4.2 Integer Representations and Algorithms

4.2.1 Introduction

Integers can be expressed using any integer greater than one as a base, as we will show in this section. Although we commonly use decimal (base 10), representations, binary (base 2), octal (base 8), and hexadecimal (base 16) representations are often used, especially in computer science. Given a base b and an integer n, we will show how to construct the base b representation of this integer. We will also explain how to quickly convert between binary and octal and between binary and hexadecimal notations.

As mentioned in Section 3.1, 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.

We will also introduce an algorithm for finding a **div** d and a **mod** d where a and d are integers with $d > 1$. Finally, we will describe an efficient algorithm for modular exponentiation, which is a particularly important algorithm for cryptography, as we will see in Section 4.6.

4.2.2 Representations of Integers

In everyday life we use decimal notation to express integers. In decimal notation, an integer n is written as a sum of the form $a_k 10^k + a_{k-1} 10^{k-1} + \cdots + a_1 10 + a_0$, where a_j is an integer with $0 \le a_j \le 9$ for $j = 0, 1, \ldots, k$. 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 integer greater than 1 as the base when expressing integers. This is stated in Theorem 1.

THEOREM 1 Let b be an 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 $a_k \neq 0$.

A proof of this theorem can be constructed using mathematical induction, a proof method that is discussed in Section 5.1. It can also be found in [Ro10]. 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 $(a_k a_{k-1} \ldots a_1 a_0)_b$. For instance, $(245)_8$ represents $2 \cdot 8^2 + 4 \cdot 8 + 5 = 165$. Typically, the subscript 10 is omitted for base 10 expansions of integers because base 10, or **decimal expansions**, are commonly used to represent integers.

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

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

OCTAL AND HEXADECIMAL EXPANSIONS Among the most important bases in computer science are base 2, base 8, and base 16. Base 8 expansions are called **octal** expansions and base 16 expansions are **hexadecimal** expansions.

EXAMPLE 2 What is the decimal expansion of the number with octal expansion $(7016)_8$?

Solution: Using the definition of a base b expansion with $b = 8$ tells us that

$$(7016)_8 = 7\cdot 8^3 + 0\cdot 8^2 + 1\cdot 8 + 6 = 3598.$$ ◀

Sixteen different digits are required for hexadecimal 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 3 What is the decimal expansion of the number with hexadecimal expansion $(2AE0B)_{16}$?

Solution: Using the definition of a base b expansion with $b = 16$ tells us that

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

Each hexadecimal digit can be represented using four bits. For instance, we see that $(1110\ \ 0101)_2 = (E5)_{16}$ because $(1110)_2 = (E)_{16}$ 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,

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

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

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

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. It produces the base b digits of n from the right to the left.

EXAMPLE 4 Find the 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

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

The successive remainders that we have found, 1, 7, 0, 0, and 3, are the digits from the right to the left of 12345 in base 8. Hence,

$$(12345)_{10} = (30071)_8.$$ ◀

EXAMPLE 5 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

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

The successive remainders that we have found, 10, 14, 3, 11, 2, give us the digits from the right to the left of 177130 in the hexadecimal (base 16) expansion of $(177130)_{10}$. It follows that

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

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

EXAMPLE 6 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

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

The successive remainders that we have found, 1, 0, 0, 0, 1, 1, 1, 1, are the digits from the right to the left in the binary (base 2) expansion of $(241)_{10}$. Hence,

$$(241)_{10} = (1111\ 0001)_2.$$ ◀

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

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

ALGORITHM 1 **Constructing Base b Expansions.**

```
procedure base b expansion($n, b$: positive integers with $b > 1$)
$q := n$
$k := 0$
while $q \neq 0$
    $a_k := q$ mod $b$
    $q := q$ div $b$
    $k := k + 1$
return $(a_{k-1}, \ldots, a_1, a_0)$ $\{(a_{k-1} \ldots a_1 a_0)_b$ is the base $b$ expansion of $n\}$
```

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, because the base b digits are taken as large as possible in each step.

CONVERSION BETWEEN BINARY, OCTAL, AND HEXADECIMAL EXPANSIONS

Conversion between binary and octal and between binary and hexadecimal expansions is extremely easy because each octal digit corresponds to a block of three binary digits and each hexadecimal digit corresponds to a block of four binary digits, with these correspondences shown in Table 1 without initial 0s shown. (We leave it as Exercises 13–16 to show that this is the case.) This conversion is illustrated in Example 7.

EXAMPLE 7 Find the octal and hexadecimal expansions of $(11\ 1110\ 1011\ 1100)_2$ and the binary expansions of $(765)_8$ and $(\mathrm{A8D})_{16}$.

Solution: To convert $(11\ 1110\ 1011\ 1100)_2$ into octal notation we group the binary digits into blocks of three, adding initial zeros at the start of the leftmost block if necessary. These blocks, from left to right, are 011, 111, 010, 111, and 100, corresponding to 3, 7, 2, 7, and 4, respectively. Consequently, $(11\ 1110\ 1011\ 1100)_2 = (37274)_8$. 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, from left to right, are 0011, 1110, 1011, and 1100, corresponding to the hexadecimal digits 3, E, B, and C, respectively. Consequently, $(11\ 1110\ 1011\ 1100)_2 = (\mathrm{3EBC})_{16}$.

To convert $(765)_8$ into binary notation, we replace each octal digit by a block of three binary digits. These blocks are 111, 110, and 101. Hence, $(765)_8 = (1\ 1111\ 0101)_2$. To convert $(\mathrm{A8D})_{16}$ into binary notation, we replace each hexadecimal digit by a block of four binary digits. These blocks are 1010, 1000, and 1101. Hence, $(\mathrm{A8D})_{16} = (1010\ 1000\ 1101)_2$. ◀

4.2.3 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

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

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.

ADDITION ALGORITHM 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, $a + b = (s_ns_{n-1}s_{n-2} \dots s_1s_0)_2$.

EXAMPLE 8 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, because

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

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, because

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

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, where carries are shown in color. ◀

$$\begin{array}{r} \mathit{1\ \ 1\ 1\ \ \ } \\ 1\ 1\ 1\ 0 \\ +\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 1 \end{array}$$

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 $(a_{n-1}a_{n-2} \ldots a_1a_0)_2$
  and $(b_{n-1}b_{n-2} \ldots b_1b_0)_2$, respectively}
$c := 0$
for $j := 0$ to $n - 1$
    $d := \lfloor (a_j + b_j + c)/2 \rfloor$
    $s_j := a_j + b_j + c - 2d$
    $c := d$
$s_n := c$
return $(s_0, s_1, \ldots, s_n)$ {the binary expansion of the sum is $(s_n s_{n-1} \ldots s_0)_2$}
```

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

EXAMPLE 9 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 two additions of bits. Thus, the total number of additions of bits used is less than twice 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 $O(n)$. ◀

MULTIPLICATION ALGORITHM 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 $ab_j 2^j$, $j = 0, 1, 2, \ldots, n - 1$.

Algorithm 3 displays this procedure for multiplication.

ALGORITHM 3 Multiplication of Integers.

```
procedure multiply($a, b$: positive integers)
{the binary expansions of $a$ and $b$ are $(a_{n-1}a_{n-2} \dots a_1a_0)_2$
  and $(b_{n-1}b_{n-2} \dots b_1b_0)_2$, respectively}
for $j := 0$ to $n - 1$
    if $b_j = 1$ then $c_j := a$ shifted $j$ places
    else $c_j := 0$
{$c_0, c_1, \dots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ to $n - 1$
    $p := add(p, c_j)$
return $p$ {$p$ is the value of $ab$}
```

Example 10 illustrates the use of this algorithm.

EXAMPLE 10 Find the product of $a = (110)_2$ and $b = (101)_2$.

Solution: First note that

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

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

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. ◀

$$\begin{array}{r} 1\,1\,0 \\ \times\ 1\,0\,1 \\ \hline 1\,1\,0 \\ 0\,0\,0 \\ 1\,1\,0 \\ \hline 1\,1\,1\,1\,0 \end{array}$$

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

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

EXAMPLE 11 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 $c_0, c_1, c_2, \dots,$ and c_{n-1}. When $b_j = 1$, we compute the partial product c_j by shifting the binary expansion of a by j bits. When $b_j = 0$, no shifts are required because $c_j = 0$. Hence, to find all n of the integers $ab_j2^j, j = 0, 1, \dots, n - 1$, requires at most

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

shifts. Hence, by Example 5 in Section 3.2 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, $\dots$, and a $(2n)$-bit integer. We know from Example 9 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 8.3.

ALGORITHM FOR div AND mod Given integers a and d, $d > 0$, we can find $q = a$ **div** d and $r = a$ **mod** d using Algorithm 4. In this brute-force 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. This algorithm finds the quotient q and remainder r when $|a|$ is divided by d. Then, when $a < 0$ and $r > 0$, it 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 65) to show that, assuming that $a > d$, this algorithm uses $O(q \log a)$ bit operations.

ALGORITHM 4 Computing div and mod.

```
procedure division algorithm(a: integer, d: positive integer)
q := 0
r := |a|
while r ≥ d
    r := r − d
    q := q + 1
if a < 0 and r > 0 then
    r := d − r
    q := −(q + 1)
return (q, r) {q = a div d is the quotient, r = a mod d is the remainder}
```

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.

4.2.4 Modular Exponentiation

part of Lecture 14

In cryptography it is important to be able to find b^n **mod** m efficiently without using an excessive amount of memory, 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 can be a huge number and we will need a huge amount of computer memory to store such numbers. Instead, we can avoid time and memory problems by using an algorithm that employs the binary expansion of the exponent n.

Before we present an algorithm for fast modular exponentiation based on the binary expansion of the exponent, first observe that we can avoid using large amount of memory if we compute b^n **mod** m by successively computing b^k **mod** m for $k = 1, 2, \ldots, n$ using the fact that b^{k+1} **mod** $m = b(b^k$ **mod** $m)$ **mod** m (by Corollary 2 of Theorem 5 of Section 4.1). (Recall that $1 \le b < m$.) However, this approach is impractical because it requires $n - 1$ multiplications of integers and n might be huge.

To motivate the fast modular exponentiation algorithm, we illustrate its basic idea. We will explain how to use the binary expansion of n, say $n = (a_{k-1} \ldots a_1 a_0)_2$, to compute b^n. First, note that

$$b^n = b^{a_{k-1}\cdot 2^{k-1} + \cdots + a_1 \cdot 2 + a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1 \cdot 2} \cdot b^{a_0}.$$

If you are rusty with the laws for exponents, this is the time to review them! See Theorem 1 in Appendix 2.

This shows that to compute b^n, we need only compute the values of b, b^2, $(b^2)^2 = b^4$, $(b^4)^2 = b^8, \ldots, b^{2^k}$. Once we have these values, we multiply the terms b^{2^j} in this list, where $a_j = 1$. (For efficiency and to reduce space requirements, after multiplying by each term, we reduce the result modulo m.)

Note that $(b^{2^n})^2 = b^{2^{n+1}}$ when n is a nonnegative integer.

This gives us b^n. For example, to compute 3^{11} we first note that $11 = (1011)_2$, so that $3^{11} = 3^8 3^2 3^1$. By successively squaring, we find that $3^2 = 9$, $3^4 = 9^2 = 81$, and $3^8 = (81)^2 = 6561$. Consequently, $3^{11} = 3^8 3^2 3^1 = 6561 \cdot 9 \cdot 3 = 177{,}147$.

Be sure to reduce modulo m after each multiplication!

The algorithm successively finds b **mod** m, b^2 **mod** m, b^4 **mod** $m, \ldots, b^{2^{k-1}}$ **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. Note that we need only perform $O(\log_2(n))$ multiplications. Pseudocode for this algorithm is shown in Algorithm 5. Note that in Algorithm 5 we can use the most efficient algorithm available to compute values of the **mod** function, not necessarily Algorithm 4.

ALGORITHM 5 Fast Modular Exponentiation.

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

We illustrate how Algorithm 5 works in Example 12.

EXAMPLE 12 Use Algorithm 5 to find 3^{644} **mod** 645.

Solution: Algorithm 5 initially sets $x = 1$ and $power = 3$ **mod** $645 = 3$. In the computation of 3^{644} **mod** 645, this algorithm determines 3^{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, which is $(1010000100)_2$), it multiplies the current value of x by 3^{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 $power = 3^2$ **mod** $645 = 9$ **mod** $645 = 9$;
$i = 1$: Because $a_1 = 0$, we have $x = 1$ and $power = 9^2$ **mod** $645 = 81$ **mod** $645 = 81$;
$i = 2$: Because $a_2 = 1$, we have $x = 1 \cdot 81$ **mod** $645 = 81$ and $power = 81^2$ **mod** $645 = 6561$ **mod** $645 = 111$;
$i = 3$: Because $a_3 = 0$, we have $x = 81$ and $power = 111^2$ **mod** $645 = 12{,}321$ **mod** $645 = 66$;
$i = 4$: Because $a_4 = 0$, we have $x = 81$ and $power = 66^2$ **mod** $645 = 4356$ **mod** $645 = 486$;
$i = 5$: Because $a_5 = 0$, we have $x = 81$ and $power = 486^2$ **mod** $645 = 236{,}196$ **mod** $645 = 126$;
$i = 6$: Because $a_6 = 0$, we have $x = 81$ and $power = 126^2$ **mod** $645 = 15{,}876$ **mod** $645 = 396$;
$i = 7$: Because $a_7 = 1$, we find that $x = (81 \cdot 396)$ **mod** $645 = 471$ and $power = 396^2$ **mod** $645 = 156{,}816$ **mod** $645 = 81$;
$i = 8$: Because $a_8 = 0$, we have $x = 471$ and $power = 81^2$ **mod** $645 = 6561$ **mod** $645 = 111$;
$i = 9$: Because $a_9 = 1$, we find that $x = (471 \cdot 111)$ **mod** $645 = 36$.

This shows that following the steps of Algorithm 5 produces the result 3^{644} **mod** $645 = 36$. ◀

Algorithm 5 is quite efficient; it uses $O((\log m)^2 \log n)$ bit operations to find b^n **mod** m (see Exercise 64).

Exercises

1. Convert the decimal expansion of each of these integers to a binary expansion.

a) 231 **b)** 4532 **c)** 97644

2. Convert the decimal expansion of each of these integers to a binary expansion.

a) 321 **b)** 1023 **c)** 100632

3. Convert the binary expansion of each of these integers to a decimal expansion.

a) $(1\ 1111)_2$ **b)** $(10\ 0000\ 0001)_2$
c) $(1\ 0101\ 0101)_2$ **d)** $(110\ 1001\ 0001\ 0000)_2$

4. Convert the binary expansion of each of these integers to a decimal expansion.

a) $(1\ 1011)_2$ **b)** $(10\ 1011\ 0101)_2$
c) $(11\ 1011\ 1110)_2$ **d)** $(111\ 1100\ 0001\ 1111)_2$

5. Convert the octal expansion of each of these integers to a binary expansion.

a) $(572)_8$ **b)** $(1604)_8$
c) $(423)_8$ **d)** $(2417)_8$

6. Convert the binary expansion of each of these integers to an octal expansion.

a) $(1111\ 0111)_2$
b) $(1010\ 1010\ 1010)_2$
c) $(111\ 0111\ 0111\ 0111)_2$
d) $(101\ 0101\ 0101\ 0101)_2$

7. Convert the hexadecimal expansion of each of these integers to a binary expansion.

a) $(80E)_{16}$ **b)** $(135AB)_{16}$
c) $(ABBA)_{16}$ **d)** $(DEFACED)_{16}$

8. Convert $(BADFACED)_{16}$ from its hexadecimal expansion to its binary expansion.

9. Convert $(ABCDEF)_{16}$ from its hexadecimal expansion to its binary expansion.

10. Convert each of the integers in Exercise 6 from a binary expansion to a hexadecimal expansion.

11. Convert $(1011\ 0111\ 1011)_2$ from its binary expansion to its hexadecimal expansion.

12. Convert $(1\ 1000\ 0110\ 0011)_2$ from its binary expansion to its hexadecimal expansion.

13. 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 zeros if necessary, and translating each block of four binary digits into a single hexadecimal digit.

14. 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.

15. Show that the octal expansion of a positive integer can be obtained from its binary expansion by grouping together blocks of three binary digits, adding initial zeros if necessary, and translating each block of three binary digits into a single octal digit.

16. Show that the binary expansion of a positive integer can be obtained from its octal expansion by translating each octal digit into a block of three binary digits.

17. Convert $(7345321)_8$ to its binary expansion and $(10\ 1011\ 1011)_2$ to its octal expansion.

18. Give a procedure for converting from the hexadecimal expansion of an integer to its octal expansion using binary notation as an intermediate step.

19. Give a procedure for converting from the octal expansion of an integer to its hexadecimal expansion using binary notation as an intermediate step.

20. Explain how to convert from binary to base 64 expansions and from base 64 expansions to binary expansions and from octal to base 64 expansions and from base 64 expansions to octal expansions.

21. Find the sum and the product of each of these pairs of numbers. Express your answers as a binary expansion.

a) $(100\ 0111)_2$, $(111\ 0111)_2$
b) $(1110\ 1111)_2$, $(1011\ 1101)_2$
c) $(10\ 1010\ 1010)_2$, $(1\ 1111\ 0000)_2$
d) $(10\ 0000\ 0001)_2$, $(11\ 1111\ 1111)_2$

22. Find the sum and product of each of these pairs of numbers. Express your answers as a base 3 expansion.

a) $(112)_3$, $(210)_3$
b) $(2112)_3$, $(12021)_3$
c) $(20001)_3$, $(1111)_3$
d) $(120021)_3$, $(2002)_3$

23. Find the sum and product of each of these pairs of numbers. Express your answers as an octal expansion.

a) $(763)_8$, $(147)_8$
b) $(6001)_8$, $(272)_8$
c) $(1111)_8$, $(777)_8$
d) $(54321)_8$, $(3456)_8$

24. Find the sum and product of each of these pairs of numbers. Express your answers as a hexadecimal expansion.

a) $(1AE)_{16}$, $(BBC)_{16}$
b) $(20CBA)_{16}$, $(A01)_{16}$
c) $(ABCDE)_{16}$, $(1111)_{16}$
d) $(E0000E)_{16}$, $(BAAA)_{16}$

25. Use Algorithm 5 to find 7^{644} **mod** 645.

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

27. Use Algorithm 5 to find 3^{2003} **mod** 99.

28. Use Algorithm 5 to find 123^{1001} **mod** 101.

29. Show that every positive integer can be represented uniquely as the sum of distinct powers of 2. [*Hint:* Consider binary expansions of integers.]

30. 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.

31. Show that a positive integer is divisible by 3 if and only if the sum of its decimal digits is divisible by 3.

32. 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.

33. 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.

34. Determine how we can use the decimal expansion of an integer n to determine whether n is divisible by

a) 2 **b)** 5 **c)** 10

35. Determine how we can use the decimal expansion of an integer n to determine whether n is divisible by

a) 4 **b)** 25 **c)** 20

36. Suppose that n and b are positive integers with $b \geq 2$ and the base b expansion of n is $n = (a_m a_{m-1} \dots a_1 a_0)_b$. Find the base b expansion of

a) bn. **b)** $b^2 n$;
c) $\lfloor n/b \rfloor$, **d)** $\lfloor n/b^2 \rfloor$.

37. Prove that if n and b are positive integers with $b \geq 2$ the base b representation of n has $\lfloor \log_b n \rfloor + 1$ digits.

38. Find the decimal expansion of the number with the n-digit base seven expansion $(111 \dots 111)_7$ (with n 1's). [*Hint*: Use the formula for the sum of the terms of a geometric progression.]

39. Find the decimal expansion of the number with the $3n$ bit binary expansion $(101101....101101)_2$ (so that the binary expansion is made of n copies of 101). [*Hint*: Use the formula for the sum of the terms of a geometric progression.]

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.

40. Find the one's complement representations, using bit strings of length six, of the following integers.

a) 22 **b)** 31 **c)** -7 **d)** -19

41. What integer does each of the following one's complement representations of length five represent?

a) 11001 **b)** 01101
c) 10001 **d)** 11111

42. 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?

43. How is the one's complement representation of the sum of two integers obtained from the one's complement representations of these integers?

44. How is the one's complement representation of the difference of two integers obtained from the one's complement representations of these integers?

45. Show that the integer m with one's complement representation $(a_{n-1}a_{n-2} \dots a_1 a_0)$ 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.

46. Answer Exercise 40, but this time find the two's complement expansion using bit strings of length six.

47. Answer Exercise 41 if each expansion is a two's complement expansion of length five.

48. Answer Exercise 42 for two's complement expansions.

49. Answer Exercise 43 for two's complement expansions.

50. Answer Exercise 44 for two's complement expansions.

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

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

53. 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 $0 \leq a_i \leq i$ for $i = 1, 2, \dots, n$.

54. Find the Cantor expansions of

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

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

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

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

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

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

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

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

62. How many bit operations does the comparison algorithm from Exercise 61 use when the larger of a and b has n bits in its binary expansion?

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

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

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

4.3 Primes and Greatest Common Divisors

4.3.1 Introduction

In Section 4.1 we studied the concept of divisibility of integers. One important concept based on divisibility is that of a prime number. A prime is an integer greater than 1 that is divisible by no positive integers other than 1 and itself. The study of prime numbers goes back to ancient times. Thousands of years ago it was known that there are infinitely many primes; the proof of this fact, found in the works of Euclid, is famous for its elegance and beauty.

We will discuss the distribution of primes among the integers. We will describe some of the results about primes found by mathematicians in the last 400 years. In particular, we will introduce an important theorem, the fundamental theorem of arithmetic. This theorem, which asserts that every positive integer can be written uniquely as the product of primes in nondecreasing order, has many interesting consequences. We will also discuss some of the many old conjectures about primes that remain unsettled today.

Primes have become essential in modern cryptographic systems, and we will develop some of their properties important in cryptography. For example, finding large primes is essential in modern cryptography. The length of time required to factor large integers into their prime factors is the basis for the strength of some important modern cryptographic systems.

In this section we will also study the greatest common divisor of two integers, as well as the least common multiple of two integers. We will develop an important algorithm for computing greatest common divisors, called the Euclidean algorithm.

4.3.2 Primes

Every integer greater than 1 is divisible by at least two integers, because a positive integer is divisible by 1 and by itself. Positive integers that have exactly two different positive integer factors are called **primes**.

Definition 1 An integer p greater than 1 is called *prime* if the only positive factors of p are 1 and p. A positive integer that is greater than 1 and is not prime is called *composite*.

Remark: The integer 1 is not prime, because it has only one positive factor. Note also that an integer n is composite if and only if there exists an integer a such that $a \mid n$ and $1 < a < n$.