2.5 Integers and Algorithms

INTRODUCTION

As mentioned in Section 2.1, the term *algorithm* originally referred to procedures for performing arithmetic operations using the decimal representations of integers. These algorithms, adapted for use with binary representations, are the basis for computer arithmetic. They provide good illustrations of the concept of an algorithm and the complexity of algorithms. For these reasons, they will be discussed in this section.

There are many important algorithms involving integers besides those used in arithmetic, including the Euclidean algorithm, which is one of the most useful algorithms, and perhaps the oldest algorithm, in mathematics. We will also describe an algorithm for finding the base b expansion of a positive integer for any base b and for modular exponentiation, an algorithm important in cryptography.

REPRESENTATIONS OF INTEGERS

In everyday life we use decimal notation to express integers. For example, 965 is used to denote $9 \cdot 10^2 + 6 \cdot 10 + 5$. However, it is often convenient to use bases other than 10. In particular, computers usually use binary notation (with 2 as the base) when carrying out arithmetic, and octal (base 8) or hexadecimal (base 16) notation when expressing characters, such as letters or digits. In fact, we can use any positive integer greater than 1 as the base when expressing integers. This is stated in Theorem 1.

THEOREM 1 Let b be a positive integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0,$$

where k is a nonnegative integer, $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

The proof of this theorem can be found in [Ro99]. The representation of n given in Theorem 1 is called the **base b expansion of n.** The base b expansion of n is denoted by $(a_k a_{k-1} \cdots a_1 a_0)_b$. For instance, $(245)_8$ represents $2 \cdot 8^2 + 4 \cdot 8 + 5 = 165$.

BINARY EXPANSIONS Choosing 2 as the base gives **binary expansions** of integers. In binary notation each digit is either a 0 or a 1. In other words, the binary expansion of an integer is just a bit string. Binary expansions (and related expansions that are variants of binary expansions) are used by computers to represent and do arithmetic with integers.

EXAMPLE 1 What is the decimal expansion of the integer that has $(1\ 0101\ 1111)_2$ as its binary expansion?

Solution: We have

$$(1\ 0101\ 1111)_2 = 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 351. \quad \blacktriangleleft$$

HEXADECIMAL EXPANSIONS Sixteen is another base used in computer science. The base 16 expansion of an integer is called its **hexadecimal** expansion. Sixteen different digits are required for such expansions. Usually, the hexadecimal digits used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, where the letters A through F represent the digits corresponding to the numbers 10 through 15 (in decimal notation).

EXAMPLE 2 What is the decimal expansion of the hexadecimal expansion of $(2AE0B)_{16}$?

Solution: We have

$$(2AE0B)_{16} = 2 \cdot 16^4 + 10 \cdot 16^3 + 14 \cdot 16^2 + 0 \cdot 16 + 11 = (175627)_{10}.$$ ◀

Each hexadecimal digit can be represented using four bits. For instance, we see that $(1110\ 0101)_2 = (E5)_{16}$ since $(1110)_2 = (E)_{16}$ and $(0101)_2 = (5)_{16}$. **Bytes,** which are bit strings of length eight, can be represented by two hexadecimal digits.

BASE CONVERSION We will now describe an algorithm for constructing the base b expansion of an integer n. First, divide n by b to obtain a quotient and remainder, that is,

$$n = bq_0 + a_0, \qquad 0 \leq a_0 < b.$$

The remainder, a_0, is the rightmost digit in the base b expansion of n. Next, divide q_0 by b to obtain

$$q_0 = bq_1 + a_1, \qquad 0 \leq a_1 < b.$$

We see that a_1 is the second digit from the right in the base b expansion of n. Continue this process, successively dividing the quotients by b, obtaining additional base b digits as the remainders. This process terminates when we obtain a quotient equal to zero.

EXAMPLE 3 Find the base 8, or **octal,** expansion of $(12345)_{10}$.

Extra Examples

Solution: First, divide 12345 by 8 to obtain

$$12345 = 8 \cdot 1543 + 1.$$

Successively dividing quotients by 8 gives

$$\begin{aligned} 1543 &= 8 \cdot 192 + 7, \\ 192 &= 8 \cdot 24 + 0, \\ 24 &= 8 \cdot 3 + 0, \\ 3 &= 8 \cdot 0 + 3. \end{aligned}$$

Since the remainders are the digits of the base 8 expansion of 12345, it follows that

$$(12345)_{10} = (30071)_8.$$ ◀

EXAMPLE 4 Find the hexadecimal expansion of $(177130)_{10}$.

Solution: First divide 177130 by 16 to obtain

$$177130 = 16 \cdot 11070 + 10.$$

Successively dividing quotients by 16 gives

$$\begin{aligned} 11070 &= 16 \cdot 691 + 14, \\ 691 &= 16 \cdot 43 + 3, \\ 43 &= 16 \cdot 2 + 11, \\ 2 &= 16 \cdot 0 + 2. \end{aligned}$$

Since the remainders are the digits of the hexadecimal (base 16) expansion of $(177130)_{10}$, it follows that

$$(177130)_{10} = (2\text{B3EA})_{16}.$$

(Recall that the integers 10, 11, and 14 correspond to the hexadecimal digits A, B, and E, respectively.) ◀

EXAMPLE 5 Find the binary expansion of $(241)_{10}$.

Solution: First divide 241 by 2 to obtain

$$241 = 2 \cdot 120 + 1.$$

Successively dividing quotients by 2 gives

$$\begin{aligned} 120 &= 2 \cdot 60 + 0, \\ 60 &= 2 \cdot 30 + 0, \\ 30 &= 2 \cdot 15 + 0, \\ 15 &= 2 \cdot 7 + 1, \\ 7 &= 2 \cdot 3 + 1, \\ 3 &= 2 \cdot 1 + 1, \\ 1 &= 2 \cdot 0 + 1. \end{aligned}$$

Since the remainders are the digits of the binary (base 2) expansion of $(241)_{10}$, it follows that

$$(241)_{10} = (1111\ 0001)_2.$$ ◀

The pseudocode given in Algorithm 1 finds the base b expansion $(a_{k-1} \cdots a_1 a_0)_b$ of the integer n.

ALGORITHM 1 Constructing Base b Expansions.

procedure *base b expansion*(n: positive integer)
$q := n$
$k := 0$
while $q \neq 0$
begin
  $a_k := q$ **mod** b
  $q := \lfloor q/b \rfloor$
  $k := k + 1$
end {the base b expansion of n is $(a_{k-1} \cdots a_1 a_0)_b$}

TABLE 1 Hexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

In Algorithm 1, q represents the quotient obtained by successive divisions by b, starting with $q = n$. The digits in the base b expansion are the remainders of these divisions and are given by q **mod** b. The algorithm terminates when a quotient $q = 0$ is reached.

Remark: Note that Algorithm 1 can be thought of as a greedy algorithm.

Conversion between binary and hexadecimal expansions is extremely easy because each hexadecimal digit corresponds to a block of four binary digits, with this correspondence shown in Table 1 without initial 0s shown. (We leave it as Exercises 11 and 12 at the end of this section to show that this is the case.) This conversion is illustrated in Example 6.

EXAMPLE 6 Find the hexadecimal expansion of $(11\ 1110\ 1011\ 1100)_2$ and the binary expansion of $(A8D)_{16}$.

Solution: To convert $(11\ 1110\ 1011\ 1100)_2$ into hexadecimal notation we group the binary digits into blocks of four, adding initial zeros at the start of the leftmost block if necessary. These blocks are 0011, 1110, 1011, and 1100, which correspond to the hexadecimal digits 3, E, B, and C, respectively. Consequently, $(11\ 1110\ 1011\ 1100)_2 = (3EBC)_{16}$.

To convert $(A8D)_{16}$ into binary notation, we replace each hexadecimal digit by a block of four binary digits. These blocks are 1010, 1000, 1101. Consequently, $(A8D)_{16} = (1010\ 1000\ 1101)_2$. ◀

ALGORITHMS FOR INTEGER OPERATIONS

The algorithms for performing operations with integers using their binary expansions are extremely important in computer arithmetic. We will describe algorithms for the addition and the multiplication of two integers expressed in binary notation. We will also analyze the computational complexity of these algorithms, in terms of the actual number of bit operations used. Throughout this discussion, suppose that the binary expansions of a and b are

$$a = (a_{n-1}a_{n-2}\cdots a_1a_0)_2,\ b = (b_{n-1}b_{n-2}\cdots b_1b_0)_2,$$

so that a and b each have n bits (putting bits equal to 0 at the beginning of one of these expansions if necessary).

We will measure the complexity of algorithms for integer arithmetic in terms of the number of bits in these numbers.

Consider the problem of adding two integers in binary notation. A procedure to perform addition can be based on the usual method for adding numbers with pencil and paper. This method proceeds by adding pairs of binary digits together with carries, when they occur, to compute the sum of two integers. This procedure will now be specified in detail.

To add a and b, first add their rightmost bits. This gives

$$a_0 + b_0 = c_0 \cdot 2 + s_0,$$

where s_0 is the rightmost bit in the binary expansion of $a + b$ and c_0 is the **carry,** which is either 0 or 1. Then add the next pair of bits and the carry,

$$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1,$$

where s_1 is the next bit (from the right) in the binary expansion of $a + b$, and c_1 is the carry. Continue this process, adding the corresponding bits in the two binary expansions and the carry, to determine the next bit from the right in the binary expansion of $a + b$. At the last stage, add a_{n-1}, b_{n-1}, and c_{n-2} to obtain $c_{n-1} \cdot 2 + s_{n-1}$. The leading bit of the sum is $s_n = c_{n-1}$. This procedure produces the binary expansion of the sum, namely, $a + b = (s_n s_{n-1} s_{n-2} \cdots s_1 s_0)_2$.

EXAMPLE 7 Add $a = (1110)_2$ and $b = (1011)_2$.

Solution: Following the procedure specified in the algorithm, first note that

$$a_0 + b_0 = 0 + 1 = 0 \cdot 2 + 1,$$

so that $c_0 = 0$ and $s_0 = 1$. Then, since

$$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1 \cdot 2 + 0,$$

it follows that $c_1 = 1$ and $s_1 = 0$. Continuing,

$$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1 \cdot 2 + 0,$$

so that $c_2 = 1$ and $s_2 = 0$. Finally, since

$$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1 \cdot 2 + 1,$$

it follows that $c_3 = 1$ and $s_3 = 1$. This means that $s_4 = c_3 = 1$. Therefore, $s = a + b = (1\ 1001)_2$. This addition is displayed in Figure 1. ◀

```
 1 1 1
   1 1 1 0
   1 0 1 1
 ---------
 1 1 0 0 1
```

FIGURE 1 Adding $(1110)_2$ and $(1011)_2$.

The algorithm for addition can be described using pseudocode as follows.

ALGORITHM 2 Addition of Integers.

procedure *add*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}a_{n-2} \cdots a_1 a_0)_2$ and $(b_{n-1}b_{n-2} \cdots b_1 b_0)_2$, respectively}
$c := 0$
for $j := 0$ **to** $n - 1$
begin
  $d := \lfloor (a_j + b_j + c)/2 \rfloor$
  $s_j := a_j + b_j + c - 2d$
  $c := d$
end
$s_n := c$
{the binary expansion of the sum is $(s_n s_{n-1} \cdots s_0)_2$}

Next, the number of additions of bits used by Algorithm 2 will be analyzed.

EXAMPLE 8 How many additions of bits are required to use Algorithm 2 to add two integers with n bits (or less) in their binary representations?

Solution: Two integers are added by successively adding pairs of bits and, when it occurs, a carry. Adding each pair of bits and the carry requires three or fewer additions of bits. Thus, the total number of additions of bits used is less than three times the number of bits in the expansion. Hence, the number of additions of bits used by Algorithm 2 to add two n-bit integers is $O(n)$. ◀

Next, consider the multiplication of two n-bit integers a and b. The conventional algorithm (used when multiplying with pencil and paper) works as follows. Using the distributive law, we see that

$$\begin{aligned} ab &= a(b_0 2^0 + b_1 2^1 + \cdots + b_{n-1} 2^{n-1}) \\ &= a(b_0 2^0) + a(b_1 2^1) + \cdots + a(b_{n-1} 2^{n-1}). \end{aligned}$$

We can compute ab using this equation. We first note that $ab_j = a$ if $b_j = 1$ and $ab_j = 0$ if $b_j = 0$. Each time we multiply a term by 2, we shift its binary expansion one place to the left and add a zero at the tail end of the expansion. Consequently, we can obtain $(ab_j)2^j$ by **shifting** the binary expansion of ab_j j places to the left, adding j zero bits at the tail end of this binary expansion. Finally, we obtain ab by adding the n integers $ab_j 2^j$, $j = 0, 1, 2, \ldots, n-1$.

This procedure for multiplication can be described using the pseudocode shown as Algorithm 3.

ALGORITHM 3 Multiplying Integers.

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}a_{n-2}\cdots a_1 a_0)_2$ and $(b_{n-1}b_{n-2}\cdots b_1 b_0)_2$, respectively}
for $j := 0$ **to** $n-1$
begin
  if $b_j = 1$ **then** $c_j := a$ shifted j places
  else $c_j := 0$
end
{$c_0, c_1, \ldots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ **to** $n-1$
  $p := p + c_j$
{p is the value of ab}

Example 9 illustrates the use of this algorithm.

EXAMPLE 9 Find the product of $a = (110)_2$ and $b = (101)_2$.

Solution: First note that

$$ab_0 \cdot 2^0 = (110)_2 \cdot 1 \cdot 2^0 = (110)_2,$$

$$ab_1 \cdot 2^1 = (110)_2 \cdot 0 \cdot 2^1 = (0000)_2,$$

```
  1 1 0
  1 0 1
  -----
  1 1 0
0 0 0
1 1 0
-------
1 1 1 1 0
```

FIGURE 2 **Multiplying $(110)_2$ and $(101)_2$.**

and

$$ab_2 \cdot 2^2 = (110)_2 \cdot 1 \cdot 2^2 = (11000)_2.$$

To find the product, add $(110)_2$, $(0000)_2$, and $(11000)_2$. Carrying out these additions (using Algorithm 2, including initial zero bits when necessary) shows that $ab = (1\ 1110)_2$. This multiplication is displayed in Figure 2. ◀

Next, we determine the number of additions of bits and shifts of bits used by Algorithm 3 to multiply two integers.

EXAMPLE 10 How many additions of bits and shifts of bits are used to multiply a and b using Algorithm 3?

Solution: Algorithm 3 computes the products of a and b by adding the partial products $c_0, c_1, c_2, \ldots,$ and c_{n-1}. When $b_j = 1$, we compute the partial product c_j by shifting the binary expansion of a j bits. When $b_j = 0$, no shifts are required since $c_j = 0$. Hence, to find all n of the integers $ab_j 2^j$, $j = 0, 1, \ldots, n-1$, requires at most

$$0 + 1 + 2 + \cdots + n - 1$$

shifts. Hence, by Example 4 in Section 2.2 the number of shifts required is $O(n^2)$.

To add the integers ab_j from $j = 0$ to $j = n-1$ requires the addition of an n-bit integer, an $(n+1)$-bit integer, ..., and a $(2n)$-bit integer. We know from Example 8 that each of these additions requires $O(n)$ additions of bits. Consequently, a total of $O(n^2)$ additions of bits are required for all n additions. ◀

Surprisingly, there are more efficient algorithms than the conventional algorithm for multiplying integers. One such algorithm, which uses $O(n^{1.585})$ bit operations to multiply n-bit numbers, will be described in Section 6.3.

Given integers a and d, $d > 0$, we can find $q = a$ **div** d and $r = a$ **mod** d using Algorithm 4. In this algorithm, when a is positive we subtract d from a as many times as necessary until what is left is less than d. The number of times we perform this subtraction is the quotient and what is left over after all these subtractions is the remainder. Algorithm 4 also covers the case where a is negative. It finds the quotient q, and remainder r when $|a|$ is divided by d and when $r > 0$, uses these to find the quotient $-(q+1)$ and remainder $d - r$ when a is divided by d. We leave it to the reader (Exercise 55) to show that, assuming that $a > d$, this algorithm uses $O(q \log a)$ bit operations.

There are more efficient algorithms than Algorithm 4 for determining the quotient $q = a$ **div** d and the remainder $r = a$ **mod** d when a positive integer a is divided by a positive integer d (see [Kn98] for details). These algorithms require $O(\log a \cdot \log d)$ bit operations. If both of the binary expansions of a and d contain n or fewer bits, then we can replace $\log a \cdot \log d$ by n^2. This means that we need $O(n^2)$ bit operations to find the quotient and remainder when a is divided by d.

MODULAR EXPONENTIATION

In cryptography it is important to be able to efficiently find b^n **mod** m, where b, n, and m are large integers. It is impractical to first compute b^n and then find its remainder when divided by m because b^n will be a huge number. Instead, we can use an algorithm that employs the binary expansion of the exponent n, say $n = (a_{k-1} \cdots a_1 a_0)_2$. The algorithm

ALGORITHM 4 Computing div and mod.

procedure *division algorithm*(a: integer, d: positive integer)
$q := 0$
$r := |a|$
while $r \geq d$
begin
    $r := r - d$
    $q := q + 1$
end
if $a < 0$ and $r > 0$ **then**
begin
    $r := d - r$
    $q := -(q + 1)$
end
{$q = a$ **div** d is the quotient, $r = a$ **mod** d is the remainder}

ALGORITHM 5 Modular Exponentiation.

procedure *modular exponentiation*(b: integer, $n = (a_{k-1}a_{k-2}\cdots a_1a_0)_2$,
    m: positive integers)
$x := 1$
$power := b$ **mod** m
for $i := 0$ **to** $k - 1$
begin
    if $a_i = 1$ **then** $x := (x \cdot power)$ **mod** m
    $power := (power \cdot power)$ **mod** m
end
{x equals b^n **mod** m}

successively finds b **mod** m, b^2 **mod** m, b^4 **mod** m, ..., $b^{2^{k-1}}$ **mod** m and multiplies together those terms b^{2^j} **mod** m where $a_j = 1$, finding the remainder of the product when divided by m after each multiplication. Pseudocode for this algorithm is shown in Algorithm 5.

We illustrate how Algorithm 5 works in Example 11.

EXAMPLE 11 Use Algorithm 5 to find 2^{644} **mod** 645.

Solution: Algorithm 5 initially sets $x = 1$ and $power = 2$ **mod** $645 = 2$. In the computation of 2^{644} **mod** 645, this algorithm determines 2^{2^j} **mod** 645 for $j = 1, 2, \ldots, 9$ by successively squaring and reducing modulo 645. If $a_j = 1$ (where a_j is the bit in the jth position in the binary expansion of 644), it multiplies the current value of x by 2^{2^j} **mod** 645 and reduces the result modulo 645. Here are the steps used:

$i = 0$: Because $a_0 = 0$, we have $x = 1$ and $power = 2^2 = 4$ **mod** $645 = 4$;
$i = 1$: Because $a_1 = 0$, we have $x = 1$ and $power = 4^2 = 16$ **mod** $645 = 16$;
$i = 2$: Because $a_2 = 1$, we have $x = 1 \cdot 16$ **mod** $645 = 16$ and $power = 16^2 = 256$ **mod** $645 = 256$;
$i = 3$: Because $a_3 = 0$, we have $x = 16$ and $power = 256^2 = 65{,}536$ **mod** $645 = 391$;
$i = 4$: Because $a_4 = 0$, we have $x = 16$ and $power = 391^2 = 152{,}881$ **mod** $645 = 16$;
$i = 5$: Because $a_5 = 0$, we have $x = 16$ and $power = 16^2 = 256$ **mod** $645 = 256$;
$i = 6$: Because $a_6 = 0$, we have $x = 16$ and $power = 256^2 = 65{,}536$ **mod** $645 = 391$;
$i = 7$: Because $a_7 = 1$, we find that $x = (16 \cdot 391)$ **mod** $645 = 451$ and $power = 391^2 = 152{,}881$ **mod** $645 = 16$;
$i = 8$: Because $a_8 = 0$, we have $x = 451$ and $power = 16^2 = 256$ **mod** $645 = 256$;
$i = 9$: Because $a_9 = 1$, we find that $x = (451 \cdot 256)$ **mod** $645 = 1$.

This shows that following the steps of Algorithm 5 produces the result 2^{644} **mod** $645 = 1$. ◀

Algorithm 5 is quite efficient; it uses $O((\log m)^2 \log n)$ bit operations to find b^n **mod** m (see Exercise 54).

THE EUCLIDEAN ALGORITHM

Links

The method described in Section 2.4 for computing the greatest common divisor of two integers, using the prime factorizations of these integers, is inefficient. The reason is that it is time-consuming to find prime factorizations. We will give a more efficient method of finding the greatest common divisor, called the **Euclidean algorithm.** This algorithm has been known since ancient times. It is named after the ancient Greek mathematician Euclid, who included a description of this algorithm in his *Elements.*

Before describing the Euclidean algorithm, we will show how it is used to find gcd(91, 287). First, divide 287, the larger of the two integers, by 91, the smaller, to obtain

$$287 = 91 \cdot 3 + 14.$$

Any divisor of 91 and 287 must also be a divisor of $287 - 91 \cdot 3 = 14$. Also, any divisor of 91 and 14 must also be a divisor of $287 = 91 \cdot 3 + 14$. Hence, the greatest common divisor of 91 and 287 is the same as the greatest common divisor of 91 and 14. This means that the problem of finding gcd(91, 287) has been reduced to the problem of finding gcd(91, 14).

Next, divide 91 by 14 to obtain

$$91 = 14 \cdot 6 + 7.$$

Links

Since any common divisor of 91 and 14 also divides $91 - 14 \cdot 6 = 7$ and any common divisor of 14 and 7 divides 91, it follows that gcd(91, 14) = gcd(14, 7).

EUCLID (C. 325–265 B.C.E.) Euclid was the author of the most successful mathematics book ever written, the *Elements,* which appeared in over 1000 different editions from ancient to modern times. Little is known about Euclid's life, other than that he taught at the famous academy at Alexandria. Apparently, Euclid did not stress applications. When a student asked what he would get by learning geometry, Euclid explained that knowledge was worth acquiring for its own sake and told his servant to give the student a coin "since he must make a profit from what he learns."

Continue by dividing 14 by 7, to obtain

$$14 = 7 \cdot 2.$$

Since 7 divides 14, it follows that $\gcd(14, 7) = 7$. Furthermore, because $\gcd(287, 91) = \gcd(91, 14) = \gcd(14, 7) = 7$, the original problem has been solved.

We now describe how the Euclidean algorithm works in generality. We will use successive divisions to reduce the problem of finding the greatest common divisor of two positive integers to the same problem with smaller integers, until one of the integers is zero.

The Euclidean algorithm is based on the following result about greatest common divisors and the division algorithm.

LEMMA 1 Let $a = bq + r$, where a, b, q, and r are integers. Then $\gcd(a, b) = \gcd(b, r)$.

Proof: If we can show that the common divisors of a and b are the same as the common divisors of b and r, we will have shown that $\gcd(a, b) = \gcd(b, r)$, since both pairs must have the same *greatest* common divisor.

So suppose that d divides both a and b. Then it follows that d also divides $a - bq = r$ (from Theorem 1 of Section 2.4). Hence, any common divisor of a and b is also a common divisor of b and r.

Likewise, suppose that d divides both b and r. Then d also divides $bq + r = a$. Hence, any common divisor of b and r is also a common divisor of a and b.

Consequently, $\gcd(a, b) = \gcd(b, r)$. ◁

Suppose that a and b are positive integers with $a \geq b$. Let $r_0 = a$ and $r_1 = b$. When we successively apply the division algorithm, we obtain

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

Eventually a remainder of zero occurs in this sequence of successive divisions, since the sequence of remainders $a = r_0 > r_1 > r_2 > \cdots \geq 0$ cannot contain more than a terms. Furthermore, it follows from Lemma 1 that

$$\begin{aligned}
\gcd(a, b) &= \gcd(r_0, r_1) = \gcd(r_1, r_2) = \cdots = \gcd(r_{n-2}, r_{n-1}) \\
&= \gcd(r_{n-1}, r_n) = \gcd(r_n, 0) = r_n.
\end{aligned}$$

Hence, the greatest common divisor is the last nonzero remainder in the sequence of divisions.

EXAMPLE 12 Find the greatest common divisor of 414 and 662 using the Euclidean algorithm.

Solution: Successive uses of the division algorithm give:

$$662 = 414 \cdot 1 + 248$$
$$414 = 248 \cdot 1 + 166$$
$$248 = 166 \cdot 1 + 82$$
$$166 = 82 \cdot 2 + 2$$
$$82 = 2 \cdot 41.$$

Hence, $\gcd(414, 662) = 2$, since 2 is the last nonzero remainder. ◀

The Euclidean algorithm is expressed in pseudocode in Algorithm 6.

ALGORITHM 6 The Euclidean Algorithm.

procedure *gcd*(a, b: positive integers)
$x := a$
$y := b$
while $y \neq 0$
begin
    $r := x$ **mod** y
    $x := y$
    $y := r$
end {gcd(a, b) is x}

In Algorithm 6, the initial values of x and y are a and b, respectively. At each stage of the procedure, x is replaced by y, and y is replaced by x **mod** y, which is the remainder when x is divided by y. This process is repeated as long as $y \neq 0$. The algorithm terminates when $y = 0$, and the value of x at that point, the last nonzero remainder in the procedure, is the greatest common divisor of a and b.

We will study the time complexity of the Euclidean algorithm in Section 3.4, where we will show that the number of divisions required to find the greatest common divisor of a and b, where $a \geq b$, is $O(\log b)$.

Exercises

1. Convert these integers from decimal notation to binary notation.
   a) 231 **b)** 4532 **c)** 97644
2. Convert these integers from decimal notation to binary notation.
   a) 321 **b)** 1023 **c)** 100632
3. Convert these integers from binary notation to decimal notation.
   a) 1 1111 **b)** 10 0000 0001
   c) 1 0101 0101 **d)** 110 1001 0001 0000
4. Convert these integers from binary notation to decimal notation.
   a) 1 1011 **b)** 10 1011 0101
   c) 11 1011 1110 **d)** 111 1100 0001 1111
5. Convert these integers from hexadecimal notation to binary notation.
   a) 80E **b)** 135AB
   c) ABBA **d)** DEFACED
6. Convert $(\text{BADFACED})_{16}$ from its hexadecimal expansion to its binary expansion.

7. Convert $(\text{ABCDEF})_{16}$ from its hexadecimal expansion to its binary expansion.
8. Convert each of these integers from binary notation to hexadecimal notation.
   a) 1111 0111
   b) 1010 1010 1010
   c) 111 0111 0111 0111
9. Convert $(1011\ 0111\ 1011)_2$ from its binary expansion to its hexadecimal expansion.
10. Convert $(1\ 1000\ 0110\ 0011)_2$ from its binary expansion to its hexadecimal expansion.
11. Show that the hexadecimal expansion of a positive integer can be obtained from its binary expansion by grouping together blocks of four binary digits, adding initial digits if necessary, and translating each block of four binary digits into a single hexadecimal digit.
12. Show that the binary expansion of a positive integer can be obtained from its hexadecimal expansion by translating each hexadecimal digit into a block of four binary digits.
13. Give a simple procedure for converting from the binary expansion of an integer to its octal expansion.
14. Give a simple procedure for converting from the octal expansion of an integer to its binary expansion.
15. Convert $(7345321)_8$ to its binary expansion and $(10\ 1011\ 1011)_2$ to its octal expansion.
16. Give a procedure for converting from the hexadecimal expansion of an integer to its octal expansion using binary notation as an intermediate step.
17. Give a procedure for converting from the octal expansion of an integer to its hexadecimal expansion using binary notation as an intermediate step.
18. Convert $(12345670)_8$ to its hexadecimal expansion and $(\text{ABB093BABBA})_{16}$ to its octal expansion.
19. Use Algorithm 5 to find 3^{2003} **mod** 99.
20. Use Algorithm 5 to find 123^{1001} **mod** 101.
21. Use the Euclidean algorithm to find
    a) gcd(12, 18). **b)** gcd(111, 201).
    c) gcd(1001, 1331). **d)** gcd(12345, 54321).
    e) gcd(1000, 5040). **f)** gcd(9888, 6060).
22. Use the Euclidean algorithm to find
    a) gcd(1, 5). **b)** gcd(100, 101).
    c) gcd(123, 277). **d)** gcd(1529, 14039).
    e) gcd(1529, 14038). **f)** gcd(11111, 111111).
23. How many divisions are required to find gcd(21, 34) using the Euclidean algorithm?
24. How many divisions are required to find gcd(34, 55) using the Euclidean algorithm?
25. Show that every positive integer can be represented uniquely as the sum of distinct powers of 2. (*Hint:* Consider binary expansions of integers.)
26. It can be shown that every integer can be uniquely represented in the form

$$e_k 3^k + e_{k-1} 3^{k-1} + \cdots + e_1 3 + e_0,$$

where $e_j = -1, 0$, or 1 for $j = 0, 1, 2, \ldots, k$. Expansions of this type are called **balanced ternary expansions.** Find the balanced ternary expansions of
a) 5. **b)** 13. **c)** 37. **d)** 79.

27. Show that a positive integer is divisible by 3 if and only if the sum of its decimal digits is divisible by 3.
28. Show that a positive integer is divisible by 11 if and only if the difference of the sum of its decimal digits in even-numbered positions and the sum of its decimal digits in odd-numbered positions is divisible by 11.
29. Show that a positive integer is divisible by 3 if and only if the difference of the sum of its binary digits in even-numbered positions and the sum of its binary digits in odd-numbered positions is divisible by 3.

One's complement representations of integers are used to simplify computer arithmetic. To represent positive and negative integers with absolute value less than 2^{n-1}, a total of n bits is used. The leftmost bit is used to represent the sign. A 0 bit in this position is used for positive integers, and a 1 bit in this position is used for negative integers. For positive integers the remaining bits are identical to the binary expansion of the integer. For negative integers, the remaining bits are obtained by first finding the binary expansion of the absolute value of the integer, and then taking the complement of each of these bits, where the complement of a 1 is a 0 and the complement of a 0 is a 1.

30. Find the one's complement representations, using bit strings of length six, of the following integers.
    a) 22 **b)** 31 **c)** -7 **d)** -19
31. What integer does each of the following one's complement representations of length five represent?
    a) 11001 **b)** 01101 **c)** 10001 **d)** 11111
32. If m is a positive integer less than 2^{n-1}, how is the one's complement representation of $-m$ obtained from the one's complement of m, when bit strings of length n are used?
33. How is the one's complement representation of the sum of two integers obtained from the one's complement representations of these integers?
34. How is the one's complement representation of the difference of two integers obtained from the one's complement representations of these integers?
35. Show that the integer m with one's complement representation $(a_{n-1}a_{n-2}\cdots a_1a_0)$ can be found using the equation $m = -a_{n-1}(2^{n-1} - 1) + a_{n-2}2^{n-2} + \cdots + a_1 \cdot 2 + a_0$.

Two's complement representations of integers are also used to simplify computer arithmetic and are used more commonly than one's complement representations. To represent an integer x with $-2^{n-1} \leq x \leq 2^{n-1} - 1$ for a specified positive integer n, a total of n bits is used. The leftmost bit is used to represent the sign. A 0 bit

in this position is used for positive integers, and a 1 bit in this position is used for negative integers, just as in one's complement expansions. For a positive integer, the remaining bits are identical to the binary expansion of the integer. For a negative integer, the remaining bits are the bits of the binary expansion of $2^{n-1} - |x|$. Two's complement expansions of integers are often used by computers because addition and subtraction of integers can be performed easily using these expansions, where these integers can be either positive or negative.

36. Answer Exercise 30, but this time find the two's complement expansion using bit strings of length six.

37. Answer Exercise 31 if each expansion is a two's complement expansion of length five.

38. Answer Exercise 32 for two's complement expansions.

39. Answer Exercise 33 for two's complement expansions.

40. Answer Exercise 34 for two's complement expansions.

41. Show that the integer m with two's complement representation $(a_{n-1}a_{n-2}\cdots a_1a_0)$ can be found using the equation $m = -a_{n-1}\cdot 2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1\cdot 2 + a_0$.

42. Give a simple algorithm for forming the two's complement representation of an integer from its one's complement representation.

43. Sometimes integers are encoded by using four-digit binary expansions to represent each decimal digit. This produces the **binary coded decimal** form of the integer. For instance, 791 is encoded in this way by 011110010001. How many bits are required to represent a number with n decimal digits using this type of encoding?

A **Cantor expansion** is a sum of the form

$$a_n n! + a_{n-1}(n-1)! + \cdots + a_2 2! + a_1 1!,$$

where a_i is an integer with $0 \le a_i \le i$ for $i = 1, 2, \ldots, n$.

44. Find the Cantor expansions of

a) 2. **b)** 7. **c)** 19.
d) 87. **e)** 1000. **f)** 1,000,000.

***45.** Describe an algorithm that finds the Cantor expansion of an integer.

***46.** Describe an algorithm to add two integers from their Cantor expansions.

47. Add $(10111)_2$ and $(11010)_2$ by working through each step of the algorithm for addition given in the text.

48. Multiply $(1110)_2$ and $(1010)_2$ by working through each step of the algorithm for multiplication given in the text.

49. Describe an algorithm for finding the difference of two binary expansions.

50. Estimate the number of bit operations used to subtract two binary expansions.

51. Devise an algorithm that, given the binary expansions of the integers a and b, determines whether $a > b$, $a = b$, or $a < b$.

52. How many bit operations does the comparison algorithm from Exercise 51 use when the larger of a and b has n bits in its binary expansion?

53. Estimate the complexity of Algorithm 1 for finding the base b expansion of an integer n in terms of the number of divisions used.

***54.** Show that Algorithm 5 uses $O((\log m)^2 \log n)$ bit operations to find b^n **mod** m.

55. Show that Algorithm 4 uses $O(q \log |a|)$ bit operations, assuming that $a > d$.

2.6 Applications of Number Theory

INTRODUCTION

Number theory has many applications, especially to computer science. In Section 2.4 we described several of these applications, including hashing functions, the generation of pseudorandom numbers, and shift ciphers. This section continues our introduction to number theory, developing some key results and presenting two important applications: a method for performing arithmetic with large integers and a recently invented type of cryptosystem, called a *public key system*. In such a cryptosystem, we do not have to keep encryption keys secret, since knowledge of an encryption key does not help someone decrypt messages in a realistic amount of time. Privately held decryption keys are used to decrypt messages.

Before developing these applications, we will introduce some key results that play a central role in number theory and its applications. For example, we will show how to solve systems of linear congruences modulo pairwise relatively prime integers using the Chinese Remainder Theorem, and then show how to use this result as a basis for