is invoked for a list with k elements. By our inductive hypothesis, the algorithm works for such a list.

Thus in both cases, the algorithm returns the correct value of location. Therefore, P($k + 1$) is true.

Consequently, P(n) holds for $n \geq 0$ by induction; that is, the algorithm works correctly for every list. ■

Next we verify the correctness of the recursive version of the bubble sort algorithm, given in Algorithm 5.9. To get used to it, you may use it to sort the list $X = [13, 5, 47, 7, 11, 8, 3]$.

```
Algorithm Bubble Sort(X,n)
(* This algorithm sorts a list X of n items using recursion. *)
0.  Begin (* algorithm *)
1.    if n > 1 then (* list contains at least two elements *)
2.    begin (* if *)
3.      for i = 1 to n - 1 do
4.       if xi > xi+1 then (* they are out of order *)
5.         swap xi and xi+1
6.      bubble sort(X,n - 1)
7.    endif
8.  End (* algorithm *)
```

Algorithm 5.9

EXAMPLE 5.36 Establish the correctness of Algorithm 5.9.

PROOF (by PMI):
Let P(n): The algorithm works for every list of size n.

Basis step When $n = 0$, the list contains no elements. So the algorithm works by default. Thus, P(0) is true.

Induction step Assume P(k) is true for an arbitrary integer $k \geq 0$; that is, the algorithm correctly sorts every list of k (≥ 0) elements. To prove that P($k+1$) is true, invoke the algorithm for a list X with $k+1$ elements, where $k+1 \geq 1$.

If $k + 1 = 1$, the **for** loop is not entered. So P(k+1) is true, by default.

If $k + 1 > 1$, the **for** loop is entered. Consecutive elements x_i and x_{i+1} are compared in line 4 and switched in line 5 if necessary. When we exit the loop, the largest of the $k + 1$ elements is placed in the correct position, in location $k + 1$.

This leaves a sublist of k elements, $x_1, \ldots, x_k$. By the inductive hypothesis, the algorithm correctly sorts such a list.

Thus if P(k) is true, then P(k+1) is also true.

Therefore, by induction, P(n) is true for every $n \geq 0$: the algorithm sorts every list of every size $n \geq 0$. ■

The following exercises provide additional opportunities to establish the correctness of recursive algorithms.

Exercises 5.6

Establish the correctness of each algorithm.

1. The factorial algorithm in Example 5.28.
2. The handshake algorithm in Example 5.29.
3. The Tower of Brahma algorithm in Example 5.30.
4. The Fibonacci algorithm in Example 5.31.
5. The binary search algorithm in Example 5.33.
6. The merge sort algorithm in Algorithm 5.7.

7–17. The algorithms in Exercises 19–29 of Section 5.5.

Algorithm 5.10 computes the nth power of a positive real number x, where $n \geq 0$. Use it to answer Exercises 18–24.

```
Algorithm exponentiation(x,n)
(* This algorithm computes the nth power of x using recursion
   and returns the value in the variable answer. *)
 0.  Begin (* algorithm *)
 1.    if n = 0 then
 2.      answer ← 1
 3.    else if n = 1 then
 4.            answer ← x
 5.    else
```

```
 6.       begin (* else *)
 7.        value ← exponentiation(x,⌊n/2⌋)
 8.        answer ← value · value
 9.        if n is odd then
10.         answer ← answer · x
11.       endelse
12.  End (* algorithm *)
```

Algorithm 5.10

Let a_n denote the number of multiplications (lines 7–10) required by the algorithm to compute x^n. Compute each.

18. a_0 **19.** a_1 **20.** a_4 **21.** a_5

22. Find the recurrence relation satisfied by a_n.

23. Solve the recurrence relation in Exercise 22, where $n = 2^k$.

24. Establish the correctness of Algorithm 5.10.

25. Prove the correctness of the iterative Fibonacci algorithm in Exercise 11 of Section 5.5.

*5.7 Complexities of Recursive Algorithms (optional)

Using the big-oh and big-theta notations, we now investigate the complexities of a few standard recursive algorithms: linear search, Fibonacci, selection sort, binary search, and merge sort. In addition, using Fibonacci numbers, we estimate the number of divisions needed to compute $\gcd\{a, b\}$ using the euclidean algorithm.

We begin our analysis with the recursive linear search algorithm.

EXAMPLE 5.37 Use the recursive linear search in Algorithm 5.8 to estimate the worst time required to search for a *key* in a list X of n items.

SOLUTION:

Let c_n denote the maximum number of element comparisons needed in line 3 of the algorithm. To find a big-oh estimate of c_n, first define it recursively.

Clearly, $c_0 = 0$. When $n \geq 1$

$$c_n = \begin{pmatrix}\text{maximum number of calls}\\ \text{from the recursive call in}\\ \text{line 6}\end{pmatrix} + \begin{pmatrix}\text{number of}\\ \text{comparisons}\\ \text{in line 3}\end{pmatrix}$$

$$= c_{n-1} + 1$$

Thus

$$c_0 = 0$$
$$c_n = c_{n-1} + 1, \quad n \geq 1$$

Solving this recurrence relation (try) yields $c_n = n, n \geq 0$; so $c_n = O(n) = \Theta(n)$. Thus, in the worst case, the algorithm takes $O(n) = \Theta(n)$ comparisons to locate the key, the same as the iterative version. ■

Next we analyze the recursive and iterative Fibonacci algorithms.

EXAMPLE 5.38 Using the recursive algorithm in Example 5.31, estimate the number of additions a_n needed to compute the nth Fibonacci number.

SOLUTION:
By Exercises 9 and 10 in Section 5.5, $a_n = F_n - 1, n \geq 1$. But, by Exercise 43 in Section 5.1, $F_n \leq 2^n$, where $n \geq 1$. Therefore,

$$\begin{aligned} a_n &\leq 2^n - 1 \\ &< 2^n \\ &= O(2^n) \end{aligned}$$

Thus, the recursive Fibonacci algorithm takes $O(2^n)$ additions. ■

For comparison, we now study the complexity of the iterative version of the Fibonacci algorithm.

EXAMPLE 5.39 Estimate the number of additions a_n required in line 5 to compute the nth Fibonacci number F_n by Algorithm 5.11.

```
Algorithm iterative Fibonacci(n)
(* This iterative algorithm uses the values of the
   variables of the last and the current Fibonacci
   numbers to compute the next Fibonacci number. *)
0.  Begin (* algorithm *)
1.    last ← 1
2.    current ← 1
3.    for i = 2 to n do
4.    begin (* for *)
5.          next ← last + current
6.          last ← current
7.          current ← next
8.    endfor
9.  End (* algorithm *)
```

Algorithm 5.11

SOLUTION:
The first two Fibonacci numbers need no computations; therefore, $a_1 = 0 = a_2$. Suppose $n > 2$. It takes one addition to compute the next item

F_n from the current term F_{n-1}. So $a_n = a_{n-1} + 1$. Solving this recurrence relation (try), we get

$$a_n = n - 2, \quad n \geq 2$$
$$= \Theta(n)$$

Thus the iterative version takes $\Theta(n)$ additions to compute F_n. ■

The time it takes to compute F_n by the recursive algorithm grows exponentially with n, whereas by the iterative algorithm it grows only linearly. As n gets larger and larger, it takes more time to compute F_n by recursion than by iteration. Thus, by dividing and conquering the problem, we have made it complicated.

Should we prefer the iterative method to the recursive method? Since every recursive algorithm has a nonrecursive version, if the algorithm makes just one recursive call to itself, as in the factorial algorithm, the iterative approach will, in general, save time. On the other hand, if the problem has a recursive definition, it will be easy to write a recursive algorithm for the problem. Writing the nonrecursive version of a recursive algorithm is often a painful task and the resulting algorithm is often much longer, complicated, and difficult to understand. For instance, the nonrecursive version of the Tower of Brahma algorithm is longer and that of quicksort is rather complicated.

Next we estimate the number of element-comparisons required by the recursive selection sort algorithm presented in Algorithm 5.12. (See Algorithm 4.11 in Chapter 4 for an iterative version.)

```
Algorithm selection sort(X,n)
(* This algorithm invokes a subalgorithm called swap
   which switches two elements. Maxindex denotes the
   index of the largest of the n elements. *)
0.  Begin (* algorithm *)
1.    maxindex ← n (*initialize maxindex at each pass *)
2.    for i = 1 to n - 1 do
3.      if x_i > x_maxindex then
4.        maxindex ← i
5.    if maxindex ≠ n then (* swap the corresponding items *)
6.      swap x_maxindex and x_n
7.    selection sort(X,n - 1)
8.  End (* algorithm *)
```

Algorithm 5.12

EXAMPLE 5.40 Estimate the number c_n of comparisons (lines 3 and 5) required by Algorithm 5.12.

SOLUTION:

To estimate c_n, first define it recursively.

If the list contains just one element, lines 3 and 5 are not executed; therefore, $c_1 = 0$.

Suppose $n \geq 2$. Since the **for** loop is executed $n - 1$ times, line 3 is executed $n - 1$ times. Furthermore, line 5 is executed once. Therefore,

$$\begin{aligned} c_n &= c_{n-1} + (n-1) + 1 \\ &= c_{n-1} + n, \quad n \geq 2 \end{aligned}$$

Solving the recurrence relation by the iterative method, we get

$$\begin{aligned} c_n &= \frac{n(n+1)}{2} - 1, \quad n \geq 1 \\ &= \Theta(n^2) \end{aligned}$$

Thus the algorithm takes $\Theta(n^2)$ comparisons to sort a list of n items, as in the iterative version. ■

Example 5.41 investigates one of the many properties of Fibonacci numbers. Example 5.42 uses the property to estimate the number of divisions in the euclidean algorithm.

EXAMPLE 5.41 Let F_n denote the nth Fibonacci number and $\alpha = \dfrac{1+\sqrt{5}}{2}$. Prove that $\alpha^{n-2} < F_n < \alpha^{n-1}$, $n \geq 3$.

PROOF (by strong induction):
(We shall prove that $\alpha^{n-2} < F_n$ and leave the other half as an exercise.)
You may verify that α is a solution of the equation $x^2 = x + 1$, so $\alpha^2 = \alpha + 1$.
Let P(n): $\alpha^{n-2} < F_n$, where $n \geq 3$.

Basis step Since the induction step below uses the recurrence relation $F_{k+1} = F_k + F_{k-1}$, the basis step involves verifying that both P(3) and P(4) are true.

- *To show that P(3) is true*: *When* $n = 3$,

$$\alpha^{n-2} = \alpha = \frac{1+\sqrt{5}}{2} < \frac{1+3}{2} = 2 = F_3$$

So P(3) is true.

- *To show that P(4) is true*:

$$\begin{aligned} \alpha^2 &= \left(\frac{1+\sqrt{5}}{2}\right)^2 = \frac{3+\sqrt{5}}{2} \\ &< \frac{3+3}{2} = 3 = F_4 \end{aligned}$$

Thus P(4) is also true.

Induction step Assume P(3), P(4),..., P(k) are true; that is, assume $\alpha^{i-2} < F_i$ for $3 \le i \le k$. We must show that $P(k+1)$ is true; that is, $\alpha^{k-1} < F_{k+1}$.

We have

$$\alpha^2 = \alpha + 1.$$

Multiplying both sides by α^{k-3},

$$\begin{aligned} \alpha^{k-1} &= \alpha^{k-2} + \alpha^{k-3} && \text{(Note: } k-3 \ge 2.\text{)} \\ &< F_k + F_{k-1}, && \text{by the inductive hypothesis} \\ &= F_{k+1}, && \text{by the recurrence relation} \end{aligned}$$

Thus P($k+1$) is true.

Therefore, by the strong version of induction, P(n) is true for $n \ge 3$; that is, $\alpha^{n-2} < F_n$ for every $n \ge 3$. ■

Now we can estimate the number of divisions required by the euclidean algorithm to compute gcd$\{a, b\}$.

EXAMPLE 5.42 **(Lamé's Theorem)** The number of divisions needed to compute g$\{a, b\}$ by the euclidean algorithm is no more than five times the number of decimal digits in b, where $a \ge b \ge 2$.

PROOF:
Let F_n denote the nth Fibonacci number, $a = r_0$, and $b = r_1$. By the repeated application of the division algorithm we have:

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

Clearly, it takes n divisions to evaluate gcd$\{a, b\} = r_n$. Since $r_i < r_{i-1}$, $q_i \ge 1$ for $1 \le i \le n$. In particular, since $r_n < r_{n-1}$, $q_n \ge 2$; so $r_n \ge 1$ and $r_{n-1} \ge 2 = F_3$. Consequently, we have:

$$\begin{aligned} r_{n-2} &= r_{n-1} q_{n-1} + r_n \\ &\ge r_{n-1} + r_n \\ &\ge F_3 + 1 \\ &= F_3 + F_2 = F_4 \end{aligned}$$

$$\begin{aligned} r_{n-3} &= r_{n-2}q_{n-2} + r_{n-1} \\ &\geq r_{n-2} + r_{n-1} \\ &\geq F_4 + F_3 = F_5 \end{aligned}$$

Continuing like this,

$$\begin{aligned} r_1 &= r_2 q_2 + r_3 \\ &\geq r_2 + r_3 \\ &\geq F_n + F_{n-1} = F_{n+1} \end{aligned}$$

That is,

$$b \geq F_{n+1}$$

By Example 5.41, $F_{n+1} > \alpha^{n-1}$, where $\alpha = \dfrac{1+\sqrt{5}}{2}$. Therefore,

$$b > \alpha^{n-1}$$

Then

$$\log b > (n-1)\log \alpha$$

Since $\alpha = \dfrac{1+\sqrt{5}}{2} \approx 1.618033989$, $\log \alpha \approx 0.2089876403 > \dfrac{1}{5}$. So

$$\log b > \frac{n-1}{5}$$

Suppose b contains k decimal digits. Then $b < 10^k$. Therefore, $\log b < k$ and hence $k > \dfrac{n-1}{5}$. Thus $n < 5k + 1$ or $n \leq 5k$. That is, the number of divisions needed by the algorithm is no more than five times the number of decimal digits in n. ■

Let us pursue this example a bit further. Since $\log b > \dfrac{n-1}{5}$, $n < 1 + 5\log b$. Also, since $b \geq 2$,

$$\begin{aligned} 5\log b &\geq 5\log 2 \\ &> 1 \end{aligned}$$

Thus

$$\begin{aligned} n &< 1 + 5\log b \\ &< 5\log b + 5\log b \\ &= 10\log b \\ &= O(\log b) \end{aligned}$$

Thus it takes $O(\log b)$ divisions to compute $\gcd\{a, b\}$ by the euclidean algorithm.

Gabriel Lamé *(1795–1870) was born in Tours, France. After graduating from the École Polytechnique in 1817, he continued his studies at the École des Mines, from which he graduated in 1820.*

The same year Lamé was appointed director of the School of Highways and Transportation in St. Petersburg, Russia. There he taught mathematics, physics, and chemistry and planned roads and bridges in and around the city. In 1832, he returned to Paris to form an engineering firm. Within a few months, however, he left it to become the chair of physics at the École Polytechnique, where he remained until 1844. While teaching, he served as a consulting engineer, becoming the chief engineer of mines in 1836. He helped build the railroads from Paris to Versailles and to St. Germain.

In 1844, Lamé became graduate examiner for the University of Paris in mathematical physics and probability, and professor 7 years later. In 1862, he became deaf and resigned his positions. He died in Paris.

Although Lamé did original work in number theory and mathematical physics, his greatest contribution was the development of the curvilinear coordinates and their applications. His work on the curvilinear system led him to number theory. In 1840, he proved Fermat's Last Theorem for n = 7.

Gauss considered Lamé the foremost French mathematician of his time. French mathematicians, however, considered him too practical, and French scientists, too theoretical.

The next example, due to S. H. Friedberg, explores the number of multiplications needed to compute the determinant of an $n \times n$ matrix by cofactor expansion. (It may be omitted by those not familiar with determinants and calculus.)

EXAMPLE 5.43 (optional) Let f_n denote the number of multiplications needed to compute $\det A$, the determinant of an arbitrary $n \times n$ matrix $A = (a_{ij})$ by cofactor expansion. Estimate f_n.

SOLUTION:
We estimate f_n in three steps:

- Define f_n recursively.
- Solve the recurrence relation.
- Use the solution to estimate f_n.
- *To define f_n recursively*:
  Let C_{ij} denote the $(n-1) \times (n-1)$ determinant obtained from $\det A$ by deleting its ith row and jth column. By expanding $\det A$ with respect to the first row, we have

$$\det A = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} C_{1j} \quad \leftarrow \text{cofactor expansion by row 1}$$

In particular, let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Then $\det A = aC_{11} - bC_{12} = ad - bc$. Clearly, two multiplications are needed to evaluate $\det A$ and hence $f_2 = 2$. Also $f_1 = 0$.

Suppose $n \geq 3$. Then, by definition, it takes f_{n-1} multiplications to compute C_{1j}. Therefore, it takes $f_{n-1} + 1$ multiplications to evaluate $a_{1j}C_{1j}$ and hence $n(f_{n-1} + 1)$ multiplications to compute $\det A$.

Thus f_n can be defined recursively as follows:

$$\begin{aligned} f_1 &= 0 \\ f_n &= n(f_{n-1} + 1), \quad n \geq 2 \end{aligned} \tag{5.20}$$

(This is a linear nonhomogeneous recurrence relation with nonconstant coefficients.)

- *To solve the recurrence relation (5.20)*:
  Let $f_n = n!g_n$. Since $f_1 = 0$, $g_1 = 0$. Substituting for f_n in Equation (5.20), we get

$$\begin{aligned} n!g_n &= n[(n-1)!g_{n-1} + 1] \\ &= n!g_{n-1} + n \end{aligned}$$

So

$$\begin{aligned} (g_n - g_{n-1})n! &= n \\ g_n - g_{n-1} &= \frac{1}{(n-1)!} \quad (\text{Note: } g_1 = 0.) \end{aligned}$$

Solving this yields (see Exercise 64)

$$g_n = \sum_{k=1}^{n-1} \frac{1}{k!}, \text{ since } g_1 = 0$$

So,

$$\begin{aligned} f_n = n!g_n &= n!\left(\sum_{k=1}^{n-1} \frac{1}{k!}\right) \\ &= n!\left(\sum_{k=1}^{n} \frac{1}{k!}\right) - 1 \end{aligned}$$

Therefore,

$$\begin{aligned} f_n &\leq n!\left(\sum_{k=1}^{\infty} \frac{1}{k!}\right) - 1 \\ &= n!(e-1) - 1, \text{ by calculus} \\ &\leq en! \\ &= O(n!) \end{aligned}$$

Thus the evaluation of $\det A$ by cofactor expansion takes $O(n!)$ multiplications. ■

Divide-and-Conquer Algorithms

We can now analyze the complexities of a special class of recursive algorithms called divide-and-conquer algorithms.

The binary search algorithm presented in Algorithm 5.6 is based on the divide-and-conquer approach. To search an ordered list of n items for a given key, we divide the list into two smaller and similar sublists of about the same size. If the middle value $\neq$ *key*, then we search either the lower half or the upper half, and continue this procedure until we are done. This exemplifies a divide-and-conquer algorithm.

More generally, consider a problem of size n. Suppose the problem can be solved for a small initial value of n, and it can be broken up into a smaller and similar subproblems of approximately the same size, usually $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$, where $a, b \in \mathbb{N}$, $1 \leq a < n$, and $1 < b < n$. Assume that we can solve each of the subproblems and employ their solutions to solve the original problem. Such an algorithm is a **divide-and-conquer algorithm**.

Let $f(n)$ denote the number of operations required to solve the original problem and $g(n)$ the number of operations resulting from the splitting. Then, assuming b is a factor of n,

$$f(n) = af(n/b) + g(n)$$

This is the **divide-and-conquer recurrence relation** resulting from the algorithm.

The binary search algorithm manifests the complexities of the divide-and-conquer technique.

EXAMPLE 5.44 **(binary search)** Using the recursive binary search in Algorithm 5.6, let c_n denote the maximum number of element comparisons needed to search for a given item (key) in an ordered list X of n items. If $n = 1$, then *low* = *high* = *mid* = 1 and the condition in line 5 is tested exactly once; so $c_1 = 1$.

Suppose $n > 1$. Then the middle term is $x_{\lfloor (n+1)/2 \rfloor}$. Compare *key* to $x_{\lfloor (n+1)/2 \rfloor}$. If they are *not* equal, search the lower sublist or the upper sublist, but *not* both. If n is even, $\left\lfloor \frac{n+1}{2} \right\rfloor = \left\lfloor \frac{n}{2} \right\rfloor$; so the upper half contains $\frac{n}{2} = \left\lfloor \frac{n}{2} \right\rfloor$ elements and the lower half contains $\frac{n}{2} - 1 \left(< \left\lfloor \frac{n}{2} \right\rfloor \right)$ elements. On the other hand, if n is odd, then $\left\lfloor \frac{n+1}{2} \right\rfloor = \frac{n+1}{2}$; so both sublists contain $\frac{n-1}{2} = \left\lfloor \frac{n}{2} \right\rfloor$ elements each. Thus, in any case, the maximum number of comparisons needed is $c_{\lfloor n/2 \rfloor} + 1$. So

$$\begin{aligned} c_1 &= 1 \\ c_n &= c_{\lfloor n/2 \rfloor} + 1, \quad n \geq 2 \end{aligned} \tag{5.21}$$

To solve this recurrence relation, assume, for convenience, that n is a power of 2, say $n = 2^k$, where $k \geq 0$. Let $c_n = a_k$. Then the recurrence relation (5.21) becomes $a_k = a_{k-1} + 1$, where $a_0 = 1$. Solving this recurrence relation yields $a_k = k + 1$, $k \geq 0$ (Verify.). Since $n = 2^k$, $k = \lg n$, so $c_n = 1 + \lg n$, $n \geq 1$. Thus, if n is a power of 2, then $c_n = \Theta(\lg n)$.

Suppose n is *not* a power of 2. Then, by induction, it can be shown that $c_n = 1 + \lfloor \lg n \rfloor$, where $n \geq 1$ (see Exercise 44), so $c_n = \Theta(\lg n)$.

Thus, in both cases, the algorithm takes $\Theta(\lg n)$ element comparisons in the worst case. ■

The preceding example is a special case of the following theorem. Since the proof is somewhat complicated, we skip it (see Exercises 65 and 66).

THEOREM 5.7 Let $a, b \in \mathbb{N}$ and $c, d \in \mathbb{R}^+$ with $b \geq 2$. Let f be a nondecreasing function* such that $f(n) = af(n/b) + c$ and $f(1) = d$. Then

$$f(n) = \begin{cases} O(\lg n) & \text{if } a = 1 \\ O(n^{\log_b a}) & \text{otherwise} \end{cases}$$

■

For example, let f be a nondecreasing function such that $f(n) = 3f(n/2) + 5$ and $f(1) = 8$. Then, by Theorem 5.7, $f(n) = O(n^{\lg 3})$.

The next theorem is a generalization of Theorem 5.7. We state it without proof (see Exercises 67–69 for special cases of the theorem) and apply it in Example 5.45.

THEOREM 5.8 Let $a, b \in \mathbb{N}$ and $c, d \in \mathbb{R}^+$ with $b \geq 2$. Let f be a nondecreasing function such that $f(n) = af(n/b) + cn^d$. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \lg n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

■

EXAMPLE 5.45 (optional) Let $A = (a_{ij})$ and $B = (b_{ij})$ be two $n \times n$ matrices. Let $C = (c_{ij})$ be their product where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Since C has n^2 entries and each takes n multiplications, the product C can be computed using $n^3 = O(n^3)$ multiplications; in fact, it can be computed using $O(n^3)$ computations (additions and multiplications), as Exercises 40 and 41 indicate. ■

We close this section with an analysis of the merge sort algorithm, a divide-and-conquer strategy.

EXAMPLE 5.46 **(merge sort)** The merge sort method in Algorithm 5.7 sorts a list of n elements. Assume, for convenience, that n is a power of 2, say, $n = 2^k, k \geq 0$.

*Let $S \subseteq \mathbb{R}$. A function $f : S \to \mathbb{R}^+$ is said to be **nondecreasing** if $x < y$ implies $f(x) \leq f(y)$.

Let c_n denote the maximum number of element comparisons needed in line 6. Show that $c_n = O(n \lg n)$.

SOLUTION:
When $n = 2$, one comparison is needed in line 6; therefore, $c_2 = 1$. So, let $n > 2$. The list is split into two, with each sublist containing $n/2$ elements. In the worst case, the number of comparisons resulting from line 4 is $c_{n/2}$, as it is from line 5. When the merge algorithm is invoked in line 6, each sublist contains $n/2$ elements; so the maximum number of comparisons from line 6 is $n - 1$. Thus

$$\begin{aligned} c_2 &= 1 \\ c_n &= 2c_{n/2} + (n-1), \quad n \geq 3 \end{aligned}$$

Let $a_k = c_n$ where $n = 2^k$, $k \geq 0$. Then

$$\begin{aligned} a_1 &= 1 \\ a_k &= 2a_{k-1} + (2^k - 1), \quad k \geq 2 \end{aligned}$$

This recurrence relation (see Exercise 8 in Section 5.2) yields

$$\begin{aligned} a_k &= (k-1)2^k + 1, \quad k \geq 1 \\ &= k \cdot 2^k - 2^k + 1 \end{aligned}$$

Thus

$$\begin{aligned} c_n &= (\lg n)n - n + 1 \\ &\leq n \lg n + 1 \\ &< 2n \lg n, \quad n \geq 2 \\ &= O(n \lg n) \end{aligned}$$

■

More generally, it can be shown that in the worst case the merge sort requires $O(n \lg n)$ element comparisons for a list of n elements. This time estimate is the best among all sorting algorithms.

Exercises 5.7

Find a big-oh estimate for each.

1. The number $h(n)$ of handshakes made by n guests at a party, using Example 5.3.
2. The number b_n of moves needed to transfer n disks in the Tower of Brahma puzzle in Example 5.4.
3. The number f_n of regions formed by n lines, using Example 5.5.

Estimate the solution f_n of each recurrence relation (see Exercises 5.2).

4. $f_1 = 1$
$f_n = f_{n-1} + (2n - 1),\ n \geq 2$

5. $f_0 = 0$
$f_n = f_{n-1} + 4n,\ n \geq 1$

6. $f_1 = 2$
$f_n = f_{n-1} + n,\ n \geq 2$

7. $f_1 = 1$
$f_n = 2f_{n-1} + (2^n - 1),\ n \geq 2$

Find the number of comparisons needed to search for $key = 13$ in each ordered list using the recursive binary search algorithm in Example 5.33.

8. 1, 2, 3, 5, 8, 13

9. 5, 8, 13, 21, 34

10. 3, 7, 8, 13, 21

11. 15, 16, 19, 21

Compute the maximum number of comparisons needed to search for a particular item in an ordered list containing the following number of items, using the recursive binary search algorithm.

12. 8 **13.** 20 **14.** 25 **15.** 31

Let b_n denote the number of multiplications needed to compute $n!$ using the recursive factorial algorithm in Example 5.1.

16. Define b_n recursively.

17. Solve the recurrence relation satisfied by b_n.

18. Show that $b_n = O(n)$.

19–22. Estimate the number of times a_n the assignment statement, $x \leftarrow x + 1$, is executed by the nested **for** loops in Exercises 35–38 of Section 4.4.

Estimate the number a_n of times the statement, $x \leftarrow x + 1$, is executed by each nested **for** loop.

23.
```
for i = 1 to n do
  for j = 1 to ⌊i/2⌋ do
    x ← x + 1
```

24.
```
for i = 1 to n do
  for j = 1 to ⌈i/2⌉ do
    x ← x + 1
```

***25.**
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to j do
      for l = 1 to j do
        x ← x + 1
```

***26.**
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to j do
      for l = 1 to k do
        x ← x + 1
```

Let b_n denote the number of element-comparisons needed by the bubble sort algorithm in Algorithm 5.9.

27. Define b_n recursively.

28. Solve the recurrence relation.

29. Find a big-oh estimate of b_n.

30. Let a_n denote the number of additions needed to compute the nth Fibonacci number F_n, using Algorithm 5.4. Prove that $a_n = \sum_{i=1}^{n-2} F_i$, $n \geq 3$.

Solve each recurrence relation.

31. $c_0 = 1$

$c_n = c_{n-1} + b,\ n \geq 1$

32. $a_2 = 0$

$a_n = a_{n-1} + b,\ n \geq 3$

33. $c_1 = 0$

$c_n = c_{n-1} + bn,\ n \geq 2$

34. $c_1 = a$

$c_n = c_{n-1} + bn^3,\ n \geq 2$

The number of operations $f(n)$ required by an algorithm is given by $f(n) = f(n-1) + (n-1) + (n-2)$, where $f(1) = 1$.

35. Find an explicit formula for $f(n)$.

36. Show that $f(n) = O(n^2)$.

Let $f(n)$ denote the number of bits in the binary representation of a positive integer n.

37. Find a formula for $f(n)$.

38. Show that $f(n) = O(\lg n)$.

39. Let $x \in \mathbb{R}^+$ and $n \in \mathbb{N}$. The technique of **successive squaring** can be applied to compute x^n faster than multiplying x by itself $n-1$ times. For example, to find x^{43}, first evaluate x^2, x^4, x^8, x^{16}, and x^{32}; then multiply x^{32}, x^8, x^2, and x^1: $x^{43} = x^{32} \cdot x^8 \cdot x^2 \cdot x^1$. This process takes only $5 + 3 = 8$ multiplications instead of the conventional method's 42. The powers of x used in computing x^n are the place values of the bits in the binary representation of n; in fact, the number of powers of x used equals the number of nonzero bits in the binary representation of n. Let $f(n)$ denote the number of multiplications needed to compute x^n by successive squaring. Show that $f(n) = O(\lg n)$.

Let $A = (a_{ij})$ and $B = (b_{ij})$ be two $n \times n$ matrices. Let f_n denote the number of computations (additions and multiplications) to compute their product $C = (c_{ij})$, where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

40. Evaluate f_n.

41. Estimate f_n.

42. Solve the recurrence relation $C_n = 2C_{n/2} + 1$, where $C_1 = a$ and n is a power of 2.

43. Show that $C_n = O(n)$.

44. Let c_n denote the maximum number of comparisons needed to search for a *key* in an ordered list X of n elements, using the recursive binary search algorithm. Prove that $c_n = 1 + \lfloor \lg n \rfloor$, for every $n \geq 1$.

45. Let $a, b, k \in \mathbb{N}$, $b \geq 2$, and $n = b^k$. Consider the function f defined by $f(n) = af(n/b) + g(n)$. Show that $f(n) = a^k f(1) + \sum_{i=0}^{k-1} a^i g(n/b^i)$.

46. Solve the recurrence relation $a_n = 2a_{n/2} + n$, where $a_1 = 0$ and $n = 2^k$.

47. Use Exercise 46 to show that $a_n = O(n \lg n)$.

Let f be a function defined by $f(n) = af(n/b) + cn$, where $a, b \in \mathbb{N}$, $b \geq 2$, $c \in R^+$, and $f(1) = d$. Assume n is a power of b.

48. Solve the recurrence relation.

49. Let $a = b$ and $d = 0$. Show that $f(n) = O(n \lg n)$.

Consider the recurrence relation $c_n = c_{\lfloor n/2 \rfloor} + c_{\lfloor (n+1)/2 \rfloor} + 2$, where $c_1 = 0$.

50. Compute c_3 and c_4.

51. Solve the recurrence relation when n is a power of 2.

52. Find the order of magnitude of c_n when n is a power of 2.

Let t be a function defined by

$$t(n) = \begin{cases} a & \text{if } n = 1 \\ t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + bn & \text{otherwise} \end{cases}$$

where $a, b \in \mathbb{R}^+$. (Such a function occurs in the analysis of merge sort.)

53. Evaluate $t(5)$ and $t(6)$.

54. Prove that $t(n)$ is a nondecreasing function; that is, $t(n) \leq t(n+1)$, where $n \geq 1$.

55. Show that $t(n) = O(n \lg n)$, where n is a power of 2.

Let $f(n) = 2f(n/2) + cn^2$, where $f(1) = d$ and n is a power of 2.

56. Solve the recurrence relation. **57.** Show that $f(n) = O(n^2)$.

The number $h_n = \sum_{i=1}^{n} \left(\frac{1}{i}\right)$, called the **harmonic number**, occurs frequently in the analysis of algorithms.

58. Compute h_4 and h_5. **59.** Define h_n recursively.

60. Prove that $\sum_{i=1}^{n} h_i = (n+1)h_n - n$, $n \geq 1$.

61. Prove that $h_2 m \geq 1 + \frac{m}{2}$, $m \geq 0$.

62. Prove that $h_n \leq \frac{n+1}{2}$.

***63.** (For those familiar with calculus) Let h_n denote the nth harmonic number $h_n = \sum_{i=1}^{n} \left(\frac{1}{i}\right)$. Show that $h_n = O(\lg n)$.
(*Hint*: Use integration.)

64. Solve the recurrence relation $g_n - g_{n-1} = 1/(n-1)!$, where $g_1 = 0$.

Let $a, b \in \mathbb{N}$ and $c, d \in \mathbb{R}^+$ with $b \geq 2$. Let f be a nondecreasing function such that $f(n) = af(n/b) + c$ and $f(1) = d$. Prove each.

****65.** If $a = 1$, then $f(n) = O(\lg n)$.

****66.** If $a \neq 1$, then $f(n) = O(n^{\log_b a})$.

Let $a, b, n \in \mathbb{N}$, $b \geq 2$, $c, d \in \mathbb{R}^+$, $f(1) = d$, and n is a power of b. Let f be a nondecreasing function such that $f(n) = af(n/b) + cn^2$. Prove each.

****67.** If $a = b^2$, then $f(n) = n^2 d + cn^2 \log_b n$.

****68.** If $a \neq b^2$, then $f(n) = An^2 + Bn^{\log_b a}$, where $A = \frac{b^2 c}{b^2 - a}$ and $B = d + \frac{b^2 c}{a - b^2}$.

****69.**
$$f(n) = \begin{cases} O(n^2) & \text{if } a < b^2 \\ O(n^2 \lg n) & \text{if } a = b^2 \\ O(n^{\log_b a}) & \text{if } a > b^2 \end{cases}$$

Chapter Summary

This chapter presented a new class of functions and hence sequences: recursively defined functions. The definitions of such functions can be translated into recursive algorithms. Just as the big-oh and big-theta notations worked well in analyzing the time complexities of algorithms, so does induction in proving the correctness of recursive algorithms.

Recursion

- The **recursive definition** of a function consists of one or more initial conditions and a recurrence relation (page 262).

Solving Recurrence Relations

- A simple class of recurrence relations can be solved using the iterative method (page 279).
- Every solution of the recurrence relation $a_n = a_{n-1} + f(n)$ is of the form $a_n = a_0 + \sum_{i=1}^{n} f(i)$ (page 280).

- Every solution of the recurrence relation $a_n = ca_{n-1} + 1$ is of the form $a_n = c^n a_0 + \dfrac{c^n - 1}{c - 1}$, where $c \neq 1$ (page 282).
- A ***k*th-order LHRRWCC** is of the form $a_n = \sum_{i=1}^{k} c_i a_{n-i}$, where $c_k \neq 0$ (page 287).
- The **characteristic equation** of this recurrence relation is $x^k - \sum_{i=1}^{k} c_i x^{k-i} = 0$ (page 287).
- The characteristic roots of a LHRRWCCs can be used to solve the LHRRWCCs (page 288).
- The general solution of a LNHRRWCCs is given by $a_n = a_n^{(h)} + a_n^{(p)}$ (page 294).

Generating Functions

- $g(x) = \sum_{n=0}^{\infty} a_n x^n$ is the **generating function** of the real number sequence $a_0, a_1, a_2, \ldots$ (page 298).
- Generating functions and the partial fraction decomposition rule can be used to solve LHRRWCCs (page 301).

Recursive Algorithms

- A **recursive algorithm** consists of two cases: base case(s) and a general case (page 307).
- **Lamé's Theorem** The euclidean algorithm for computing $\gcd\{a, b\}$ takes no more than five times the number of decimal digits in b, where $a \geq b \geq 2$ (page 323).

Divide-and-Conquer Algorithms

- The recurrence relation of a divide-and-conquer algorithm is of the form $f(n) = af(n/b) + g(n)$ (page 327).

Review Exercises

In Exercises 1 and 2, the nth term a_n of a number sequence is defined recursively. Compute a_5.

1. $a_1 = a_2 = 1, a_3 = 2$

$a_n = a_{n-1} + a_{n-2} + a_{n-3}, n \geq 4$

2. $a_1 = 0, a_2 = a_3 = 1$

$a_n = a_{n-1} + 2a_{n-2} + 3a_{n-3}, n \geq 4$

3. The number of additions a_n needed to compute the nth Fibonacci number F_n by recursion is given by $a_n = F_n - 1$, $n \geq 1$. Find the recurrence relation satisfied by a_n.

(**A modified handshake problem**) Mr. and Mrs. Matrix hosted a party for n married couples. At the party, each person shook hands with everyone else, except the spouse. Let $h(n)$ denote the total number of handshakes made.

4. Define $h(n)$ recursively.

5. Predict an explicit formula for $h(n)$.

6. Prove the formula obtained in Exercise 5, where $n \geq 1$.

Using the iterative method, predict an explicit formula satisfied by each recurrence relation.

7. $a_1 = 1 \cdot 2$

$a_n = a_{n-1} + n(n+1), n \geq 2$

8. $a_1 = 2 \cdot 3$

$a_n = 3a_{n-1}, n \geq 2$

9. $a_1 = 1$

$a_n = a_{n-1} + 2^{n-1}, n \geq 2$

10. $a_0 = 0$

$a_n = a_{n-1} + (3n - 1), n \geq 1$

11–14. Using induction, prove the formulas obtained in Exercises 7–10.

Solve each recurrence relation.

15. $a_n = a_{n-1} + a_{n-2}, a_1 = 2, a_2 = 3$

16. $a_n = a_{n-1} + a_{n-2}, a_1 = a_2 = a$

17. $a_n = 2a_{n-1} + 7a_{n-2} - 8a_{n-3} - 12a_{n-4}, a_0 = 4, a_1 = 10, a_2 = 18, a_3 = 58$

18. $a_n = 4a_{n-1} + 2a_{n-2} - 12a_{n-3} - 9a_{n-4}, a_0 = 4, a_1 = 0, a_2 = 4, a_3 = -32$

19. $a_n = 10a_{n-1} - 21a_{n-2} + 5n, a_0 = 0, a_1 = 3$

20. $a_n = 8a_{n-1} - 15a_{n-2} + 4n5^n, a_0 = 1, a_1 = 3$

21. Let a_n denote the number of multiplications (lines 7–10) in Algorithm 5.10. Show that $a_n = O(n)$.

Let c_n denote the number of element comparisons made (line 4) by the recursive bubble sort algorithm in Algorithm 5.9.

22. Define c_n recursively.

23. Solve the recurrence relation.

24. Show that $c_n = O(n^2)$.

Algorithm 5.13 evaluates the polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ at $x = \alpha$. Use it for Exercises 25–29.

```
Algorithm evaluate poly(f,n,α,answer)
(* This algorithm returns the value of a polynomial f
   of degree n at α in the variable answer. *)
0. Begin (* algorithm *)
1.    answer ← a0
2.    power ← 1
3.    for i = 1 to n do
4.    begin (* for *)
5.         power ← power * α
6.         answer ← answer + ai * power
7.    endfor
8. End (* algorithm *)
```

Algorithm 5.13

Evaluate each polynomial at $x = -1$.

25. $f(x) = x^3 + 2x^2 - 3x + 4$ **26.** $f(x) = 2x^3 + 5x - 6$

Let c_n denote the number of operations (lines 5–6) required to evaluate a polynomial at $x = \alpha$.

27. Define c_n recursively. **28.** Solve the recurrence relation.

29. Show that $c_n = O(n^2)$.

Use **Horner's algorithm** (Algorithm 5.14) to evaluate the polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ at $x = \alpha$ for Exercises 30–35.

```
Algorithm Horner(f,n,i,α)
(* This algorithm evaluates a polynomial f of degree n at
   x = α by recursion and is invoked by Horner(f,n,0,α). *)
0. Begin (* algorithm *)
1.   if i = n then
2.      Horner ← an
3.   else
4.      Horner ← Horner(f,n,i + 1,α) · α + ai
5. End (* algorithm *)
```

Algorithm 5.14

Evaluate each polynomial at $x = 2$.

30. $f(x) = 3x^2 + 4x - 5$ **31.** $f(x) = 2x^3 - 5x + 3$

Let b_n denote the number of operations (addition and multiplication) needed in line 4.

32. Define b_n recursively.

33. Solve the recurrence relation.

34. Show that $b_n = O(n)$.

35. Let a_n denote the number of n-bit words that do not contain the pattern 111. Define a_n recursively.

Let a_n denote the number of ways a $2 \times n$ rectangular board can be covered with 2×1 dominoes.

36. Define a_n recursively. **37.** Find an explicit formula for a_n. (*Hint*: Consider $2 \times (n-1)$ and $2 \times (n-2)$ boards.)

Write a recursive algorithm to compute each sum.

38. The sum of the first n even positive integers.

39. The sum of the first n odd positive integers.

40–41. Establish the correctness of the algorithms in Exercises 38 and 39.

42. Write an iterative algorithm to find the minimum and the maximum of a list X of n elements.

Let c_n denote the number of element comparisons made by the minmax algorithm in Exercise 42.

43. Define c_n recursively. **44.** Solve the recurrence relation.

45. Show that $b_n = O(n)$.

Prove each, where α and β are the solutions of the equation $x^2 = x + 1$, F_n the nth Fibonacci number, and L_n the nth Lucas number. Identities in Exercises 46–53 were discovered in 1876 by Lucas.

46. $\sum_{i=1}^{n} F_i = F_{n+2} - 1$ **47.** $\sum_{i=1}^{n} F_{2i-1} = F_{2n}$

48. $\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$ **49.** $\sum_{i=1}^{n} L_i = L_{n+2} - 3$

50. $\sum_{i=1}^{n} L_{2i-1} = L_{2n} - 2$ **51.** $\sum_{i=1}^{n} L_{2i} = L_{2n+1} - 1$

52. $F_{n+1}^2 + F_n^2 = F_{2n+1}$ **53.** $F_{n+1}^2 - F_{n-1}^2 = F_{2n}$

54. $\gcd\{F_n, F_{n+1}\} = 1, n \geq 1$ **55.** $x^n = F_n x + F_{n-1}, n \geq 2$

56. $F_n = \dfrac{\alpha^n - \beta^n}{\alpha - \beta}, n \geq 1$

Let $C(n)$ denote the number of comparisons needed by quicksort to sort a list of n items. In the worst case, $C(n) = C(n-1) + (n-1)$, where $C(0) = 0 = C(1)$.

57. Solve the recurrence relation. **58.** Show that $C(n) = O(n^2)$.

(*Note*: This shows that, in the worst case, quicksort is as bad as selection sort.)

Let $A(n)$ denote the average number of comparisons needed by quicksort. Then $A(n) = (n+1) + \frac{1}{n}\sum_{i=1}^{n}[A(i-1) + A(n-i)]$, where $A(0) = 0 = A(1)$. Use this fact to answer Exercises 59 and 60.

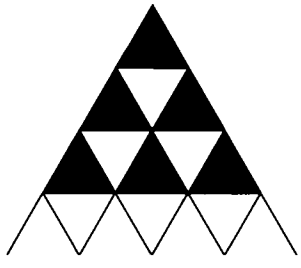
***59.** Show that $\frac{A(n)}{n+1} = 2\sum_{i=3}^{n+1}\left(\frac{1}{i}\right)$.

c ***60.** Show that $A(n) = O(n \lg n)$.
(*Hint*: Use integration.)

Supplementary Exercises

A side of the equilateral triangle in Figure 5.20 is n units long. Let a_n denote the number of triangles pointing north.

Figure 5.20



1. Define a_n recursively. **2.** Solve the recurrence relation.

The nth **Fermat number** f_n is defined by $f_n = 2^{2^n} + 1$, $n \geq 0$.

3. Prove that $f_{n+1} = f_n^2 - 2f_n + 2$. (J. M. Schram, 1983)

4. Using Exercise 3, compute f_1, f_2, f_3, and f_4.

5. Let a_n be an infinite sequence with $a_1 = 1$, $a_5 = 5$, $a_{12} = 144$, and $a_n + a_{n+3} = 2a_{n+2}$. Prove that $a_n = F_n$. (H. Larson, 1977)

6. Let $\alpha = \frac{1+\sqrt{5}}{2}$ and F_n the nth Fibonacci number. Prove that $\lim_{n\to\infty}\frac{F_{n+1}}{F_n} = \alpha$.

***7.** Let S_n denote the sum of the numbers in the nth term of the sequence of sets of pentagonal numbers $\{1\}$, $\{5, 12\}$, $\{22, 35, 51\}$, $\{70, 92, 117, 145\}, \ldots$. Find a formula for S_n.

***8.** Let S_n denote the sum of the numbers in the nth term of the sequence of sets of Fibonacci numbers $\{1\}$, $\{1, 2\}$, $\{3, 5, 8\}$, $\{13, 21, 34, 55\}, \ldots$. Find a formula for S_n.

Describe the behavior of each number sequence $\{a_n\}$, where $a_0 = a$, $a_1 = b$, and $a_2 = c$ are positive numbers. (R. L. Graham, 1991)

9. $a_{n+2} = (1 + a_{n+1})/a_n$ **10.** $a_{n+3} = (1 + a_{n+1} + a_{n+2})/a_n$

Let $n \in \mathbb{N}$ and φ Euler's phi-function. Define $\varphi^k = \varphi^{k-1} \circ \varphi$, where $\varphi^1 = \varphi$ and $\circ$ denotes composition. Let $f(n) = \varphi(n) + \varphi^2(n) + \varphi^3(n) + \cdots + \varphi(1)$. (D. L. Silverman, 1981)

11. Compute $f(5)$ and $f(8)$.

12. Prove that if $n = 2^k$, then $f(n) = n$.

13. Prove that $f(n)$ is even. [*Hint*: $\varphi(n)$ is even for $n > 2$.]

14. Consider the sequence of right triangles T_n, $n \geq 1$, with legs A_n and B_n, and hypotenuse C_n such that $A_{n+1} = B_n$ and $B_{n+1} = C_n$. Compute $\lim_{n\to\infty} \frac{B_n}{A_n}$ and $\lim_{n\to\infty} \frac{C_n}{B_n}$. (M. Flavio, 1980)

A set of integers A is **fat** if each of its elements is $\geq |A|$. For example, $\{5, 7, 91\}$ is a fat set, but $\{3, 7, 36, 41\}$ is not. $\emptyset$ is considered a fat set. Let f_n denote the number of fat subsets of the set $\{1, 2, \ldots, n\}$. (G. F. Andrews)

***15.** Define f_n recursively. ***16.** Find an explicit formula for f_n.

Let $f(n, k)$ denote the number of k-element subsets of the set $S = \{1, 2, \ldots, n\}$ that do not contain consecutive integers. Let f_n denote the total number of subsets of S that do not contain consecutive integers. (I. Kaplansky)

***17.** Define $f(n, k)$ recursively. ***18.** Find an explicit formula for f_n.

Computer Exercises

Write a program to perform each task.

1. Read in a positive integer $n \leq 20$, and print the various moves and the number of moves needed to transfer n disks from peg X to peg Z, using the rules in Example 5.4.
2. Read in a positive integer n, and print the first n triangular and tetrahedral numbers.
3. Print the triangular numbers ≤ 1000 that are perfect squares.
4. Print the triangular numbers ≤ 1000 that are primes.
5. There are eight palindromic triangular numbers < 1000. Find them.
6. Search for two triangular numbers t_n such that t_n and n are palindromic, where $9 \leq n \leq 100$.
7. Read in a positive integer n and print the first n Fibonacci numbers, using recursion and iteration.

8. Read in a positive integer $n \leq 20$ and print the first n Lucas numbers.
9. Read in a positive integer $n \leq 25$ and print the values of $\frac{F_{n+1}}{F_n}$ and $\frac{L_{n+1}}{L_n}$ correct to 10 decimal places, where F_n denotes the nth Fibonacci number and L_n the nth Lucas number.

Read in a list of n positive integers. Use recursion to print each.

10. Their sum, product, maximum, and minimum.
11. The list in the given order.
12. The list in the reverse order.
13. Read in a *key* and search the list for *key*. Print the location if the search is successful; otherwise, print a suitable message.
14. Read in a *key* and a sorted list of n items; determine if *key* occurs in the list using recursion and iteration. Print the location of *key* if the search is successful.
15. Read in a list of n words and determine if each is a palindrome, using recursion.
16. Read in two lists of n integers. Determine if they are identical, using recursion.
17. Read in a nonnegative real number x and a nonnegative integer n; compute the nth power of x.
18. Read in a positive integer $n \leq 100$ and a positive real number $x \leq 2$. Use the binary representation of n and the technique of successive squaring to compute x^n. Print the number of multiplications needed to compute it.
19. Read in a number α, and a polynomial $\sum_{i=0}^{n} a_i x^i$ (that is, coefficients and the corresponding exponents); print the value of the polynomial at α, using Horner's method.
20. Read in n positive integers and print their minimum and maximum, using both iteration and recursion.
21. Read in a positive integer $n \leq 10$ and arrange the Stirling numbers of the second kind $S(n, r)$ in a triangular form, where $1 \leq r \leq n$.
22. Read in n positive integers and sort them using bubble sort, selection sort, and insertion sort. Print the number of element-comparisons needed by each algorithm.
23. Read in n four-letter words. Sort them, using merge sort and quicksort. Print the number of element comparisons needed by each sort.

Exploratory Writing Projects

Using library and Internet resources, write a team report on each of the following in your own words. Provide a well-documented bibliography.

1. Describe the properties of Fibonacci numbers, their occurrences in nature, applications to various disciplines, and relationships to Lucas numbers.
2. Explain how the golden ratio is related to Fibonacci and Lucas numbers. Describe its various occurrences in nature.
3. Describe the various forms of Ackermann's function. Investigate its importance in the study of recursive functions and the analysis of algorithms.
4. Investigate the *Josephus problem*, named for the first century Jewish historian Flavius Josephus (37?–100?).
5. Describe how, using Fibonacci numbers F_n $(n \geq 2)$ as bases, nonnegative integers can be represented as binary numbers with no two adjacent 1's. Express the integers 1–25 as such binary numbers.
6. Define continued fractions and describe their relationship to Fibonacci numbers.
7. Describe the *Game of Life*, invented in 1970 by British mathematician John H. Conway, now at Princeton University.
8. Describe the *Game of Halma*, invented in 1883 by George H. Monks, a Harvard Medical School graduate.
9. Examine the history of Catalan numbers and their properties and applications. Include a biography of E. C. Catalan.
10. Write an essay on the Tower of Brahma (Hanoi).
11. Write an essay on Quicksort.
12. Discuss *the fifteen puzzle*, invented by American puzzlist Samuel Loyd (1841–1911).
13. Discuss *Markov chains*, named after Russian mathematician Andrei A. Markov (1856–1922), who developed the theory of stochastic processes, and their applications to business.

Enrichment Readings

1. G. Brassard and P. Bratley, *Algorithmics: Theory & Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 26–34, 48–61.

2. R. P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction*, 4th edition, Addison-Wesley, Boston, MA, 1999, pp. 351–403.
3. B. W. Jackson and D. Thro, *Applied Combinatorics with Problem Solving*, Addison-Wesley, Reading, MA, 1990, pp. 226–252.
4. T. Koshy, *Fibonacci and Lucas Numbers with Applications*, Wiley, New York, 2001.
5. C. Oliver, "The Twelve days of Christmas," *Mathematics Teacher*, Vol. 70 (Dec. 1977), pp. 752–754.
6. S. Sahni, *Concepts in Discrete Mathematics*, 2nd ed., Camelot, Fridley, MN, 1985, pp. 205–335.
7. R. Sedgewick, *Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1988, pp. 3–189.
8. K. B. Strangeman, "The Sum of n Polygonal Numbers," *Mathematics Teacher*, Vol. 67 (Nov. 1974), pp. 655–658.
9. C. W. Trigg, "Palindromic Triangular Numbers," *J. Recreational Mathematics*, Vol. 6 (Spring 1973), pp. 146–147.
10. A. Tucker, *Applied Combinatorics*, Wiley, New York, 1984, pp. 222–298.
11. H. S. Wilf, *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 26–34, 48–61.