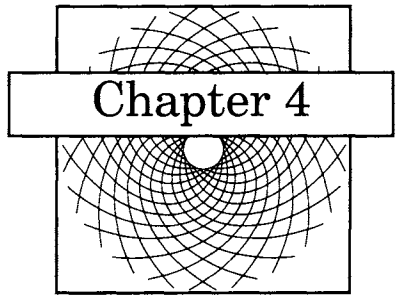


Induction and Algorithms

God created the natural numbers; all else is the work of man.

—L. KRONECKER

This chapter presents the well-ordering principle, the division algorithm with which you are already familiar, and some fundamental divisibility properties. In addition, through the well-ordering principle we will establish an additional proof technique, the principle of mathematical induction. Interesting applications of this principle, as well as the pigeonhole principle from Chapter 3, will be investigated.

Some of the intriguing problems pursued in this chapter lie below:

- Are there integers between 0 and 1?
- If n is a positive integer ≥ 2 and $a_1, a_2, \ldots, a_n \in \mathbf{Z}$, are there consecutive elements $a_{k+1}, a_{k+2}, \ldots, a_\ell$ such that $a_{k+1} + a_{k+2} + \cdots + a_\ell$ is divisible by n, where $k < \ell$?
- If $a_1, a_2, \ldots, a_n$ are the first n positive integers in some order, arranged around a circle, is it true that there must be a set of k consecutive elements in the cyclic arrangement whose sum is greater than $\lfloor [kn(n+1) - 2]/2n \rfloor$?
- Can any postage of $n \geq 2$ cents be paid using two- and three-cent stamps?

4.1 The Division Algorithm

The division algorithm, with which you are already familiar, is often employed to verify the correctness of a division problem. Its proof is based on the following cardinal fact, which is accepted as an axiom. (An **axiom** is a proposition that is accepted as true. It is usually a self-evident proposition and is consistent with known facts.)

The Well-Ordering Principle

Every nonempty set of positive integers has a least element. ■

For example, the set $\{13, 5, 8, 23\}$ has a least element, 5. The well-ordering principle applies to any nonempty subset S of $T = \{n \in \mathbf{Z} \mid n \geq n_0\}$, where n_0 is any integer. To see this, let $S^* = \{n - n_0 + 1 \mid n \in S\}$ and $T^* = \{n - n_0 + 1 \mid n \in T\}$. Since $S^* \subseteq T^*$ and $T^* \subseteq N$, by the well-ordering principle, S^* contains a least element ℓ^*. Then $n_0 + \ell^* - 1$ is a least element of S (why?).

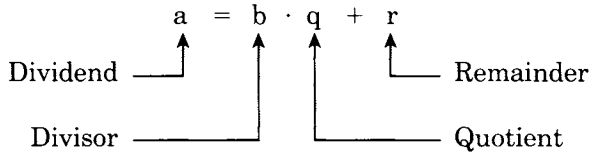
For example, let $S = \{-3, -1, 0, 1, 3, 5\}$ and $T = \{n \in \mathbf{Z} \mid n \geq -5\}$. Then $S^* = \{3, 5, 6, 7, 9, 11\}$ has a least element $\ell^* = 3$, so $n_0 + \ell^* - 1 = -5 + 3 - 1 = -3$ is the least element of S.

Next we present the division algorithm. Its proof is a bit complicated, so we omit it here; but a proof, using the well-ordering principle, can be established (see, for instance, the author's number theory book).

The Division Algorithm

When an integer a is divided by a positive integer b, we get a unique (integer) **quotient** q and a unique (integer) **remainder** r, where $0 \leq r < b$. The integer a is the **dividend** and b the **divisor**. This is formally stated as follows.

THEOREM 4.1 **(The Division Algorithm)** Let a be any integer and b any positive integer. Then there exist unique integers q and r such that



where $0 \leq r < b$. ■

Although this theorem does not present an algorithm for finding q and r, it has been traditionally called the division algorithm. The values of q and r can be found using the familiar long division method.

Notice that the equation $a = bq + r$ can be written as

$$\frac{a}{b} = q + \frac{r}{b}$$

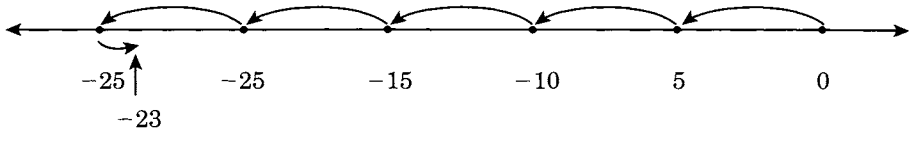
so $q = a \text{ div } b = \lfloor a/b \rfloor$ and $r = a - bq = a \bmod b$.

The next example shows that we should be careful in finding the quotient and the remainder when the dividend is negative.

EXAMPLE 4.1 Find the quotient q and the remainder r when -23 is divided by 5.

SOLUTION:
Since $-23 = 5 \cdot (-4) + (-3)$, you might be tempted to say that $q = -4$ and $r = -3$. Recall that the remainder can *never* be negative, so we rewrite -23 as $-23 = 5 \cdot (-5) + 2$, where $0 \leq r(= 2) < 5$ (see the number line in Figure 4.1). Thus $q = -5$ and $r = 2$; in other words, -23 div $5 = -5$ and -23 mod $5 = 2$.

Figure 4.1



We close this section with two applications of the division algorithm and the pigeonhole principle. ■

EXAMPLE 4.2 Let b be an integer ≥ 2. If $b + 1$ distinct integers are randomly selected, prove that the difference of some two of them must be divisible by b.

PROOF
Let q be the quotient and r the remainder when an integer a is divisible by b. Then, by the division algorithm, $a = bq + r$ where $0 \leq r < b$. The $b + 1$ distinct integers yield $b + 1$ remainders (pigeons); but there are only b possible remainders (pigeonholes). Therefore, by the pigeonhole principle, two of the remainders must be equal.

Let x and y be the corresponding integers. Then $x = bq_1 + r$ and $y = bq_2 + r$ for some quotients q_1 and q_2. Then

$$\begin{aligned} x - y &= (bq_1 + r) - (bq_2 + r) \\ &= b(q_1 - q_2) \end{aligned}$$

Thus, $x - y$ is divisible by b. ■

EXAMPLE 4.3 Let n be an integer ≥ 2 and let $a_1, a_2, \ldots, a_n \in \mathbf{Z}$. Prove that there exist integers k and ℓ such that $a_{k+1} + a_{k+2} + \cdots + a_\ell$ is divisible by n, where $1 \leq k < \ell \leq n$; that is, there exist consecutive elements $a_{k+1}, a_{k+2}, \ldots, a_\ell$ whose sum is divisible by n.

PROOF (by cases):
Consider the n sums $S_i = a_1 + a_2 + \cdots + a_i$, where $1 \leq i \leq n$.

Case 1 If any of the sums S_i is divisible by n, then the statement is true.

Case 2 Suppose none of the sums S_i is divisible by n. When S_i is divided by n, the remainder must be nonzero. So, by the division algorithm, the possible remainders are $1, 2, \ldots, (n-1)$. Since there are n sums and $n-1$ possible remainders, by the pigeonhole principle, two of the sums S_k and S_ℓ must yield the same remainder r when divided by n, where $k < \ell$.

Therefore, there must exist integers q_1 and q_2 such that $a_1 + a_2 + \cdots + a_k = nq_1 + r$ and $a_1 + a_2 + \cdots + a_\ell = nq_2 + r$, where $k < \ell$. Subtracting, we get $a_{k+1} + a_{k+2} + \cdots + a_\ell = n(q_1 - q_2)$. Thus $a_{k+1} + a_{k+2} + \cdots + a_\ell$ is divisible by n. ■

To cite a specific example, consider the seven integers 2, 3, 8, 15, 23, 29, and 57. Then $S_1 = a_1 = 2 = 0 \cdot 7 + 2$ and $S_5 = a_1 + a_2 + a_3 + a_4 + a_5 = 2 + 3 + 8 + 15 + 23 = 51 = 7 \cdot 7 + 2$. Then $S_5 - S_1 = a_2 + a_3 + a_4 + a_5 = 3 + 8 + 15 + 23 = 49$ is divisible by 7. Here $k = 1$ and $\ell = 5$. (You may notice that $S_4 = a_1 + a_2 + a_3 + a_4 = 2 + 3 + 8 + 15$ is also divisible by 7.)

Exercises 4.1

1. Is the set of positive odd integers well-ordered?

2. Is the set of positive even integers well-ordered?

In Exercises 3–6, find the quotient and the remainder when the first integer is divided by the second.

3. 137, 11 **4.** 15, 23 **5.** −43, 16 **6.** −37, 73

Find the set of possible remainders when an integer is divided by the given integer.

7. Two **8.** Five **9.** Seven **10.** Twelve

11. Prove that there exists no integer between 0 and 1.

12. Let $a \in \mathbf{Z}$. Prove that no integer exists between a and $a + 1$.

13. Let $n_0 \in \mathbf{Z}, S$ be a nonempty subset of the set $T = \{n \in \mathbf{Z} \mid n \geq n_0\}$, and ℓ^* be a least element of the set $T^* = \{n - n_0 + 1 \mid n \in T\}$. Prove that $n_0 + \ell^* - 1$ is a least element of S.

14. Using the well-ordering principle, prove that 1 is the smallest positive integer.
(*Hint*: Prove by contradiction.)

***15.** Let $a \in \mathbf{Z}, S = \{a, a+1, \ldots\}, T \subseteq S$, and $a \in T$. Let k be any element of S such that whenever $k \in T$, $k + 1 \in T$. Prove that $S = T$.

***16.** Let $a \in \mathbf{Z}$ and $S = \{a, a+1, \ldots\}$. Let $P(n)$ be a predicate on S such that the following conditions are satisfied: (1) $P(a)$ is true; (2) If $P(a)$,

$P(a+1), \ldots, P(k)$ are true for any $k \geq a$, then $P(k+1)$ is also true. Prove that $P(n)$ is true for every $n \geq a$.

4.2 Divisibility Properties

The celebrated euclidean algorithm can be used to find the greatest common divisor of two positive integers, but first a very few properties of prime and composite numbers, and some divisibility properties.

Let a and b ($\neq 0$) be any two integers. If there is an integer q such that $a = bq$, we say b **divides** a, b is a **factor** of a, a is **divisible** by b, or a is a **multiple** of b. We then write $b \mid a$; otherwise, $b \nmid a$. (Again, the meaning of the vertical bar should be clear from the context.) For instance, $3 \mid 6$, $8 \mid 24$, but $6 \nmid 14$.

A positive factor b of a positive integer a is a **proper factor** of a if $b \neq a$. For example, the proper factors of 6 are 1, 2, and 3.

There are positive integers with exactly two positive factors. Accordingly, we make the following definition.

Prime Numbers and Composite Numbers

A positive integer > 1 is a **prime number** (or simply a **prime**) if its only positive factors are 1 and itself. A positive integer > 1 is a **composite number** if it is not a prime.

For example, 2 and 19 are primes, whereas 6 and 21 are composite numbers (why?).

There is a systematic procedure for determining whether or not a positive integer $n \geq 2$ is a prime. It is based on the next theorem.

THEOREM 4.2 Any composite number n has a prime factor $\leq \lfloor \sqrt{n} \rfloor$.

PROOF (by contradiction):
Since n is composite, there are positive integers a and b such that $n = ab$ where $1 < a < n$ and $1 < b < n$. Suppose $a > \sqrt{n}$ and $b > \sqrt{n}$. Then $n = ab > \sqrt{n} \cdot \sqrt{n} = n$, which is impossible. Therefore, either $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Since both a and b are integers, it follows that either $a \leq \lfloor \sqrt{n} \rfloor$ or $b \leq \lfloor \sqrt{n} \rfloor$.

By the fundamental theorem of arithmetic (see Theorem 4.13), every positive integer has a prime factor. Any such factor of a or b is also a factor of $a \cdot b = n$, so n must have a prime factor $\lfloor \sqrt{n} \rfloor$. ■

> It follows from Theorem 4.2 that if n has *no* prime factors $\leq \lfloor \sqrt{n} \rfloor$, then n is a prime; otherwise, it is a composite number.

This fact can be used to determine whether or not an integer $n \geq 2$ is a prime, as the next example illustrates.

EXAMPLE 4.4 Determine if 1601 is a prime number.

SOLUTION:
First list all primes $\leq \lfloor\sqrt{1601}\rfloor$. They are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, and 37. None of them is a factor of 1601 (verify); so 1601 is a prime. ■

An algorithm for determining the primality of a positive integer n ≥ 2 is given in Algorithm 4.1.

```
Algorithm prime number(n)
(* This algorithm determines if a positive integer n≥2 is
   prime or not using Theorem 4.2. *)
 Begin (* algorithm *)
   list all primes ≤ ⌊√n⌋
   if any of them is a factor of n then
     n is not a prime
   else
     n is a prime
 End (* algorithm *)
```

Algorithm 4.1

In the remainder of this section we discuss some useful divisibility properties. We begin with a simple and straightforward property.

THEOREM 4.3 If a and b are positive integers such that $a \mid b$ and $b \mid a$, then $a = b$. ■

> Notice that this theorem does *not* hold if a and b are any integers. For example, $3 \mid (-3)$ and $(-3) \mid 3$, but $3 \neq -3$.

THEOREM 4.4 Let a, b, and c be any integers. Then:

(1) If $a \mid b$ and $b \mid c$, then $a \mid c$ (**transitive property**).
(2) If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.
(3) If $a \mid b$ and $a \mid c$, then $a \mid (b - c)$.
(4) If $a \mid b$, then $a \mid bc$.

PROOF:
We shall prove properties 1 and 2, and leave the others as exercises.

(1) Since $a \mid b$, there exists an integer q_1 such that $b = aq_1$. Similarly, there exists an integer q_2 such that $c = bq_2$. Then $c = bq_2 = (aq_1)q_2 = a(q_1q_2)$. Thus, there exists an integer $q = q_1q_2$ such that $c = aq$. Therefore, $a \mid c$.

(2) As above, we have $b = aq_1$ and $c = aq_3$. Then $b + c = aq_1 + aq_3 = a(q_1 + q_3)$. Since $q_1 + q_3$ is an integer, it follows that $a \mid (b + c)$. ■

The Greatest Common Divisor

A positive integer can be a factor of two positive integers a and b. Such a positive integer is a **common factor** of a and b. The largest such common factor is the **greatest common divisor** (gcd) of a and b, denoted by $\gcd\{a, b\}$.

For instance, $\gcd\{6, 9\} = 3$, $\gcd\{12, 24\} = 12$, and $\gcd\{6, 35\} = 1$.

This definition of gcd, although simple and clear, is not practical, so we give an alternate, equivalent definition below.

An Alternate Definition of GCD

A positive integer d is the **gcd** of two positive integers a and b if:

- $d \mid a$ and $d \mid b$; and
- if $d' \mid a$ and $d' \mid b$, then $d' \mid d$, where d' is a positive integer.

Thus, d is $\gcd\{a, b\}$ if (1) d is a common divisor of both a and b; and (2) any common divisor of a and b is also a divisor of d.

The next theorem, an extremely useful and powerful result, can be applied to develop an algorithm to compute $\gcd\{a, b\}$.

THEOREM 4.5 Let a and b be any positive integers, and r the remainder when a is divided by b. Then $\gcd\{a, b\} = \gcd\{b, r\}$.

PROOF

Let $\gcd\{a, b\} = d$ and $\gcd\{b, r\} = d'$. To prove that $d = d'$, it suffices to show that $d \mid d'$ and $d' \mid d$. By the division algorithm, a unique quotient q exists such that

$$a = bq + r \tag{4.1}$$

To show that $d \mid d'$:

Since $d = \gcd\{a, b\}$, $d \mid a$ and $d \mid b$. Therefore, $d \mid bq$, by Theorem 4.4. Then $d \mid (a - bq)$, again by Theorem 4.4. In other words, $d \mid r$, by Equation (4.1). Thus, $d \mid b$ and $d \mid r$. Therefore, $d \mid \gcd\{b, r\}$; that is, $d \mid d'$.

Similarly, it can be shown that $d' \mid d$. (See Exercise 33.) Thus, by Theorem 4.3, $d = d'$; that is, $\gcd\{a, b\} = \gcd\{b, r\}$. ■

EXAMPLE 4.5 Illustrate Theorem 4.5, using $a = 108$ and $b = 20$.

SOLUTION:

$\gcd\{108, 20\} = 4$ (verify). When 108 is divided by 20, the remainder is 8. $\gcd\{20, 8\} = 4$ (verify). Thus, $\gcd\{108, 20\} = \gcd\{20, 8\}$. ■

Euclidean Algorithm

Among several procedures for finding the gcd of two positive integers, one efficient algorithm is the **euclidean algorithm**, named after the

Little is known about Euclid's life. He taught at the University of Alexandria and founded the Alexandrian School of Mathematics. When the Egyptian ruler King Ptolemy I asked Euclid if there were an easier way to learn geometry than by studying The Elements, *he replied, "There is no royal road to geometry." Euclid is called the* father of geometry.

No work, except for the Bible, has been more widely read, studied, or edited," according to J. E. Lightner of Western Maryland College, Westminister, Maryland. "More than 2000 editions of the work have appeared since the first printed one in 1482; however, no extant copy of The Elements *dates from Euclid's own time."*

Greek mathematician Euclid (330?–275 B.C.), who included it in his extraordinary work *The Elements*. The algorithm repeatedly applies the division algorithm and Theorem 4.5. Before formally discussing the algorithm, we illustrate it in the next example.

EXAMPLE 4.6 Find $\gcd\{1976, 1776\}$.

SOLUTION:
Apply the division algorithm with 1976 (the larger of the two numbers) as the dividend and 1776 as the divisor:

$$1976 = 1 \cdot 1776 + 200$$

Apply the division algorithm again with 1776 and 200, using 1776 as the dividend and 200 as the divisor:

$$1776 = 8 \cdot 200 + 176$$

Continue this procedure until a zero remainder is obtained:

$$\begin{aligned} 1976 &= 1 \cdot 1776 + 200 \\ 1776 &= 8 \cdot 200 + 176 \\ 200 &= 1 \cdot 176 + 24 \\ 176 &= 7 \cdot 24 + 8 \qquad \longleftarrow \text{last nonzero remainder} \\ 24 &= 3 \cdot 8 + 0 \end{aligned}$$

The last nonzero remainder in this procedure is the gcd. Thus $\gcd\{1976, 1776\} = 8$. ■

Will this method work for any two positive integers a and b? If $a = b$, then $\gcd\{a, b\} = a$. So assume, for convenience, $a > b$. (If this is not true,

simply switch them.) Let $r_0 = b$. Then by successive application of the division algorithm, we get a sequence of equations:

$$\begin{aligned} a &= q_0 r_0 + r_1 & 0 \le r_1 < r_0 \\ r_0 &= q_1 r_1 + r_2 & 0 \le r_2 < r_1 \\ r_1 &= q_2 r_2 + r_3 & 0 \le r_3 < r_2 \\ &\vdots \end{aligned}$$

Continuing like this, we get the following sequence of remainders:

$$b = r_0 > r_1 > r_2 > r_3 > \cdots \ge 0$$

Since the remainders are nonnegative and getting smaller and smaller, this sequence must eventually terminate with remainder $r_n = 0$. Thus, the last two equations in the above procedure are:

$$r_{n-2} = q_{n-1} r_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$$

and

$$r_{n-1} = q_n r_n$$

It then follows that $\gcd\{a, b\} = \gcd\{a, r_0\} = \gcd\{r_0, r_1\} = \gcd\{r_1, r_2\} = \cdots = \gcd\{r_{n-1}, r_n\} = r_n$, the last nonzero remainder. (This can be established by using mathematical induction; see Exercise 56 in Section 4.4.)

EXAMPLE 4.7 Apply the euclidean algorithm to find $\gcd\{2076, 1024\}$.

SOLUTION:
By the successive application of the division algorithm, we get:

$$\begin{aligned} 2076 &= 2 \cdot 1024 + 28 \\ 1024 &= 36 \cdot 28 + 16 \\ 28 &= 1 \cdot 16 + 12 \\ 16 &= 1 \cdot 12 + 4 \qquad \longleftarrow \text{last nonzero remainder} \\ 12 &= 3 \cdot 4 + 0 \end{aligned}$$

Since the last nonzero remainder is 4, $\gcd\{2076, 1024\} = 4$. ■

The euclidean algorithm is formally presented in Algorithm 4.2.

```
Algorithm Euclid(x,y,divisor)
(* This algorithm returns gcd{x,y} in divisor, where
   x ≥ y > 0. *)
 0. Begin (* algorithm *)
 1.   dividend ← x
 2.   divisor ← y
```

```
3.   remainder ← dividend mod divisor
4.   while remainder > 0 do (* update dividend,
                               divisor, and remainder *)
5.   begin (* while *)
6.     dividend ← divisor
7.     divisor ← remainder
8.     remainder ← dividend mod divisor
9.   endwhile
10. End (* algorithm *)
```

Algorithm 4.2

The euclidean algorithm provides a procedure for expressing the gcd of two positive integers in terms of themselves, as the next example shows.

EXAMPLE 4.8 Example 4.7 showed that $\gcd\{2076, 1024\} = 4$. Express the gcd in terms of 2076 and 1024.

SOLUTION:
We use the equations in Example 4.7 in the reverse order:

$$\begin{aligned} 4 &= 16 - 1 \cdot 12 & &= 16 - 1 \cdot (28 - 1 \cdot 16) \\ &= 2 \cdot 16 - 1 \cdot 28 & &= (1024 - 36 \cdot 28) - 1 \cdot 28 \\ &= 2 \cdot 1024 - 72 \cdot 28 - 1 \cdot 28 & &= 2 \cdot 1024 - 73 \cdot 28 \\ &= 2 \cdot 1024 - 73(2076 - 2 \cdot 1024) & &= 2 \cdot 1024 - 73 \cdot 2076 + 146 \cdot 1024 \\ &= (-73) \cdot 2076 + 148 \cdot 1024 \end{aligned}$$

(You may verify this by direct computation.) ■

Example 4.8 can be generalized as in the following theorem. We omit its proof.

THEOREM 4.6 Let a and b be any positive integers, and $d = \gcd\{a, b\}$. Then there exist integers s and t such that $d = sa + tb$. ■

> *Note*: (1) The expression $sa + tb$ is called a **linear combination** of a and b. (2) The integers s and t are *not* unique. For example, $\gcd\{28, 12\} = 4$ and $4 = 1 \cdot 28 + (-2) \cdot 12 = (-2) \cdot 28 + 5 \cdot 12$. (3) The integers s and t can be found by using the various equations in the euclidean algorithm, or by trial and error especially when a and b are fairly small.

Theorem 4.6 can be used to derive other divisibility properties. To this end, we define two positive integers to be **relatively prime** if their gcd is 1. For example, 6 and 35 are relatively prime, whereas 12 and 18 are not relatively prime.

THEOREM 4.7 Let a and b be relatively prime numbers. If $a \mid bc$, then $a \mid c$.

PROOF:
Since a and b are relatively prime, Theorem 4.6 indicates integers s and t exist such that $sa + tb = 1$. Then $sac + tbc = c$. By Theorem 4.4, $a \mid (sac)$ and $a \mid (tbc)$. Therefore, by Theorem 4.4, $a \mid (sac + tbc)$; that is, $a \mid c$. ■

The following exercises offer additional divisibility properties to verify; again, consult a number theory book.

Exercises 4.2

Determine if each positive integer is a prime.

1. 727 **2.** 1001 **3.** 1681 **4.** 1723

5. Prove or disprove: Every prime is a perfect number.

Using the euclidean algorithm, find the gcd of the given integers.

6. 2024, 1024 **7.** 2076, 1076 **8.** 2076, 1776 **9.** 3076, 1976

In Exercises 10–13, express the gcd of the given integers as a linear combination of them.

10. 12, 9 **11.** 18, 28 **12.** 12, 29 **13.** 28, 15

14. Two prime numbers that differ by 2 are called **twin primes**. For example, 5 and 7 are twin primes. Prove that one more than the product of two twin primes is a perfect square. (Twin primes played a key role in 1994 in establishing a flaw in the Pentium chip, manufactured by Intel Corporation.)

Evaluate each sum, where d is a positive integer.

15. $\sum_{d \mid 6} d$ **16.** $\sum_{d \mid 12} 1$ **17.** $\sum_{d \mid 18} \left(\frac{1}{d}\right)$ **18.** $\sum_{d \mid 18} \left(\frac{18}{d}\right)$

Disprove each statement, where a, b, and c are arbitrary integers.

19. If $a \mid (b + c)$, then $a \mid b$ and $a \mid c$. **20.** If $a \mid bc$, then $a \mid b$ and $a \mid c$.

(Easter Sunday) Here is a second method* for determining Easter Sunday in a given year N. Let $a = N \mod 19$, $b = N \operatorname{div} 100$, $c = N \mod 100$, $d = b \operatorname{div} 4$, $e = b \mod 4$, $f = (b+8) \operatorname{div} 25$, $g = (b-f+1) \operatorname{div} 3$, $h = (19a + b - d - g + 15) \mod 30$, $i = c \operatorname{div} 4$, $j = c \mod 4$, $k = (32 + 2e + 2i - h - j) \mod 7$, $\ell = (a + 11h + 22k) \operatorname{div} 451$, $m = (h + k - 7\ell + 114) \operatorname{div} 31$, and $n = (h + k - 7\ell + 114) \mod 31$. Then Easter Sunday falls on the $(n + 1)$st

*Based on "To Find Easter," *Nature* (April 20, 1876). For bringing this method to his attention, the author would like to thank Thomas Moore of Bridgewater State College.

day of the mth month of the year. Compute the date for Easter Sunday in each year.

21. 2000 **22.** 2076 **23.** 3000 **24.** 3663

Euler's phi-function φ is another important number-theoretic function on $\mathbb{N}$, defined by $\varphi(n)$ = number of positive integers $\leq n$ and relatively prime to n. For example, $\varphi(1) = 1 = \varphi(2), \varphi(3) = 2 = \varphi(4)$, and $\varphi(5) = 4$. Evaluate $\varphi(n)$ for each value of n.

25. 10 **26.** 15 **27.** 17 **28.** 24

29. Compute $\sum_{d|n} \varphi(d)$ for $n = 5$, 6, 10, and 12.

30. Using Exercise 29, predict a formula for $\sum_{d|n} \varphi(d)$.

Let a, b, c, and n be any positive integers and p be any prime. Prove each.

31. If $a \mid b$ and $a \mid c$, then $a \mid (b - c)$.

32. If $a \mid b$, then $a \mid bc$.

33. Let r be the remainder when a is divided by b. Let $d = \gcd\{a, b\}$ and $d' = \gcd\{b, r\}$. Then $d' \mid d$.

34. Let $a > b$. Then $\gcd\{a, b\} = \gcd\{a, a - b\}$.

35. Let $a > b$. Then $\gcd\{a, b\} = \gcd\{b, a + b\}$.

36. The gcd of a and b is unique.
(*Hint*: Assume two gcd's d and d'; show that $d = d'$.)

37. If $p \mid ab$, then $p \mid a$ or $p \mid b$.
[*Hint*: Assume $p \mid ab$ and $p \not\mid a$. Since $p \not\mid a$, $\gcd\{p, a\} = 1$.]

38. Any two consecutive integers are relatively prime.

39. Let $d = \gcd\{a, b\}$. Then a/d and b/d are relatively prime.

40. $\gcd\{na, nb\} = n \cdot \gcd\{a, b\}$ **41.** $\gcd\{\gcd\{a, b\}, c\} = \gcd\{a, \gcd\{b, c\}\}$

42. Let $a \mid c$ and $b \mid c$, where a and b are relatively prime numbers. Then $ab \mid c$.

43. 2 and 3 are the only two consecutive integers that are primes.

44. 3, 5, and 7 are the only three consecutive odd integers that are primes.

45. If p and $p^2 + 8$ are primes, then $p^3 + 4$ is also a prime. (D. L. Silverman, 1968)

46. If p and $p + 2$ are twin primes, then p must be odd.

47. Suppose p and q are primes such that $p - q = 3$. Then $p = 5$.

48. Every odd prime is of the form $4n + 1$ or $4n + 3$.

Disprove each statement.

49. If $\gcd\{a,b\} = 1$ and $\gcd\{b,c\} = 1$, then $\gcd\{a,c\} = 1$, where a, b, and c are positive integers.

50. $n! + 1$ is a prime for every $n \geq 0$.

51. $E_n = p_1 p_2 \cdots p_n + 1$ is a prime, where p_i denotes the ith prime and $i \geq 1$.

52. Let n be a positive integer. Prove that $(n+1)! + 2$, $(n+1)! + 3$, ... , $(n+1)! + (n+1)$ are n consecutive composite numbers.

4.3 Nondecimal Bases

In everyday life we use the decimal notation, base ten, to represent any real number. For example, $234 = 2(10^2) + 3\ (10^1) + 4(10^0)$, which is the **decimal expansion** of 234. Likewise, $23 \cdot 45 = 2(10^1) + 3(10^0) + 4(10^{-1}) + 5(10^{-2})$. Computers use base two (*binary*), and very long binary numbers are often handled by humans (as opposed to computers) using bases eight (*octal*) and sixteen (*hexadecimal*).

Actually, any positive integer $b \geq 2$ is a valid choice for a base. This is a consequence of the following fundamental result.

THEOREM 4.8 Let b be a positive integer ≥ 2. Then every positive integer a can be expressed uniquely in the form $a = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$, where $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, $a_k \neq 0$, and $k \geq 0$. ■

This leads us to the following definition.

Base-*b* Representation

The expression $a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$ is the **base-*b* expansion** of the integer a. Accordingly, we write $a = (a_k a_{k-1} \cdots a_1 a_0)_b$ in base b. The base is omitted when it is 10.

For example, $234 = 234_{\text{ten}}$ and $22 = 10110_{\text{two}}$ (see Example 4.9).

When the base is greater than 10, to avoid confusion we use the letters A, B, C, ... to represent the *digits* 10, 11, 12, ... , respectively. It is easy to find the decimal value of an integer from its base-b representation, as the next example illustrates.

EXAMPLE 4.9 Express 10110_{two} in base 10.

SOLUTION:

$$\begin{aligned} 10110_{\text{two}} &= 1(2^4) + 0(2^3) + 1(2^2) + 1(2^1) + 0(2^0) \quad \longleftarrow \text{binary expansion} \\ &= 16 + 0 + 4 + 2 + 0 \\ &= 22 \end{aligned}$$

■

Conversely, suppose we are given a decimal integer. How do we express it in another base b? By Theorem 4.8, all we have to do is express it as a sum of powers of b, then simply collect the coefficients in the right order. Always remember to account for missing coefficients.

A Brainteaser

Take a look at the tablets A, B, C, D, and E in Figure 4.2. Assuming you are under 32 years old, identify the tablets on which your age appears; we can then easily tell your age. For example, if your age appears on tablets A, B, C, and E, then you must be 23. Can you explain how this puzzle works?

Figure 4.2

A		B		C		D		E	
1	17	2	18	4	20	8	24	16	24
3	19	3	19	5	21	9	25	17	25
5	21	6	22	6	22	10	26	18	26
7	23	7	23	7	23	11	27	19	27
9	25	10	26	12	28	12	28	20	28
11	27	11	27	13	29	13	29	21	29
13	29	14	30	14	30	14	30	22	30
15	31	15	31	15	31	15	31	23	31

Returning to nondecimal representations, a simple algorithm expresses an integer a in any nondecimal base b: divide a, and its successive quotients by b until a zero quotient is reached, then pick the remainders in the reverse order. These steps can be translated into the elegant algorithm given in Algorithm 4.3.

```
Algorithm nondecimal base(n,b)
(* This algorithm finds the base-b representation (a_m a_{m-1} ... a_1 a_0)_b
   of a positive integer n. The variables q and r denote the quotient
   of the remainder of the division algorithm, and i is a subscript. *)
  Begin (* algorithm *)
    (* initialize the variables q, r, and i *)
    q ← n; i ← 0
    while q > 0 do
    begin (* while *)
      r ← q mod b
      a_i ← r
      q ← q div b
```

```
    i ← i+1
  endwhile
End (* algorithm *)
```

Algorithm 4.3

The next example illustrates this algorithm.

EXAMPLE 4.10 Represent 15,036 in the hexadecimal system, that is, in base 16.

SOLUTION:
Applying Algorithm 4.3 we have:

$$\begin{aligned} 15036 &= 939 \cdot 16 + 12 \\ 939 &= 58 \cdot 16 + 11 \quad \uparrow \\ 58 &= 3 \cdot 16 + 10 \quad \text{read up} \\ 3 &= 0 \cdot 16 + 3 \end{aligned}$$

Thus $15{,}036 = 3\text{ABC}_{\text{sixteen}}$. ■

Addition in Base b

Before we discuss how to add nondecimal numbers, let us examine the familiar addition algorithm in base 10.

To find the sum of any two decimal digits a and b, we find the remainder $r = (a + b) \bmod 10$ and the quotient $q = (a + b) \text{ div } 10$. Then $a + b = (qr)_{\text{ten}}$; q is the **carry** resulting from the addition of a and b. Using this idea we can add any two decimal integers.

Fortunately, the addition algorithm can be extended to any nondecimal base b in an obvious way. For example, let $x = (x_m \ldots x_0)_b$ and $y = (y_n \ldots y_0)_b$ where $m \geq n$. If $m > n$, we could assume that $y_{n+1} = \cdots = y_m = 0$. We add the corresponding digits in x and y in a right-to-left fashion. Let $s_i = (x_i + y_i + c_i) \bmod b$ and $c_{i+1} = (x_i + y_i + c_i) \text{ div } b$, where $c_0 = 0$. Then $x + y = (s_{m+1}s_m \ldots s_0)_b$ where s_{m+1} may be 0 or 1. (Leading zeros are deleted from the answer.)

These steps translate into a straightforward algorithm, as in Algorithm 4.4.

```
Algorithm addition (x,y,s,b)
(* This algorithm computes the sum s = (s_{m+1}s_m...s_0) of two
   integers x = x_m...x_0 and y = y_n...y_0 in base b, where m ≥ n. *)
  Begin (* algorithm *)
    carry ← 0 (* initialize carry *)
    for i = 0 to n do
    begin (* for *)
      s_i ← (x_i + y_i + carry) mod b
      carry ← (x_i + y_i + carry) div b
    endfor
    for i = n+1 to m do
```

```
    begin (* for *)
      s_i ← (x_i + carry) mod b
      carry ← (x_i + carry) div b
    endfor
    if carry > 0 then
      s_{m+1} ← carry
End (* algorithm *)
```

Algorithm 4.4

This algorithm is illustrated in the next two examples.

EXAMPLE 4.11 Add the binary integers 10110_{two} and 1011_{two}.

SOLUTION:
First write the integers one below the other in such a way that the corresponding bits are vertically aligned. See Figure 4.3. (For convenience, the base two is not shown.)

Figure 4.3

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & 0 \\ + & & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Figure 4.4

$$\begin{array}{cccccc} & & & & \textcircled{0} & \\ & 1 & 0 & 1 & 1 & 0 \\ + & & 1 & 0 & 1 & 1 \\ \hline & & & & & 1 \end{array}$$

Figure 4.5

$$\begin{array}{cccccc} & & & \textcircled{1} & \textcircled{0} & \\ & 1 & 0 & 1 & 1 & 0 \\ + & & 1 & 0 & 1 & 1 \\ \hline & & & & 0 & 1 \end{array}$$

Figure 4.6

$$\begin{array}{cccccc} \textcircled{1} & \textcircled{1} & \textcircled{1} & \textcircled{1} & \textcircled{0} & \\ & 1 & 0 & 1 & 1 & 0 \\ + & & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Add the corresponding bits from right to left, beginning with the one's column: $0 + 1 = 1$. Since $1 \bmod 2 = 1$, enter 1 as the one's bit in the sum. Since $1 \operatorname{div} 2 = 0$, the resulting carry is 0, shown circled in Figure 4.4. (In practice when the carry is 0, it is simply ignored.) Now add the bits 0, 1, and 1 in the twos column: $0 + 1 + 1 = 2$. Since $2 \bmod 2 = 0$ and $2 \operatorname{div} 2 = 1$, enter 0 in the twos column and the new carry is 1 (see Figure 4.5). Continuing like this, we get the sum 100001_{two}. See Figure 4.6. ■

The addition of binary numbers can be made easy by observing that $0+0=0$, $0+1=1=1+0$, and $1+1=10$, all in base two.

Next we illustrate the multiplication algorithm in base b.

Multiplication in Base b

The traditional algorithm for multiplying two integers x and y works for any base in an obvious way: multiply every digit in x by every digit in y as in base b and add up the partial products, as in Example 4.12.

EXAMPLE 4.12 Multiply 1011_{two} and 101_{two}.

SOLUTION:
The various steps unfold in Figures 4.7–4.9. The product is 110111_{two}.

Figure 4.7

```
    1 0 1 1
×     1 0 1
-----------
    1 0 1 1     ← multiply 1011 by 1
```

Figure 4.8

```
      1 0 1 1
    ×   1 0 1
    ---------
      1 0 1 1
    0 0 0 0       ← multiply 1011 by 0
1 0 1 1           ← multiply 1011 by 1
```

Figure 4.9

```
      1 0 1 1
    ×   1 0 1
    ---------
      1 0 1 1   }
    0 0 0 0     } add the partial products
1 0 1 1         }
-------------
1 1 0 1 1 1
```

■

Shifting and Binary Multiplication

If you found these two examples confusing, don't be discouraged. Fortunately, most computers do binary multiplications using a technique called **shifting**, as discussed below.

Consider the binary number $x = (x_m x_{m-1} \dots x_1 x_0)_{\text{two}} = \sum_{i=0}^{m} x_i 2^i$. What is the effect of multiplying x by 2^j? Since

$$x2^j = \sum_{i=0}^{m} x_i 2^{i+j} = x_m \dots x_1 x_0 \underbrace{00\dots0}_{j \text{ zeros}}{}_{\text{two}},$$

every bit in x is shifted to the left by j columns.

More generally, let a be any bit. Then

$$x(a2^j) = \sum_{i=0}^{m} (ax_i)2^{i+j} = (ax_m)\dots(ax_0)\underbrace{00\dots0}_{j \text{ zeros}}{}_{\text{two}}$$

The bit ax_i equals x_i if $a = 1$ and equals 0 if $a = 0$. Thus, the effect of multiplying the number $x = (x_m \dots x_0)_{\text{two}}$ by the bit y_j in the multiplicand $y = (y_n \dots y_j \dots y_0)_{\text{two}}$ is the same as multiplying each bit x_i by y_j and shifting the result to the left by j columns. Then add the partial products to get the desired product, as illustrated below.

EXAMPLE 4.13 Evaluate $1011_{\text{two}} \times 101_{\text{two}}$.

SOLUTION:
The various steps are displayed in Figures 4.10–4.13. It follows from Figure 4.13 that the resulting product is 110111_{two}.

Figure 4.10

```
    1 0 1 1
  ×   1 0 1
  -----------
    1 0 1 1   ← multiply 1011 by 1; no shifting.
```

Figure 4.11

```
    1 0 1 1
  ×   1 0 1
  -----------
    1 0 1 1
0 0 0 0       ← multiply 1011 by 0; shift by one column.
```

Figure 4.12

```
      1 0 1 1
    ×   1 0 1
    -----------
      1 0 1 1
    0 0 0 0
1 0 1 1       ← multiply 1011 by 1; shift by 2 columns.
```

Figure 4.13

$$
\begin{array}{cccccc}
 & & 1 & 0 & 1 & 1 \\
 & \times & & 1 & 0 & 1 \\
\hline
 & & 1 & 0 & 1 & 1 \\
 & 0 & 0 & 0 & 0 & \\
1 & 0 & 1 & 1 & & \\
\hline
1 & 1 & 0 & 1 & 1 & 1
\end{array}
\quad \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} \text{add the partial products.}
$$

■

The shifting method of multiplication leads to Algorithm 4.5 for multiplying two binary numbers.

```
Algorithm binary multiplication (x, y, p)
(* This algorithm computes the product p = (p_{m+n} p_{m+n-1} ... p_0)_two
   of the binary numbers x = (x_m x_{m-1} ... x_1 x_0)_two and
   y = (y_n y_{n-1} ... y_1 y_0)_two, using shifting. *)
 Begin (* algorithm *)
   for j = 0 to n do
     begin (* for *)
       multiply each bit x_i by y_j
       shift the resulting binary word to the left
       by j columns
       w_j ← resulting binary word
     endfor
   add the partial products w_j
   p ← resulting sum
 End (* algorithm *)
```

Algorithm 4.5

Binary Subtraction

We can subtract binary numbers without the bother of "borrows," using one's complement and addition. The **one's complement** x' of a binary number x is obtained by replacing each 0 in x with a 1 and vice versa. For example, the one's complement of 1011_{two} is 0100_{two} and that of 1001_{two} is 0110_{two}. The **two's complement** of x is $x' + 1$. For instance, the two's complement of 1011_{two} is $0100_{\text{two}} + 1 = 0101_{\text{two}}$.

The next example illustrates this new technique step-by-step before it is justified in a formal discussion.

EXAMPLE 4.14 Subtract 1011_{two} from 100001_{two}.

SOLUTION:
For convenience, we shall drop the base two.

Step 1 Find the one's complement of the subtrahend 1011. Since the minuend 100001 contains six bits, keep the same number of bits in the subtrahend by padding it with two 0's at the beginning. The one's complement of $1011 = 001011$ is 110100.

Step 2 Find the two's complement by adding 1 to the one's complement: $110100 + 1 = 110101$.

Step 3 Add the two's complement in step 2 to the minuend 100001:

$$\begin{array}{rcccccc} & 1 & 0 & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \textcircled{1} & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

delete ⟵↑

Step 4 Delete the leading carry 1. The resulting number $010110 = 10110$ is the desired answer.

Thus $100001_{\text{two}} - 1011_{\text{two}} = 10110_{\text{two}}$. (To check this, you may verify that $1011_{\text{two}} + 10110_{\text{two}} = 100001_{\text{two}}$.) ■

How can this technique work? To justify the algorithm illustrated, first notice that $x - y = x + (-y)$; that is, subtracting y from x is equivalent to adding the additive inverse $-y$ of y to x. This is the basic idea behind the binary subtraction algorithm.

Now how to find $-y$? First, assume that $\|x\| = \|y\| = n$. (If $\|y\| < \|x\|$, pad y with enough 0's at the beginning so the length of the resulting word is n.) Let y' denote the one's complement of y. Then $y + y'$ is an n-bit word w containing all 1's:

$$w = \underset{\substack{n-1 \quad n-2}}{\boxed{1}\boxed{1}\boxed{1}\boxed{\ldots}\boxed{1}\boxed{1}\boxed{1}}$$

For example, let $y = 10110$. Then $y' = 01001$, so $y + y' = 11111$.

The value of the n-bit word w is $2^n - 1$ (see Section 4.4). Thus $y + y' = w = 2^n - 1$, so $-y = y' + 1 - 2^n = y'' - 2^n$, where $y'' = y' + 1$ denotes the two's complement of y. Therefore, $x + (-y) = x + y'' - 2^n = (x + y'') - 2^n$. Thus, to subtract y from x, it suffices to add y'' to x and drop the leading carry 1. This explains why the above subtraction algorithm works.

The algorithm for the case $\|x\| < \|y\|$ is complicated, so we omit its discussion here.*

We close this section with an intriguing numeric puzzle that will test your mastery of both nondecimal addition and subtraction.

---

*For a discussion of negative binary numbers, see A. S. Tanerbaum, *Structured Computer Organization*, Prentice Hall, Englewood, NJ, 1976, pp. 420–423.

A Nondecimal Puzzle

Write down a three-digit number in base eight. Reverse its digits. Subtract the smaller number from the other (in base eight); save all leading zeros. Reverse its digits. Add the last two numbers. Is your answer 1067_{eight}? Now redo the puzzle in base 12; your answer should be $10AB_{\text{twelve}}$.

Exercises 4.3

Express each number in base 10.

1. 1101_{two} **2.** 11011_{two} **3.** 1776_{eight} **4.** 1976_{sixteen}

Express each decimal number as required.

5. $1076 = (\quad)_{\text{two}}$ **6.** $676 = (\quad)_{\text{eight}}$

7. $1776 = (\quad)_{\text{eight}}$ **8.** $2076 = (\quad)_{\text{sixteen}}$

The binary representation of an integer can conveniently be used to find its octal representation. Group the bits in threes from right to left and replace each group with the corresponding octal digit. For example,

$$243 = 11110011_{\text{two}} = 011\ \ 110\ \ 011_{\text{two}} = 363_{\text{eight}}$$

Using this short cut, rewrite each binary number as an octal integer.

9. 1101_{two} **10.** 11011_{two} **11.** 111010_{two} **12.** 10110101_{two}

The binary representation of an integer can also be used to find its hexadecimal representation. Group the bits in fours from right to left and then replace each group with the equivalent hexadecimal digit. For instance,

$$243 = 11110011_{\text{two}} = 1111\ \ 0011_{\text{two}} = \text{F3}_{\text{sixteen}}$$

Using this method express each binary number in base 16.

13. 11101_{two} **14.** 110111_{two} **15.** 1110101_{two} **16.** 10110101_{two}

The techniques explained in Exercises 9–12 are reversible; that is, the octal and hexadecimal representations of integers can be used to find their binary representations. For example,

$$345_{\text{eight}} = 011\ \ 100\ \ 101_{\text{two}} = 11100101_{\text{two}}$$

Using this technique, rewrite each number in base two.

17. 36_{sixteen} **18.** 237_{eight} **19.** 237_{sixteen} **20.** $\text{3AD}_{\text{sixteen}}$

In Exercises 21–28, perform the indicated operations.

21. $1111_{\text{two}} + 1011_{\text{two}}$ **22.** $1076_{\text{eight}} + 2076_{\text{eight}}$ **23.** $3076_{\text{sixteen}} + 5776_{\text{sixteen}}$ **24.** $101101_{\text{two}} - 10011_{\text{two}}$

25. $11000_{\text{two}} - 100_{\text{two}}$ **26.** $10111_{\text{two}} \times 1101_{\text{two}}$ **27.** $1024_{\text{eight}} \times 2776_{\text{eight}}$ **28.** $3ABC_{\text{sixteen}} \times 4CBA_{\text{sixteen}}$

29. Arrange the binary numbers 1011, 110, 11011, 10110, and 101010 in order of increasing magnitude.

30. Arrange the hexadecimal numbers 1076, 3056, 3CAB, 5ABC, and CACB in order of increasing magnitude.

31. What can you say about the ones bit in the binary representation of an even integer? An odd integer?

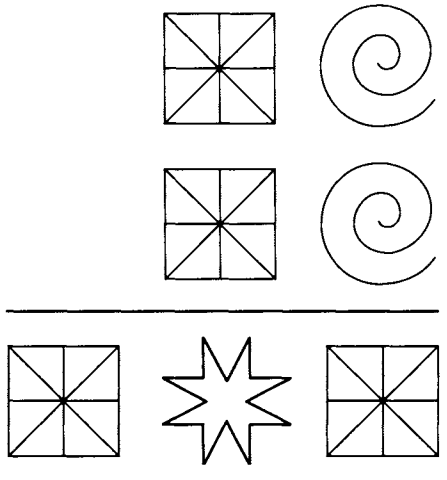
Find the value of the base b in each case.

32. $54_b = 64$ **33.** $1001_b = 9$ **34.** $1001_b = 126$ **35.** $144_b = 49$

36. Suppose a space investigative team to Venus sends back the picture of an addition problem scratched on a wall, as shown in Figure 4.14. The Venusian numeration system is a place value system, just like ours. The base of the system is the same as the number of fingers on a Venusian hand. Determine the base of the Venusian numeration system. (This puzzle is due to H. L. Nelson.**)

Figure 4.14

The sum in "Venusian" notation.



Define recursively each set S of binary words.

37. Set of binary words that represent even positive integers.

38. Set of binary words that represent odd positive integers.

**M. Gardner, "Mathematical Games," *Scientific American*, Vol. 219, Sept. 1968, pp. 218–230.

39. Set of binary words that represent positive integers with no leading zeros.

40. Set of palindromic binary words.

Polynomials can be evaluated efficiently using the technique of **nested multiplication**, called **Horner's method**. [This method is named after the English schoolmaster, William G. Horner (1786–1837), who published it in 1819.] For instance, the polynomial $f(x) = 4x^3 + 5x^2 + 6x + 7$ can be evaluated as $f(x) = ((4x + 5)x + 6)x + 7$. Using this method, express each integer as a decimal integer.

41. 245_{eight} **42.** 101101_{two} **43.** 1100101_{two} **44.** $43\text{BC}_{\text{sixteen}}$

***45.** Let x be a three-digit hexadecimal number with distinct digits. Reverse the digits. Subtract the smaller number from the other number (save all the digits in your answer). Reverse the digits in the difference. Add this number to x. Find the sum.

4.4 Mathematical Induction

The principle of mathematical induction[†] (PMI) is a frequently used proof technique in both mathematics and computer science, as will be seen shortly.

Many interesting results in mathematics hold true for all positive integers. For example, the following statements are true for every positive integer n, where x, y, and x_i are any positive real numbers:

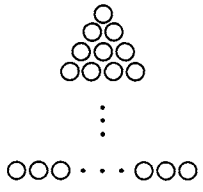
- $(x \cdot y)^n = x^n \cdot y^n$
- $\log(x_1 \ldots x_n) = \sum_{i=1}^{n} \log x_i$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=0}^{n-1} r^i = \frac{r^n - 1}{r-1} (r \neq 1)$

How do we prove that these results hold for every positive integer n? Obviously, it is impossible to substitute each positive integer for n and verify that the formula holds. The principle of induction can establish the validity of such formulas.

To begin with, suppose the orange cans in a collection can be arranged as in Figure 4.15. Row 1 contains one can, row 2 contains two cans, ..., row n contains n cans. Can you predict a formula for the total number of cans in the collection? See Example 4.15 for a formula.

[†]Although the Venetian scientist Francesco Maurocylus (1491–1575) applied it in proofs in a book he wrote in 1575, the term *mathematical induction* was coined by De Morgan.

Figure 4.15



The next result is the cornerstone of the principle of induction. Its proof, as we shall see shortly, follows by the well-ordering principle in Section 4.1.

THEOREM 4.9 Let S be a subset of $\mathbb{N}$ satisfying the following properties:

(1) $1 \in S$.
(2) If k is an arbitrary positive integer in S, then $k + 1 \in S$. Then $S = \mathbb{N}$.

PROOF (by contradiction):
Suppose $S \neq \mathbb{N}$. Let $S' = n \in N \mid n \notin S$. Since $S' \neq \emptyset$, by the well-ordering principle, S' contains a least element ℓ'. Then $\ell' > 1$ by condition 1. Since ℓ' is the least element in S', $\ell' - 1 \notin S'$; so $\ell' - 1 \in S$. Consequently, by condition 2, $(\ell' - 1) + 1 = \ell' \in S$. This contradiction establishes the theorem. ■

This theorem can be generalized as in Theorem 4.10. We leave its proof as an exercise.

THEOREM 4.10 Let n_0 be a fixed integer. Let S be a subset of $\mathbf{Z}$ satisfying the following conditions:

- $n_0 \in S$.
- If k is an arbitrary integer $\geq n_0$ such that $k \in S$, then $k + 1 \in S$.

Then $S \supseteq \{n \in \mathbf{Z} \mid n \geq n_0\}$. ■

Weak Version of Induction

Before we formalize the principle of induction, let's look at a trivial example. Consider an infinite number of dominoes arranged in a row (see Figure 4.16a). Suppose we knock down the first domino.

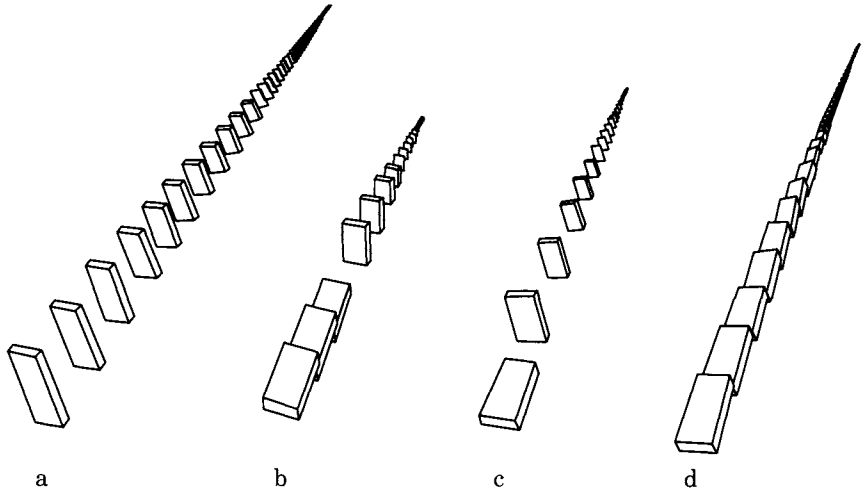
What happens to the rest of the dominoes? Do they all fall? Not necessarily; see Figures 4.16b and c.

So let's further assume the following: If the kth domino is knocked down, then the $(k + 1)$st domino also falls down. If we topple the first domino, what would happen to the rest? They all would fall; see Figure 4.16d.

This illustration can be expressed in symbols. Let $P(n)$ denote the predicate that the nth domino falls. (*Note*: UD = $\mathbb{N}$.) Assume the following propositions are true:

- $P(1)$.
- $P(k) \rightarrow P(k + 1)$ for every positive integer k.

Figure 4.16



Then P(n) is true for every positive integer n; that is, every domino would fall. This leads us to the **weak version** of the principle.

THEOREM 4.11 **(The Principle of Mathematical Induction)** Let P(n) be a predicate satisfying the following conditions, where n is an integer:

(1) P(n_0) is true for some integer n_0.
(2) If P(k) is true for an arbitrary integer $k \geq n_0$, then P($k + 1$) is also true.

Then P(n) is true for every integer $n \geq n_0$.

PROOF:
Let S denote the set of integers $\geq n_0$ for which P(n) is true. Since P(n_0) is true, $n_0 \in S$. By condition 2, whenever $k \in S$, $k+1 \in S$. Therefore, by Theorem 4.10, S consists of all integers $\geq n_0$. Consequently, P(n) is true for every integer $n \geq n_0$. This establishes the validity of the principle. ■

Condition 1 assumes the proposition P(n) is true when $n = n_0$. Look at condition 2: If P(n) is true for an arbitrary integer $k \geq n_0$, it is also true for $n = k + 1$. Then, by the repeated applications of condition 2 and the law of detachment, it follows that P($n_0 + 1$), P($n_0 + 2$), ... all hold true. In other words, P(n) holds for every $n \geq n_0$.

Proving a result by PMI involves two key steps:

1. **Basis step** Verify that P(n_0) is true.
2. **Induction step** Assume P(k) is true for an arbitrary integer $k \geq n_0$ (**inductive hypothesis**).
   Then verify that P($k + 1$) is also true.

A word of caution: A question frequently asked is, "Isn't this cyclic reasoning? Are you not assuming what you are asked to prove?" The confusion stems from misinterpreting step 2 for the conclusion. The induction step involves showing that the implication $P(k) \rightarrow P(k+1)$ is a tautology; that is, if $P(k)$ is true, then so is $P(k+1)$. The conclusion is "$P(n)$ is true for every $n \geq n_0$." So be careful.

A variety of interesting examples will show how useful this important proof technique is.

The next example gives a nice formula for computing the total number of cans in the collection in Figure 4.15.

EXAMPLE 4.15 Using PMI, prove that, for every positive integer n,

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

PROOF (by induction):

Let $P(n)$: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Basis step To verify that $P(1)$ is true (*Note*: Here $n_0 = 1$):

When $n = 1$, RHS $= \frac{1(1+1)}{2} = 1 = \sum_{i=1}^{1} i =$ LHS; so $P(1)$ is true.

Induction step Let k be an arbitrary positive integer. We would like to show that $P(k) \rightarrow P(k+1)$: Assume $P(k)$ is true; that is,

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \qquad \longleftarrow \text{inductive hypothesis}$$

To establish that $P(k) \rightarrow P(k+1)$ is true, that is,

$$\sum i = \frac{(k+1)(k+2)}{2}$$

we start with the LHS of this equation:

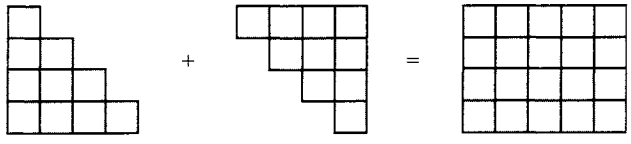
$$\begin{aligned} \text{LHS} &= \sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + (k+1) \qquad \left(\text{Note}: \sum_{i=1}^{k+1} x_i = \sum_{i=1}^{k} x_i + x_{k+1} \right) \\ &= \frac{k(k+1)}{2} + (k+1), \qquad \text{by the inductive hypothesis} \\ &= \frac{(k+1)(k+2)}{2} \\ &= \text{RHS} \end{aligned}$$

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

Therefore, by PMI, P(n) is true for every $n \geq 1$; that is, the formula holds for every positive integer n. ■

Figure 4.17 provides a geometric proof of this formula without words.

Figure 4.17



The next example, again an application of induction, employs a divisibility property, so we follow it in some detail.

EXAMPLE 4.16 Prove that $2n^3 + 3n^2 + n$ is divisible by 6 for every integer $n \geq 1$.

PROOF (by PMI):
Let P(n): $2n^3 + 3n^2 + n$ is divisible by 6.

Basis step When $n = 1$, $2n^3 + 3n^2 + n = 2(1) + 3(1) + 1 = 6$ is clearly divisible by 6. Therefore, P(1) is true.

Induction step Assume P(k) is true, that is, $2k^3 + 3k^2 + k$ is divisible by 6 for any $k \geq 1$. Then $2k^3 + 3k^2 + k = 6m$ for some integer m (inductive hypothesis). We must show that P($k+1$) is true; that is, $2(k+1)^3 + 3(k+1)^2 + (k+1)$ is divisible by 6. Notice that

$$\begin{aligned}
&2(k+1)^3 + 3(k+1)^2 + (k+1) \\
&\quad = 2(k^3 + 3k^2 + 3k + 1) + 3(k^2 + 2k + 1) + (k+1) \\
&\quad = (2k^3 + 3k^2 + k) + 6(k^2 + 2k + 1) \\
&\quad = 6m + 6(k^2 + 2k + 1) \qquad \text{by the inductive hypothesis} \\
&\quad = 6(m + k^2 + 2k + 1),
\end{aligned}$$

which is clearly divisible by 6. Thus P($k+1$) is true.

Thus, by induction, the given statement is true for every $n \geq 1$. ■

Notice that in the above examples, $n_0 = 1$, but it need not always be 1, as the next example shows.

Jacob I. Bernoulli *(1654–1705), a member of the most distinguished family of mathematicans (see the family tree in Section 9.1), was born in Basel, Switzerland. His grandfather, a pharmacist in Amsterdam, had become a Swiss through marriage, and his father was a town councilor and a magistrate.*

Bernoulli received his M.A. in philosophy in 1671 and a theological degree 5 years later. During this time, he studied mathematics and astronomy against his father's will. He spent the next 2 years tutoring in Geneva. In 1687 he became professor of mathematics at the University of Basel, remaining there until his death. His brother Johann succeeded him at Basel.

In May 1690 he used the term integral *in the calculus sense known today. Bernoulli's most famous work,* Ars Conjectandi, *was published posthumously in 1713. It contains significant contributions to probability theory, the theory of series, and gravitational theory.*

EXAMPLE 4.17 **(Bernoulli's Inequality)** Let x be any real number greater than -1. Prove that $(1+x)^n \geq 1+nx$ for every $n \geq 0$.

PROOF (by PMI):
Let x be any real number > -1. Let P(n): $(1+x)^n \geq 1+nx$. (*Note*: The induction is on the discrete variable n and not on the "continuous" variable x.)

Basis step To verify that P(0) is true: Notice that

$$\begin{aligned}(1+x)^0 &= 1\\ &\geq 1+0x\end{aligned}$$

So P(0) is true. (*Note*: Here $n_0 = 0$.)

Induction step Assume P(k) is true; that is, $(1+x)^k \geq 1+kx$ for an arbitrary integer $k \geq 0$. We need to show that P($k+1$) is true; that is, $(1+x)^{k+1} \geq 1+(k+1)x$.

By the inductive hypothesis, we have $(1+x)^k \geq 1+kx$. Then

$$\begin{aligned}(1+x)^{k+1} &= (1+x)(1+x)^k,\\ &\geq (1+x)(1+kx), && \text{by IH and since } 1+x > 0\\ &= 1+(k+1)x+kx^2\\ &\geq 1+(k+1)x, && \text{since } kx^2 \geq 0\end{aligned}$$

Therefore, P($k+1$) is also true.
Thus, by PMI, $(1+x)^n \geq 1+nx$ for every $n \geq 0$. ■

The next example inductively establishes Theorem 2.3 from Chapter 2.

EXAMPLE 4.18 A finite set A with n elements has exactly 2^n subsets.

PROOF (by PMI):

Basis step When $n = 0$, $A = \emptyset$, so A has exactly $1 = 2^0$ subset. Thus the result is true when $n = 0$.

Induction step Assume any finite set with k elements has 2^k subsets, where $k \geq 0$. Let A be a set with $k + 1$ elements. We would like to show that A has 2^{k+1} subsets.

To this end, let $x \in A$. Let $B = A - \{x\}$. Since $|B| = k$, B has 2^k subsets by the inductive hypothesis. Each of the subsets of B is a subset of A. Now add x to each of them. The resulting 2^k sets are also subsets of A. Since every subset of A either contains x or does not contain x, by the addition principle, A has $2^k + 2^k = 2^{k+1}$ subsets.

Thus, by the principle of induction, the result holds for every finite set. ■

Both the basis and the induction steps are essential in the principle of induction, as the next two examples illustrate.

EXAMPLE 4.19 Let $g(n)$ denote the maximum number of nonoverlapping regions formed inside a circle by joining n distinct points on it. Figures 4.18–4.22 show the cases $n = 1$, 2, 3, 4, and 5, where the various regions are numbered 1, 2, 3, etc. The results are summarized in Table 4.1.

Figure 4.18

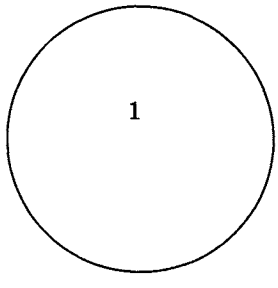


Figure 4.19

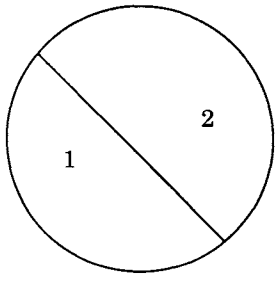


Figure 4.20

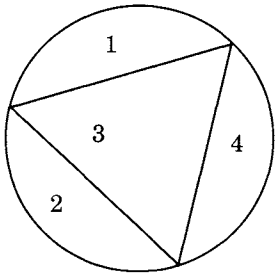


Figure 4.21

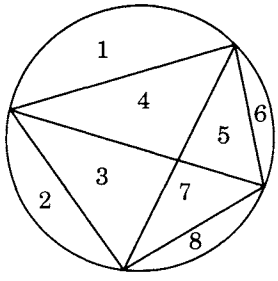


Figure 4.22

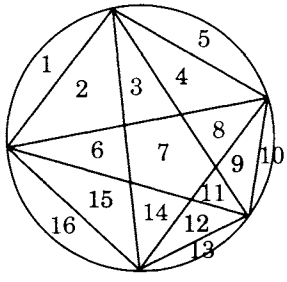
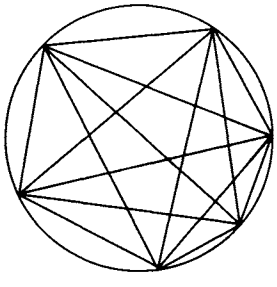


Figure 4.23



It appears from the table that $g(n) = 2^{n-1}$. Then $g(1) = 2^0 = 1$, which is true (basis step). Nonetheless, this does not guarantee that $g(n) = 2^{n-1}$ for every $n \geq 1$. If the formula were true, there would be $g(6) = 2^5 = 32$ nonoverlapping regions with six points. Unfortunately, there are only 31 such regions (see Figure 4.23) We shall derive the correct formula in Chapter 6.

Table 4.1

Number of points n	1	2	3	4	5	6
Maximum number of nonoverlapping regions $g(n)$	1	2	4	8	16	?

We can conclude that the truthfulness of the basis step and an apparent pattern do *not* ensure that P(n) is true for every n. ■

The following example shows that the validity of the induction step is necessary, but not sufficient, to guarantee that P(n) is true for all integers in the UD.

EXAMPLE 4.20 Consider the "formula" P(n) : $1 + 3 + 5 + \cdots + (2n - 1) = n^2 + 1$. Suppose P($k$) is true: $\sum_{i=1}^{k}(2i - 1) = k^2 + 1$. Then:

$$\begin{aligned}\sum_{i=1}^{k+1}(2i - 1) &= \sum_{i=1}^{k}(2i - 1) + (2k + 1)\\ &= (k^2 + 1) + (2k + 1)\\ &= (k + 1)^2 + 1\end{aligned}$$

So if P(k) is true, P($k + 1$) is true. Nevertheless, the formula does not hold for any positive integer n. Try P(1) (see Exercise 5). ■

Using induction, the next example "proves" that every person is of the same sex.

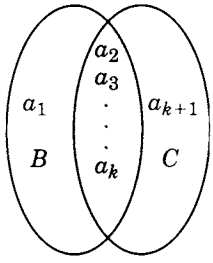
EXAMPLE 4.21 "Prove" that every person in a set of n people is of the same sex.

"PROOF":
Let P(n): Everyone in a set of n people is of the same sex. Clearly, P(1) is true. Let k be a positive integer such that P(k) is true; that is, everyone in a set of k people is of the same sex. To show that P($k + 1$) is true, consider a set $A = \{a_1, a_2, \ldots, a_{k+1}\}$ of $k+1$ people. Partition A into two overlapping sets, $B = \{a_1, a_2, \ldots, a_k\}$ and $C = \{a_2, \ldots, a_{k+1}\}$, as in Figure 4.24. Since $|B| = k = |C|$, by the inductive hypothesis, everyone in B is of the same sex and everyone in C is of the same sex. Since B and C overlap, everyone in $B \cup C$ must be of the same sex; that is, everyone in A is of the same sex.

Thus, by PMI, P(n) is true for every $n \geq 1$. This concludes the "proof." ■

Note: The assertion that everyone is of the same sex is clearly false. Can you find the flaw in the "proof"? See Exercise 46.

Figure 4.24



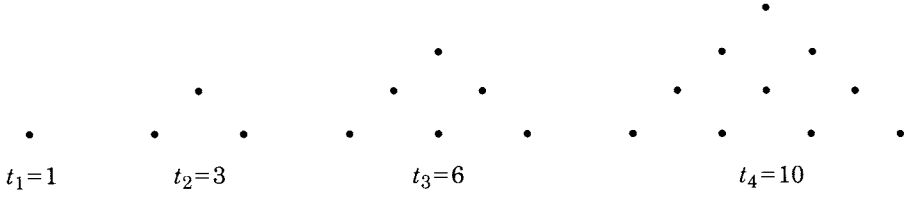
Before discussing the second version of the principle of induction, we will look at a few applications of the formula in Example 4.15. First a definition.

Polygonal Number

A **polygonal number** is a positive integer n that can be represented by n dots in a polygonal array in a systematic fashion. For example, the integers 1, 3, 6, 10, ... are **triangular numbers** since they can be represented by triangular arrays, as shown in Figure 4.25; the number of pins in a bowling alley and that of balls in the game of pool are triangular numbers. Let t_n denote the nth triangular number. Then

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

Figure 4.25



Triangular numbers manifest delightful properties. For example, $t_n + t_{n-1} = n^2$; Figures 4.26 and 4.27 provide a nonverbal, geometric proof of this result. See Exercises 47–50.

Figure 4.26

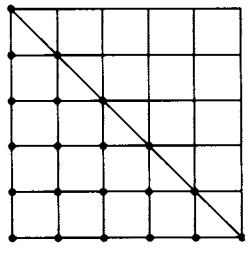
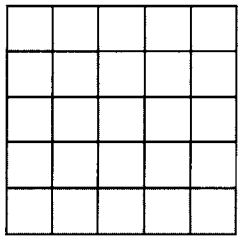


Figure 4.27

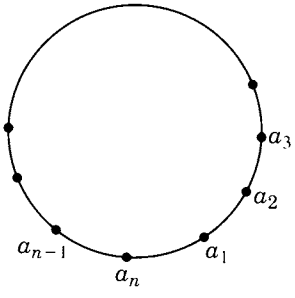


The next example is another application of the formula in Example 4.15 and the generalized pigeonhole principle.

EXAMPLE 4.22

Let $a_1, a_2, \ldots, a_n$ be the first n positive integers in some order. Suppose they are arranged around a circle (see Figure 4.28). Let k be any positive integer $\leq n$. Prove that there exists a set of k consecutive elements in the arrangement with a sum $\lfloor [kn(n+1) - 2]/2n \rfloor$, where $\lfloor x \rfloor$ denotes the floor of x.

Figure 4.28



PROOF:
Consider the following sums:

$$\begin{aligned} S_1 &= a_1 + a_2 + \cdots + a_k \\ S_2 &= a_2 + a_3 + \cdots + a_{k+1} \\ &\vdots \\ S_n &= a_n + a_1 + \cdots + a_{k-1} \end{aligned}$$

Each of the first n positive integers appears k times in this set of sums. Then

$$\sum_{i=1}^{n} S_i = k\left(\sum_{i=1}^{n} a_i\right) = k\left(\sum_{i=1}^{n} i\right) = \frac{kn(n+1)}{2}, \text{ by Example 4.15}$$

Consider $kn(n+1)/2$ pigeons. We would like to distribute them among n pigeonholes, called $S_1, S_2, \ldots, S_n$. By the generalized pigeonhole principle, at least one of the pigeonholes S_i must contain more than $\lfloor kn(n+1)/2n - 1/n \rfloor = \lfloor \lfloor kn(n+1-2) \rfloor / 2n \rfloor$ pigeons. In other words, $s_i > \lfloor kn(n+1) - 2/2n \rfloor$, as desired. ■

In particular, if numbers 1 through 10 are randomly placed around a circle, at least three consecutive integers in the arrangement must have a sum exceeding $\lfloor \lfloor 3 \cdot 10 \cdot 11 - 2 \rfloor / (2 \cdot 10) \rfloor = 16$.

We now discuss the strong version of the principle of induction.

Strong Version of Induction

Sometimes the truth of $P(k)$ might not be enough to establish that of $P(k+1)$. In other words, the truthfulness of $P(k+1)$ may require more than that of $P(k)$. In such cases, we have to assume a stronger inductive hypothesis that $P(n_0)$, $P(n_0+1)$, ... , $P(k)$ are all true; then verify that $P(k+1)$ is also true. This **strong version,** which can be proved using the weak version (see Exercise 57), is stated as follows.

THEOREM 4.12 **(The Second Principle of Mathematical Induction)** Let $P(n)$ be a predicate satisfying the following conditions, where n is any integer:

- $P(n_0)$ is true for some integer n_0.
- If k is an arbitrary integer $\geq n_0$ such that $P(n_0) \wedge P(n_0+1) \wedge \cdots \wedge P(k)$ is true, then $P(k+1)$ is also true. Then $P(n)$ is true for every $n \geq n_0$.

The next theorem illustrates this proof technique. ■

THEOREM 4.13 **(The Fundamental Theorem of Arithmetic)** Every positive integer $n \geq 2$ either is a prime or can be written as a product of primes.

PROOF (by strong induction):
Let $P(n)$ denote the given predicate.

Basis step Choose $n_0 = 2$. Since 2 is itself a prime, $P(2)$ is true.

Inductive step Let k be a positive integer ≥ 2 such that $P(2)$, $P(3)$, ... , $P(k)$ are true; that is, assume that integers 2 through k are primes or can be written as products of primes. We would like to show that $P(k+1)$ is also true; that is, integer $k+1$ is a prime or can be expressed as a product of primes.

If $k+1$ is itself a prime, then we are done. If $k+1$ is not a prime, it must be the product of two positive integers x and y, where $1 < x, y < k+1$. By the inductive hypothesis, both x and y are primes or products of primes. Therefore, $k+1 = x \times y$ is also a product of two or more prime numbers. In other words, $P(k+1)$ also holds:

Thus, by the strong version of induction, $P(n)$ is true for every $n \geq 2$. ■

We now present an interesting application of the fundamental theorem of arithmetic, which is the cornerstone of number theory, and the floor function.

EXAMPLE 4.23 Find the number of trailing zeros in 123!

SOLUTION:
By the fundamental theorem of arithmetic, 123! can be factored as $2^a 5^b c$, where c denotes the product of primes other than 2 and 5. Clearly a > b. Each trailing zero in 123! corresponds to a factor of 10 and vice versa.

$$\therefore \text{ Number of trailing zeros} = \begin{pmatrix}\text{Number of products of the form} \\ 2 \cdot 5 \text{ in the prime factorization}\end{pmatrix}$$
$$= \text{minimum of } a \text{ and } b$$
$$= b, \text{ since } a > b$$

We proceed to find b:

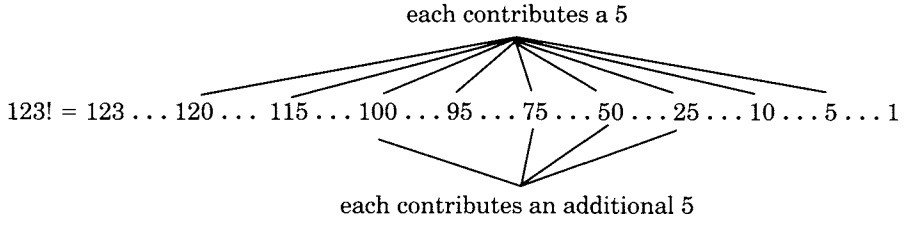
Number of positive integers ≤ 123 and divisible by $5 = \lfloor 123/5 \rfloor = 24$

Each of them contributes a 5 to the prime factorization of 123!

Number of positive integers ≤ 123 and divisible by $25 = \lfloor 123/25 \rfloor = 4$

(See Figure 4.29.) Each of them contributes an additional 5 to the prime factorization. Since no higher power of 5 contributes a 5 in the prime factorization of 123!, the total number of 5's in the prime factorization equals $24 + 4 = 28$. Thus the total number of trailing zeros in 123! is 28.

Figure 4.29



■

The next example is another interesting application of the floor function. It employs the following facts from number theory:

- *Every positive integer that is not a square has an even number of positive factors.* For example, 18 has six positive factors: 1, 2, 3, 6, 9, 18; 21 has four: 1, 3, 7, 21; 19 has two: 1, 19.
- *Every perfect square has an odd number of positive factors.* For example, 25 has three positive factors, namely, 1, 5, and 25; 64 has seven: 1, 2, 4, 8, 16, 32, and 64.

- *There are* $\lfloor \sqrt{n} \rfloor$ *perfect squares* $\leq n$.
- For example, there are $\lfloor \sqrt{27} \rfloor = 5$ perfect squares not exceeding $27 : 1, 4, 9, 16, 25$; there are $\lfloor \sqrt{68} \rfloor = 8$ perfect squares $< 68 : 1, 4, 9, 16, 25, 36, 49, 64$.

EXAMPLE 4.24 There are 1000 rooms in a hotel and every room is occupied by a guest. The first guest opens the door to every room. The nth guest closes every nth door if it is open and opens it otherwise, where $2 \leq n \leq 1000$. How many doors will be open at the end?*

SOLUTION:
Before applying these results to solve the puzzle, let us study a mini-version with 10 tenants and 10 apartments. The first tenant opens all 10 doors; the second tenant closes the 2nd, 4th, 6th, 8th, and 10th doors; the third closes the 3rd door, opens the 6th door, and closes the 9th door; the fourth tenant opens the 4th and 8th doors. Continuing like this, the 10th tenant closes the 10th door. These data are summarized in Table 4.2, where O indicates the door is open and C indicates the door is closed.

Table 4.2

	Door									
Tenant	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	O	O	O	O	O	O	O	O	O	O
2	.	C	.	C	.	C	.	C	.	C
3	.	.	C	.	.	O	.	.	C	.
4	.	.	.	O	.	.	.	O	.	.
5	.	.	.	.	C	.	.	.	.	O
6	.	.	.	.	.	C	.	.	.	.
7	.	.	.	.	.	.	C	.	.	.
8	.	.	.	.	.	.	.	C	.	.
9	.	.	.	.	.	.	.	.	O	.
10	.	.	.	.	.	.	.	.	.	C

It follows from the table that doors 1, 4, and 9 remain open at the end, so the number of such doors is three. (Notice that $3 = \lfloor \sqrt{10} \rfloor$; so can you predict the answer to the given problem? Construct tables like Table 4.2 for 13 tenants and 13 apartments, 18 tenants and 18 apartments, and 25 tenants and 25 apartments, and look for a pattern.)

Let us now return to the original problem. The first tenant opens all doors. Consider the kth tenant, where $2 \leq k \leq 1000$.

Case 1 Let n be a perfect square, where $n^2 \leq 1000$. Since n has an odd number of positive factors, the last person to touch the door will open it. Thus every nth door will remain open if n is a perfect square. The number

*Based on M. vos Savant, *Ask Marilyn*, St. Martin Press, New York, 1992, p. 228.

of such doors equals the number of perfect squares ≤ 1000, namely, $\lfloor\sqrt{1000}\rfloor = 31$.

Case 2 Suppose n is not a perfect square, where $n^2 \leq 1000$. Since n has an even number of positive factors, the last person to touch the door will close it. In other words, every nth door will remain closed if n is not a perfect square.

Thus, by the addition principle, $31 + 0 = 31$ doors will remain open. They are doors numbered $1, 4, 9, 16, 25, \ldots, 900$, and 961. ■

More generally, suppose there are m tenants and m apartments, and the first tenant opens all doors. The jth tenant closes every jth door if it is open, and opens it otherwise, where $2 \leq j \leq m$. How many doors will remain open at the end?

Exercises 4.4

1. Compute the 36th triangular number. (It is the so-called *beastly number*.)
2. Prove that the sum of two consecutive triangular numbers is a perfect square.

(**Twelve Days of Christmas**) Suppose you sent your love 1 gift on the first day of Christmas, $1 + 2$ gifts on the second day, $1 + 2 + 3$ gifts on the third day and so on.

3. How many gifts did you send on the 12th day of Christmas?
4. How many gifts did your love receive in the 12 days of Christmas?

Using PMI, prove each for every integer $n \geq 1$.

5. $\sum_{i=1}^{n}(2i - 1) = n^2$
6. $\sum_{i=1}^{n} i^2 = \dfrac{(n+1)(2n+1)}{6}$
7. $\sum_{i=1}^{n} i^3 = \left[\dfrac{n(n+1)}{2}\right]^2$
8. $\sum_{i=1}^{n} ar^{i-1} = \dfrac{a(r^n - 1)}{r - 1}$ $(r \neq 1)$
9. $n^2 + n$ is divisible by 2.
10. $n^4 + 2n^3 + n^2$ is divisible by 4.
11. The number of lines formed by joining n (≥ 2) distinct points in a plane, no three of which being collinear, is $n(n - 1)/2$.
12. The number of diagonals of a convex n-gon* is $n(n - 1)/2 \geq 3$.
13. Let a be a positive integer and p a prime number such that $p \mid a^n$. Then $p \mid a$, where $n \geq 1$.
    (*Hint*: Use Exercise 37 in Section 4.2.)

*An n-gon is a polygon with n sides. An n-gon such that the line segment joining any two points inside it lies within it is a **convex polygon.**

14. Prove that $1 + 2 + \cdots + n = n(n + 1)/2$ by considering the sum in the reverse order.* (Do not use induction.)

Evaluate each sum.

15. $\sum_{k=1}^{30} (3k^2 - 1)$ **16.** $\sum_{k=1}^{50} (k^3 + 2)$ **17.** $\sum_{i=1}^{n} \lfloor i/2 \rfloor$ **18.** $\sum_{i=1}^{n} \lceil i/2 \rceil$

Find the value of x resulting from executing each algorithm fragment.

19.
```
x ← 0
for i = 1 to n do
  x ← x + (2i − 1)
```

20.
```
x ← 0
for i = 1 to n do
  x ← x + i(i + 1)
```

21.
```
x ← 0
for i = 1 to n do
  for j = 1 to i do
  x ← x + 1
```

Evaluate each sum and product.

22. $\sum_{i=1}^{n} \sum_{j=1}^{i} i$ **23.** $\sum_{i=1}^{n} \sum_{j=1}^{i} j$ **24.** $\sum_{i=1}^{n} \sum_{j=1}^{i} j^2$ **25.** $\sum_{i=1}^{n} \sum_{j=1}^{i} (2j - 1)$

26. $\prod_{i=1}^{n} 2^{2i}$ **27.** $\prod_{i=1}^{n} i^2$ **28.** $\prod_{i=1}^{n} \prod_{j=1}^{n} i^j$ **29.** $\prod_{i=1}^{n} \prod_{j=1}^{n} 2^{i+j}$

30. A **magic square** of order n is a square arrangement of the positive integers 1 through n^2 such that the sum of the integers along each row, column, and diagonal is a constant k, called the **magic constant**. Figure 4.30 shows two magic squares, one of order 3 and the other of order 4. Prove that the magic constant of a magic square of order n is $n(n^2 + 1)/2$.

Figure 4.30

8	1	6
3	5	7
4	9	2

$k = 15$

1	14	15	4
12	7	6	9
8	11	10	5
13	2	3	16

$k = 34$

*An interesting anecdote is told about Karl Frederich Gauss (1777–1855), one of the great mathematicians. When he was a child, his teacher asked his pupils to compute the sum of the first 100 positive integers. According to the story, the teacher did so to get some time to grade his papers. To the teacher's dismay, Gauss found the answer in a few moments by pairing the numbers from both ends:

$$1 + 2 + 3 + \cdots + 50 + 51 + \cdots + 98 + 99 + 100$$

The sum of each pair is 101 and there are 50 pairs. So the total sum is $50 \cdot 101 = 5050$.

Let p, q, and r be prime numbers, and i, j, and k whole numbers. Find the sum of the positive divisors of each.

31. p^i **32.** $p^i q^j$ **33.** $p^i q^j r^k$

34. Let p be a prime and $n \in \mathbb{N}$. Prove that p^n is not a perfect number. (*Hint*: Prove by contradiction.)

Find the number of times the statement x ← x + 1 is executed by each loop.

35.
```
for i = 1 to n do
  for j = 1 to i do
    x ← x + 1
```

36.
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to i do
      x ← x + 1
```

37.
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to j do
      x ← x + 1
```

38.
```
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to i do
      for l = 1 to i do
        x ← x + 1
```

According to legend, King Shirham of India was so pleased with the invention of chess that he offered to reward its inventor Sissa Ben Dahir with anything he wished. His request was a seemingly modest one: one grain of wheat on the first square of a chessboard, two on the second, four on the third, and so on. The king was delighted with this simple request, but soon realized he could not fulfill it. The last square alone would take 2^{63} = 9,223,372,036,854,775,808 grains of wheat. Find each for an $n \times n$ chessboard.

39. The number of grains on the last square.

40. The total number of grains on the chessboard.

41. Let a_n denote the number of times the statement $x \leftarrow x + 1$ is executed in the following loop:

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

Show that

$$a_n = \begin{cases} \dfrac{n^2}{4} & \text{if } n \text{ is even} \\ \dfrac{n^2 - 1}{4} & \text{if } n \text{ is odd} \end{cases}$$

Find the number of trailing zeros in the decimal value of each.

42. 100! **43.** 378! **44.** 500! **45.** 1000!

46. Find the flaw in the "proof" in Example 4.21.

Prove each, where t_n denotes the nth triangular number and $n \geq 2$.

47. $8t_n + 1 = (2n + 1)^2$

48. $8t_{n-1} + 4n = (2n)^2$

49. $t_{n-1}^2 + t_n^2 = t_{n^2}$

50. $\sum_{i=1}^{n} t_i = \dfrac{n(n+1)(n+2)}{6}$

Let $A, A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n$ be any sets, and $p_1, p_2, \ldots, p_n, q, q_1, q_2, \ldots, q_n$ be any propositions. Using induction prove each.

***51.** $A \cup \left(\bigcap_{i=1}^{n} B_i \right) = \bigcap_{i=1}^{n} (A \cup B_i)$

***52.** $A \bigcap_{i=1}^{n} (\cup B_i) = \bigcup_{i=1}^{n} (A \cap B_i)$

***53.** $\sim (p_1 \wedge p_2 \wedge \cdots \wedge p_n) \equiv (\sim p_1) \vee (\sim p_2) \vee \cdots \vee (\sim p_n)$

***54.** $\sim (p_1 \vee p_2 \vee \cdots \vee p_n) \equiv (\sim p_1) \wedge (\sim p_2) \wedge \cdots \wedge (\sim p_n)$

***55.** Prove that any postage of n (≥ 2) cents can be made using two- and three-cent stamps. (*Hint*: Use the division algorithm and induction.)

***56.** Let a and b be any two positive integers with $a \geq b$. Using the sequence of equations in the euclidean algorithm prove that $\gcd\{a, b\} = \gcd\{r_{n-1}, r_n\}$, $n \geq 1$.

***57.** Prove the strong version of mathematical induction, using the weak version.

***58.** Prove the weak version of induction, using the well-ordering principle.

****59.** Let S_n denote the sum of the elements in the nth set of the sequence of sets of squares $\{1\}, \{4, 9\}, \{16, 25, 36\}, \ldots$. Find a formula for S_n. (J. M. Howell, 1989)

****60.** Redo Exercise 59 using the sequence of triangular numbers $\{1\}$, $\{3, 6\}$, $\{10, 15, 21\}, \ldots$. (J. M. Howell, 1988)

4.5 Algorithm Correctness

Suppose we wrote an algorithm to solve a problem and translated the algorithm into a computer program. Since it is impossible to test the program for all sets of input values, we rely on a mathematical proof to ensure that the program will always yield the correct output. The principle of induction can certify the correctness of algorithms.

Correct Program

A **correct** program yields the correct result for all legal input values, assuming the program contains no compilation and execution errors.

Proving the correctness of a program, especially a complex one, is not at all an easy task. It consists of two steps:

(1) Proving that the program will always terminate; and
(2) proving that it will always produce the correct result. The second step constitutes the **partial correctness** of the program.

Loop Invariant

First, we will establish the partial correctness of simple **while** loops. Let n denote the number of iterations of a **while** loop. Assume a predicate P(n). A relationship among the variables holds true before the loop is executed and after each iteration of the loop, no matter how large n is. As the algorithm execution progresses, the values of the variables in the loop may vary, but the relationship remains unaffected. Such a predicate is a **loop invariant.**

To prove that P(n) is a loop invariant, we apply PMI, as the next two examples demonstrate.

EXAMPLE 4.25 Algorithm 4.6 computes the product of two positive integers x and y. Notice that the values of the variables x and y are not affected by the loop in lines 3–7. But the values of i and *answer* do get changed during each iteration of the loop.

```
Algorithm multiplication(x,y)
(* This algorithm computes the product of the positive integers x and y,
   and prints the answer. *)
0.  Begin (* algorithm *)
1.    answer ← 0 (* initialize answer *)
2.    i ← 0    (* counter *)
3.    while i < x do
4.    begin (* while *)
5.      answer ← answer + y
6.      i ← i + 1
7.    endwhile
8.  End (* algorithm *)
```

Algorithm 4.6

Let a_n and i_n denote the values of *answer* and i at the end of n iterations. Let P(n): $a_n = i_n \cdot y$. We shall prove that the predicate P(n) is a loop invariant.

PROOF (by PMI):
Let P(n): $a_n = i_n \cdot y$, $n \geq 0$.

Basis step The value $n = 0$ means zero iterations; it corresponds to the situation before the loop is entered. When $n = 0, a_0 = 0$ and $i_0 = 0$. Therefore, $a_0 = i_0 \cdot y$; so, P(0) is true.

Induction step Assume P(k) is true; that is, $a_k = i_k \cdot y$ after k iterations. Then $a_{k+1} = a_k + y$ and $i_{k+1} = i_k + 1$, by lines 5 and 6. Thus:

$$\begin{aligned} a_{k+1} &= i_k \cdot y + y, \text{ by the inductive hypothesis} \\ &= (i_k + 1)y \\ &= i_{k+1} \cdot y \end{aligned}$$

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

Thus, by PMI, P(n) is true for every $n \geq 0$; that is, P(n) is a loop invariant. ■

How is the property that P(n) is a loop invariant useful? Since $a_n = i_n \cdot y$ after n iterations, it must be true even when we exit the loop. The loop is terminated when $i_n = x$. Then answer $= a_n = x \cdot y$, as expected. Since P(n) is a loop invariant, the algorithm does indeed work correctly.

What exactly is the iteration method? Suppose we would like to compute the value $f(n)$ of a function f at an integer $n \geq n_0$. In the **iteration method**, we use $f(n_0)$ to compute $f(n_0 + 1)$, then use the successive values $f(n_0 + 2)$, $f(n_0 + 3)$,... to evaluate $f(n)$. For instance, to evaluate $n!$ by iteration, we successively evaluate 0!, 1!, 2!,..., $(n - 1)!$ and then evaluate $n!$.

EXAMPLE 4.26 Algorithm 4.7 is an iterative algorithm for computing $n!$, where $n \geq 0$. Let fact(n) be the value of *factorial* at the end of n iterations of the loop. Prove that *P(n): fact(n) = n!* is a loop invariant.

```
Algorithm factorial (n)
(* This algorithm computes and prints the value of
   n! for every n ≥ 0. *)
0.  Begin (* algorithm *)
1.    factorial ← 1          (* initialize *)
2.    i ← 1                  (* counter *)
3.    while i < n do
4.    begin (* while *)
5.      i ← i + 1
6.      factorial ← factorial * i
7.    endwhile
9.  End (* algorithm *)
```

Algorithm 4.7

PROOF (by PMI):
Let P(n): fact(n) = $n!$, $n \geq 0$.

Basis step When $n = 0$, fact(0) = 1 = 1! by line 1; so P(0) is true.

Induction step Assume P(k) is true: fact(k) = k!. Then:

$$\begin{aligned}\text{fact}(k+1) &= \text{fact}(k) \cdot (k+1), \text{ by line 6} \\ &= k! \cdot (k+1), \text{ by the inductive hypothesis} \\ &= (k+1)!\end{aligned}$$

Therefore, P(k + 1) is true.

Thus, by induction, P(n) holds true for every $n \geq 0$; that is, P(n) is a loop invariant and hence the algorithm correctly computes the value of n!, for every $n \geq 0$. ■

Searching and Sorting Algorithms

The remainder of this section establishes the partial correctness of a few standard searching and sorting algorithms. We begin with two searching algorithms, linear and binary.

Linear Search Algorithm

Let $X = [x_1, x_2, \ldots, x_n]$ be an unordered list (also known as a one-dimensional array or simply an array) of n distinct items. We would like to search the list for a specific item, called *key*. If *key* exists in the list, the algorithm should return the location of *key*.

We search the list from right to left for convenience. Compare x_n and *key*. If $x_n = key$, *key* occurs and location = n. Otherwise, compare x_{n-1} and *key*. If they are equal, we are done. Otherwise, continue the search until it is successful or the list is empty. This algorithm is the **linear search algorithm**.

For example, let X = [Dallas, Boston, Nashville, Albany, Portland] and *key* = Albany. Then *key* occurs in the list at location 4.

In general, we cannot assume *key* occurs in the list. To make the search process always successful, we store *key* in location 0: $x_0 \leftarrow key$. So if the search routine returns the value zero for location, it implies *key* does not occur in the list.

An iterative version of the linear search algorithm is given in Algorithm 4.8.

```
Algorithm linear search (X,n,key,location)
(* This algorithm searches a list by the linear search method for
   a key and returns its location in the list. To make the search
   always successful, we store key in x0. If the algorithm returns
   the value 0 for location, key does not occur in the list. *)
0.  Begin (* algorithm *)
1.    x0 ← key
2.    i ← n
3.    while xi ≠ key do
4.      i ← i - 1
```

```
5.    location ← i
6.  End (* algorithm *)
```

Algorithm 4.8

EXAMPLE 4.27 Prove that the linear search algorithm in Algorithm 4.8 works correctly for every $n \geq 0$.

PROOF (by PMI):
Let P(n): The algorithm returns the correct location for every list of size $n \geq 0$.

Basis step When $n = 0$, the **while** loop is skipped. The algorithm returns the value 0 in location by line 5, which is correct. So P(0) is true.

Induction step Assume P(k) is true for an arbitrary integer $k \geq 0$; that is, the algorithm works when the list contains k items.

To show that P($k + 1$) is true, consider a list X with $k + 1$ elements.

Case 1 If $x_{k+1} = key$ in line 3, the **while** loop will not be entered and the algorithm returns the correct value $k + 1$ for location in line 5.

Case 2 If $x_{k+1} \neq key$, $i = k$ at the end of the first iteration. This restricts us to a sublist with k elements. By the inductive hypothesis, the algorithm works correctly for such a list.

In both cases, P($k + 1$) holds. Thus, by induction, P(n) is true for $n \geq 0$. In other words, the algorithm returns the correct location for every list with $n \geq 0$ elements.

Binary Search Algorithm

The **binary search algorithm** searches for a given *key* if the list X is ordered. The technique employed is **divide and conquer**. First compute the *middle* (*mid*) of the list, where $mid = \lfloor (1 + n)/2 \rfloor$. The middle item is x_{mid}.

Now partition the list into three disjoint sublists: $[x_1, \ldots, x_{\text{mid}-1}]$, $[x_{\text{mid}}]$, and $[x_{\text{mid}+1}, \ldots, x_n]$. If $x_{\text{mid}} = key$, the search is successful and $location = mid$. If they are not equal, we search only the lower half or the upper half of the list. If $key < x_{\text{mid}}$, search the sublist $[x_1, \ldots, x_{\text{mid}-1}]$; otherwise, search the sublist $[x_{\text{mid}+1}, \ldots, x_n]$. Continue like this until the search is successful or the sublist is empty.

EXAMPLE 4.28 Use the binary search algorithm to search the list
$X = [3, 5, 8, 13, 21, 34, 55, 89]$ for $key = 5$.

SOLUTION:
Let x_i denote the ith element of the list X, where $1 \leq i \leq n$ and $n = 8$.

Step 1 Compute *mid* for the list X:

$$mid = \lfloor (1+n)/2 \rfloor = \lfloor (1+8)/2 \rfloor = 4.$$

Therefore, the middle term is $x_{mid} = 13$.

Step 2 Compare x_{mid} and *key*:

Since $x_4 \neq 5$, *key*, if it occurs, must exist in the lower sublist $[x_1, x_2, x_3] = [3, 5, 8]$ or in the upper sublist $[x_5, x_6, x_7, x_8] = [21, 34, 55, 89]$. Since $key < x_4$, search the first sublist and continue steps 1 and 2 until either *key* is located or the sublist becomes empty.

Step 3 Compute *mid* for the list $[x_1, x_2, x_3]$:

$$mid = \lfloor (1+3)/2 \rfloor = 2$$

So $x_{mid} = x_2 = 5$.

Step 4 Compare x_{mid} and *key*:

Since $x_{mid} = key$, the search is successful. *Key* occurs at location 2 and we are done. (As an exercise, use the algorithm to search the list X with $key = 23$.) ■

The steps in this example can be translated into an algorithm. See Algorithm 4.9.

```
Algorithm binary search(X,1,n,key,mid)
(* This algorithm searches an ordered list X of n elements for a special
   item (key). It returns the location of key if the search is
   successful and zero otherwise. The variable mid returns such a value.
   The variables low and high denote the lower and upper indices of the
   list being searched. *)
 0. Begin (* algorithm *)
 1. low ← 1
 2. high ← n
 3. while low ≤ high do      (* list is nonempty *)
 4. begin (* while *)
 5.   mid ← ⌊(low + high)/2⌋
 6.   if key = x_mid then     (* key exists in the list*)
 7.     exit the loop
 8.   else if key < x_mid then      (* search lower half*)
 9.         high ← mid - 1
10.       else  (* search the upper half *)
11.         low ← mid + 1
12. endwhile
13.   if low > high then      (* search is unsuccessful *)
14.     mid ← 0
15. End (* algorithm *)
```

Algorithm 4.9

The next example establishes the partial correctness of this algorithm using strong induction.

EXAMPLE 4.29 Prove that the binary search algorithm (Algorithm 4.9) works correctly for every ordered list of size $n \geq 0$.

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

Basis step When $n = 0$, $low = 1$ and $high = 0$. Since $low \leq high$ is false in line 3, the **while** loop is not executed. So the algorithm returns the correct value 0 from line 14, as expected, and P(0) is true.

Induction step Assume P(i) holds for every $i \leq k$, where $k \geq 0$; that is, the algorithm returns the correct value for any list of size $i \leq k$.

To show that P($k + 1$) is true, consider an ordered list X of size $k + 1$. Since $high = k + 1 \geq 1 = low$, the loop is entered and the middle index is computed in line 5.

Case 1 If $key = x_{mid}$, we exit the loop (line 7) and the value of *mid* is returned, so the algorithm works.

Case 2 If $key < x_{mid}$, search the sublist $x_1, \ldots, x_{mid-1}$; otherwise, search the sublist $x_{mid+1}, \ldots, x_n$. In both cases, the sublists contain fewer than $k + 1$ elements, so the algorithm works in either case by the inductive hypothesis.

Thus P($k + 1$) is true. So, by PMI, P(n) is true for $n \geq 0$; that is, the algorithm works correctly for every ordered list of zero or more items. ■

Next we present two standard sorting algorithms and prove their correctness.

Sorting Algorithms

Suppose we are given a list of n items and would like to sort them in "ascending order." Several methods are available. Two algorithms that can do the job are bubble sort and selection sort.

Bubble Sort

Bubble sort is a simple, elegant algorithm for sorting a list of n items. It "bubbles up" smaller items to the top and pushes larger items to the bottom: Compare consecutive elements, beginning with the first pair. Swap them if they are out of order. Compare the next pair and swap them if necessary. Continue like this to the end of the list. This ends the first pass. Now place the largest element at the end of the list. Repeat these steps with all but the largest element until the resulting sublist consists of one element. The list is now ordered.

The following example demonstrates this method.

EXAMPLE 4.30 Using bubble sort, sort the list $X = [34, 13, 21, 3, 89]$.

SOLUTION:

Let x_i denote the ith element in the list, where $1 \leq i \leq 5$. The given list is

	1	2	3	4	5
X	34	13	21	3	89

Step 1 Compare x_1 and x_2. Since $x_1 > x_2$, swap them. This yields the list

	1	2	3	4	5
X	13	34	21	3	89

Now compare x_2 and x_3. Since $x_2 > x_3$, interchange x_2 and x_3. This produces the list

	1	2	3	4	5
X	13	21	34	3	89

Since $x_3 > x_4$, switch them, yielding the list

	1	2	3	4	5
X	13	21	3	34	89

Compare x_4 and x_5. Since $x_4 < x_5$, they are in the correct order and no interchanging is needed. This completes the first pass. At the end of the first pass, the largest element in the list is placed in proper position:

	1	2	3	4	5
X	13	21	3	34	89

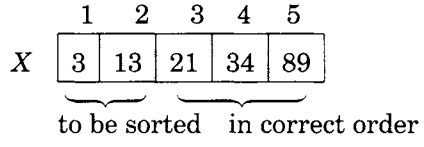
to be sorted ↑ in correct position

Step 2 In the second pass, compare the elements x_1 through x_4 and swap them if necessary. This results in the two largest elements being placed correctly:

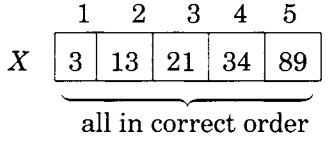
	1	2	3	4	5
X	13	3	21	34	89

to be sorted correctly sorted

Step 3 The third pass involves the elements x_1 through x_3. At the end of this pass, the three largest elements are correctly placed:



Step 4 At the end of the fourth pass the list is completely sorted:



Two important observations:

- At the end of the ith pass, the i largest elements are correctly placed at the end of the list, where $1 \leq i \leq n$. So the $(i + 1)$st pass involves the elements x_1 through x_{n-i}.
- Bubble sort takes $n - 1$ passes to sort a list of n items, even if the list becomes ordered at the end of the ith pass, where $i < n - 1$. Once the list is sorted, it makes no sense to go through the remaining passes, so the additional passes can be avoided with a boolean variable.

The various steps in Example 4.30 can be developed into an algorithm for bubble sort, as presented in Algorithm 4.10.

```
Algorithm bubble sort(X,n)
(* This algorithm sorts a list X of n elements
   using the bubble algorithm. *)
0.  Begin (* algorithm *)
1.    for i=1 to n - 1 do
2.      for j=1 to n - i do
3.        if xj > xj+1 then
4.          swap xj and xj+1
5.  End (* algorithm *)
```

Algorithm 4.10

EXAMPLE 4.31 Establish the correctness of the bubble sort algorithm.

PROOF (by PMI):
Let P(n): The algorithm sorts every list of size $n \geq 1$.

Basis step When $n = 1$, the list contains just one element and hence is clearly sorted, so P(1) is true.

Induction step Assume P(k) is true; that is, the algorithm sorts correctly every list of k (≥ 1) items.

To show that P($k + 1$) is true, consider a list $X = [x_1, x_2, \ldots, x_{k+1}]$. Since $k + 1 \geq 2$, the **for** loop in line 1 is entered. When $i = 1$, j runs from 1 through $n - 1$. Lines 3 and 4 are executed: the consecutive elements x_j and x_{j+1} are compared and swapped if out of order. The inner **for** loop places the largest of the elements $x_1, x_2, \ldots, x_{k+1}$ in position $k + 1$. This leaves a sublist of k elements, $[x_1, x_2, \ldots, x_k]$. By the inductive hypothesis, the algorithm correctly sorts it. It follows that the algorithm correctly sorts the entire list X; that is, P($k + 1$) is true.

Thus, by the principle of induction, P(n) is true for $n \geq 1$; that is, the bubble sort algorithm always works. ■

Selection Sort

Unlike bubble sort, **selection sort** finds the largest element and swaps it with x_n if x_n is not the largest element. Find the largest of the remaining elements $x_1, x_2, \ldots, x_{n-1}$, and switch it with x_{n-1} if it isn't x_{n-1}. Continue like this until the list is completely sorted.

> In each pass, unlike in bubble sort, if two elements are out of order, we do not swap them right away but wait to find the largest element of the sublist. At the end of the ith pass, the largest of the elements $x_1, x_2, \ldots, x_{n-i+1}$ is swapped with x_{n-i+1}, where $1 \leq i < n$.

This outline of the selection sort algorithm can be a bit refined. In the ith pass, initially assume x_{n-i+1} is the largest element. Find the largest of the elements $x_1, x_2, \ldots, x_{n-i}$. Swap it with x_{n-i+1} if necessary. Algorithm 4.11 results.

```
Algorithm selection sort(X,n)
(* This algorithm sorts a list X of n items using the iterative version
   of selection sort. Maxindex denotes the index of the largest element
   in a given pass. *)
 0. Begin (* algorithm *)
 1.   if n > 1 then(* list contains at least two elements *)
 2.    for i=1 to n - 1 do
```

```
3.      begin (* for *)
4.        maxindex ← n − i + 1 (* assume x_{n−i+1} is the
              largest element; save its index. *)
5.        for j=1 to n − i do
6.          if x_j > x_maxindex, then      (* update maxindex *)
7.            maxindex ← j
8.          if maxindex ≠ n − i + 1, then (* found a larger
              element; swap the corresponding elements *)
9.            swap x_maxindex and x_{n−i+1}
10.       endfor
11.   End (* algorithm *)
```

Algorithm 4.11

EXAMPLE 4.32 Establish the correctness of Algorithm 4.11.

PROOF (by PMI):
Let P(n): The algorithm works correctly for every list of size $n \geq 1$.

Basis step When $n = 1$, the list contains one element and is clearly sorted, so P(1) is true.

Induction step Assume P(k) is true; that is, the algorithm sorts correctly every list of size $k \geq 1$.

To show that P($k + 1$) is true, consider a list $X = [x_1, x_2, \ldots, x_{k+1}]$ with $k + 1$ elements, where $k + 1 \geq 2$. Since $k + 1 \geq 2$, the condition in line 1 is satisfied, and we enter the loop in line 2. When $i = 1$, $maxindex = (k + 1) - 1 + 1 = k + 1$. The **for** loop in lines 5–7 compares each of the elements x_1, $x_2, \ldots, x_k$ with x_{maxindex} and updates it as needed. Line 8 updates *maxindex* if we have found an element larger than x_{k+1}. If maxindex $\neq k + 1$, then the elements x_{k+1} and x_{maxindex} are swapped. This stores the largest of the $k + 1$ elements in position $k + 1$, leaving a sublist of k elements, namely, $x_1, x_2, \ldots, x_k$ to be sorted.

Therefore, by the inductive hypothesis, the algorithm sorts correctly the list X containing $k + 1$ elements.

Thus, by induction, P(n) is true for every $n \geq 1$; that is, the algorithm correctly sorts every list of size n. ■

These searching and sorting algorithms are pursued again in Section 4.7. Additional sorting algorithms appear in the exercises.

Exercises 4.5

Prove that the given predicate P(n) in each algorithm is a loop invariant.

1.
```
Algorithm exponential(x,n)
(* This algorithm computes
x^n, where x∈ℝ⁺ and
n ∈ W. *)
```

2.
```
Algorithm division(x,y)
(* This algorithm computes
the quotient and the
remainder when a positive
```

```
0. Begin (* algorithm *)
1.  answer ← 1
2.  while n > 0 do
3.  begin (* while *)
4.   answer ← answer · x
5.   n ← n − 1
6.  endwhile
7. End  (* algorithm *)
```

P(n): $a_n = x^n$, where a_n denotes the value of answer after n iterations of the **while** loop.

3.

```
Algorithm Euclid(x,y,divisor)
(* See Algorithm 4.2 *)
```

P(n): $\gcd\{x_n, y_n\} = \gcd\{x, y\}$ where x_n and y_n denote the values of x = dividend and y = divisor after n iterations.

4.

```
Algorithm gcd(x,y)
(* This algorithm computes
the gcd of two positive
integers x and y. *)
0. Begin (* algorithm *)
1.  while x ≠ y do
2.   if x > y then
3.    x ← x − y
4.   else
5.    y ← y − x
6.   gcd ← x
7. End (* algorithm *)
```

P(n): $\gcd\{x_n, y_n\} = \gcd\{x, y\}$, where x_n and y_n denote the values of x and y at the end of n iterations of the loop.

5.

```
Algorithm sum (x,y) (* This
algorithm prints the sum of
two nonnegative integers x
and y. *)
0. Begin (* algorithm *)
1.  sum ← x
2.  count ← 0 (* counter *)
3.  while count < y do
4.  begin (* while *)
5.   sum ← sum + 1
6.   count ← count + 1
7.  endwhile
8. End (* algorithm *)
```

P(n): $x = q_n y + r_n$, where q_n and r_n denote the quotient and the remainder after n iterations.

6.

```
Algorithm square (x) (* This algorithm prints the square of x∈W. *)
0.  Begin (* algorithm *)
1.  answer ← 0
2.  i ← 0 (* counter *)
3.  While i < x do
4.  begin (* while *)
5.   answer ← answer + (2i + 1):
6.  i ← i + 1
7.  endwhile
8.  End (* algorithm *)
```

P(n): $a_n = n^2$, where a_n denotes the value of answer at the end of n iterations.

```
integer x is
divided by a positive
integer y using addition and
subtraction. *)
 0. Begin (* algorithm *)
 1.  dividend ← x
 2.  divisor ← y
 3.  quotient ← 0
 4.  remainder ← dividend
 5.  while dividend ≥
       divisor do
 6.  begin (* while *)
 7.   dividend ← dividend
        − divisor
 8.   quotient ← quotient + 1
 9.   remainder ← dividend
10.  endwhile
11. End  (* algorithm *)
```

Using the algorithm in Exercise 4, compute the gcd of each pair of integers.

7. 18, 3 **8.** 28, 12 **9.** 28, 48 **10.** 24, 112

Sort the following lists using the bubble sort algorithm.

11. 23, 7, 18, 19, 53 **12.** 19, 17, 13, 8, 5

13–14. Sort each list in Exercises 11 and 12 using the selection sort algorithm.

Write an iterative algorithm to do the tasks in Exercises 15–17.

15. Compute $n!$, $n \geq 0$.

16. Determine if two $n \times n$ matrices A and B are equal.

17. Compute the product of two $n \times n$ matrices A and B.

18. Let $A = (a_{ij})_{n \times n}$ and $B = (b_{ij})_{n \times n}$. A is **less than or equal to** B, denoted by $A \leq B$, if $a_{ij} \leq b_{ij}$ for every i and j. Write an algorithm to determine if $A \leq B$.

Consider a list X of n numbers $x_1, x_2, \ldots, x_n$. Write iterative algorithms to do the tasks in Exercises 19–25.

19. Find the sum of the numbers.

20. Find the product of the numbers.

21. Find the maximum of the numbers.

22. Find the minimum of the numbers.

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

24. Print the numbers in the reverse order $x_1, x_2, \ldots, x_n$.

25. Write an algorithm to determine if a string S of n characters is a palindrome.

26–36. Establish the correctness of each algorithm in Exercises 15–25.

Use the **insertion sort** algorithm in Algorithm 4.12 to answer Exercises 37–39.

```
Algorithm insertion sort(X,n)
(* This algorithm sorts a list X of n elements into
   ascending order by inserting a new element in the
   proper place at the end of each pass. *)
 0. Begin (* algorithm *)
 1.  for i=2 to n do
 2.  begin (* for *)
 3.   temp ← xi   (* temp is a temporary variable *)
 4.   j ← i − 1
 5.   while j ≥ 1 do
```

```
 6.   begin (* while *)
 7.    if x_j > temp then
 8.     x_{j+1} ← x_j
 9.    j ← j − 1
10.   endwhile
11.   x_{j+1} ← temp
12.  endfor
13. End (* algorithm *)
```

Algorithm 4.12

Sort each list.

37. 3, 13, 8, 6, 5, 2

38. 11, 7, 4, 15, 6, 2, 9

39. Establish the correctness of the algorithm.

4.6 The Growth of Functions

The growth of functions can be investigated using three important notations: the big-oh (O), the big-omega (Ω), and the big-theta (Θ) notations.* We will employ it in Sections 4.7 and 5.7 to analyze some standard algorithms.

Suppose we have developed two algorithms to solve a problem. To determine if one is better than the other, we need some type of yardstick to measure their efficiency. Since the complexity of an algorithm is a function of the input size n, we measure efficiency in terms of n. To this end, we begin with the big-oh notation, introduced in 1892 by the German mathematician Paul Gustav Heinrich Bachmann. The big-oh symbol is also known as the **Landau symbol** after the German mathematician Edmund Landau who popularized it.

The Big-Oh Notation

Let $f, g: \mathbb{N} \to \mathbb{R}$. Then $f(n)$ is of **order at most** $g(n)$, if a positive constant C and a positive integer n_0 exist such that $|f(n)| \leq C|g(n)|$ for every $n \geq n_0$. In symbols, we write $f(n) = O(g(n))$. (Read this as *$f(n)$ is big-oh of $g(n)$.*)

In this definition, if we can find one value for C, any value greater than that also will work, so the value of C is not unique.

When we say the time needed to execute an algorithm is $O(g(n))$, it simply means the time needed is not more than some constant times $|g(n)|$ when n is sufficiently large. For instance, let c_n denote the maximum number of element comparisons required in line 3 of the linear search algorithm (Algorithm 4.8), where n denotes the input size. Using c_n as an

*Ω and Θ are the uppercase Greek letters *omega* and *theta*, respectively.

Paul Gustav Heinrich Bachmann *(1837–1920), the son of a Lutheran minister, was born in Berlin. He inherited a pious attitude and a great love for music. During his early years, he had difficulties in mathematical studies, but his talent was discovered by one of his teachers.*

After recovering from tuberculosis in Switzerland, Bachmann studied mathematics, first at the University of Berlin and then at the University of Göttingen, where he attended Dirichlet's lectures. In 1862 he received his doctorate from Berlin under the guidance of the famous German mathematician Ernst Kummer, for a thesis on group theory. He became a professor at Breslau and later at Munster.

Around 1890, he resigned his position and moved to Weimar, Germany, where he continued his mathematical writing, composed music, played the piano, and wrote music criticism for newspapers. His writings include several volumes on number theory and a book on Fermat's Last Theorem. Bachmann died in Weimar.

Edmund Landau *(1877–1938), the son of a gynecologist, was born in Berlin. After attending high school, he studied mathematics at the University of Berlin, receiving his doctorate under the German mathematician Georg Frobenius in 1899. He taught at Berlin until 1909 and then moved to the University of Göttingen, where both David Hilbert and Felix Klien were colleagues. After the Nazis forced him to quit teaching, he never gave another lecture in Germany.*

Landau's principal contributions were to analytic number theory, especially to the distribution of primes. He wrote several books and more than 250 papers, and exercised tremendous influence on the development of number theory. Landau died suddenly in Berlin.

estimate of the execution of the algorithm, it can be shown that $c_n = O(n)$ (see Example 4.44). This means c_n grows no faster than n, when n is sufficiently large.

Before we analyze the execution times of algorithms, we will study a few simple examples to show how to use the big-oh notation.

EXAMPLE 4.33 Let $f(n) = 50n^3 - 6n + 23$. Show that $f(n) = O(n^3)$.

SOLUTION:

$$f(n) = 50n^3 - 6n + 23$$

Therefore,

$$|f(n)| = |50n^3 - 6n + 23|$$

$$\leq |50n^3| + |-6n| + |23|, \text{ by the triangle inequality}$$

$$
\begin{aligned}
&= 50n^3 + 6n + 23 \\
&\leq 50n^3 + 6n^3 + 23n^3, \text{ when } n \geq 1 \quad (\textit{Note}: n_0 = 1) \\
&= 79n^3
\end{aligned}
$$

Thus, by taking $C = 79$, it follows that $f(n) = O(n^3)$. ■

More generally, we have the following result.

THEOREM 4.14 Let $f(n) = \sum_{i=0}^{m} a_i n^i$ be a polynomial in n of degree m. Then $f(n) = O(n^m)$.

PROOF:
$f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$. By the triangle inequality, we have:

$$
\begin{aligned}
|f(n)| &\leq |a_m| n^m + |a_{m-1}| n^{m-1} + \cdots + |a_1| n + |a_0| \\
&\leq |a_m| n^m + |a_{m-1}| n^m + \cdots + |a_1| n^m + |a_0| n^m, \quad n \geq 1 \\
&= \left(\sum_{i=1}^{m} |a_i| \right) n^m = Cn^m, \text{ where } C = \sum_{i=1}^{m} |a_i| \\
&= O(n^m)
\end{aligned}
$$

Thus, when n is sufficiently large, the leading term dominates the value of the polynomial. ■

In Example 4.33, although $f(n) = O(n^3)$, it is also true that $f(n) \leq 79n^5$ and $f(n) \leq 79n^6$. So we could say correctly, but meaninglessly, that $f(n) = O(n^5)$ and also $f(n) = O(n^6)$. To make comparisons meaningful, however, we shall always choose the smallest possible order of magnitude.

Commonly Used Order Functions

The most common order functions and their names are listed below, arranged in increasing order of magnitude:

- Constant $O(1)$
- Logarithmic $O(\lg n)$
- Linear $O(n)$
- (no name exists) $O(n \lg n)$
- Quadratic $O(n^2)$
- Cubic $O(n^3)$
- Polynomial $O(n^m)$

- Exponential $O(2^n)$
- Factorial $O(n!)$

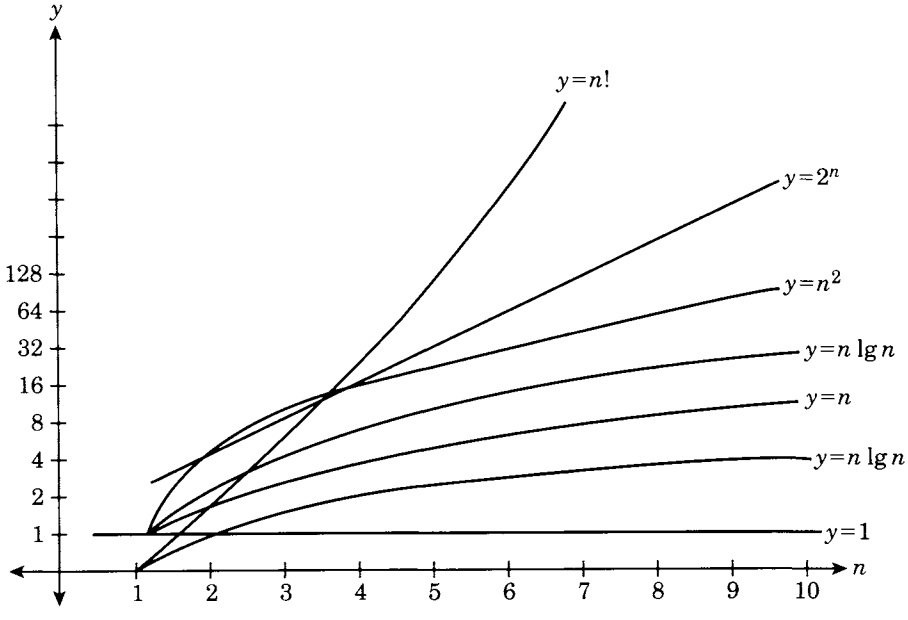
When we say that the order of magnitude of an algorithm is a constant, we mean that the execution time is bounded by a constant; that is, it is independent of the input size n. If the order is linear, the execution time grows linearly; it is directly proportional to the input size.

Approximate values of some of the order functions are given in Table 4.3 for comparison; the graphs of a few of them are given in Figure 4.31.

Table 4.3

lg n	n	n lg n	n^2
3	10	30	100
6	100	600	10,000
9	1,000	9,000	100,000
13	10,000	130,000	100,000,000
16	100,000	1,600,000	10,000,000,000
19	1,000,000	19,000,000	one trillion

Figure 4.31



The order functions satisfy the following relationships among the frequently used execution times, when n is sufficiently large: $O(1) < O(\lg n) < O(n) < O(n \lg n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$. They give us an idea of how long algorithms of varying orders will take to execute jobs.

For instance, if two algorithms solve a problem, one with $O(n)$ and the other with $O(\lg n)$, then (other things being equal) the second algorithm will work faster.

The next two examples also illustrate how to estimate the growth of functions.

EXAMPLE 4.34 Show that $n! = O(n^n)$ and $\lg n! = O(n \lg n)$.

SOLUTION:

- $$\begin{aligned} n! &= n(n-1)\cdots 3 \cdot 2 \cdot 1 \\ &\le n \cdot n \cdots n \cdot n \cdot n, \quad \text{where } n \ge 1 \\ &= n^n \\ &= O(n^n) \qquad (\textit{Note}: \text{Use } C = 1.) \end{aligned}$$
- Since $n! \le n^n$ from above,
  $$\begin{aligned} \lg n! &\le n \lg n \qquad (\textit{Note}: \text{If } 0 < x \le y, \text{ then } \lg x \le \lg y.) \\ &= O(n \lg n) \end{aligned}$$

■

The following example shows how to estimate in a nested **for** loop the growth of the number of times an assignment statement is executed.

EXAMPLE 4.35 Estimate $f(n)$, the number of times the statement $x \leftarrow x + 1$, is executed in the following **for** loop.

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

SOLUTION:
Since the statement $x \leftarrow x + 1$ is executed i times for each value of i, where $1 \le i \le n$,

$$f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{n} = O(n^2)$$

As n increases, $f(n)$ grows as n^2. ■

The Growth of a Sum of Two Functions

Imagine an algorithm consisting of two subalgorithms. Suppose the orders of execution times of the subalgorithms are given by $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. The next theorem shows how to compute the order of the algorithm.

THEOREM 4.15 Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. Then $(f_1 + f_2)(n) = O(max\{|g_1(n)|, |g_2(n)|\})$.

PROOF
By definition, there exist positive constants C_1, C_2, n_1, and n_2 such that $|f_1(n)| \le C_1\ |g_1(n)|$ for $n \ge n_1$, and $|f_2(n)| \le C_2|g_2(n)|$ for $n \ge n_2$. Let $C = \max\{C_1, C_2\}$, $n_0 = \max\{n_1, n_2\}$, and $g(n) = \max\{|g_1(n)|, |g_2(n)|\}$. Then:

$$\begin{aligned} |f_1(n) + f_2(n)| &\le C_1|g_1(n)| + C_2|g_2(n)| \\ &\le C|g(n)| + C|g(n)|, \text{ where } n \ge n_0 \\ &= 2C|g(n)| \end{aligned}$$

Thus $f_1(n) + f_2(n) = O(g(n))$; that is, $(f_1 + f_2)(n) = O(\max\{|g_1(n)|, |g_2(n)|\})$. ■

It follows by this theorem that if $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$, then $(f_1 + f_2)(n) = O(g(n))$. Why?

The Growth of a Product of Two Functions

The next theorem helps us to estimate the growth of $(f_1 \cdot f_2)(n)$, the product of the functions f_1 and f_2.

THEOREM 4.16 Let $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. Then $(f_1 \cdot f_2)(n) = O(g_1(n) \cdot g_2(n))$.

PROOF
Again, by definition, there are constants C_1, C_2, n_1, and n_2 such that $|f_1(n)| \le C_1|g_1(n)|$ for $n \ge n_1$, and $|f_2(n)| \le C_2|g_2(n)|$ for $n \ge n_2$. Let $C = C_1C_2$ and $n_0 = \max\{n_1, n_2\}$. Then:

$$\begin{aligned} |(f_1 \cdot f_2)(n)| &= |f_1(n) \cdot f_2(n)| \\ &= |f_1(n)| \cdot |f_2(n)| \\ &\le C_1|g_1(n)| \cdot C_2|g_2(n)| \\ &= C|g_1(n)g_2(n)|, \text{ where } n \ge n_0 \end{aligned}$$

Thus $(f_1 \cdot f_2)(n) = O(g_1(n)g_2(n))$. ■

The next two examples employ this handy theorem along with the earlier theorems.

EXAMPLE 4.36 Let $f(n) = 6n^2 + 5n + 7 \lg n!$. Estimate the growth of $f(n)$.

SOLUTION:
Since $6n^2 = O(n^2)$ and $5n = O(n)$, $6n^2 + 5n = O(n^2)$ by Theorem 4.15. Furthermore, $7 = O(1)$, and $\lg n! = O(n \lg n)$ by Example 4.34. So

$$\begin{aligned} 7 \lg n! &= O(1) \cdot O(n \lg n) \\ &= O(1 \cdot n \lg n), \text{ by Theorem 4. 16} \\ &= O(n \lg n) \end{aligned}$$

Since $\lg n \leq n$, $n \lg n \leq n^2$ for $n \geq 1$ (see Figure 4.31), it follows by Theorem 4.15 that $f(n) = O(n^2) + O(n \lg n) = O(n^2)$. ■

EXAMPLE 4.37 Let $f(n) = (3n^2 + 4n - 5) \lg n$. Estimate the growth of $f(n)$.

SOLUTION:
$3n^2 + 4n - 5 = O(n^2)$, by Theorem 4.14
Clearly,

$$\lg n = O(\lg n)$$

So

$$\begin{aligned} f(n) &= (3n^2 + 4n - 5) \lg n \\ &= O(n^2) \cdot O(\lg n) \\ &= O(n^2 \lg n), \text{ by Theorem 4.16} \end{aligned}$$

■

We now turn to the big-omega and the big-theta notations for investigating the growth of functions.

The Big-Omega and Big-Theta Notations

The big-oh notation has been widely used in the study of the growth of functions; however, it does not give us an exact order of growth. For instance, $f(n) = O(g(n))$ just implies that the function f does not grow any faster than g. In other words, it simply provides an upper bound for the size of $f(n)$ for large values of n, but no lower bound.

When we need the lower bound, we employ the big-omega notation. When we need both bounds to estimate the growth of f, we use the big-theta notation. Both notations were introduced in the 1970s by Donald Knuth of Stanford University.

We now pursue the big-omega notation. As you could imagine by now, its definition closely resembles that of the big-oh notation; it can be obtained by simply changing $\leq$ to $\geq$.

The Big-Omega Notation

Let $f, g: \mathbb{N} \to \mathbb{R}$. Suppose there is a positive constant C and a positive integer n_0 such that $|f(n)| \geq C|g(n)|$ for every $n \geq n_0$. Then $f(n)$ is $\Omega(g(n))$; that is, $f(n) = \Omega(g(n))$. [As above, read this as $f(n)$ is big-omega of $g(n)$.]

The following example illustrates this definition.

EXAMPLE 4.38 Let $f(n) = 50n^3 - 6n + 23$. When $n \geq 0$, $50n^3 - 6n + 23 \geq 50n^3$. So, with $C = 50$ and $g(n) = n^3$, it follows that $f(n) \geq C \cdot g(n)$ for every $n \geq 0$. Thus $f(n) = \Omega(n^3)$. (Notice that here $n_0 = 0$.) ■

Donald Ervin Knuth *(1938–), a pioneer in the development of the theory of compilers, programming languages, and the analysis of algorithms, is also a prolific writer in computer science. He was born in Milwaukee, Wisconsin, where his father, the first college graduate in the Knuth family, taught bookkeeping at a Lutheran high school; his talent for mathematics and music played a significant role in the intellectual development and pursuit of the young Knuth.*

As a youngster, Knuth had a marvelous gift for solving complex problems. As an eighth grader, he entered the Ziegler's Candies Contest to find the number of words that can be formed from the letters in Ziegler's Giant Bar. *Knuth listed 4500 such words, 2000 more than in Ziegler's master list. This won a television set for the school and enough Ziegler candy for the entire student body.*

In high school, Knuth entered the prestigious Westinghouse Science Talent Search (now Intel Science Talent Search) with his project, The Prtrzebie System of Weights and Measures, *that would replace the cumbersome British system. His project won an honorable mention, and $25 from* MAD Magazine *for publishing it. When he graduated from high school, he was already an accomplished mathematician, musician, and writer.*

He majored in physics at the Case Institute of Technology (now Case Western Reserve University) and was introduced to an IBM 650 computer, one of the earliest mainframes. After studying the manual from cover to cover, he decided that he could do better and wrote assembler and compiler code for the school's IBM 650.

In 1958, Knuth developed a system for analyzing the value of a basketball player, which the coach then used to help the team win a league championship. Newsweek *wrote an article about Knuth's system and Walter Cronkite carried it on the* CBS Evening News.

In his sophomore year, Knuth switched his major to mathematics. His work at Case was so distinguished that when he was awarded his B.S. in 1960, the faculty made an unprecedented decision to grant him an M.S. concurrently.

Knuth then entered the California Institute of Technology for graduate work and received his Ph.D. in mathematics 3 years later. He joined the faculty there, also consulting for the Burroughs Corporation writing compilers for various programming languages, including ALGOL 58 and FORTRAN II.

From 1968–1969, he worked at the Institute for Defense Analyses, Princeton, New Jersey. In 1969, Knuth joined the faculty at Stanford University.

Knuth's landmark project, The Art of Computer Programming, *was initiated by Addison-Wesley Publishing Co. in early 1962, while he was still in graduate school. Dedicated to the study of algorithms, it would be a seven-volume series when completed. A revered work, it was the pioneer textbook in the 1970s and continues to be an invaluable resource. Knuth developed two computer languages to deal with mathematics typography,* TEX, *a typesetting program, and* Metafont, *a program to develop the shapes of letters.*

He has received numerous honorary degrees from universities around the world: the Grace Murray Hopper Award (1971), the Alan M. Turing Award (1974), the Lester R. Ford Award (1975), the National Medal of Science (1979), the McDowell Award (1980), the Computer Pioneer Award (1982), and the Steele Prize (1987).

An accomplished church organist and composer of music for the organ, Knuth retired from Stanford in 1992.

We now make an interesting observation. To this end, let $f(n) = \Omega(g(n))$; so $|f(n)| \geq C|g(n)|$ for $n \geq n_0$. Then $|g(n)| \leq C'|f(n)|$ for some positive constant $C' = 1/C$; so $g(n) = O(f(n))$. Conversely, let $g(n) = O(f(n))$. By retracing these steps, it follows that $f(n) = \Omega(g(n))$. Thus $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

We now define the big-theta notation, using the big-oh and big-omega notations.

The Big-Theta Notation

Let $f, g : \mathbb{N} \to \mathbb{R}$ such that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. Then $f(n)$ is said to be of **order** $g(n)$. We then write $f(n) = \Theta(g(n))$; read this as *$f(n)$ is big-theta of $g(n)$.*

The next two examples illustrate this definition.

EXAMPLE 4.39 Let $f(n) = (3n^2 + 4n - 5) \lg n$. By Example 4.37, $f(n) = O(n^2 \lg n)$. When $n \geq 1$, we also have:

$$(3n^2 + 4n - 5)\lg n \geq 3n^2 \lg n$$

That is,

$$f(n) \geq 3(n^2 \lg n)$$

So

$$f(n) = \Omega(n^2 \lg n)$$

Thus $f(n) = O(n^2 \lg n) = \Omega(n^2 \lg n)$, so $f(n) = \Theta(n^2 \lg n)$. ■

EXAMPLE 4.40 Let $f(n)$ show the number of times the assignment statement $x \leftarrow x + 1$ is executed by the nested for loops in Example 4.35. Recall that $f(n) = n(n + 1)/2 = O(n^2)$.

Since $n + 1 \geq n$ for every $n \geq 1$, it follows that $n(n + 1)/2 \geq n^2/2$; so $f(n) = \Omega(n^2)$. Thus $f(n) = \Theta(n^2)$. ■

We now make two interesting observations from Examples 4.39 and 4.40:

- If $f(n)$ is a polynomial in n of degree m, then $f(n) = \theta(n^m)$.
- $f(n) = \Theta(g(n))$ if and only if $A|g(n)| \leq |f(n)| \leq B \mid g(n) \mid$ for some constants A and B.

See Exercises 50 and 51.

Before closing this section, we add that the definitions of the big-oh, big-omega, and big-theta notations remain valid even if the domain of f consists of real numbers.

Exercises 4.6

Using the big-oh notation, estimate the growth of each function.

1. $f(n) = 2n + 3$ **2.** $f(n) = 4n^2 + 2n - 3$ **3.** $f(n) = 2n^3 - 3n^2 + 4n$

4. $f(n) = 3 + \lg n$ **5.** $f(n) = 3 \lg n + 2$ **6.** $f(n) = (3n)!$

7. $f(n) = \lg (5n)!$ **8.** $f(n) = 23$ **9.** $f(n) = \sum_{k=1}^{n} k^2$

10. $f(n) = \sum_{k=1}^{n} k^3$ **11.** $f(n) = \sum_{i=1}^{n} \lfloor i/2 \rfloor$ **12.** $f(n) = \sum_{i=1}^{n} \lceil i/2 \rceil$

Verify each.

13. $2^n = O(n!)$ **14.** $\sum_{i=0}^{n-1} 2^i = O(2^n)$ **15.** $\sum_{i=0}^{n} i^k = O(n^{k+1})$

16. $\sum_{i=1}^{n} \frac{1}{i(i+1)} = O(1)$ **17.** $\sum_{i=1}^{n} i(i+1) = O(n^3)$ **18.** $\sum_{i=1}^{n} (2i-1)^2 = O(n^3)$

19–22. Let a_n denote the number of times the statement $x \leftarrow x + 1$ is executed by each loop in Exercises 35–38 in Section 4.4. Using the big-oh notation, estimate the growth of a_n in each case.

23–32. Using the big-omega notation, estimate the growth of each function in Exercises 1–5 and 8–12.

Verify each.

33. $(3n)! = \Omega(6^n)$

34. $\sum_{i=1}^{n} i(i+1) = \Omega(n^3)$

35. $\sum_{i=1}^{n} (2i-1) = \Omega(n^2)$

36. $\sum_{i=1}^{n} (2i-1)^2 = \Omega(n^3)$

37. $2n + 3 = \Omega(n)$

38. $4n^2 + 2n - 3 = \Omega(n^2)$

39. $2n^3 - 3n^2 + 4n = \Omega(n^3)$

40. $3 + \lg n = \Omega(\lg n)$

41. $3 \lg n + 2 = \Omega(\lg n)$

42. $23 = \Omega(1)$

43. $\sum_{i=1}^{n} \lfloor i/2 \rfloor = \Omega(n^2)$

44. $\sum_{i=1}^{n} \lceil i/2 \rceil = \Omega(n^2)$

45. Let $f_1(n) = O(g(n))$ and $f_2(n) = kf_1(n)$, where k is a positive constant. Show that $f_2(n) = O(g(n))$.

46. Consider the constant function $f(n) = k$. Show that $f(n) = O(1)$.

Let $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Verify each.

47. $(f + g)(n) = O(h(n))$ **48.** $(f \cdot g)(n) = O((h(n))^2)$

49. Let f, g, and h be three functions such that $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Show that $f(n) = O(h(n))$.

50. Let $f(n) = \sum_{i=0}^{m} a_i n^i$, where each a_i is a real number and $a_m \neq 0$. Prove that $f(n) = \Theta(n^m)$.

51. Let $f, g : \mathbb{N} \to \mathbb{R}$. Prove that $f(n) = \Theta(g(n))$ if and only if $A\,|g(n)| \leq |f(n)| \leq B\,|g(n)|$ for some constants A and B.

4.7 Complexities of Algorithms (optional)

The time complexities of standard algorithms can be used to estimate theoretically using the big-oh and big-theta notations. Before beginning to code an algorithm we should make sure it will do its job. Why is analyzing the algorithm important? Several routines can perform the same task, but not necessarily with the same efficiency, so we should employ the one that is most efficient.

Two norms are used to measure the efficiency of an algorithm: space complexity and time complexity.

Space Complexity

Space complexity refers to how much storage space the algorithm needs. Since this depends on factors such as the computer used and methods of data storage, we restrict our discussion to time complexity.

Time Complexity

The **time complexity** of an algorithm refers to the time it takes to run the algorithm. It is often measured by the number of fundamental operations performed by the algorithm. In the case of a sorting or searching algorithm, we shall use element-comparison as the basic operation. Since the time required by an algorithm depends on the input size n, we measure time complexity in terms of n.

Often we are interested in three cases:

- The **best-case time** is the minimum time needed to execute an algorithm for an input of size n.

- The **worst-case-time** is the maximum time needed to execute the algorithm for an input of size n.
- The **average-case-time** is the average time needed to execute the algorithm for an input of size n. Estimating the average time is often a difficult task, involving probability.

We begin our analysis with the algorithm for matrix multiplication.

EXAMPLE 4.41 Estimate the number a_n of operations (additions and multiplications) needed to compute the product C of two matrices A and B of order n.

SOLUTION:

Let $A = (a_{ij})_{n\times n}$, $B = (b_{ij})_{n\times n}$, and $C = (c_{ij})_{n\times n}$. Since $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$, it takes n multiplications and $n - 1$ additions to compute each c_{ij}. There are n^2 elements in C and each takes a total of $n + (n - 1) = 2n - 1$ operations. Therefore, $a_n = n^2(2n - 1) = O(n^3) = \Theta(n^3)$. Thus the product takes $O(n^3) = \Theta(n^3)$ operations. ■

Next we estimate the number of operations required to compute the product of two binary integers.

EXAMPLE 4.42 Use Algorithm 4.5 to estimate the maximum number a_n of operations (shifting and additions) required to compute the product of two binary integers $x = (x_n \dots x_0)_{\text{two}}$ and $y = (y_n \dots y_0)_{\text{two}}$.

SOLUTION:

The worst case occurs when $y_j = 1$ for every j. Each y_j contributes a shift of j places to the left. Therefore, the total number of shifts $= \sum_{j=0}^{n} j = n(n + 1)/2$, by Example 4.15.

There are $n + 1$ partial products. Adding them involves an $(n + 1)$-bit integer, an $(n + 2)$-bit integer, ..., a $(2n + 1)$-bit integer. Therefore, the total number of bit additions required is $2n + 1$. Thus:

$$\begin{aligned} a_n &= (\text{maximum no. of shifts}) + (\text{maximum no. of additions}) \\ &= \frac{n(n+1)}{2} + 2n + 1 \\ &= O(n^3) = \Theta(n^3) \end{aligned}$$

■

Next, we estimate the number of comparisons required by the bubble sort algorithm, so review it before proceeding any further.

EXAMPLE 4.43 Let c_n denote the number of comparisons required in line 3 of the bubble sort algorithm (see Algorithm 4.10). Estimate the order of magnitude of c_n.

SOLUTION:

In line 3 of the algorithm, the consecutive elements x_j and x_{j+1} are compared for every value of j. Since j varies from 1 to $n - i$, the

number of comparisons is $n - i$, by virtue of the inner loop, where $1 \leq i \leq n - 1$. So

$$\begin{aligned} c_n &= \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= n(n-1) - \frac{(n-1)n}{2}, \text{ by Example 4.15} \\ &= \frac{n(n-1)}{2} \\ &= O(n^2) = \Theta(n^2) \end{aligned}$$

Thus the bubble sort algorithm takes $O(n^2) = \Theta(n^2)$ comparisons. ■

We turn our analysis to the search algorithms presented in Section 5. Review them before proceeding any further.

EXAMPLE 4.44 Use the linear search algorithm (Algorithm 4.8) to estimate the best time, the worst time, and the average time required to search for a *key* in a list X of n elements.

SOLUTION:
Let a_n, b_n, and c_n denote the number of element comparisons needed in line 3 in the average case, the best case, and the worst case, respectively.

- The best case is realized if $x_n = key$. Since this takes only one comparison for all inputs of size n, $b_n = 1$. So $b_n = O(1)$ and the execution time is a constant.
- To compute c_n, notice that the worst case occurs when *key* does not exist in the list, in which case the **while** loop is executed $n + 1$ times. Therefore,

$$\begin{aligned} c_n &= n + 1 \\ &\leq n + n, \text{ when } n \geq 1 \\ &= 2n = O(n) \end{aligned}$$

Thus, in the worst case, the linear search algorithm takes $O(n)$ comparisons. The run time varies linearly with input size.

- To compute the average time a_n, we need to consider two cases: *key* occurs or does not occur in the list. If *key* occurs in position i, $n-i+1$ element comparisons will be required, where $1 \leq i \leq n$. If *key* does not occur in the list, $n + 1$ comparisons will be needed. So the average time

taken is given by

$$\begin{aligned} a_n &= \frac{(1+2+\cdots+n)+(n+1)}{n+1} \\ &\leq \frac{(n+1)(n+2)}{2(n+1)} \\ &= \frac{n}{2}+1 = O(n) \end{aligned}$$

Again, it takes $O(n)$ element comparisons. Thus, the average case, from the complexity point, is no better than the worst case in linear search. ■

Note: In the average case analysis, we assumed *key* could occur in any of the n positions with an equal chance. We also assumed that it had the same chance of not occurring in the list. If that were not the case, we would need to apply the concept of expected value in probability theory to compute a_n.

Next we examine the complexity of the binary search algorithm.

EXAMPLE 4.45 Let c_n denote the maximum number of comparisons in lines 6 through 8 of the binary search algorithm (Algorithm 4.9). Show that $c_n = O(\lg n)$.

SOLUTION:

Case 1 Let n be a power of 2, say, $n = 2^k$ where $k \geq 0$. Initially, $mid = \lfloor (low + high)/2 \rfloor = \lfloor (1 + 2^k)/2 \rfloor = 2^{k-1}$, so the lower sublist contains $2^{k-1} - 1$ elements and the upper sublist 2^{k-1} elements. By now two comparisons have taken place, one in line 6 and the other in line 8. Since the upper sublist contains more elements, partition it into three sublists. This time the maximum number of elements in a sublist is 2^{k-2} and two more comparisons are needed. At the next stage, two more comparisons are needed. Continue like this until the list contains one element, when $k = 0$. Again, two more comparisons ensue.

Thus, in the worst case, two comparisons are needed for each power i of 2, where $0 \leq i \leq k$. Therefore,

$$\begin{aligned} c_n &= 2(k+1) = 2k+2 \\ &= 2 \lg n + 2, \text{ since } n = 2^k \\ &= O(\lg n) \end{aligned}$$

Case 2 Suppose n is not a power of 2. Let n be an integer such that $2^j < n < 2^{j+1}$. Then $j < \lg n$. Let $N = 2^{j+1}$. Clearly, $c_n < c_N$. By the above analysis, $c_N = 2(j+2)$. Thus:

$$\begin{aligned} c_n &< c_N \\ &= 2(j+2) \end{aligned}$$

$$
\begin{aligned}
&< 2(\lg n + 2) \\
&\leq 2(\lg n + \lg n), \text{ when } n \geq 4 \\
&= 4 \lg n \\
&= O(\lg n)
\end{aligned}
$$

Thus, whether or not n is a power of 2, $c_n = O(\lg n)$, so the algorithm takes $O(\lg n)$ comparisons in the worst case. ■

Additional examples of analyzing the complexities of algorithms appear in the exercises and the next chapter.

Exercises 4.7

1. Show that it takes $O(n^2)$ additions to compute the sum of two square matrices of order n.
2. Let A and B be two square matrices of order n. Let c_n denote the number of comparisons needed to determine whether or not $A \leq B$. Show that $c_n = O(n^2)$.

Let A be a square matrix of order n. Let s_n denote the number of swappings of elements needed to find the transpose A^T of A.

3. Find a formula for s_n.
4. Show that $s_n = O(n^2)$.
5. Show that the number of additions of two n-bit integers is $O(n)$.

Let a_n denote the number of additions (lines 5 and 6) required to compute the square of an integer using the algorithm in Exercise 6 of Section 5.

6. Find a formula for a_n.
7. Show that $a_n = O(n)$.

Algorithm 4.13 finds the maximum value in a list X of n items. Use it to answer Exercises 8 and 9.

```
Algorithm findmax(X,n,max)
(* This algorithm returns the largest item in a list X of n
   items in a variable called max. *)
0.  Begin (* algorithm *)
1.    max ← x1     (* initialize max *)
2.    i ← 2
3.    while i ≤ n do
4.    begin (* while *)
5.      if xi > max then     (* update max *)
6.        max ← xi
7.      i ← i + 1
8.    endwhile
9.  End (* algorithm *)
```

Algorithm 4.13

8. Establish the correctness of the algorithm.
9. Let c_n denote the number of comparisons needed in line 5. Show that $c_n = O(n)$.
10. Let c_n denote the number of element-comparisons in line 6 of the insertion sort algorithm in Algorithm 4.12. Show that $c_n = O(n^2)$.

Use the minmax algorithm in Algorithm 4.14 to answer Exercises 11–13.

```
Algorithm iterative minmax(X,n,min,max)
(* This algorithm returns the minimum and the maximum
   of a list X of n elements. *)
 0.  Begin (* algorithm *)
 1.    if n ≥ 1 then
 2.    begin  (* if *)
 3.      min ← x1
 4.      max ← x1
 5.      for i = 2 to n do
 6.      begin  (* for *)
 7.        if xi < min then
 8.          min ← xi
 9.        if xi > max then
10.          max ← xi
11.      endfor
12.    endif
13.  End (* algorithm *)
```

Algorithm 4.14

11. Find the maximum and the minimum of the list 12, 23, 6, 2, 19, 15, 37.
12. Establish the correctness of the algorithm.
13. Using the big-oh notation, estimate the number c_n of comparisons in lines 7 and 9 of the algorithm.
14. Let c_n denote the maximum number of comparisons in lines 6 through 8 of the binary search algorithm (Algorithm 4.9). Show that $c_n = \Theta(\lg n)$.

Chapter Summary

This chapter provided a quick introduction to number theory, one of the oldest branches of mathematics. By accepting the well-ordering principle as an axiom, we established the principle of induction. We saw many examples of how pivotal induction is in proving loop invariants.

We also illustrated how to add and multiply any two nondecimal numbers, and how to subtract binary integers using complements.

Finally, we established the partial correctness of algorithms and discussed the time complexities of some standard algorithms using the big-oh and big-theta notations.

The Well-Ordering Principle

- Every nonempty subset of N has a least element (page 186).

The Division Algorithm

- The **division algorithm** When an integer a is divided by a positive integer b, there exist a unique quotient q and a unique remainder r such that $a = bq + r$, where $0 \leq x < b$ (page 186).
- An integer $p \geq 2$ is a **prime** if its only positive factors are 1 and p (page 189).

The Greatest Common Divisor (gcd)

- A positive integer d is the gcd of two positive integers a and b if:
  - $d \mid a$ and $d \mid b$; and
  - if $d' \mid a$ and $d' \mid b$, then $d' \mid d$. (page 191).
- The **euclidean algorithm**, which uses successive applications of the division algorithm, provides a procedure to compute $\gcd\{a,b\}$ (page 193).
- Two positive integers a and b are **relatively prime** if $\gcd\{a,b\} = 1$ (page 194).
- Every decimal integer has a unique nondecimal representation in a given base and every nondecimal integer has a unique decimal value (page 197).
- Binary subtraction can be performed using two's complement (page 203).

Mathematical Induction

- **Weak version** Let $P(n)$ be a predicate such that
  - $P(n_0)$ is true; and
  - for every $k \geq n_0$, if $P(k)$ is true, $P(k+1)$ is also true.

  Then $P(n)$ is true for every $n \geq n_0$ (page 209).
- **Strong version** Let $P(n)$ be a predicate such that
  - $P(n_0)$ is true; and

- for every $k \geq n_0$, if $P(n_0)$, $P(n_0 + 1)$, ..., $P(k)$ are true, $P(k + 1)$ is also true. Then $P(n)$ is true for $n \geq n_0$ (page 218).

- **The Fundamental Theorem of Arithmetic** Every positive integer ≥ 2 is either a prime or can be expressed as a product of primes (page 218).

Algorithm Correctness

- Using induction, we verified the partial correctness of several standard algorithms: linear search (page 228), binary search (page 230), bubble sort (page 233), and selection sort (page 234).

The Big-Oh Notation

- $f(n) = O(g(n))$, if there are positive constants C and n_0 such that $|f(n)| \geq C|g(n)|$ for every $n \geq n_0$ (page 237).
- $f(n) = \Omega(g(n))$, if $|f(n)| \geq C|g(n)|$ for every $n \geq n_0$ (page 243).
- $f(n) = \Theta g(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ (page 245).
- The time complexity of an algorithm is the execution time of the algorithm (page 245).

Review Exercises

Using the euclidean algorithm, find the gcd of each pair of integers.

1. 18, 28 **2.** 36, 12 **3.** 15, 24 **4.** 1024, 3076

Express each number in base 10.

5. 2000_{eight} **6.** 2345_{sixteen} **7.** $\text{BAD}_{\text{sixteen}}$ ***8.** $\text{BAD.CA}_{\text{sixteen}}$

Rewrite each number in the indicated base b.

9. 245, $b = 2$ **10.** 348, $b = 8$ **11.** 1221, $b = 8$ **12.** 1976, $b = 16$

In Exercises 13–16, perform the indicated operation.

13. 11010_{two} $+111_{\text{two}}$ **14.** 5768_{sixteen} $+78\text{CB}_{\text{sixteen}}$ **15.** $5\text{AB8}_{\text{sixteen}}$ $\times\text{BAD}_{\text{sixteen}}$ **16.** 110110_{two} -11011_{two}

Rewrite each binary integer in base eight.

17. 10110101 **18.** 1101101101 **19.** 100110011 **20.** 10011011001

21–24. Rewrite the binary integers in Exercises 17–20 in base 16.

Find the value of x resulting from the execution of each algorithm fragment.

25.
```
x ← 0
for i = 1 to n do
  for j = 1 to n do
    x ← x + 1
```

26.
```
x ← 0
for i = 1 to n do
  for j = 1 to i do
    for k = 1 to j do
      x ← x + 1
```

27. Find a formula for the number a_n of times the statement $x \leftarrow x + 1$ is executed by the following loop:

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

28. Let $a,b,c,d \in \mathbb{N}$. Let $d \mid ab$, $d \mid ac$, and b and c be relatively prime numbers. Prove that $d \mid a$.

29. Let $a,b \in \mathbb{N}$ and $\gcd\{a,b\} = 1$. Prove that $\gcd\{a - b, a + b\} = 1$ or 2.

Using induction prove each, where n is a positive integer.

30. $n^2 - n$ is divisible by 2.

31. $n^3 - n$ is divisible by 3.

32. $\displaystyle\sum_{i=1}^{n}(2i-1)^2 = \frac{n(4n^2-1)}{3}$

33. $\displaystyle\sum_{i=1}^{n}\frac{1}{(2i-1)(2i+1)} = \frac{n}{2n+1}$

34. The product of any two consecutive positive integers is even.

35. Suppose you have an unlimited supply of identical black and white socks. Using induction and the pigeonhole principle, show that you must select at least $2n + 1$ socks in order to ensure n matching pairs. (C. T. Long)

Evaluate each sum and product.

36. $\displaystyle\sum_{i=1}^{n} i(i+1)$

37. $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}(2i+3j)$

38. $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} 2^i 3^j$

39. $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} 2^j$

40. $\displaystyle\prod_{i=1}^{n}\prod_{j=1}^{n} 2^i 3^j$

41. $\displaystyle\prod_{i=1}^{n}\prod_{j=1}^{n} 3^{2j}$

42. $\displaystyle\prod_{i=1}^{n}\prod_{j=1}^{n} 2^i$

***43.** $\displaystyle\sum_{i=1}^{n} i \prod_{j=1}^{i} j$

44. Let S_n denote the value of *sum* after n iterations of the while loop in Algorithm 4.15. Prove that $P(n)$: $S_n = n(n + 1)$ is a loop invariant.

```
Algorithm evensum (n)
(* This algorithm computes the sum of the first x
   positive even integers. *)
0.  Begin (* algorithm *)
1.    sum ← 0
```

```
2.    i ← 0 (* counter *)
3.    while i < n do
4.    begin (* while *)
5.      i ← i + 1
6.      sum ← sum + 2 * i
7.    endwhile
8.  End (* algorithm *)
```

Algorithm 4.15

45. Using Example 4.23 predict a formula for the number of trailing zeros in $n!$, where $n \geq 1$.

46. Let a_n denote the number of operations (additions and multiplications) in line 6 of the algorithm in Exercise 44. Find the order of magnitude of a_n.

47. Add two lines to the following number pattern, where t_n denotes the nth triangular number.

$$t_1 + t_2 + t_3 = t_4$$
$$t_5 + t_6 + t_7 + t_8 = t_9 + t_{10}$$
$$t_{11} + t_{12} + t_{13} + t_{14} + t_{15} = t_{16} + t_{17} + t_{18}$$

Prove each, where t_n denotes the nth triangular number.

48. $t_n^2 - t_{n-1}^2 = n^3$ **49.** $t_n^2 = t_n + t_{n-1}t_{n+1}$ **50.** $2t_nt_{n-1} = t_{n^2-1}$

Supplementary Exercises

1. Prove that $(m^2 - n^2, 2mn, m^2 + n^2)$ is a solution of the equation $x^2 + y^2 = z^2$.

2. Prove that the product of the sums of two squares of two integers can be written as a sum of two squares.

3. Let t_k denote the kth triangular number and n any triangular number. Prove that $(2k + 1)^2n + t_k$ is also a triangular number. (R. F. Jordan, 1991)

4. In 1950, P. A. Piza discovered the following formula about sums of powers of triangular numbers t_i: $[3\sum_{i=1}^{n} t_i]^3 = \sum_{i=1}^{n} t_i^3 + 2\sum_{i=1}^{n} t_i^4$. Verify it for $n = 3$ and $n = 4$.

5. Show that 111 cannot be a square in any base.

***6.** Prove that one more than the product of four consecutive integers is a perfect square, and the square root of the resulting number is the average of the product of the smaller and larger numbers and the product of the two middle integers. (W. M. Waters, 1990)

A composite number n is **Duffinian** if none of its positive factors, except 1, is a factor of the sum s of its proper factors. For example, let $n = 21$. The sum of its proper factors $= 1 + 3 + 7 = 11$. Since both 3 and 7 are not factors of 11, 21 is Duffinian. (You may verify that 10 is not Duffinian.)

7. Determine if 18, 25, 36, and 43 are Duffinian.

8. Let p be a prime and k a positive integer ≥ 2. Prove that p^k is Duffinian.

9. Prove that n is Duffinian if and only if none of the factors of n, except 1, is a factor of n.

10. Prove or disprove: The product of two Duffinian numbers is Duffinian.

Prove each, where n is a positive integer.

***11.** $n(3n^4 + 7n^2 + 2)$ is divisible by 12.

***12.** $n(3n^4 + 13n^2 + 8)$ is divisible by 24.

****13.** Let S_n denote the sum of the elements in the nth set in the sequence of sets of positive integers $\{1\}, \{3, 5\}, \{7, 9, 11\}, \{13, 15, 17, 19\}, \ldots$. Find a formula for S_n. (R. Euler, 1988)

****14.** Let S_n denote the sum of the elements in the nth set in the sequence of positive integers $\{1\}, \{2, 3, \ldots, 8\}, \{9, 10, \ldots, 21\}, \{22, 23, \ldots, 40\}, \ldots$. Find a formula for S_n. (C. W. Trigg, 1980)

****15.** Three schools in each state, Alabama, Georgia, and Florida, enter one person in each of the events in a track meet. The number of events and the scoring system are unknown, but the number of points for the third place is less than that for the second place, which in turn is less than the number of points for the first place. Georgia scored 22 points, and Alabama and Florida tied with 9 each. Florida won the high jump. Who won the mile run? (M. vos Savant, 1993)

Computer Exercises

Write a program to perform each task.

1. Read in an integer $b \geq 2$ and select $b + 1$ integers at random. Find two integers in the list such that their difference is divisible by b.

2. Read in an integer $n \geq 2$ and select n positive integers at random. Find a sequence of integers from the list whose sum is divisible by n.

3. Read in a positive integer ≥ 2 and determine if it is a prime.

4. Determine if each value of $f(n) = n^2 - n + 41$ is a prime, where $0 \leq n \leq 41$.
5. Redo Program 4 with $f(n) = n^2 - 79n + 1601$, where $0 \leq n \leq 80$.
6. Determine if the nth **Fermat number** $f(n) = 2^{2^n} + 1$ is a prime, where $0 \leq n \leq 4$.
7. Find all perfect numbers ≤ 1000. (There are three such numbers.)
8. Find the $\gcd\{x,y\}$ using the euclidean algorithm.
9. Read in a sequence of pairs of integers n and b. For each integer n, determine its base-b representation and use this representation to compute the corresponding decimal value. Print each integer n, base b, base-b representation, and its decimal value in a tabular form.
10. Read in a positive integer n and find the number of trailing zeros in $n!$.
11. A **palindrome** is a positive integer that reads the same backward and forward. Find the eight palindromic triangular numbers < 1000.
12. Compute the total number of grains of wheat needed for each of the squares on an 8×8 chessboard, as in Exercises 39 and 40 in Section 4.4. (*Hint*: The answer is 18,446,744,073,709,551,615 grains, which may be too large for an integer variable to hold, so think of a suitable data structure.)
13. Read in a positive integer $N \leq 1000$. Using Example 4.24, determine how many doors will remain open at the end. Do *not* use the fact that there are $\lfloor \sqrt{n} \rfloor$ perfect squares $\leq n$.
14. Print the ages 1–31 on five tablets A, B, C, D, and E, as in Figure 4.2. Read in some tablets at random and compute the corresponding age. Extend the puzzle to six tablets to include ages through 63.
15. Read in a positive integer n and determine if it is a prime.
16. Construct a table of values of the function $E(n) = n^2 - n + 41$, where $0 \leq n \leq 41$, and identify each value as prime or composite.
17. Redo program 16 with $L(n) = n^2 + n + 41$, where $0 \leq n \leq 41$, and identify each value as prime or composite.
18. Redo program 16 with $H(n) = 9n^2 - 471n + 6203$, where $0 \leq n \leq 39$, and identify each value as prime or composite.
19. Redo program 16 with $G(n) = n^2 - 2999n + 2248541$, where $1460 \leq n \leq 1539$, and identify each value as prime or composite.
20. Read in a positive integer n, and list all primes $\leq n$ and are of the form $k^2 + 1$.

21. Read in a positive integer n and find a prime between:
(a) n and $2n$. (b) n^2 and $n^2 + 1$.

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 how twin primes were used in 1994 by Thomas Nicely of Lynchburg College, Virginia, to detect defects in the Pentium chip.
2. Explain how to construct Tables A–E in Figure 4.2 and how the puzzle works. Extend the puzzle to cover ages through 63.
3. Describe the origin of mathematical induction. Include biographies of those who developed this proof technique. Comment on its importance in computer science.
4. Describe the origin of *figurate numbers*. Explain the various types and their properties. Include the relationships between the 12 days of Christmas puzzle, and polygonal numbers and tetrahedral numbers.
5. Explore the history of magic squares. Do they have any practical applications?
6. Describe the origin of the big-oh, big-omega, and big-theta notations. Include biographies of mathematicians who developed them.
7. Investigate the various classes of prime numbers.
8. Describe the history of finding larger and larger primes, and their practical applications. Comment on the Greatest Internet Mersenne Prime Search (GIMPS), founded in 1996 by George Woltman.
9. Discuss the game of *Nim* and its relationship to binary numbers.
10. Discuss *Eleusis*, a card game devised by R. Abbott of New York.

Enrichment Readings

1. R. G. Archibald, *An Introduction to the Theory of Numbers*, Merrill, Columbus, OH, 1970, pp. 1–95.
2. G. Brassard and P. Bratley, *Algorithmics: Theory & Practice*, Prentice Hall, Englewood Cliffs, NJ, 1988.
3. J. Dugle, "The Twelve Days of Christmas and Pascal's Triangle," *Mathematics Teacher*, Vol. 75 (Dec. 1982), pp. 755–757.
4. G. H. Hardy, *A Mathematician's Apology*, Cambridge University Press, Cambridge, 1941.

5. T. Koshy, *Elementary Number Theory with Applications*, Harcourt/ Academic Press, Boston, 2002, pp. 1–189.

6. C. Oliver, "The Twelve Days of Christmas," *Mathematics Teacher*, Vol. 70 (Dec. 1977), pp. 752–754.

7. H. S. Wilf, *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 8–22, 137–175.