31 Number-Theoretic Algorithms

Number theory was once viewed as a beautiful but largely useless subject in pure mathematics. Today number-theoretic algorithms are used widely, due in large part to the invention of cryptographic schemes based on large prime numbers. These schemes are feasible because we can find large primes quickly, and they are secure because we do not know how to factor the product of large primes (or solve related problems, such as computing discrete logarithms) efficiently. This chapter presents some of the number theory and related algorithms that underlie such applications.

We start in Section 31.1 by introducing basic concepts of number theory, such as divisibility, modular equivalence, and unique prime factorization. Section 31.2 studies one of the world's oldest algorithms: Euclid's algorithm for computing the greatest common divisor of two integers, and Section 31.3 reviews concepts of modular arithmetic. Section 31.4 then explores the set of multiples of a given number a, modulo n, and shows how to find all solutions to the equation $ax = b \pmod{n}$ by using Euclid's algorithm. The Chinese remainder theorem is presented in Section 31.5. Section 31.6 considers powers of a given number a, modulo n, and presents a repeated-squaring algorithm for efficiently computing $a^b \bmod n$, given a, b, and n. This operation is at the heart of efficient primality testing and of much modern cryptography, such as the RSA public-key cryptosystem described in Section 31.7. We wrap up in Section 31.8, which examines a randomized primality test. This test finds large primes efficiently, an essential step in creating keys for the RSA cryptosystem.

Size of inputs and cost of arithmetic computations

Because we'll be working with large integers, we need to adjust how to think about the size of an input and about the cost of elementary arithmetic operations.

In this chapter, a "large input" typically means an input containing "large integers" rather than an input containing "many integers" (as for sorting). Thus, the size of an input depends on the *number of bits* required to represent that input, not just the number of integers in the input. An algorithm with integer in-

puts $a_1, a_2, \ldots, a_k$ is a ***polynomial-time algorithm*** if it runs in time polynomial in $\lg a_1, \lg a_2, \ldots, \lg a_k$, that is, polynomial in the lengths of its binary-encoded inputs.

Most of this book considers the elementary arithmetic operations (multiplications, divisions, or computing remainders) as primitive operations that take one unit of time. Counting the number of such arithmetic operations that an algorithm performs provides a basis for making a reasonable estimate of the algorithm's actual running time on a computer. Elementary operations can be time-consuming, however, when their inputs are large. It thus becomes appropriate to measure how many ***bit operations*** a number-theoretic algorithm requires. In this model, multiplying two β-bit integers by the ordinary method uses $\Theta(\beta^2)$ bit operations. Similarly, dividing a β-bit integer by a shorter integer or taking the remainder of a β-bit integer when divided by a shorter integer requires $\Theta(\beta^2)$ time by simple algorithms. (See Exercise 31.1-12.) Faster methods are known. For example, a simple divide-and-conquer method for multiplying two β-bit integers has a running time of $\Theta(\beta^{\lg 3})$, and $O(\beta \lg \beta \lg\lg \beta)$ time is possible. For practical purposes, however, the $\Theta(\beta^2)$ algorithm is often best, and we use this bound as a basis for our analyses. In this chapter, we'll usually analyze algorithms in terms of both the number of arithmetic operations and the number of bit operations they require.

31.1 Elementary number-theoretic notions

This section provides a brief review of notions from elementary number theory concerning the set $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ of integers and the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ of natural numbers.

Divisibility and divisors

The notion of one integer being divisible by another is key to the theory of numbers. The notation $d \mid a$ (read "d ***divides*** a") means that $a = kd$ for some integer k. Every integer divides 0. If $a > 0$ and $d \mid a$, then $|d| \le |a|$. If $d \mid a$, then we also say that a is a ***multiple*** of d. If d does not divide a, we write $d \nmid a$.

If $d \mid a$ and $d \ge 0$, then d is a ***divisor*** of a. Since $d \mid a$ if and only if $-d \mid a$, without loss of generality, we define the divisors of a to be nonnegative, with the understanding that the negative of any divisor of a also divides a. A divisor of a nonzero integer a is at least 1 but not greater than $|a|$. For example, the divisors of 24 are 1, 2, 3, 4, 6, 8, 12, and 24.

Every positive integer a is divisible by the ***trivial divisors*** 1 and a. The nontrivial divisors of a are the ***factors*** of a. For example, the factors of 20 are 2, 4, 5, and 10.

Prime and composite numbers

An integer $a > 1$ whose only divisors are the trivial divisors 1 and a is a ***prime number*** or, more simply, a ***prime***. Primes have many special properties and play a critical role in number theory. The first 20 primes, in order, are

2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71.

Exercise 31.1-2 asks you to prove that there are infinitely many primes. An integer $a > 1$ that is not prime is a ***composite number*** or, more simply, a ***composite***. For example, 39 is composite because $3 \mid 39$. We call the integer 1 a ***unit***, and it is neither prime nor composite. Similarly, the integer 0 and all negative integers are neither prime nor composite.

The division theorem, remainders, and modular equivalence

Given an integer n, we can partition the integers into those that are multiples of n and those that are not multiples of n. Much number theory is based upon refining this partition by classifying the integers that are not multiples of n according to their remainders when divided by n. The following theorem provides the basis for this refinement. We omit the proof (but see, for example, Niven and Zuckerman [345]).

Theorem 31.1 (Division theorem)
For any integer a and any positive integer n, there exist unique integers q and r such that $0 \le r < n$ and $a = qn + r$. ■

The value $q = \lfloor a/n \rfloor$ is the ***quotient*** of the division. The value $r = a \bmod n$ is the ***remainder*** (or ***residue***) of the division, so that $n \mid a$ if and only if $a \bmod n = 0$.

The integers partition into n equivalence classes according to their remainders modulo n. The ***equivalence class modulo n*** containing an integer a is

$$[a]_n = \{a + kn : k \in \mathbb{Z}\} \ .$$

For example, $[3]_7 = \{\ldots, -11, -4, 3, 10, 17, \ldots\}$, and $[-4]_7$ and $[10]_7$ also denote this set. With the notation defined on page 64, writing $a \in [b]_n$ is the same as writing $a = b \pmod{n}$. The set of all such equivalence classes is

$$\mathbb{Z}_n = \{[a]_n : 0 \le a \le n - 1\} \ . \tag{31.1}$$

When you see the definition

$$\mathbb{Z}_n = \{0, 1, \ldots, n - 1\} \ , \tag{31.2}$$

you should read it as equivalent to equation (31.1) with the understanding that 0 represents $[0]_n$, 1 represents $[1]_n$, and so on. Each class is represented by its

smallest nonnegative element. You should keep the underlying equivalence classes in mind, however. For example, if we refer to -1 as a member of $\mathbb{Z}_n$, we are really referring to $[n-1]_n$, since $-1 = n - 1 \pmod{n}$.

Common divisors and greatest common divisors

If d is a divisor of a and d is also a divisor of b, then d is a ***common divisor*** of a and b. For example, the divisors of 30 are 1, 2, 3, 5, 6, 10, 15, and 30, and so the common divisors of 24 and 30 are 1, 2, 3, and 6. Any pair of integers has a common divisor of 1.

An important property of common divisors is that

$$\text{if } d \mid a \text{ and } d \mid b, \text{ then } d \mid (a+b) \text{ and } d \mid (a-b)\ . \tag{31.3}$$

More generally, for any integers x and y,

$$\text{if } d \mid a \text{ and } d \mid b, \text{ then } d \mid (ax+by)\ . \tag{31.4}$$

Also, if $a \mid b$, then either $|a| \le |b|$ or $b = 0$, which implies that

$$\text{if } a \mid b \text{ and } b \mid a, \text{ then } a = \pm b\ . \tag{31.5}$$

The ***greatest common divisor*** of two integers a and b which are not both 0, denoted by $\gcd(a,b)$, is the largest of the common divisors of a and b. For example, $\gcd(24,30) = 6$, $\gcd(5,7) = 1$, and $\gcd(0,9) = 9$. If a and b are both nonzero, then $\gcd(a,b)$ is an integer between 1 and $\min\{|a|,|b|\}$. We define $\gcd(0,0)$ to be 0, so that standard properties of the gcd function (such as equation (31.9) below) hold universally.

Exercise 31.1-9 asks you to prove the following elementary properties of the gcd function:

$$\gcd(a,b) = \gcd(b,a)\ , \tag{31.6}$$

$$\gcd(a,b) = \gcd(-a,b)\ , \tag{31.7}$$

$$\gcd(a,b) = \gcd(|a|,|b|)\ , \tag{31.8}$$

$$\gcd(a,0) = |a|\ , \tag{31.9}$$

$$\gcd(a,ka) = |a| \qquad \text{for any } k \in \mathbb{Z}\ . \tag{31.10}$$

The following theorem provides an alternative and useful way to characterize $\gcd(a,b)$.

Theorem 31.2

If a and b are any integers, not both zero, then $\gcd(a,b)$ is the smallest positive element of the set $\{ax + by : x, y \in \mathbb{Z}\}$ of linear combinations of a and b.

Proof Let s be the smallest positive such linear combination of a and b, and let $s = ax + by$ for some $x, y \in \mathbb{Z}$. Let $q = \lfloor a/s \rfloor$. Equation (3.11) on page 64 then implies

$$\begin{aligned} a \bmod s &= a - qs \\ &= a - q(ax + by) \\ &= a\,(1 - qx) + b\,(-qy), \end{aligned}$$

so that $a \bmod s$ is a linear combination of a and b as well. Because s is the smallest *positive* such linear combination and $0 \le a \bmod s < s$ (inequality (3.12) on page 64), $a \bmod s$ cannot be positive. Hence, $a \bmod s = 0$. Therefore, we have that $s \mid a$ and, by analogous reasoning, $s \mid b$. Thus, s is a common divisor of a and b, so that $\gcd(a, b) \ge s$. By definition, $\gcd(a, b)$ divides both a and b, and s is defined as a linear combination of a and b. Equation (31.4) therefore implies that $\gcd(a, b) \mid s$. But $\gcd(a, b) \mid s$ and $s > 0$ imply that $\gcd(a, b) \le s$. Combining $\gcd(a, b) \ge s$ and $\gcd(a, b) \le s$ yields $\gcd(a, b) = s$. We conclude that s, the smallest positive linear combination of a and b, is also their greatest common divisor. ■

Theorem 31.2 engenders three useful corollaries.

Corollary 31.3
For any integers a and b, if $d \mid a$ and $d \mid b$, then $d \mid \gcd(a, b)$.

Proof This corollary follows from equation (31.4) and Theorem 31.2, because $\gcd(a, b)$ is a linear combination of a and b, ■

Corollary 31.4
For all integers a and b and any nonnegative integer n, we have

$$\gcd(an, bn) = n \gcd(a, b)\ .$$

Proof If $n = 0$, the corollary is trivial. If $n > 0$, then $\gcd(an, bn)$ is the smallest positive element of the set $\{anx + bny : x, y \in \mathbb{Z}\}$, which in turn is n times the smallest positive element of the set $\{ax + by : x, y \in \mathbb{Z}\}$. ■

Corollary 31.5
For all positive integers n, a, and b, if $n \mid ab$ and $\gcd(a, n) = 1$, then $n \mid b$.

Proof Exercise 31.1-5 asks you to provide the proof. ■

Relatively prime integers

Two integers a and b are ***relatively prime*** if their only common divisor is 1, that is, if $\gcd(a, b) = 1$. For example, 8 and 15 are relatively prime, since the divisors of 8 are 1, 2, 4, and 8, and the divisors of 15 are 1, 3, 5, and 15. The following theorem states that if two integers are each relatively prime to an integer p, then their product is relatively prime to p.

Theorem 31.6
For any integers a, b, and p, we have $\gcd(ab, p) = 1$ if and only if $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$ both hold.

Proof If $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$, then it follows from Theorem 31.2 that there exist integers x, y, x', and y' such that

$$ax + py = 1 ,$$
$$bx' + py' = 1 .$$

Multiplying these equations and rearranging gives

$$ab(xx') + p(ybx' + y'ax + pyy') = 1 .$$

Since 1 is thus a positive linear combination of ab and p, it is the smallest positive linear combination. Applying Theorem 31.2 implies $\gcd(ab, p) = 1$, completing the proof in this direction.

Conversely, if $\gcd(ab, p) = 1$, then Theorem 31.2 implies that there exist integers x and y such that

$$abx + py = 1 .$$

Writing abx as $a(bx)$ and applying Theorem 31.2 again proves that $\gcd(a, p) = 1$. Proving that $\gcd(b, p) = 1$ is similar. ■

Integers $n_1, n_2, \ldots, n_k$ are ***pairwise relatively prime*** if $\gcd(n_i, n_j) = 1$ for $1 \le i < j \le k$.

Unique prime factorization

An elementary but important fact about divisibility by primes is the following.

Theorem 31.7
For all primes p and all integers a and b, if $p \mid ab$, then $p \mid a$ or $p \mid b$ (or both).

Proof Assume for the purpose of contradiction that $p \mid ab$, but that $p \nmid a$ and $p \nmid b$. Because $p > 1$ and $ab = kp$ for some $k \in \mathbb{Z}$, equation (31.10) gives

that $\gcd(ab, p) = p$. We also have that $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$, since the only divisors of p are 1 and p, and we assumed that p divides neither a nor b. Theorem 31.6 then implies that $\gcd(ab, p) = 1$, contradicting $\gcd(ab, p) = p$. This contradiction completes the proof. ■

A consequence of Theorem 31.7 is that any composite integer can be uniquely factored into a product of primes. Exercise 31.1-11 asks you to provide a proof.

Theorem 31.8 (Unique prime factorization)
There is exactly one way to write any composite integer a as a product of the form

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r} ,$$

where the p_i are prime, $p_1 < p_2 < \cdots < p_r$, and the e_i are positive integers. ■

As an example, the unique prime factorization of the number 6000 is $2^4 \cdot 3^1 \cdot 5^3$.

Exercises

31.1-1
Prove that if $a > b > 0$ and $c = a + b$, then $c \bmod a = b$.

31.1-2
Prove that there are infinitely many primes. (*Hint:* Show that none of the primes $p_1, p_2, \ldots, p_k$ divide $(p_1 p_2 \cdots p_k) + 1$.)

31.1-3
Prove that if $a \mid b$ and $b \mid c$, then $a \mid c$.

31.1-4
Prove that if p is prime and $0 < k < p$, then $\gcd(k, p) = 1$.

31.1-5
Prove Corollary 31.5.

31.1-6
Prove that if p is prime and $0 < k < p$, then $p \mid \binom{p}{k}$. Conclude that for all integers a and b and all primes p,

$$(a + b)^p = a^p + b^p \pmod{p} .$$

31.1-7
Prove that if a and b are any positive integers such that $a \mid b$, then

$(x \bmod b) \bmod a = x \bmod a$

for any x. Prove, under the same assumptions, that

$x = y \pmod b$ implies $x = y \pmod a$

for any integers x and y.

31.1-8
For any integer $k > 0$, an integer n is a ***kth power*** if there exists an integer a such that $a^k = n$. Furthermore, $n > 1$ is a ***nontrivial power*** if it is a kth power for some integer $k > 1$. Show how to determine whether a given β-bit integer n is a nontrivial power in time polynomial in β.

31.1-9
Prove equations (31.6)–(31.10).

31.1-10
Show that the gcd operator is associative. That is, prove that for all integers a, b, and c, we have

$$\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c) .$$

★ ***31.1-11***
Prove Theorem 31.8.

31.1-12
Give efficient algorithms for the operations of dividing a β-bit integer by a shorter integer and of taking the remainder of a β-bit integer when divided by a shorter integer. Your algorithms should run in $\Theta(\beta^2)$ time.

31.1-13
Give an efficient algorithm to convert a given β-bit (binary) integer to a decimal representation. Argue that if multiplication or division of integers whose length is at most β takes $M(\beta)$ time, where $M(\beta) = \Omega(\beta)$, then you can convert binary to decimal in $O(M(\beta) \lg \beta)$ time. (*Hint:* Use a divide-and-conquer approach, obtaining the top and bottom halves of the result with separate recursions.)

31.1-14
Professor Marshall sets up n lightbulbs in a row. The lightbulbs all have switches, so that if he presses a bulb, it toggles on if it was off and off if it was on. The lightbulbs all start off. For $i = 1, 2, 3, \ldots, n$, the professor presses bulb $i, 2i, 3i, \ldots$. After the last press, which lightbulbs are on? Prove your answer.

31.2 Greatest common divisor

In this section, we describe Euclid's algorithm for efficiently computing the greatest common divisor of two integers. When we analyze the running time, we'll see a surprising connection with the Fibonacci numbers, which yield a worst-case input for Euclid's algorithm.

We restrict ourselves in this section to nonnegative integers. This restriction is justified by equation (31.8), which states that $\gcd(a, b) = \gcd(|a|, |b|)$.

In principle, for positive integers a and b, their prime factorizations suffice to compute $\gcd(a, b)$. Indeed, if

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r} , \tag{31.11}$$

$$b = p_1^{f_1} p_2^{f_2} \cdots p_r^{f_r} , \tag{31.12}$$

with 0 exponents being used to make the set of primes $p_1, p_2, \ldots, p_r$ the same for both a and b, then, as Exercise 31.2-1 asks you to show,

$$\gcd(a, b) = p_1^{\min\{e_1, f_1\}} p_2^{\min\{e_2, f_2\}} \cdots p_r^{\min\{e_r, f_r\}} . \tag{31.13}$$

The best algorithms to date for factoring do not run in polynomial time. Thus, this approach to computing greatest common divisors seems unlikely to yield an efficient algorithm.

Euclid's algorithm for computing greatest common divisors relies on the following theorem.

Theorem 31.9 (GCD recursion theorem)
For any nonnegative integer a and any positive integer b,

$$\gcd(a, b) = \gcd(b, a \bmod b) .$$

Proof We will show that $\gcd(a, b)$ and $\gcd(b, a \bmod b)$ divide each other. Since they are both nonnegative, equation (31.5) then implies that they must be equal.

We first show that $\gcd(a, b) \mid \gcd(b, a \bmod b)$. If we let $d = \gcd(a, b)$, then $d \mid a$ and $d \mid b$. By equation (3.11) on page 64, $a \bmod b = a - qb$, where $q = \lfloor a/b \rfloor$. Since $a \bmod b$ is thus a linear combination of a and b, equation (31.4) implies that $d \mid (a \bmod b)$. Therefore, since $d \mid b$ and $d \mid (a \bmod b)$, Corollary 31.3 implies that $d \mid \gcd(b, a \bmod b)$, that is,

$$\gcd(a, b) \mid \gcd(b, a \bmod b) . \tag{31.14}$$

Showing that $\gcd(b, a \bmod b) \mid \gcd(a, b)$ is almost the same. If we now let $d = \gcd(b, a \bmod b)$, then $d \mid b$ and $d \mid (a \bmod b)$. Since $a = qb + (a \bmod b)$,

where $q = \lfloor a/b \rfloor$, we have that a is a linear combination of b and $(a \bmod b)$. By equation (31.4), we conclude that $d \mid a$. Since $d \mid b$ and $d \mid a$, we have that $d \mid \gcd(a, b)$ by Corollary 31.3, so that

$$\gcd(b, a \bmod b) \mid \gcd(a, b) \; . \tag{31.15}$$

Using equation (31.5) to combine equations (31.14) and (31.15) completes the proof. ■

Euclid's algorithm

Euclid's *Elements* (circa 300 B.C.E.) describes the following gcd algorithm, although its origin might be even earlier. The recursive procedure EUCLID implements Euclid's algorithm, based directly on Theorem 31.9. The inputs a and b are arbitrary nonnegative integers.

```
EUCLID(a, b)
1  if b == 0
2        return a
3  else return EUCLID(b, a mod b)
```

For example, here is how the procedure computes $\gcd(30, 21)$:

$$\begin{aligned} \text{EUCLID}(30, 21) &= \text{EUCLID}(21, 9) \\ &= \text{EUCLID}(9, 3) \\ &= \text{EUCLID}(3, 0) \\ &= 3 \; . \end{aligned}$$

This computation calls EUCLID recursively three times.

The correctness of EUCLID follows from Theorem 31.9 and the property that if the algorithm returns a in line 2, then $b = 0$, so that by equation (31.9), $\gcd(a, b) = \gcd(a, 0) = a$. The algorithm cannot recurse indefinitely, since the second argument strictly decreases in each recursive call and is always nonnegative. Therefore, EUCLID always terminates with the correct answer.

The running time of Euclid's algorithm

Let's analyze the worst-case running time of EUCLID as a function of the size of a and b. The overall running time of EUCLID is proportional to the number of recursive calls it makes. The analysis assumes that $a > b \geq 0$, that is, the first argument is greater than the second argument. Why? If $b = a > 0$, then $a \bmod b = 0$ and the procedure terminates after one recursive call. If $b > a \geq 0$,

then the procedure makes just one more recursive call than when $a > b$, because in this case EUCLID(a, b) immediately makes the recursive call EUCLID(b, a), and now the first argument is greater than the second.

Our analysis relies on the Fibonacci numbers F_k, defined by the recurrence equation (3.31) on page 69.

Lemma 31.10
If $a > b \geq 1$ and the call EUCLID(a, b) performs $k \geq 1$ recursive calls, then $a \geq F_{k+2}$ and $b \geq F_{k+1}$.

Proof The proof proceeds by induction on k. For the base case of the induction, let $k = 1$. Then, $b \geq 1 = F_2$, and since $a > b$, we must have $a \geq 2 = F_3$. Since $b > (a \bmod b)$, in each recursive call the first argument is strictly larger than the second. The assumption that $a > b$ therefore holds for each recursive call.

Assuming inductively that the lemma holds if the procedure makes $k - 1$ recursive calls, we shall prove that the lemma holds for k recursive calls. Since $k > 0$, we have $b > 0$, and EUCLID(a, b) calls EUCLID$(b, a \bmod b)$ recursively, which in turn makes $k - 1$ recursive calls. The inductive hypothesis then implies that $b \geq F_{k+1}$ (thus proving part of the lemma), and $a \bmod b \geq F_k$. We have

$$\begin{aligned} b + (a \bmod b) &= b + (a - b \lfloor a/b \rfloor) \qquad \text{(by equation (3.11))} \\ &\leq a \ , \end{aligned}$$

since $a > b > 0$ implies $\lfloor a/b \rfloor \geq 1$. Thus,

$$\begin{aligned} a &\geq b + (a \bmod b) \\ &\geq F_{k+1} + F_k \\ &= F_{k+2} \ . \end{aligned}$$

■

The following theorem is an immediate corollary of this lemma.

Theorem 31.11 (Lamé's theorem)
For any integer $k \geq 1$, if $a > b \geq 1$ and $b < F_{k+1}$, then the call EUCLID(a, b) makes fewer than k recursive calls. ■

To show that the upper bound of Theorem 31.11 is the best possible, we'll show that the call EUCLID(F_{k+1}, F_k) makes exactly $k - 1$ recursive calls when $k \geq 2$. We use induction on k. For the base case, $k = 2$, and the call EUCLID(F_3, F_2) makes exactly one recursive call, to EUCLID$(1, 0)$. (We have to start at $k = 2$, because when $k = 1$ we do not have $F_2 > F_1$.) For the inductive step, assume that EUCLID(F_k, F_{k-1}) makes exactly $k - 2$ recursive calls. For $k > 2$, we have $F_k > F_{k-1} > 0$ and $F_{k+1} = F_k + F_{k-1}$, and so by Exercise 31.1-1, we

have $F_{k+1} \bmod F_k = F_{k-1}$. Because EUCLID$(a, b)$ calls EUCLID$(b, a \bmod b)$ when $b > 0$, the call EUCLID(F_{k+1}, F_k) recurses one time more than the call EUCLID(F_k, F_{k-1}), or exactly $k - 1$ times, which meets the upper bound given by Theorem 31.11.

Since F_k is approximately $\phi^k/\sqrt{5}$, where ϕ is the golden ratio $(1 + \sqrt{5})/2$ defined by equation (3.32) on page 69, the number of recursive calls in EUCLID is $O(\lg b)$. (See Exercise 31.2-5 for a tighter bound.) Therefore, a call of EUCLID on two β-bit numbers performs $O(\beta)$ arithmetic operations and $O(\beta^3)$ bit operations (assuming that multiplication and division of β-bit numbers take $O(\beta^2)$ bit operations). Problem 31-2 asks you to prove an $O(\beta^2)$ bound on the number of bit operations.

The extended form of Euclid's algorithm

By rewriting Euclid's algorithm, we can gain additional useful information. Specifically, let's extend the algorithm to compute the integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by \,, \tag{31.16}$$

where either or both of x and y may be zero or negative. These coefficients will prove useful later for computing modular multiplicative inverses. The procedure EXTENDED-EUCLID takes as input a pair of nonnegative integers and returns a triple of the form (d, x, y) that satisfies equation (31.16). As an example, Figure 31.1 traces out the call EXTENDED-EUCLID$(99, 78)$.

```
EXTENDED-EUCLID(a, b)
1  if b == 0
2       return (a, 1, 0)
3  else (d', x', y') = EXTENDED-EUCLID(b, a mod b)
4       (d, x, y) = (d', y', x' − ⌊a/b⌋ y')
5       return (d, x, y)
```

The EXTENDED-EUCLID procedure is a variation of the EUCLID procedure. Line 1 is equivalent to the test "$b == 0$" in line 1 of EUCLID. If $b = 0$, then EXTENDED-EUCLID returns not only $d = a$ in line 2, but also the coefficients $x = 1$ and $y = 0$, so that $a = ax + by$. If $b \neq 0$, EXTENDED-EUCLID first computes (d', x', y') such that $d' = \gcd(b, a \bmod b)$ and

$$d' = bx' + (a \bmod b)y' \,. \tag{31.17}$$

As in the EUCLID procedure, we have $d = \gcd(a, b) = d' = \gcd(b, a \bmod b)$. To obtain x and y such that $d = ax + by$, let's rewrite equation (31.17), setting

a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	−11	14
78	21	3	3	3	−11
21	15	1	3	−2	3
15	6	2	3	1	−2
6	3	2	3	0	1
3	0	—	3	1	0

Figure 31.1 How EXTENDED-EUCLID computes gcd(99, 78). Each line shows one level of the recursion: the values of the inputs a and b, the computed value $\lfloor a/b \rfloor$, and the values d, x, and y returned. The triple (d, x, y) returned becomes the triple (d', x', y') used at the next higher level of recursion. The call EXTENDED-EUCLID(99, 78) returns $(3, -11, 14)$, so that $\gcd(99, 78) = 3 = 99 \cdot (-11) + 78 \cdot 14$.

$d = d'$ and using equation (3.11):

$$\begin{aligned} d &= bx' + (a - b\lfloor a/b \rfloor)y' \\ &= ay' + b(x' - \lfloor a/b \rfloor y') \, . \end{aligned}$$

Thus, choosing $x = y'$ and $y = x' - \lfloor a/b \rfloor y'$ satisfies the equation $d = ax + by$, thereby proving the correctness of EXTENDED-EUCLID.

Since the number of recursive calls made in EUCLID is equal to the number of recursive calls made in EXTENDED-EUCLID, the running times of EUCLID and EXTENDED-EUCLID are the same, to within a constant factor. That is, for $a > b > 0$, the number of recursive calls is $O(\lg b)$.

Exercises

31.2-1

Prove that equations (31.11) and (31.12) imply equation (31.13).

31.2-2

Compute the values (d, x, y) that the call EXTENDED-EUCLID(899, 493) returns.

31.2-3

Prove that for all integers a, k, and n,

$$\gcd(a, n) = \gcd(a + kn, n) \, . \tag{31.18}$$

Use equation (31.18) to show that $a = 1 \pmod{n}$ implies $\gcd(a, n) = 1$.

31.2-4

Rewrite EUCLID in an iterative form that uses only a constant amount of memory (that is, stores only a constant number of integer values).

31.2-5
If $a > b \geq 0$, show that the call EUCLID(a, b) makes at most $1 + \log_\phi b$ recursive calls. Improve this bound to $1 + \log_\phi(b/\gcd(a, b))$.

31.2-6
What does EXTENDED-EUCLID(F_{k+1}, F_k) return? Prove your answer correct.

31.2-7
Define the gcd function for more than two arguments by the recursive equation $\gcd(a_0, a_1, \ldots, a_n) = \gcd(a_0, \gcd(a_1, a_2, \ldots, a_n))$. Show that the gcd function returns the same answer independent of the order in which its arguments are specified. Also show how to find integers $x_0, x_1, \ldots, x_n$ such that $\gcd(a_0, a_1, \ldots, a_n) = a_0x_0 + a_1x_1 + \cdots + a_nx_n$. Show that the number of divisions performed by your algorithm is $O(n + \lg(\max\{a_0, a_1, \ldots, a_n\}))$.

31.2-8
The ***least common multiple*** $\text{lcm}(a_1, a_2, \ldots, a_n)$ of integers $a_1, a_2, \ldots, a_n$ is the smallest nonnegative integer that is a multiple of each a_i. Show how to compute $\text{lcm}(a_1, a_2, \ldots, a_n)$ efficiently using the (two-argument) gcd operation as a subroutine.

31.2-9
Prove that n_1, n_2, n_3, and n_4 are pairwise relatively prime if and only if

$$\gcd(n_1n_2, n_3n_4) = \gcd(n_1n_3, n_2n_4) = 1 .$$

More generally, show that $n_1, n_2, \ldots, n_k$ are pairwise relatively prime if and only if a set of $\lceil \lg k \rceil$ pairs of numbers derived from the n_i are relatively prime.

31.3 Modular arithmetic

Informally, you can think of modular arithmetic as arithmetic as usual over the integers, except that when working modulo n, then every result x is replaced by the element of $\{0, 1, \ldots, n-1\}$ that is equivalent to x, modulo n (so that x is replaced by $x \bmod n$). This informal model suffices if you stick to the operations of addition, subtraction, and multiplication. A more formal model for modular arithmetic, which follows, is best described within the framework of group theory.

Finite groups

A ***group*** $(S, \oplus)$ is a set S together with a binary operation $\oplus$ defined on S for which the following properties hold:

1. **Closure:** For all $a, b \in S$, we have $a \oplus b \in S$.
2. **Identity:** There exists an element $e \in S$, called the ***identity*** of the group, such that $e \oplus a = a \oplus e = a$ for all $a \in S$.
3. **Associativity:** For all $a, b, c \in S$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
4. **Inverses:** For each $a \in S$, there exists a unique element $b \in S$, called the ***inverse*** of a, such that $a \oplus b = b \oplus a = e$.

As an example, consider the familiar group $(\mathbb{Z}, +)$ of the integers $\mathbb{Z}$ under the operation of addition: 0 is the identity, and the inverse of a is $-a$. An ***abelian group*** $(S, \oplus)$ satisfies the ***commutative law*** $a \oplus b = b \oplus a$ for all $a, b \in S$. The ***size*** of group $(S, \oplus)$ is $|S|$, and if $|S| < \infty$, then $(S, \oplus)$ is a ***finite group***.

The groups defined by modular addition and multiplication

We can form two finite abelian groups by using addition and multiplication modulo n, where n is a positive integer. These groups are based on the equivalence classes of the integers modulo n, defined in Section 31.1.

To define a group on $\mathbb{Z}_n$, we need suitable binary operations, which we obtain by redefining the ordinary operations of addition and multiplication. We can define addition and multiplication operations for $\mathbb{Z}_n$, because the equivalence class of two integers uniquely determines the equivalence class of their sum or product. That is, if $a = a' \pmod{n}$ and $b = b' \pmod{n}$, then

$$\begin{aligned} a + b &= a' + b' \pmod{n}, \\ ab &= a'b' \pmod{n}. \end{aligned}$$

Thus, we define addition and multiplication modulo n, denoted $+_n$ and $\cdot_n$, by

$$\begin{aligned} [a]_n +_n [b]_n &= [a + b]_n, \\ [a]_n \cdot_n [b]_n &= [ab]_n. \end{aligned} \tag{31.19}$$

(We can define subtraction similarly on $\mathbb{Z}_n$ by $[a]_n -_n [b]_n = [a - b]_n$, but division is more complicated, as we'll see.) These facts justify the common and convenient practice of using the smallest nonnegative element of each equivalence class as its representative when performing computations in $\mathbb{Z}_n$. We add, subtract, and multiply as usual on the representatives, but we replace each result x by the representative of its class, that is, by $x \bmod n$.

$+_6$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

(a)

$\cdot_{15}$	1	2	4	7	8	11	13	14
1	1	2	4	7	8	11	13	14
2	2	4	8	14	1	7	11	13
4	4	8	1	13	2	14	7	11
7	7	14	13	4	11	2	1	8
8	8	1	2	11	4	13	14	7
11	11	7	14	2	13	1	8	4
13	13	11	7	1	14	8	4	2
14	14	13	11	8	7	4	2	1

(b)

Figure 31.2 Two finite groups. Equivalence classes are denoted by their representative elements. **(a)** The group $(\mathbb{Z}_6, +_6)$. **(b)** The group $(\mathbb{Z}_{15}^*, \cdot_{15})$.

Using this definition of addition modulo n, we define the ***additive group modulo n*** as $(\mathbb{Z}_n, +_n)$. The size of the additive group modulo n is $|\mathbb{Z}_n| = n$. Figure 31.2(a) gives the operation table for the group $(\mathbb{Z}_6, +_6)$.

Theorem 31.12
The system $(\mathbb{Z}_n, +_n)$ is a finite abelian group.

Proof Equation (31.19) shows that $(\mathbb{Z}_n, +_n)$ is closed. Associativity and commutativity of $+_n$ follow from the associativity and commutativity of $+$:

$$\begin{aligned}
([a]_n +_n [b]_n) +_n [c]_n &= [a + b]_n +_n [c]_n \\
&= [(a + b) + c]_n \\
&= [a + (b + c)]_n \\
&= [a]_n +_n [b + c]_n \\
&= [a]_n +_n ([b]_n +_n [c]_n) ,
\end{aligned}$$

$$\begin{aligned}
[a]_n +_n [b]_n &= [a + b]_n \\
&= [b + a]_n \\
&= [b]_n +_n [a]_n .
\end{aligned}$$

The identity element of $(\mathbb{Z}_n, +_n)$ is 0 (that is, $[0]_n$). The (additive) inverse of an element a (that is, of $[a]_n$) is the element $-a$ (that is, $[-a]_n$ or $[n - a]_n$), since $[a]_n +_n [-a]_n = [a - a]_n = [0]_n$. ■

Using the definition of multiplication modulo n, we define the ***multiplicative group modulo n*** as $(\mathbb{Z}_n^*, \cdot_n)$. The elements of this group are the set $\mathbb{Z}_n^*$ of elements in $\mathbb{Z}_n$ that are relatively prime to n, so that each one has a unique inverse, modulo n:

$$\mathbb{Z}_n^* = \{[a]_n \in \mathbb{Z}_n : \gcd(a, n) = 1\} \ .$$

To see that $\mathbb{Z}_n^*$ is well defined, note that for $0 \le a < n$, we have $a = (a + kn) \pmod{n}$ for all integers k. By Exercise 31.2-3, therefore, $\gcd(a, n) = 1$ implies $\gcd(a + kn, n) = 1$ for all integers k. Since $[a]_n = \{a + kn : k \in \mathbb{Z}\}$, the set $\mathbb{Z}_n^*$ is well defined. An example of such a group is

$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\} \ ,$$

where the group operation is multiplication modulo 15. (We have denoted an element $[a]_{15}$ as a, and thus, for example, we denote $[7]_{15}$ as 7.) Figure 31.2(b) shows the group $(\mathbb{Z}_{15}^*, \cdot_{15})$. For example, $8 \cdot 11 = 13 \pmod{15}$, working in $\mathbb{Z}_{15}^*$. The identity for this group is 1.

Theorem 31.13
The system $(\mathbb{Z}_n^*, \cdot_n)$ is a finite abelian group.

Proof Theorem 31.6 implies that $(\mathbb{Z}_n^*, \cdot_n)$ is closed. Associativity and commutativity can be proved for $\cdot_n$ as they were for $+_n$ in the proof of Theorem 31.12. The identity element is $[1]_n$. To show the existence of inverses, let a be an element of $\mathbb{Z}_n^*$ and let (d, x, y) be returned by EXTENDED-EUCLID(a, n). Then we have $d = 1$, since $a \in \mathbb{Z}_n^*$, and

$$ax + ny = 1 \ , \tag{31.20}$$

or equivalently,

$$ax = 1 \pmod{n} \ .$$

Thus $[x]_n$ is a multiplicative inverse of $[a]_n$, modulo n. Furthermore, we claim that $[x]_n \in \mathbb{Z}_n^*$. To see why, equation (31.20) demonstrates that the smallest positive linear combination of x and n must be 1. Therefore, Theorem 31.2 implies that $\gcd(x, n) = 1$. We defer the proof that inverses are uniquely defined until Corollary 31.26 in Section 31.4. ■

As an example of computing multiplicative inverses, suppose that $a = 5$ and $n = 11$. Then EXTENDED-EUCLID(a, n) returns $(d, x, y) = (1, -2, 1)$, so that $1 = 5 \cdot (-2) + 11 \cdot 1$. Thus, $[-2]_{11}$ (i.e., $[9]_{11}$) is the multiplicative inverse of $[5]_{11}$.

When working with the groups $(\mathbb{Z}_n, +_n)$ and $(\mathbb{Z}_n^*, \cdot_n)$ in the remainder of this chapter, we follow the convenient practice of denoting equivalence classes by their representative elements and denoting the operations $+_n$ and $\cdot_n$ by the usual

arithmetic notations $+$ and $\cdot$ (or juxtaposition, so that $ab = a \cdot b$) respectively. Furthermore, equivalences modulo n may also be interpreted as equations in $\mathbb{Z}_n$. For example, the following two statements are equivalent:

$$ax = b \pmod{n}$$

and

$$[a]_n \cdot_n [x]_n = [b]_n \ .$$

As a further convenience, we sometimes refer to a group $(S, \oplus)$ merely as S when the operation $\oplus$ is understood from context. We may thus refer to the groups $(\mathbb{Z}_n, +_n)$ and $(\mathbb{Z}_n^*, \cdot_n)$ as just $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$, respectively.

We denote the (multiplicative) inverse of an element a by $(a^{-1} \bmod n)$. Division in $\mathbb{Z}_n^*$ is defined by the equation $a/b = ab^{-1} \pmod{n}$. For example, in $\mathbb{Z}_{15}^*$ we have that $7^{-1} = 13 \pmod{15}$, since $7 \cdot 13 = 91 = 1 \pmod{15}$, so that $2/7 = 2 \cdot 13 = 11 \pmod{15}$.

The size of $\mathbb{Z}_n^*$ is denoted $\phi(n)$. This function, known as ***Euler's phi function***, satisfies the equation

$$\phi(n) = n \prod_{p \text{ prime such that } p \mid n} \left(1 - \frac{1}{p}\right) \ , \tag{31.21}$$

so that p runs over all the primes dividing n (including n itself, if n is prime). We won't prove this formula here. Intuitively, begin with a list of the n remainders $\{0, 1, \ldots, n-1\}$ and then, for each prime p that divides n, cross out every multiple of p in the list. For example, since the prime divisors of 45 are 3 and 5,

$$\begin{aligned} \phi(45) &= 45\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right) \\ &= 45\left(\frac{2}{3}\right)\left(\frac{4}{5}\right) \\ &= 24 \ . \end{aligned}$$

If p is prime, then $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, and

$$\begin{aligned} \phi(p) &= p\left(1 - \frac{1}{p}\right) \\ &= p - 1 \ . \end{aligned} \tag{31.22}$$

If n is composite, then $\phi(n) < n - 1$, although it can be shown that

$$\phi(n) > \frac{n}{e^\gamma \ln\ln n + 3/\ln\ln n} \tag{31.23}$$

for $n \geq 3$, where $\gamma = 0.5772156649\ldots$ is ***Euler's constant***. A somewhat simpler (but looser) lower bound for $n > 5$ is

$$\phi(n) > \frac{n}{6 \ln \ln n} . \tag{31.24}$$

The lower bound (31.23) is essentially the best possible, since

$$\liminf_{n \to \infty} \frac{\phi(n)}{n / \ln \ln n} = e^{-\gamma} . \tag{31.25}$$

Subgroups

If $(S, \oplus)$ is a group, $S' \subseteq S$, and $(S', \oplus)$ is also a group, then $(S', \oplus)$ is a ***subgroup*** of $(S, \oplus)$. For example, the even integers form a subgroup of the integers under the operation of addition. The following theorem, whose proof we leave as Exercise 31.3-3, provides a useful tool for recognizing subgroups.

Theorem 31.14 (A nonempty closed subset of a finite group is a subgroup)
If $(S, \oplus)$ is a finite group and S' is any nonempty subset of S such that $a \oplus b \in S'$ for all $a, b \in S'$, then $(S', \oplus)$ is a subgroup of $(S, \oplus)$. ■

For example, the set $\{0, 2, 4, 6\}$ forms a subgroup of $\mathbb{Z}_8$, since it is nonempty and closed under the operation $+$ (that is, it is closed under $+_8$).

The following theorem, whose proof is omitted, provides an extremely useful constraint on the size of a subgroup.

Theorem 31.15 (Lagrange's theorem)
If $(S, \oplus)$ is a finite group and $(S', \oplus)$ is a subgroup of $(S, \oplus)$, then $|S'|$ is a divisor of $|S|$. ■

A subgroup S' of a group S is a ***proper*** subgroup if $S' \neq S$. We'll use the following corollary in the analysis in Section 31.8 of the Miller-Rabin primality test procedure.

Corollary 31.16
If S' is a proper subgroup of a finite group S, then $|S'| \leq |S|/2$. ■

Subgroups generated by an element

Theorem 31.14 affords us a straightforward way to produce a subgroup of a finite group $(S, \oplus)$: choose an element a and take all elements that can be generated from a using the group operation. Specifically, define $a^{(k)}$ for $k \geq 1$ by

$$a^{(k)} = \bigoplus_{i=1}^{k} a = \underbrace{a \oplus a \oplus \cdots \oplus a}_{k} .$$

For example, taking $a = 2$ in the group $\mathbb{Z}_6$ yields the sequence

$$a^{(1)}, a^{(2)}, a^{(3)}, \ldots = 2, 4, 0, 2, 4, 0, 2, 4, 0, \ldots .$$

We have $a^{(k)} = ka \bmod n$ in the group $\mathbb{Z}_n$, and $a^{(k)} = a^k \bmod n$ in the group $\mathbb{Z}_n^*$. We define the ***subgroup generated by a***, denoted $\langle a \rangle$ or $(\langle a \rangle, \oplus)$, by

$$\langle a \rangle = \left\{ a^{(k)} : k \geq 1 \right\} .$$

We say that a ***generates*** the subgroup $\langle a \rangle$ or that a is a ***generator*** of $\langle a \rangle$. Since S is finite, $\langle a \rangle$ is a finite subset of S, possibly including all of S. Since the associativity of $\oplus$ implies

$$a^{(i)} \oplus a^{(j)} = a^{(i+j)} ,$$

$\langle a \rangle$ is closed and therefore, by Theorem 31.14, $\langle a \rangle$ is a subgroup of S. For example, in $\mathbb{Z}_6$, we have

$$\begin{aligned} \langle 0 \rangle &= \{0\} , \\ \langle 1 \rangle &= \{0, 1, 2, 3, 4, 5\} , \\ \langle 2 \rangle &= \{0, 2, 4\} . \end{aligned}$$

Similarly, in $\mathbb{Z}_7^*$, we have

$$\begin{aligned} \langle 1 \rangle &= \{1\} , \\ \langle 2 \rangle &= \{1, 2, 4\} , \\ \langle 3 \rangle &= \{1, 2, 3, 4, 5, 6\} . \end{aligned}$$

The ***order*** of a (in the group S), denoted $\mathrm{ord}(a)$, is defined as the smallest positive integer t such that $a^{(t)} = e$. (Recall that $e \in S$ is the group identity.)

Theorem 31.17
For any finite group $(S, \oplus)$ and any $a \in S$, the order of a is equal to the size of the subgroup it generates, or $\mathrm{ord}(a) = |\langle a \rangle|$.

Proof Let $t = \mathrm{ord}(a)$. Since $a^{(t)} = e$ and $a^{(t+k)} = a^{(t)} \oplus a^{(k)} = a^{(k)}$ for $k \geq 1$, if $i > t$, then $a^{(i)} = a^{(j)}$ for some $j < i$. Therefore, as we generate elements by a, we see no new elements after $a^{(t)}$. Thus, $\langle a \rangle = \left\{ a^{(1)}, a^{(2)}, \ldots, a^{(t)} \right\}$, and so $|\langle a \rangle| \leq t$. To show that $|\langle a \rangle| \geq t$, we show that each element of the sequence $a^{(1)}, a^{(2)}, \ldots, a^{(t)}$ is distinct. Suppose for the purpose of contradiction that $a^{(i)} = a^{(j)}$ for some i and j satisfying $1 \leq i < j \leq t$. Then, $a^{(i+k)} = a^{(j+k)}$ for

$k \geq 0$. But this equation implies that $a^{(i+(t-j))} = a^{(j+(t-j))} = e$, a contradiction, since $i + (t - j) < t$ but t is the least positive value such that $a^{(t)} = e$. Therefore, each element of the sequence $a^{(1)}, a^{(2)}, \ldots, a^{(t)}$ is distinct, and $|\langle a \rangle| \geq t$. We conclude that $\text{ord}(a) = |\langle a \rangle|$. ■

Corollary 31.18
The sequence $a^{(1)}, a^{(2)}, \ldots$ is periodic with period $t = \text{ord}(a)$, that is, $a^{(i)} = a^{(j)}$ if and only if $i = j \pmod{t}$. ■

Consistent with the above corollary, we define $a^{(0)}$ as e and $a^{(i)}$ as $a^{(i \bmod t)}$, where $t = \text{ord}(a)$, for all integers i.

Corollary 31.19
If $(S, \oplus)$ is a finite group with identity e, then for all $a \in S$,

$$a^{(|S|)} = e \ .$$

Proof Lagrange's theorem (Theorem 31.15) implies that $\text{ord}(a) \mid |S|$, and so $|S| = 0 \pmod{t}$, where $t = \text{ord}(a)$. Therefore, $a^{(|S|)} = a^{(0)} = e$. ■

Exercises

31.3-1
Draw the group operation tables for the groups $(\mathbb{Z}_4, +_4)$ and $(\mathbb{Z}_5^*, \cdot_5)$. Show that these groups are isomorphic by exhibiting a one-to-one correspondence f between $\mathbb{Z}_4$ and $\mathbb{Z}_5^*$ such that $a + b = c \pmod{4}$ if and only if $f(a) \cdot f(b) = f(c) \pmod{5}$.

31.3-2
List all subgroups of $\mathbb{Z}_9$ and of $\mathbb{Z}_{13}^*$.

31.3-3
Prove Theorem 31.14.

31.3-4
Show that if p is prime and e is a positive integer, then

$$\phi(p^e) = p^{e-1}(p - 1) \ .$$

31.3-5
Show that for any integer $n > 1$ and for any $a \in \mathbb{Z}_n^*$, the function $f_a : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ defined by $f_a(x) = ax \bmod n$ is a permutation of $\mathbb{Z}_n^*$.

31.4 Solving modular linear equations

We now consider the problem of finding solutions to the equation

$$ax = b \pmod{n}, \tag{31.26}$$

where $a > 0$ and $n > 0$. This problem has several applications. For example, we'll use it in Section 31.7 as part of the procedure to find keys in the RSA public-key cryptosystem. We assume that a, b, and n are given, and we wish to find all values of x, modulo n, that satisfy equation (31.26). The equation may have zero, one, or more than one such solution.

Let $\langle a \rangle$ denote the subgroup of $\mathbb{Z}_n$ generated by a. Since $\langle a \rangle = \{a^{(x)} : x > 0\} = \{ax \bmod n : x > 0\}$, equation (31.26) has a solution if and only if $[b] \in \langle a \rangle$. Lagrange's theorem (Theorem 31.15) tells us that $|\langle a \rangle|$ must be a divisor of n. The following theorem gives us a precise characterization of $\langle a \rangle$.

Theorem 31.20
For any positive integers a and n, if $d = \gcd(a, n)$, then we have

$$\begin{aligned}\langle a \rangle &= \langle d \rangle \\ &= \{0, d, 2d, \ldots, ((n/d) - 1)d\}\end{aligned}$$

in $\mathbb{Z}_n$, and thus

$$|\langle a \rangle| = n/d.$$

Proof We begin by showing that $d \in \langle a \rangle$. Recall that EXTENDED-EUCLID(a, n) returns a triple (d, x, y) such that $ax + ny = d$. Thus, $ax = d \pmod{n}$, so that $d \in \langle a \rangle$. In other words, d is a multiple of a in $\mathbb{Z}_n$.

Since $d \in \langle a \rangle$, it follows that every multiple of d belongs to $\langle a \rangle$, because any multiple of a multiple of a is itself a multiple of a. Thus, $\langle a \rangle$ contains every element in $\{0, d, 2d, \ldots, ((n/d) - 1)d\}$. That is, $\langle d \rangle \subseteq \langle a \rangle$.

We now show that $\langle a \rangle \subseteq \langle d \rangle$. If $m \in \langle a \rangle$, then $m = ax \bmod n$ for some integer x, and so $m = ax + ny$ for some integer y. Because $d = \gcd(a, n)$, we know that $d \mid a$ and $d \mid n$, and so $d \mid m$ by equation (31.4). Therefore, $m \in \langle d \rangle$.

Combining these results, we have that $\langle a \rangle = \langle d \rangle$. To see that $|\langle a \rangle| = n/d$, observe that there are exactly n/d multiples of d between 0 and $n - 1$, inclusive. ■

Corollary 31.21
The equation $ax = b \pmod{n}$ is solvable for the unknown x if and only if $d \mid b$, where $d = \gcd(a, n)$.

Proof The equation $ax = b \pmod{n}$ is solvable if and only if $[b] \in \langle a \rangle$, which is the same as saying

$$(b \bmod n) \in \{0, d, 2d, \ldots, ((n/d) - 1)d\} ,$$

by Theorem 31.20. If $0 \le b < n$, then $b \in \langle a \rangle$ if and only if $d \mid b$, since the members of $\langle a \rangle$ are precisely the multiples of d. If $b < 0$ or $b \ge n$, the corollary then follows from the observation that $d \mid b$ if and only if $d \mid (b \bmod n)$, since b and $b \bmod n$ differ by a multiple of n, which is itself a multiple of d. ■

Corollary 31.22
The equation $ax = b \pmod{n}$ either has d distinct solutions modulo n, where $d = \gcd(a, n)$, or it has no solutions.

Proof If $ax = b \pmod{n}$ has a solution, then $b \in \langle a \rangle$. By Theorem 31.17, $\mathrm{ord}(a) = |\langle a \rangle|$, and so Corollary 31.18 and Theorem 31.20 imply that the sequence $ai \bmod n$, for $i = 0, 1, \ldots$, is periodic with period $|\langle a \rangle| = n/d$. If $b \in \langle a \rangle$, then b appears exactly d times in the sequence $ai \bmod n$, for $i = 0, 1, \ldots, n-1$, since the length-(n/d) block of values $\langle a \rangle$ repeats exactly d times as i increases from 0 to $n-1$. The indices x of the d positions for which $ax \bmod n = b$ are the solutions of the equation $ax = b \pmod{n}$. ■

Theorem 31.23
Let $d = \gcd(a, n)$, and suppose that $d = ax' + ny'$ for some integers x' and y' (for example, as computed by EXTENDED-EUCLID). If $d \mid b$, then the equation $ax = b \pmod{n}$ has as one of its solutions the value x_0, where

$$x_0 = x'(b/d) \bmod n .$$

Proof We have

$$\begin{aligned} ax_0 &= ax'(b/d) \pmod{n} \\ &= d(b/d) \pmod{n} \qquad (\text{because } ax' = d \pmod{n}) \\ &= b \pmod{n} , \end{aligned}$$

and thus x_0 is a solution to $ax = b \pmod{n}$. ■

Theorem 31.24
Suppose that the equation $ax = b \pmod{n}$ is solvable (that is, $d \mid b$, where $d = \gcd(a, n)$) and that x_0 is any solution to this equation. Then, this equation has exactly d distinct solutions, modulo n, given by $x_i = x_0 + i(n/d)$ for $i = 0, 1, \ldots, d-1$.

Proof Because $n/d > 0$ and $0 \le i(n/d) < n$ for $i = 0, 1, \ldots, d-1$, the values $x_0, x_1, \ldots, x_{d-1}$ are all distinct, modulo n. Since x_0 is a solution of $ax = b \pmod{n}$, we have $ax_0 \bmod n = b \pmod{n}$. Thus, for $i = 0, 1, \ldots, d-1$, we have

$$\begin{aligned} ax_i \bmod n &= a(x_0 + in/d) \bmod n \\ &= (ax_0 + ain/d) \bmod n \\ &= ax_0 \bmod n \quad \text{(because } d \mid a \text{ implies that } ain/d \text{ is a multiple of } n\text{)} \\ &= b \pmod{n}, \end{aligned}$$

and hence $ax_i = b \pmod{n}$, making x_i a solution, too. By Corollary 31.22, the equation $ax = b \pmod{n}$ has exactly d solutions, so that $x_0, x_1, \ldots, x_{d-1}$ must be all of them. ■

We have now developed the mathematics needed to solve the equation $ax = b \pmod{n}$. The procedure MODULAR-LINEAR-EQUATION-SOLVER prints all solutions to this equation. The inputs a and n are arbitrary positive integers, and b is an arbitrary integer.

```
MODULAR-LINEAR-EQUATION-SOLVER(a, b, n)
1  (d, x', y') = EXTENDED-EUCLID(a, n)
2  if d | b
3       x_0 = x'(b/d) mod n
4       for i = 0 to d - 1
5            print (x_0 + i(n/d)) mod n
6  else print "no solutions"
```

As an example of the operation of MODULAR-LINEAR-EQUATION-SOLVER, consider the equation $14x = 30 \pmod{100}$ (and thus $a = 14$, $b = 30$, and $n = 100$). Calling EXTENDED-EUCLID in line 1 gives $(d, x', y') = (2, -7, 1)$. Since $2 \mid 30$, lines 3–5 execute. Line 3 computes $x_0 = (-7)(15) \bmod 100 = 95$. The **for** loop of lines 4–5 prints the two solutions, 95 and 45.

The procedure MODULAR-LINEAR-EQUATION-SOLVER works as follows. The call to EXTENDED-EUCLID in line 1 returns a triple (d, x', y') such that $d = \gcd(a, n)$ and $d = ax' + ny'$. Therefore, x' is a solution to the equation $ax' = d \pmod{n}$. If d does not divide b, then the equation $ax = b \pmod{n}$ has no solution, by Corollary 31.21. Line 2 checks to see whether $d \mid b$, and if not, line 6 reports that there are no solutions. Otherwise, line 3 computes a solution x_0 to $ax = b \pmod{n}$, as Theorem 31.23 suggests. Given one solution, Theorem 31.24 states that adding multiples of (n/d), modulo n, yields the other

$d - 1$ solutions. The **for** loop of lines 4–5 prints out all d solutions, beginning with x_0 and spaced n/d apart, modulo n.

MODULAR-LINEAR-EQUATION-SOLVER performs $O(\lg n + \gcd(a, n))$ arithmetic operations, since EXTENDED-EUCLID performs $O(\lg n)$ arithmetic operations, and each iteration of the **for** loop of lines 4–5 performs a constant number of arithmetic operations.

The following corollaries of Theorem 31.24 give specializations of particular interest.

Corollary 31.25

For any $n > 1$, if $\gcd(a, n) = 1$, then the equation $ax = b \pmod{n}$ has a unique solution, modulo n. ■

If $b = 1$, a common case of considerable interest, the x that solves the equation is a ***multiplicative inverse*** of a, modulo n.

Corollary 31.26

For any $n > 1$, if $\gcd(a, n) = 1$, then the equation $ax = 1 \pmod{n}$ has a unique solution, modulo n. Otherwise, it has no solution. ■

Thanks to Corollary 31.26, the notation $a^{-1} \bmod n$ refers to *the* multiplicative inverse of a, modulo n, when a and n are relatively prime. If $\gcd(a, n) = 1$, then the unique solution to the equation $ax = 1 \pmod{n}$ is the integer x returned by EXTENDED-EUCLID, since the equation

$$\gcd(a, n) = 1 = ax + ny$$

implies $ax = 1 \pmod{n}$. Thus, EXTENDED-EUCLID can compute $a^{-1} \bmod n$ efficiently.

Exercises

31.4-1

Find all solutions to the equation $35x = 10 \pmod{50}$.

31.4-2

Prove that the equation $ax = ay \pmod{n}$ implies $x = y \pmod{n}$ whenever $\gcd(a, n) = 1$. Show that the condition $\gcd(a, n) = 1$ is necessary by supplying a counterexample with $\gcd(a, n) > 1$.

31.4-3
Consider the following change to line 3 of the procedure MODULAR-LINEAR-EQUATION-SOLVER:

3 $x_0 = x'(b/d) \bmod (n/d)$

With this change, will the procedure still work? Explain why or why not.

★ ***31.4-4***
Let p be prime and $f(x) = (f_0 + f_1 x + \cdots + f_t x^t) \pmod p$ be a polynomial of degree t, with coefficients f_i drawn from $\mathbb{Z}_p$. We say that $a \in \mathbb{Z}_p$ is a ***zero*** of f if $f(a) = 0 \pmod p$. Prove that if a is a zero of f, then $f(x) = (x-a)g(x)(\bmod p)$ for some polynomial $g(x)$ of degree $t-1$. Prove by induction on t that if p is prime, then a polynomial $f(x)$ of degree t can have at most t distinct zeros modulo p.

31.5 The Chinese remainder theorem

Around 100 C.E., the Chinese mathematician Sun-Tsŭ solved the problem of finding those integers x that leave remainders 2, 3, and 2 when divided by 3, 5, and 7 respectively. One such solution is $x = 23$, and all solutions are of the form $23 + 105k$ for arbitrary integers k. The "Chinese remainder theorem" provides a correspondence between a system of equations modulo a set of pairwise relatively prime moduli (for example, 3, 5, and 7) and an equation modulo their product (for example, 105).

The Chinese remainder theorem has two major applications. Let the integer n be factored as $n = n_1 n_2 \cdots n_k$, where the factors n_i are pairwise relatively prime. First, the Chinese remainder theorem is a descriptive "structure theorem" that describes the structure of $\mathbb{Z}_n$ as identical to that of the Cartesian product $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_k}$ with componentwise addition and multiplication modulo n_i in the ith component. Second, this description helps in designing efficient algorithms, since working in each of the systems $\mathbb{Z}_{n_i}$ can be more efficient (in terms of bit operations) than working modulo n.

Theorem 31.27 (Chinese remainder theorem)
Let $n = n_1 n_2 \cdots n_k$, where the n_i are pairwise relatively prime. Consider the correspondence

$$a \leftrightarrow (a_1, a_2, \ldots, a_k) , \tag{31.27}$$

where $a \in \mathbb{Z}_n$, $a_i \in \mathbb{Z}_{n_i}$, and

$$a_i = a \bmod n_i$$

for $i = 1, 2, \ldots, k$. Then, mapping (31.27) is a one-to-one mapping (bijection) between $\mathbb{Z}_n$ and the Cartesian product $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_k}$. Operations performed on the elements of $\mathbb{Z}_n$ can be equivalently performed on the corresponding k-tuples by performing the operations independently in each coordinate position in the appropriate system. That is, if

$$a \leftrightarrow (a_1, a_2, \ldots, a_k) ,$$
$$b \leftrightarrow (b_1, b_2, \ldots, b_k) ,$$

then

$$(a + b) \bmod n \leftrightarrow ((a_1 + b_1) \bmod n_1, \ldots, (a_k + b_k) \bmod n_k) , \tag{31.28}$$
$$(a - b) \bmod n \leftrightarrow ((a_1 - b_1) \bmod n_1, \ldots, (a_k - b_k) \bmod n_k) , \tag{31.29}$$
$$(ab) \bmod n \leftrightarrow (a_1 b_1 \bmod n_1, \ldots, a_k b_k \bmod n_k) . \tag{31.30}$$

Proof Let's see how to translate between the two representations. Going from a to $(a_1, a_2, \ldots, a_k)$ requires only k "mod" operations. The reverse—computing a from inputs $(a_1, a_2, \ldots, a_k)$—is only slightly more complicated.

We begin by defining $m_i = n/n_i$ for $i = 1, 2, \ldots, k$. Thus, m_i is the product of all of the n_j's other than n_i: $m_i = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k$. We next define

$$c_i = m_i (m_i^{-1} \bmod n_i) \tag{31.31}$$

for $i = 1, 2, \ldots, k$. Equation (31.31) is well defined: since m_i and n_i are relatively prime (by Theorem 31.6), Corollary 31.26 guarantees that $m_i^{-1} \bmod n_i$ exists. Here is how to compute a as a function of the a_i and c_i:

$$a = (a_1 c_1 + a_2 c_2 + \cdots + a_k c_k) \pmod{n} . \tag{31.32}$$

We now show that equation (31.32) ensures that $a = a_i \pmod{n_i}$ for $i = 1, 2, \ldots, k$. If $j \neq i$, then $m_j = 0 \pmod{n_i}$, which implies that $c_j = m_j = 0 \pmod{n_i}$. Note also that $c_i = 1 \pmod{n_i}$, from equation (31.31). We thus have the appealing and useful correspondence

$$c_i \leftrightarrow (0, 0, \ldots, 0, 1, 0, \ldots, 0) ,$$

a vector that has 0s everywhere except in the ith coordinate, where it has a 1. The c_i thus form a "basis" for the representation, in a certain sense. For each i, therefore, we have

$$\begin{aligned} a &= a_i c_i && \pmod{n_i} \\ &= a_i m_i (m_i^{-1} \bmod n_i) && \pmod{n_i} \\ &= a_i && \pmod{n_i} , \end{aligned}$$

which is what we wished to show: our method of computing a from the a_i's produces a result a that satisfies the constraints $a = a_i \pmod{n_i}$ for $i = 1, 2, \ldots, k$. The correspondence is one-to-one, since we can transform in both directions. Finally, equations (31.28)–(31.30) follow directly from Exercise 31.1-7, since $x \bmod n_i = (x \bmod n) \bmod n_i$ for any x and $i = 1, 2, \ldots, k$. ■

We'll use the following corollaries later in this chapter.

Corollary 31.28
If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \cdots n_k$, then for any integers $a_1, a_2, \ldots, a_k$, the set of simultaneous equations

$$x = a_i \pmod{n_i} ,$$

for $i = 1, 2, \ldots, k$, has a unique solution modulo n for the unknown x. ■

Corollary 31.29
If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \cdots n_k$, then for all integers x and a,

$$x = a \pmod{n_i}$$

for $i = 1, 2, \ldots, k$ if and only if

$$x = a \pmod{n} .$$ ■

As an example of the application of the Chinese remainder theorem, suppose that you are given the two equations

$$a = 2 \pmod{5} ,$$
$$a = 3 \pmod{13} ,$$

so that $a_1 = 2$, $n_1 = m_2 = 5$, $a_2 = 3$, and $n_2 = m_1 = 13$, and you wish to compute $a \bmod 65$, since $n = n_1 n_2 = 65$. Because $13^{-1} = 2 \pmod 5$ and $5^{-1} = 8 \pmod{13}$, you compute

$$c_1 = 13 \cdot (2 \bmod 5) = 26 ,$$
$$c_2 = 5 \cdot (8 \bmod 13) = 40 ,$$

and

$$\begin{aligned} a &= 2 \cdot 26 + 3 \cdot 40 \pmod{65} \\ &= 52 + 120 \pmod{65} \\ &= 42 \pmod{65} . \end{aligned}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	40	15	55	30	5	45	20	60	35	10	50	25
1	26	1	41	16	56	31	6	46	21	61	36	11	51
2	52	27	2	42	17	57	32	7	47	22	62	37	12
3	13	53	28	3	43	18	58	33	8	48	23	63	38
4	39	14	54	29	4	44	19	59	34	9	49	24	64

Figure 31.3 An illustration of the Chinese remainder theorem for $n_1 = 5$ and $n_2 = 13$. For this example, $c_1 = 26$ and $c_2 = 40$. In row i, column j is shown the value of a, modulo 65, such that $a \bmod 5 = i$ and $a \bmod 13 = j$. Note that row 0, column 0 contains a 0. Similarly, row 4, column 12 contains a 64 (equivalent to -1). Since $c_1 = 26$, moving down a row increases a by 26. Similarly, $c_2 = 40$ means that moving right by a column increases a by 40. Increasing a by 1 corresponds to moving diagonally downward and to the right, wrapping around from the bottom to the top and from the right to the left.

See Figure 31.3 for an illustration of the Chinese remainder theorem, modulo 65.

Thus, you can work modulo n by working modulo n directly or by working in the transformed representation using separate modulo n_i computations, as convenient. The computations are entirely equivalent.

Exercises

31.5-1

Find all solutions to the equations $x = 4 \pmod 5$ and $x = 5 \pmod{11}$.

31.5-2

Find all integers x that leave remainders 1, 2, and 3 when divided by 9, 8, and 7, respectively.

31.5-3

Argue that, under the definitions of Theorem 31.27, if $\gcd(a, n) = 1$, then

$$(a^{-1} \bmod n) \leftrightarrow ((a_1^{-1} \bmod n_1), (a_2^{-1} \bmod n_2), \ldots, (a_k^{-1} \bmod n_k)) .$$

31.5-4

Under the definitions of Theorem 31.27, prove that for any polynomial f, the number of roots of the equation $f(x) = 0 \pmod n$ equals the product of the number of roots of each of the equations $f(x) = 0 \pmod{n_1}$, $f(x) = 0 \pmod{n_2}$, $\ldots$, $f(x) = 0 \pmod{n_k}$.

31.6 Powers of an element

Along with considering the multiples of a given element a, modulo n, we often consider the sequence of powers of a, modulo n, where $a \in \mathbb{Z}_n^*$:

$$a^0, a^1, a^2, a^3, \ldots,$$

modulo n. Indexing from 0, the 0th value in this sequence is $a^0 \bmod n = 1$, and the ith value is $a^i \bmod n$. For example, the powers of 3 modulo 7 are

i	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
$3^i \bmod 7$	1	3	2	6	4	5	1	3	2	6	4	5	$\cdots$

and the powers of 2 modulo 7 are

i	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
$2^i \bmod 7$	1	2	4	1	2	4	1	2	4	1	2	4	$\cdots$

In this section, let $\langle a \rangle$ denote the subgroup of $\mathbb{Z}_n^*$ generated by a through repeated multiplication, and let $\mathrm{ord}_n(a)$ (the "order of a, modulo n") denote the order of a in $\mathbb{Z}_n^*$. For example, $\langle 2 \rangle = \{1, 2, 4\}$ in $\mathbb{Z}_7^*$, and $\mathrm{ord}_7(2) = 3$. Using the definition of the Euler phi function $\phi(n)$ as the size of $\mathbb{Z}_n^*$ (see Section 31.3), we now translate Corollary 31.19 into the notation of $\mathbb{Z}_n^*$ to obtain Euler's theorem and specialize it to $\mathbb{Z}_p^*$, where p is prime, to obtain Fermat's theorem.

Theorem 31.30 (Euler's theorem)
For any integer $n > 1$,

$$a^{\phi(n)} = 1 \pmod{n} \text{ for all } a \in \mathbb{Z}_n^* .$$ ■

Theorem 31.31 (Fermat's theorem)
If p is prime, then

$$a^{p-1} = 1 \pmod{p} \text{ for all } a \in \mathbb{Z}_p^* .$$

Proof By equation (31.22), $\phi(p) = p - 1$ if p is prime. ■

Fermat's theorem applies to every element in $\mathbb{Z}_p$ except 0, since $0 \notin \mathbb{Z}_p^*$. For all $a \in \mathbb{Z}_p$, however, we have $a^p = a \pmod{p}$ if p is prime.

If $\mathrm{ord}_n(g) = |\mathbb{Z}_n^*|$, then every element in $\mathbb{Z}_n^*$ is a power of g, modulo n, and g is a ***primitive root*** or a ***generator*** of $\mathbb{Z}_n^*$. For example, 3 is a primitive root, modulo 7, but 2 is not a primitive root, modulo 7. If $\mathbb{Z}_n^*$ possesses a primitive root, the group $\mathbb{Z}_n^*$ is ***cyclic***. We omit the proof of the following theorem, which is proven by Niven and Zuckerman [345].

Theorem 31.32
The values of $n > 1$ for which $\mathbb{Z}_n^*$ is cyclic are 2, 4, p^e, and $2p^e$, for all primes $p > 2$ and all positive integers e. ■

If g is a primitive root of $\mathbb{Z}_n^*$ and a is any element of $\mathbb{Z}_n^*$, then there exists a z such that $g^z = a \pmod{n}$. This z is a ***discrete logarithm*** or an ***index*** of a, modulo n, to the base g. We denote this value as $\text{ind}_{n,g}(a)$.

Theorem 31.33 (Discrete logarithm theorem)
If g is a primitive root of $\mathbb{Z}_n^*$, then the equation $g^x = g^y \pmod{n}$ holds if and only if the equation $x = y \pmod{\phi(n)}$ holds.

Proof Suppose first that $x = y \pmod{\phi(n)}$. Then, we have $x = y + k\phi(n)$ for some integer k, and thus

$$\begin{aligned} g^x &= g^{y+k\phi(n)} && \pmod{n} \\ &= g^y \cdot (g^{\phi(n)})^k && \pmod{n} \\ &= g^y \cdot 1^k && \pmod{n} \qquad \text{(by Euler's theorem)} \\ &= g^y && \pmod{n} . \end{aligned}$$

Conversely, suppose that $g^x = g^y \pmod{n}$. Because the sequence of powers of g generates every element of $\langle g \rangle$ and $|\langle g \rangle| = \phi(n)$, Corollary 31.18 implies that the sequence of powers of g is periodic with period $\phi(n)$. Therefore, if $g^x = g^y \pmod{n}$, we must have $x = y \pmod{\phi(n)}$. ■

Let's now turn our attention to the square roots of 1, modulo a prime power. The following properties will be useful to justify the primality-testing algorithm in Section 31.8.

Theorem 31.34
If p is an odd prime and $e \geq 1$, then the equation

$$x^2 = 1 \pmod{p^e} \tag{31.33}$$

has only two solutions, namely $x = 1$ and $x = -1$.

Proof By Exercise 31.6-2, equation (31.33) is equivalent to

$$p^e \mid (x-1)(x+1) .$$

Since $p > 2$, we can have $p \mid (x-1)$ or $p \mid (x+1)$, but not both. (Otherwise, by property (31.3), p would also divide their difference $(x+1) - (x-1) = 2$.) If $p \nmid (x-1)$, then $\gcd(p^e, x-1) = 1$, and by Corollary 31.5, we would have $p^e \mid (x+1)$. That is, $x = -1 \pmod{p^e}$. Symmetrically, if $p \nmid (x+1)$,

then $\gcd(p^e, x+1) = 1$, and Corollary 31.5 implies that $p^e \mid (x-1)$, so that $x = 1 \pmod{p^e}$. Therefore, either $x = -1 \pmod{p^e}$ or $x = 1 \pmod{p^e}$. ■

A number x is a ***nontrivial square root of 1, modulo n***, if it satisfies the equation $x^2 = 1 \pmod{n}$ but x is equivalent to neither of the two "trivial" square roots: 1 or -1, modulo n. For example, 6 is a nontrivial square root of 1, modulo 35. We'll use the following corollary to Theorem 31.34 in Section 31.8 to prove the Miller-Rabin primality-testing procedure correct.

Corollary 31.35
If there exists a nontrivial square root of 1, modulo n, then n is composite.

Proof By the contrapositive of Theorem 31.34, if there exists a nontrivial square root of 1, modulo n, then n cannot be an odd prime or a power of an odd prime. Nor can n be 2, because if $x^2 = 1 \pmod 2$, then $x = 1 \pmod 2$, and therefore, all square roots of 1, modulo 2, are trivial. Thus, n cannot be prime. Finally, we must have $n > 1$ for a nontrivial square root of 1 to exist. Therefore, n must be composite. ■

Raising to powers with repeated squaring

A frequently occurring operation in number-theoretic computations is raising one number to a power modulo another number, also known as ***modular exponentiation***. More precisely, we would like an efficient way to compute $a^b \bmod n$, where a and b are nonnegative integers and n is a positive integer. Modular exponentiation is an essential operation in many primality-testing routines and in the RSA public-key cryptosystem. The method of ***repeated squaring*** solves this problem efficiently.

Repeated squaring is based on the following formula to compute a^b for nonnegative integers a and b:

$$a^b = \begin{cases} 1 & \text{if } b = 0\,, \\ (a^{b/2})^2 & \text{if } b > 0 \text{ and } b \text{ is even}\,, \\ a \cdot a^{b-1} & \text{if } b > 0 \text{ and } b \text{ is odd}\,. \end{cases} \tag{31.34}$$

The last case, where b is odd, reduces to the one of the first two cases, since if b is odd, then $b-1$ is even. The recursive procedure MODULAR-EXPONENTIATION on the next page computes $a^b \bmod n$ using equation (31.34), but performing all computations modulo n. The term "repeated squaring" comes from squaring the intermediate result $d = a^{b/2}$ in line 5. Figure 31.4 shows the values of the parameter b, the local variable d, and the value returned at each level of the recursion for the call MODULAR-EXPONENTIATION$(7, 560, 561)$, which returns the result 1.

b	560	280	140	70	35	34	17	16			8		4	2	1	0
d	67	166	298	241	355	160	103	526	157	49	7	1				–
returned value	1	67	166	298	241	355	160	103	526	157	49	7	1			

Figure 31.4 The values of the parameter b, the local variable d, and the value returned for recursive calls of MODULAR-EXPONENTIATION with parameter values $a = 7$, $b = 560$, and $n = 561$. The value returned by each recursive call is assigned directly to d. The result of the call with $a = 7$, $b = 560$, and $n = 561$ is 1.

```
MODULAR-EXPONENTIATION(a, b, n)
1  if b == 0
2      return 1
3  elseif b mod 2 == 0
4      d = MODULAR-EXPONENTIATION(a, b/2, n)     // b is even
5      return (d · d) mod n
6  else d = MODULAR-EXPONENTIATION(a, b − 1, n)  // b is odd
7      return (a · d) mod n
```

The total number of recursive calls depends on the number of bits of b and the values of these bits. Assume that $b > 0$ and that the most significant bit of b is a 1. Each 0 generates one recursive call (in line 4), and each 1 generates two recursive calls (one in line 6 followed by one in line 4 because if b is odd, then $b - 1$ is even). If the inputs a, b, and n are β-bit numbers, then there are between β and $2\beta - 1$ recursive calls altogether, the total number of arithmetic operations required is $O(\beta)$, and the total number of bit operations required is $O(\beta^3)$.

Exercises

31.6-1

Draw a table showing the order of every element in $\mathbb{Z}_{11}^*$. Pick the smallest primitive root g and compute a table giving $\text{ind}_{11,g}(x)$ for all $x \in \mathbb{Z}_{11}^*$.

31.6-2

Show that $x^2 = 1 \pmod{p^e}$ is equivalent to $p^e \mid (x-1)(x+1)$.

31.6-3

Rewrite the third case of MODULAR-EXPONENTIATION, where b is odd, so that if b has β bits and the most significant bit is 1, then there are always exactly β recursive calls.

31.6-4
Give a nonrecursive (i.e., iterative) version of MODULAR-EXPONENTIATION.

31.6-5
Assuming that you know $\phi(n)$, explain how to compute $a^{-1} \bmod n$ for any $a \in \mathbb{Z}_n^*$ using the procedure MODULAR-EXPONENTIATION.

31.7 The RSA public-key cryptosystem

With a public-key cryptosystem, you can ***encrypt*** messages sent between two communicating parties so that an eavesdropper who overhears the encrypted messages will not be able to decode, or ***decrypt***, them. A public-key cryptosystem also enables a party to append an unforgeable "digital signature" to the end of an electronic message. Such a signature is the electronic version of a handwritten signature on a paper document. It can be easily checked by anyone, forged by no one, yet loses its validity if any bit of the message is altered. It therefore provides authentication of both the identity of the signer and the contents of the signed message. It is the perfect tool for electronically signed business contracts, electronic checks, electronic purchase orders, and other electronic communications that parties wish to authenticate.

The RSA public-key cryptosystem relies on the dramatic difference between the ease of finding large prime numbers and the difficulty of factoring the product of two large prime numbers. Section 31.8 describes an efficient procedure for finding large prime numbers.

Public-key cryptosystems

In a public-key cryptosystem, each participant has both a ***public key*** and a ***secret key***. Each key is a piece of information. For example, in the RSA cryptosystem, each key consists of a pair of integers. The participants "Alice" and "Bob" are traditionally used in cryptography examples. We denote the public keys for Alice and Bob as P_A and P_B, respectively, and likewise the secret keys are S_A for Alice and S_B for Bob.

Each participant creates his or her own public and secret keys. Secret keys are kept secret, but public keys can be revealed to anyone or even published. In fact, it is often convenient to assume that everyone's public key is available in a public directory, so that any participant can easily obtain the public key of any other participant.

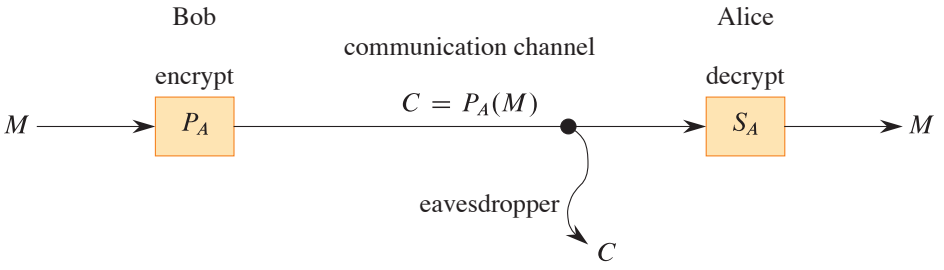

Figure 31.5 Encryption in a public key system. Bob encrypts the message M using Alice's public key P_A and transmits the resulting ciphertext $C = P_A(M)$ over a communication channel to Alice. An eavesdropper who captures the transmitted ciphertext gains no information about M. Alice receives C and decrypts it using her secret key to obtain the original message $M = S_A(C)$.

The public and secret keys specify functions that can be applied to any message. Let $\mathcal{D}$ denote the set of permissible messages. For example, $\mathcal{D}$ might be the set of all finite-length bit sequences. The simplest, and original, formulation of public-key cryptography requires one-to-one functions from $\mathcal{D}$ to itself, based on the public and secret keys. We denote the function based on Alice's public key P_A by $P_A()$ and the function based on her secret key S_A by $S_A()$. The functions $P_A()$ and $S_A()$ are thus permutations of $\mathcal{D}$. We assume that the functions $P_A()$ and $S_A()$ are efficiently computable given the corresponding keys P_A and S_A.

The public and secret keys for any participant are a "matched pair" in that they specify functions that are inverses of each other. That is,

$$M = S_A(P_A(M)) , \tag{31.35}$$

$$M = P_A(S_A(M)) \tag{31.36}$$

for any message $M \in \mathcal{D}$. Transforming M with the two keys P_A and S_A successively, in either order, yields back the original message M.

A public-key cryptosystem requires that Alice, and only Alice, be able to compute the function $S_A()$ in any practical amount of time. This assumption is crucial to keeping encrypted messages sent to Alice private and to knowing that Alice's digital signatures are authentic. Alice must keep her key S_A secret. If she does not, whoever else has access to S_A can decrypt messages intended only for Alice and can also forge her digital signature. The assumption that only Alice can reasonably compute $S_A()$ must hold even though everyone knows P_A and can compute $P_A()$, the inverse function to $S_A()$, efficiently. These requirements appear formidable, but we'll see how to satisfy them.

In a public-key cryptosystem, encryption works as shown in Figure 31.5. Suppose that Bob wishes to send Alice a message M encrypted so that it looks like

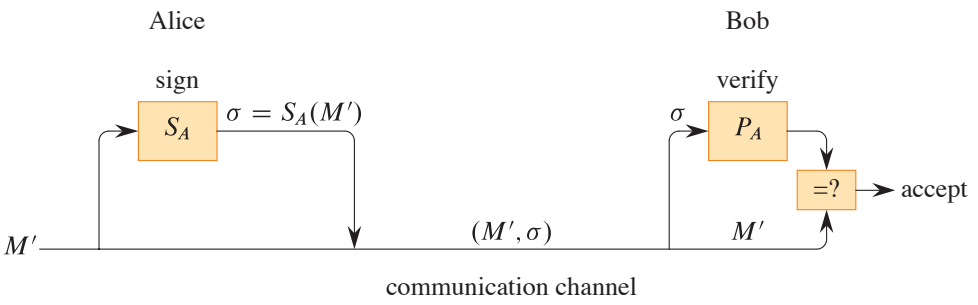

Figure 31.6 Digital signatures in a public-key system. Alice signs the message M' by appending her digital signature $\sigma = S_A(M')$ to it. She transmits the message/signature pair (M', σ) to Bob, who verifies it by checking the equation $M' = P_A(\sigma)$. If the equation holds, he accepts (M', σ) as a message that Alice has signed.

unintelligible gibberish to an eavesdropper. The scenario for sending the message goes as follows.

- Bob obtains Alice's public key P_A, perhaps from a public directory or perhaps directly from Alice.
- Bob computes the ***ciphertext*** $C = P_A(M)$ corresponding to the message M and sends C to Alice.
- When Alice receives the ciphertext C, she applies her secret key S_A to retrieve the original message: $S_A(C) = S_A(P_A(M)) = M$.

Because $S_A()$ and $P_A()$ are inverse functions, Alice can compute M from C. Because only Alice is able to compute $S_A()$, only Alice can compute M from C. Because Bob encrypts M using $P_A()$, only Alice can understand the transmitted message.

Digital signatures can be implemented within this formulation of a public-key cryptosystem. (There are other ways to construct digital signatures, but we won't go into them here.) Suppose now that Alice wishes to send Bob a digitally signed response M'. Figure 31.6 shows how the digital-signature scenario proceeds.

- Alice computes her ***digital signature*** σ for the message M' using her secret key S_A and the equation $\sigma = S_A(M')$.
- Alice sends the message/signature pair (M', σ) to Bob.
- When Bob receives (M', σ), he can verify that it originated from Alice by using Alice's public key to verify the equation $M' = P_A(\sigma)$. (Presumably, M' contains Alice's name, so that Bob knows whose public key to use.) If the equation holds, then Bob concludes that the message M' was actually signed by Alice. If

the equation fails to hold, Bob concludes either that the information he received was corrupted by transmission errors or that the pair (M', σ) is an attempted forgery.

Because a digital signature provides both authentication of the signer's identity and authentication of the contents of the signed message, it is analogous to a handwritten signature at the end of a written document.

A digital signature must be verifiable by anyone who has access to the signer's public key. A signed message can be verified by one party and then passed on to other parties who can also verify the signature. For example, the message might be an electronic check from Alice to Bob. After Bob verifies Alice's signature on the check, he can give the check to his bank, who can then also verify the signature and effect the appropriate funds transfer.

A signed message may or may not be encrypted. The message can be "in the clear" and not protected from disclosure. By composing the above protocols for encryption and for signatures, Alice can create a message to Bob that is both signed and encrypted. Alice first appends her digital signature to the message and then encrypts the resulting message/signature pair with Bob's public key. Bob decrypts the received message with his secret key to obtain both the original message and its digital signature. Bob can then verify the signature using Alice's public key. The corresponding combined process using paper-based systems would be to sign the paper document and then seal the document inside a paper envelope that is opened only by the intended recipient.

The RSA cryptosystem

In the ***RSA public-key cryptosystem***, a participant creates a public key and a secret key with the following procedure:

1. Select at random two large prime numbers p and q such that $p \neq q$. The primes p and q might be, say, 1024 bits each.
2. Compute $n = pq$.
3. Select a small odd integer e that is relatively prime to $\phi(n)$, which, by equation (31.21), equals $(p-1)(q-1)$.
4. Compute d as the multiplicative inverse of e, modulo $\phi(n)$. (Corollary 31.26 guarantees that d exists and is uniquely defined. You can use the technique of Section 31.4 to compute d, given e and $\phi(n)$.)
5. Publish the pair $P = (e, n)$ as the participant's ***RSA public key***.
6. Keep secret the pair $S = (d, n)$ as the participant's ***RSA secret key***.

For this scheme, the domain $\mathcal{D}$ is the set $\mathbb{Z}_n$. To transform a message M associated with a public key $P = (e, n)$, compute

$$P(M) = M^e \bmod n \ . \tag{31.37}$$

To transform a ciphertext C associated with a secret key $S = (d, n)$, compute

$$S(C) = C^d \bmod n \ . \tag{31.38}$$

These equations apply to both encryption and signatures. To create a signature, the signer's secret key is applied to the message to be signed, rather than to a ciphertext. To verify a signature, the public key of the signer is applied to the signature rather than to a message to be encrypted.

To implement the public-key and secret-key operations (31.37) and (31.38), you can use the procedure MODULAR-EXPONENTIATION described in Section 31.6. To analyze the running time of these operations, assume that the public key (e, n) and secret key (d, n) satisfy $\lg e = O(1)$, $\lg d \leq \beta$, and $\lg n \leq \beta$. Then, applying a public key requires $O(1)$ modular multiplications and uses $O(\beta^2)$ bit operations. Applying a secret key requires $O(\beta)$ modular multiplications, using $O(\beta^3)$ bit operations.

Theorem 31.36 (Correctness of RSA)
The RSA equations (31.37) and (31.38) define inverse transformations of $\mathbb{Z}_n$ satisfying equations (31.35) and (31.36).

Proof From equations (31.37) and (31.38), we have that for any $M \in \mathbb{Z}_n$,

$$P(S(M)) = S(P(M)) = M^{ed} \pmod{n} \ .$$

Since e and d are multiplicative inverses modulo $\phi(n) = (p-1)(q-1)$,

$$ed = 1 + k(p-1)(q-1)$$

for some integer k. But then, if $M \not\equiv 0 \pmod{p}$, we have

$$\begin{aligned} M^{ed} &= M(M^{p-1})^{k(q-1)} && \pmod{p} \\ &= M((M \bmod p)^{p-1})^{k(q-1)} && \pmod{p} \\ &= M(1)^{k(q-1)} && \pmod{p} \qquad \text{(by Theorem 31.31)} \\ &= M && \pmod{p} \ . \end{aligned}$$

Also, $M^{ed} = M \pmod{p}$ if $M = 0 \pmod{p}$. Thus,

$$M^{ed} = M \pmod{p}$$

for all M. Similarly,

$$M^{ed} = M \pmod{q}$$

for all M. Thus, by Corollary 31.29 to the Chinese remainder theorem,

$$M^{ed} = M \pmod{n}$$

for all M. ■

The security of the RSA cryptosystem rests in large part on the difficulty of factoring large integers. If an adversary can factor the modulus n in a public key, then the adversary can derive the secret key from the public key, using the knowledge of the factors p and q in the same way that the creator of the public key used them. Therefore, if factoring large integers is easy, then breaking the RSA cryptosystem is easy. The converse statement, that if factoring large integers is hard, then breaking RSA is hard, is unproven. After two decades of research, however, no easier method has been found to break the RSA public-key cryptosystem than to factor the modulus n. And factoring large integers is surprisingly difficult. By randomly selecting and multiplying together two 1024-bit primes, you can create a public key that cannot be "broken" in any feasible amount of time with current technology. In the absence of a fundamental breakthrough in the design of number-theoretic algorithms, and when implemented with care following recommended standards, the RSA cryptosystem is capable of providing a high degree of security in applications.

In order to achieve security with the RSA cryptosystem, however, you should use integers that are quite long—more than 1000 bits—to resist possible advances in the art of factoring. In 2021, RSA moduli are commonly in the range of 2048 to 4096 bits. To create moduli of such sizes, you must find large primes efficiently. Section 31.8 addresses this problem.

For efficiency, RSA is often used in a "hybrid" or "key-management" mode with fast cryptosystems that are not public-key cryptosystems. With such a ***symmetric-key*** system, the encryption and decryption keys are identical. If Alice wishes to send a long message M to Bob privately, she selects a random key K for the fast symmetric-key cryptosystem and encrypts M using K, obtaining ciphertext C, where C is as long as M, but K is quite short. Then she encrypts K using Bob's public RSA key. Since K is short, computing $P_B(K)$ is fast (much faster than computing $P_B(M)$). She then transmits $(C, P_B(K))$ to Bob, who decrypts $P_B(K)$ to obtain K and then uses K to decrypt C, obtaining M.

A similar hybrid approach creates digital signatures efficiently. This approach combines RSA with a public ***collision-resistant hash function*** h—a function that is easy to compute but for which it is computationally infeasible to find two messages M and M' such that $h(M) = h(M')$. The value $h(M)$ is a short (say, 256-bit) "fingerprint" of the message M. If Alice wishes to sign a message M, she first applies h to M to obtain the fingerprint $h(M)$, which she then encrypts with her secret key. She sends $(M, S_A(h(M)))$ to Bob as her signed version of M.

Bob can verify the signature by computing $h(M)$ and verifying that P_A applied to $S_A(h(M))$ as received equals $h(M)$. Because no one can create two messages with the same fingerprint, it is computationally infeasible to alter a signed message and preserve the validity of the signature.

One way to distribute public keys uses ***certificates***. For example, assume that there is a "trusted authority" T whose public key is known by everyone. Alice can obtain from T a signed message (her certificate) stating that "Alice's public key is P_A." This certificate is "self-authenticating" since everyone knows P_T. Alice can include her certificate with her signed messages, so that the recipient has Alice's public key immediately available in order to verify her signature. Because her key was signed by T, the recipient knows that Alice's key is really Alice's.

Exercises

31.7-1

Consider an RSA key set with $p = 11$, $q = 29$, $n = 319$, and $e = 3$. What value of d should be used in the secret key? What is the encryption of the message $M = 100$?

31.7-2

Prove that if Alice's public exponent e is 3 and an adversary obtains Alice's secret exponent d, where $0 < d < \phi(n)$, then the adversary can factor Alice's modulus n in time polynomial in the number of bits in n. (Although you are not asked to prove it, you might be interested to know that this result remains true even if the condition $e = 3$ is removed. See Miller [327].)

★ ***31.7-3***

Prove that RSA is multiplicative in the sense that

$$P_A(M_1)P_A(M_2) = P_A(M_1M_2) \pmod{n} .$$

Use this fact to prove that if an adversary had a procedure that could efficiently decrypt 1% of messages from $\mathbb{Z}_n$ encrypted with P_A, then the adversary could employ a probabilistic algorithm to decrypt every message encrypted with P_A with high probability.

★ 31.8 Primality testing

This section shows how to find large primes. We begin with a discussion of the density of primes, proceed to examine a plausible, but incomplete, approach to

primality testing, and then present an effective randomized primality test due to Miller and Rabin.

The density of prime numbers

Many applications, such as cryptography, call for finding large "random" primes. Fortunately, large primes are not too rare, so that it is feasible to test random integers of the appropriate size until you find one that is prime. The ***prime distribution function*** $\pi(n)$ specifies the number of primes that are less than or equal to n. For example, $\pi(10) = 4$, since there are 4 prime numbers less than or equal to 10, namely, 2, 3, 5, and 7. The prime number theorem gives a useful approximation to $\pi(n)$.

Theorem 31.37 (Prime number theorem)

$$\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1 \,. \qquad \blacksquare$$

The approximation $n / \ln n$ gives reasonably accurate estimates of $\pi(n)$ even for small n. For example, it is off by less than 6% at $n = 10^9$, where $\pi(n) = 50{,}847{,}534$ and $n / \ln n \approx 48{,}254{,}942$. (To a number theorist, 10^9 is a small number.)

The process of randomly selecting an integer n and determining whether it is prime is really just a Bernoulli trial (see Section C.4). By the prime number theorem, the probability of a success—that is, the probability that n is prime—is approximately $1 / \ln n$. The geometric distribution says how many trials must occur to obtain a success, and by equation (C.36) on page 1197, the expected number of trials is approximately $\ln n$. Thus, in order to find a prime that has the same length as n by testing integers chosen randomly near n, the expected number examined would be approximately $\ln n$. For example, the expectation is that finding a 1024-bit prime would require testing approximately $\ln 2^{1024} \approx 710$ randomly chosen 1024-bit numbers for primality. (Of course, to cut this figure in half, choose only odd integers.)

The remainder of this section shows how to determine whether a large odd integer n is prime. For notational convenience, we assume that n has the prime factorization

$$n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r} \,,$$

where $r \geq 1$, $p_1, p_2, \ldots, p_r$ are the prime factors of n, and $e_1, e_2, \ldots, e_r$ are positive integers. The integer n is prime if and only if $r = 1$ and $e_1 = 1$.

One simple approach to the problem of testing for primality is ***trial division***: try dividing n by each integer $2, 3, 5, 7, 9, \ldots, \lfloor \sqrt{n} \rfloor$, skipping even integers greater

than 2. We can conclude that n is prime if and only if none of the trial divisors divides n. Assuming that each trial division takes constant time, the worst-case running time is $\Theta(\sqrt{n})$, which is exponential in the length of n. (Recall that if n is encoded in binary using β bits, then $\beta = \lceil \lg(n+1) \rceil$, and so $\sqrt{n} = \Theta(2^{\beta/2})$.) Thus, trial division works well only if n is very small or happens to have a small prime factor. When it works, trial division has the advantage that it not only determines whether n is prime or composite, it also determines one of n's prime factors if n is composite.

This section focuses on finding out whether a given number n is prime. If n is composite, we won't worry about finding its prime factorization. Computing the prime factorization of a number is computationally expensive. You might be surprised that it turns out to be much easier to ascertain whether a given number is prime than it is to determine the prime factorization of the number if it is not prime.

Pseudoprimality testing

We'll start with a method for primality testing that "almost works" and, in fact, is good enough for many practical applications. Later on, we'll refine this method to remove the small defect. Let $\mathbb{Z}_n^+$ denote the nonzero elements of $\mathbb{Z}_n$:

$$\mathbb{Z}_n^+ = \{1, 2, \ldots, n-1\} .$$

If n is prime, then $\mathbb{Z}_n^+ = \mathbb{Z}_n^*$.

We say that n is a ***base-a pseudoprime*** if n is composite and

$$a^{n-1} = 1 \pmod{n} . \tag{31.39}$$

Fermat's theorem (Theorem 31.31 on page 932) implies that if n is prime, then n satisfies equation (31.39) for every a in $\mathbb{Z}_n^+$. Thus, if there is any $a \in \mathbb{Z}_n^+$ such that n does *not* satisfy equation (31.39), then n is certainly composite. Surprisingly, the converse *almost* holds, so that this criterion forms an almost perfect test for primality. Instead of trying every value of $a \in \mathbb{Z}_n^+$, test to see whether n satisfies equation (31.39) for just $a = 2$. If not, then declare n to be composite by returning COMPOSITE. Otherwise, return PRIME, guessing that n is prime (when, in fact, all we know is that n is either prime or a base-2 pseudoprime).

The procedure PSEUDOPRIME on the next page pretends in this manner to check whether n is prime. It uses the procedure MODULAR-EXPONENTIATION from Section 31.6. It assumes that the input n is an odd integer greater than 2. This procedure can make errors, but only of one type. That is, if it says that n is composite, then it is always correct. If it says that n is prime, however, then it makes an error only if n is a base-2 pseudoprime.

How often does PSEUDOPRIME err? Surprisingly rarely. There are only 22 values of n less than 10,000 for which it errs, the first four of which are 341, 561, 645,

```
PSEUDOPRIME(n)
1  if MODULAR-EXPONENTIATION(2, n − 1, n) ≠ 1 (mod n)
2       return COMPOSITE        // definitely
3  else return PRIME            // we hope!
```

and 1105. We won't prove it, but the probability that this program makes an error on a randomly chosen β-bit number goes to 0 as β approaches ∞. Using more precise estimates due to Pomerance [361] of the number of base-2 pseudoprimes of a given size, a randomly chosen 512-bit number that is called prime by PSEUDOPRIME has less than one chance in 10^{20} of being a base-2 pseudoprime, and a randomly chosen 1024-bit number that is called prime has less than one chance in 10^{41} of being a base-2 pseudoprime. Thus, if you are merely trying to find a large prime for some application, for all practical purposes you almost never go wrong by choosing large numbers at random until one of them causes PSEUDOPRIME to return PRIME. But when the numbers being tested for primality are not randomly chosen, you might need a better approach for testing primality. As we'll see, a little more cleverness, and some randomization, will yield a primality-testing method that works well on all inputs.

Since PSEUDOPRIME checks equation (31.39) for only $a = 2$, you might think that you could eliminate all the errors by simply checking equation (31.39) for a second base number, say $a = 3$. Better yet, you could check equation (31.39) for even more values of a. Unfortunately, even checking for several values of a does not eliminate all errors, because there exist composite integers n, known as ***Carmichael numbers***, that satisfy equation (31.39) for *all* $a \in \mathbb{Z}_n^*$. (The equation does fail when $\gcd(a, n) > 1$—that is, when $a \notin \mathbb{Z}_n^*$—but demonstrating that n is composite by finding such an a can be difficult if n has only large prime factors.) The first three Carmichael numbers are 561, 1105, and 1729. Carmichael numbers are extremely rare. For example, only 255 of them are less than 100,000,000. Exercise 31.8-2 helps explain why they are so rare.

Let's see how to improve the primality test so that Carmichael numbers won't fool it.

The Miller-Rabin randomized primality test

The Miller-Rabin primality test overcomes the problems of the simple procedure PSEUDOPRIME with two modifications:

- It tries several randomly chosen base values a instead of just one base value.
- While computing each modular exponentiation, it looks for a nontrivial square root of 1, modulo n, during the final set of squarings. If it finds one, it stops

and returns COMPOSITE. Corollary 31.35 from Section 31.6 justifies detecting composites in this manner.

The pseudocode for the Miller-Rabin primality test appears in the procedures MILLER-RABIN and WITNESS. The input $n > 2$ to MILLER-RABIN is the odd number to be tested for primality, and s is the number of randomly chosen base values from $\mathbb{Z}_n^+$ to be tried. The code uses the random-number generator RANDOM described on page 129: RANDOM$(2, n-2)$ returns a randomly chosen integer a satisfying $2 \le a \le n-2$. (This range of values avoids having $a = \pm 1 \pmod{n}$.) The call of the auxiliary procedure WITNESS(a, n) returns TRUE if and only if a is a "witness" to the compositeness of n—that is, if it is possible using a to prove (in a manner that we will see) that n is composite. The test WITNESS(a, n) is an extension of, but more effective than, the test in equation (31.39) that formed the basis for PSEUDOPRIME, using $a = 2$.

Let's first understand how WITNESS works, and then we'll see how the Miller-Rabin primality test uses it. Let $n - 1 = 2^t u$ where $t \ge 1$ and u is odd. That is, the binary representation of $n-1$ is the binary representation of the odd integer u followed by exactly t zeros. Therefore, $a^{n-1} = (a^u)^{2^t} \pmod{n}$, so that one way to compute $a^{n-1} \bmod n$ is to first compute $a^u \bmod n$ and then square the result t times successively.

```
MILLER-RABIN(n, s)                    // n > 2 is odd
1  for j = 1 to s
2      a = RANDOM(2, n − 2)
3      if WITNESS(a, n)
4          return COMPOSITE           // definitely
5  return PRIME                       // almost surely

WITNESS(a, n)
1  let t and u be such that t ≥ 1, u is odd, and n − 1 = 2^t u
2  x_0 = MODULAR-EXPONENTIATION(a, u, n)
3  for i = 1 to t
4      x_i = x_{i−1}^2 mod n
5      if x_i == 1 and x_{i−1} ≠ 1 and x_{i−1} ≠ n − 1
6          return TRUE                // found a nontrivial square root of 1
7  if x_t ≠ 1
8      return TRUE                    // composite, as in PSEUDOPRIME
9  return FALSE
```

This pseudocode for WITNESS computes $a^{n-1} \bmod n$ by first computing the value $x_0 = a^u \bmod n$ in line 2 and then repeatedly squaring the result t times in the **for** loop of lines 3–6. By induction on i, the sequence $x_0, x_1, \ldots, x_t$ of values computed satisfies the equation $x_i = a^{2^i u} \pmod n$ for $i = 0, 1, \ldots, t$, so that in particular $x_t = a^{n-1} \pmod n$. After line 4 performs a squaring step, however, the loop will terminate early if lines 5–6 detect that a nontrivial square root of 1 has just been discovered. (We'll explain these tests shortly.) If so, the procedure stops and returns TRUE. Lines 7–8 return TRUE if the value computed for $x_t = a^{n-1} \pmod n$ is not equal to 1, just as the PSEUDOPRIME procedure returns COMPOSITE in this case. Line 9 returns FALSE if lines 6 or 8 have not returned TRUE.

The following lemma proves the correctness of WITNESS.

Lemma 31.38
If WITNESS(a, n) returns TRUE, then a proof that n is composite can be constructed using a as a witness.

Proof If WITNESS returns TRUE from line 8, it's because line 7 determined that $x_t = a^{n-1} \bmod n \neq 1$. If n is prime, however, Fermat's theorem (Theorem 31.31) says that $a^{n-1} = 1 \pmod n$ for all $a \in \mathbb{Z}_n^*$. Since $\mathbb{Z}_n^+ = \mathbb{Z}_n^*$ if n is prime, Fermat's theorem also says that $a^{n-1} = 1 \pmod n$ for all $a \in \mathbb{Z}_n^+$. Therefore, n cannot be prime, and the equation $a^{n-1} \bmod n \neq 1$ proves this fact.

If WITNESS returns TRUE from line 6, then it has discovered that x_{i-1} is a nontrivial square root of 1, modulo n, since we have that $x_{i-1} \neq \pm 1 \pmod n$ yet $x_i = x_{i-1}^2 = 1 \pmod n$. Corollary 31.35 on page 934 states that only if n is composite can there exist a nontrivial square root of 1, modulo n, so that demonstrating that x_{i-1} is a nontrivial square root of 1, modulo n proves that n is composite. ■

Thus, if the call WITNESS(a, n) returns TRUE, then n is surely composite, and the witness a, along with the reason that the procedure returns TRUE (did it return from line 6 or from line 8?), provides a proof that n is composite.

Let's explore an alternative view of the behavior of WITNESS as a function of the sequence $X = \langle x_0, x_1, \ldots, x_t \rangle$. We'll find this view useful later on, when we analyze the error rate of the Miller-Rabin primality test. Note that if $x_i = 1$ for some $0 \leq i < t$, WITNESS might not compute the rest of the sequence. If it were to do so, however, each value $x_{i+1}, x_{i+2}, \ldots, x_t$ would be 1, so we can consider these positions in the sequence X as being all 1s. There are four cases:

1. $X = \langle \ldots, d \rangle$, where $d \neq 1$: the sequence X does not end in 1. Return TRUE in line 8, since a is a witness to the compositeness of n (by Fermat's Theorem).

2. $X = \langle 1, 1, \ldots, 1 \rangle$: the sequence X is all 1s. Return FALSE, since a is not a witness to the compositeness of n.
3. $X = \langle \ldots, -1, 1, \ldots, 1 \rangle$: the sequence X ends in 1, and the last non-1 is equal to -1. Return FALSE, since a is not a witness to the compositeness of n.
4. $X = \langle \ldots, d, 1, \ldots, 1 \rangle$, where $d \neq \pm 1$: the sequence X ends in 1, but the last non-1 is not -1. Return TRUE in line 6: a is a witness to the compositeness of n, since d is a nontrivial square root of 1.

Now, let's examine the Miller-Rabin primality test based on how it uses the WITNESS procedure. As before, assume that n is an odd integer greater than 2.

The procedure MILLER-RABIN is a probabilistic search for a proof that n is composite. The main loop (beginning on line 1) picks up to s random values of a from $\mathbb{Z}_n^+$, except for 1 and $n-1$ (line 2). If it picks a value of a that is a witness to the compositeness of n, then MILLER-RABIN returns COMPOSITE on line 4. Such a result is always correct, by the correctness of WITNESS. If MILLER-RABIN finds no witness in s trials, then the procedure assumes that it found no witness because no witnesses exist, and therefore it assumes that n is prime. We'll see that this result is likely to be correct if s is large enough, but there is still a tiny chance that the procedure could be unlucky in its choice of s random values of a, so that even though the procedure failed to find a witness, at least one witness exists.

To illustrate the operation of MILLER-RABIN, let n be the Carmichael number 561, so that $n - 1 = 560 = 2^4 \cdot 35$, $t = 4$, and $u = 35$. If the procedure chooses $a = 7$ as a base, the column for $b = 35$ in Figure 31.4 (Section 31.6) shows that WITNESS computes $x_0 = a^{35} = 241 \pmod{561}$. Because of how the MODULAR-EXPONENTIATION procedure operates recursively on its parameter b, the first four columns in Figure 31.4 represent the factor 2^4 of 560—the rightmost four zeros in the binary representation of 560—reading these four zeros from right to left in the binary representation. Thus WITNESS computes the sequence $X = \langle 241, 298, 166, 67, 1 \rangle$. Then, in the last squaring step, WITNESS discovers that a^{280} is a nontrivial square root of 1 since $a^{280} = 67 \pmod{n}$ and $(a^{280})^2 = a^{560} = 1 \pmod{n}$. Therefore, $a = 7$ is a witness to the compositeness of n, WITNESS$(7, n)$ returns TRUE, and MILLER-RABIN returns COMPOSITE.

If n is a β-bit number, MILLER-RABIN requires $O(s\beta)$ arithmetic operations and $O(s\beta^3)$ bit operations, since it requires asymptotically no more work than s modular exponentiations.

Error rate of the Miller-Rabin primality test

If MILLER-RABIN returns PRIME, then there is a very slim chance that it has made an error. Unlike PSEUDOPRIME, however, the chance of error does not depend on n: there are no bad inputs for this procedure. Rather, it depends on the size of s

and the "luck of the draw" in choosing base values a. Moreover, since each test is more stringent than a simple check of equation (31.39), we can expect on general principles that the error rate should be small for randomly chosen integers n. The following theorem presents a more precise argument.

Theorem 31.39
If n is an odd composite number, then the number of witnesses to the compositeness of n is at least $(n-1)/2$.

Proof The proof shows that the number of nonwitnesses is at most $(n-1)/2$, which implies the theorem.

We start by claiming that any nonwitness must be a member of $\mathbb{Z}_n^*$. Why? Consider any nonwitness a. It must satisfy $a^{n-1} = 1 \pmod{n}$ or, equivalently, $a \cdot a^{n-2} = 1 \pmod{n}$. Thus the equation $ax = 1 \pmod{n}$ has a solution, namely a^{n-2}. By Corollary 31.21 on page 924, $\gcd(a,n) \mid 1$, which in turn implies that $\gcd(a,n) = 1$. Therefore, a is a member of $\mathbb{Z}_n^*$, and all nonwitnesses belong to $\mathbb{Z}_n^*$.

To complete the proof, we show that not only are all nonwitnesses contained in $\mathbb{Z}_n^*$, they are all contained in a proper subgroup B of $\mathbb{Z}_n^*$ (recall that B is a *proper* subgroup of $\mathbb{Z}_n^*$ when B is subgroup of $\mathbb{Z}_n^*$ but B is not equal to $\mathbb{Z}_n^*$). By Corollary 31.16 on page 921, we then have $|B| \le |\mathbb{Z}_n^*|/2$. Since $|\mathbb{Z}_n^*| \le n-1$, we obtain $|B| \le (n-1)/2$. Therefore, if all nonwitnesses are contained in a proper subgroup of $\mathbb{Z}_n^*$, then the number of nonwitnesses is at most $(n-1)/2$, so that the number of witnesses must be at least $(n-1)/2$.

To find a proper subgroup B of $\mathbb{Z}_n^*$ containing all of the nonwitnesses, we consider two cases.

Case 1: There exists an $x \in \mathbb{Z}_n^*$ such that

$$x^{n-1} \neq 1 \pmod{n} .$$

In other words, n is not a Carmichael number. Since, as noted earlier, Carmichael numbers are extremely rare, case 1 is the more typical case (e.g., when n has been chosen randomly and is being tested for primality).

Let $B = \{b \in \mathbb{Z}_n^* : b^{n-1} = 1 \pmod{n}\}$. The set B must be nonempty, since $1 \in B$. The set B is closed under multiplication modulo n, and so B is a subgroup of $\mathbb{Z}_n^*$ by Theorem 31.14. Every nonwitness belongs to B, since a nonwitness a satisfies $a^{n-1} = 1 \pmod{n}$. Since $x \in \mathbb{Z}_n^* - B$, we have that B is a proper subgroup of $\mathbb{Z}_n^*$.

Case 2: For all $x \in \mathbb{Z}_n^*$,

$$x^{n-1} = 1 \pmod{n} . \tag{31.40}$$

In other words, n is a Carmichael number. This case is extremely rare in practice. Unlike a pseudoprimality test, however, the Miller-Rabin test can efficiently determine that Carmichael numbers are composite, as we're about to see.

In this case, n cannot be a prime power. To see why, suppose to the contrary that $n = p^e$, where p is a prime and $e > 1$. We derive a contradiction as follows. Since we assume that n is odd, p must also be odd. Theorem 31.32 on page 933 implies that $\mathbb{Z}_n^*$ is a cyclic group: it contains a generator g such that $\mathrm{ord}_n(g) = |\mathbb{Z}_n^*| = \phi(n) = p^e(1 - 1/p) = (p-1)p^{e-1}$. (The formula for $\phi(n)$ comes from equation (31.21) on page 920.) By equation (31.40), we have $g^{n-1} = 1 \pmod{n}$. Then the discrete logarithm theorem (Theorem 31.33 on page 933, taking $y = 0$) implies that $n - 1 = 0 \pmod{\phi(n)}$, or

$$(p-1)p^{e-1} \mid p^e - 1 .$$

This statement is a contradiction for $e > 1$, since $(p-1)p^{e-1}$ is divisible by the prime p, but $p^e - 1$ is not. Thus n is not a prime power.

Since the odd composite number n is not a prime power, we decompose it into a product $n_1 n_2$, where n_1 and n_2 are odd numbers greater than 1 that are relatively prime to each other. (There may be several ways to decompose n, and it does not matter which one we choose. For example, if $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, then we can choose $n_1 = p_1^{e_1}$ and $n_2 = p_2^{e_2} p_3^{e_3} \cdots p_r^{e_r}$.)

Recall that t and u are such that $n - 1 = 2^t u$, where $t \geq 1$ and u is odd, and that for an input a, the procedure WITNESS computes the sequence

$$X = \langle a^u, a^{2u}, a^{2^2 u}, \ldots, a^{2^t u} \rangle$$

where all computations are performed modulo n.

Let us call a pair (v, j) of integers ***acceptable*** if $v \in \mathbb{Z}_n^*$, $j \in \{0, 1, \ldots, t\}$, and

$$v^{2^j u} = -1 \pmod{n} .$$

Acceptable pairs certainly exist, since u is odd. Choose $v = n - 1$ and $j = 0$, and let $u = 2k + 1$, so that $v^{2^j u} = (n-1)^u = (n-1)^{2k+1}$. Taking this number modulo n gives $(n-1)^{2k+1} = (n-1)^{2k} \cdot (n-1) = (-1)^{2k} \cdot -1 = -1 \pmod{n}$. Thus, $(n-1, 0)$ is an acceptable pair. Now pick the largest possible j such that there exists an acceptable pair (v, j), and fix v so that (v, j) is an acceptable pair. Let

$$B = \{x \in \mathbb{Z}_n^* : x^{2^j u} = \pm 1 \pmod{n}\} .$$

Since B is closed under multiplication modulo n, it is a subgroup of $\mathbb{Z}_n^*$. By Theorem 31.15 on page 921, therefore, $|B|$ divides $|\mathbb{Z}_n^*|$. Every nonwitness must be a member of B, since the sequence X produced by a nonwitness must either be all 1s or else contain a -1 no later than the jth position, by the maximality of j.

(If (a, j') is acceptable, where a is a nonwitness, we must have $j' \le j$ by how we chose j.)

We now use the existence of v to demonstrate that there exists a $w \in \mathbb{Z}_n^* - B$, and hence that B is a proper subgroup of $\mathbb{Z}_n^*$. Since $v^{2^j u} = -1 \pmod{n}$, we also have $v^{2^j u} = -1 \pmod{n_1}$ by Corollary 31.29 to the Chinese remainder theorem. By Corollary 31.28, there exists a w simultaneously satisfying the equations

$$w = v \pmod{n_1},$$
$$w = 1 \pmod{n_2}.$$

Therefore,

$$w^{2^j u} = -1 \pmod{n_1},$$
$$w^{2^j u} = 1 \pmod{n_2}.$$

Corollary 31.29 gives that $w^{2^j u} \not\equiv 1 \pmod{n_1}$ implies $w^{2^j u} \not\equiv 1 \pmod{n}$ and also that $w^{2^j u} \not\equiv -1 \pmod{n_2}$ implies $w^{2^j u} \not\equiv -1 \pmod{n}$. Hence, we conclude that $w^{2^j u} \not\equiv \pm 1 \pmod{n}$, and so $w \notin B$.

It remains to show that $w \in \mathbb{Z}_n^*$. We start by working separately modulo n_1 and modulo n_2. Working modulo n_1, since $v \in \mathbb{Z}_n^*$, we have that $\gcd(v, n) = 1$. Also, we have $\gcd(v, n_1) = 1$, since if v does not have any common divisors with n, then it certainly does not have any common divisors with n_1. Since $w = v \pmod{n_1}$, we see that $\gcd(w, n_1) = 1$. Working modulo n_2, we have $w = 1 \pmod{n_2}$ implies $\gcd(w, n_2) = 1$ by Exercise 31.2-3. Since $\gcd(w, n_1) = 1$ and $\gcd(w, n_2) = 1$, Theorem 31.6 on page 908 yields $\gcd(w, n_1 n_2) = \gcd(w, n) = 1$. That is, $w \in \mathbb{Z}_n^*$.

Therefore, we have $w \in \mathbb{Z}_n^* - B$, and we can conclude in case 2 that B, which includes all nonwitnesses, is a proper subgroup of $\mathbb{Z}_n^*$ and therefore has size at most $(n-1)/2$.

In either case, the number of witnesses to the compositeness of n is at least $(n-1)/2$. ■

Theorem 31.40

For any odd integer $n > 2$ and positive integer s, the probability that MILLER-RABIN(n, s) errs is at most 2^{-s}.

Proof By Theorem 31.39, if n is composite, then each execution of the **for** loop of lines 1–4 of MILLER-RABIN has a probability of at least $1/2$ of discovering a witness to the compositeness of n. MILLER-RABIN makes an error only if it is so unlucky as to miss discovering a witness to the compositeness of n on each of the s iterations of the main loop. The probability of such a sequence of misses is at most 2^{-s}. ■

If n is prime, MILLER-RABIN always reports PRIME, and if n is composite, the chance that MILLER-RABIN reports PRIME is at most 2^{-s}.

When applying MILLER-RABIN to a large randomly chosen integer n, however, we need to consider as well the prior probability that n is prime, in order to correctly interpret MILLER-RABIN's result. Suppose that we fix a bit length β and choose at random an integer n of length β bits to be tested for primality, so that $\beta \approx \lg n \approx 1.443 \ln n$. Let A denote the event that n is prime. By the prime number theorem (Theorem 31.37), the probability that n is prime is approximately

$$\begin{aligned} \Pr\{A\} &\approx 1/\ln n \\ &\approx 1.443/\beta. \end{aligned}$$

Now let B denote the event that MILLER-RABIN returns PRIME. We have that $\Pr\left\{\overline{B} \mid A\right\} = 0$ (or equivalently, that $\Pr\{B \mid A\} = 1$) and $\Pr\left\{B \mid \overline{A}\right\} \leq 2^{-s}$ (or equivalently, that $\Pr\left\{\overline{B} \mid \overline{A}\right\} > 1 - 2^{-s}$).

But what is $\Pr\{A \mid B\}$, the probability that n is prime, given that MILLER-RABIN has returned PRIME? By the alternate form of Bayes's theorem (equation (C.20) on page 1189) and approximating $\Pr\left\{B \mid \overline{A}\right\}$ by 2^{-s}, we have

$$\begin{aligned} \Pr\{A \mid B\} &= \frac{\Pr\{A\}\Pr\{B \mid A\}}{\Pr\{A\}\Pr\{B \mid A\} + \Pr\left\{\overline{A}\right\}\Pr\left\{B \mid \overline{A}\right\}} \\ &\approx \frac{(1/\ln n) \cdot 1}{(1/\ln n) \cdot 1 + (1 - 1/\ln n) \cdot 2^{-s}} \\ &\approx \frac{1}{1 + 2^{-s}(\ln n - 1)} . \end{aligned}$$

This probability does not exceed $1/2$ until s exceeds $\lg(\ln n - 1)$. Intuitively, that many initial trials are needed just for the confidence derived from failing to find a witness to the compositeness of n to overcome the prior bias in favor of n being composite. For a number with $\beta = 1024$ bits, this initial testing requires about

$$\begin{aligned} \lg(\ln n - 1) &\approx \lg(\beta/1.443) \\ &\approx 9 \end{aligned}$$

trials. In any case, choosing $s = 50$ should suffice for almost any imaginable application.

In fact, the situation is much better. If you are trying to find large primes by applying MILLER-RABIN to large randomly chosen odd integers, then choosing a small value of s (say 3) is unlikely to lead to erroneous results, though we won't prove it here. The reason is that for a randomly chosen odd composite integer n, the expected number of nonwitnesses to the compositeness of n is likely to be considerably smaller than $(n-1)/2$.

If the integer n is not chosen randomly, however, the best that can be proven is that the number of nonwitnesses is at most $(n-1)/4$, using an improved version of Theorem 31.39. Furthermore, there do exist integers n for which the number of nonwitnesses is $(n-1)/4$.

Exercises

31.8-1
Prove that if an odd integer $n > 1$ is not a prime or a prime power, then there exists a nontrivial square root of 1, modulo n.

★ ***31.8-2***
It is possible to strengthen Euler's theorem (Theorem 31.30) slightly to the form

$$a^{\lambda(n)} = 1 \pmod{n} \quad \text{for all } a \in \mathbb{Z}_n^* ,$$

where $n = p_1^{e_1} \cdots p_r^{e_r}$ and $\lambda(n)$ is defined by

$$\lambda(n) = \operatorname{lcm}(\phi(p_1^{e_1}), \ldots, \phi(p_r^{e_r})) .$$

Prove that $\lambda(n) \mid \phi(n)$. A composite number n is a Carmichael number if $\lambda(n) \mid n-1$. The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$, for which $\lambda(n) = \operatorname{lcm}(2, 10, 16) = 80$, which divides 560. Prove that Carmichael numbers must be both "square-free" (not divisible by the square of any prime) and the product of at least three primes. (For this reason, they are not common.)

31.8-3
Prove that if x is a nontrivial square root of 1, modulo n, then $\gcd(x-1, n)$ and $\gcd(x+1, n)$ are both nontrivial divisors of n.

Problems

31-1 Binary gcd algorithm
Most computers can perform the operations of subtraction, testing the parity (odd or even) of a binary integer, and halving more quickly than computing remainders. This problem investigates the ***binary gcd algorithm***, which avoids the remainder computations used in Euclid's algorithm.

a. Prove that if a and b are both even, then $\gcd(a, b) = 2 \cdot \gcd(a/2, b/2)$.

b. Prove that if a is odd and b is even, then $\gcd(a, b) = \gcd(a, b/2)$.

c. Prove that if a and b are both odd, then $\gcd(a, b) = \gcd((a-b)/2, b)$.

d. Design an efficient binary gcd algorithm for input integers a and b, where $a \geq b$, that runs in $O(\lg a)$ time. Assume that each subtraction, parity test, and halving takes unit time.

31-2 Analysis of bit operations in Euclid's algorithm

a. Consider the ordinary "paper and pencil" algorithm for long division: dividing a by b, which yields a quotient q and remainder r. Show that this method requires $O((1 + \lg q) \lg b)$ bit operations.

b. Define $\mu(ab) = (1 + \lg a)(1 + \lg b)$. Show that the number of bit operations performed by EUCLID in reducing the problem of computing $\gcd(a, b)$ to that of computing $\gcd(b, a \bmod b)$ is at most $c(\mu(a, b) - \mu(b, a \bmod b))$ for some sufficiently large constant $c > 0$.

c. Show that EUCLID(a, b) requires $O(\mu(a, b))$ bit operations in general and $O(\beta^2)$ bit operations when applied to two β-bit inputs.

31-3 Three algorithms for Fibonacci numbers

This problem compares the efficiency of three methods for computing the nth Fibonacci number F_n, given n. Assume that the cost of adding, subtracting, or multiplying two numbers is $O(1)$, independent of the size of the numbers.

a. Show that the running time of the straightforward recursive method for computing F_n based on recurrence (3.31) on page 69 is exponential in n. (See, for example, the FIB procedure on page 751.)

b. Show how to compute F_n in $O(n)$ time using memoization.

c. Show how to compute F_n in $O(\lg n)$ time using only integer addition and multiplication. (*Hint:* Consider the matrix $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ and its powers.)

d. Assume now that adding two β-bit numbers takes $\Theta(\beta)$ time and that multiplying two β-bit numbers takes $\Theta(\beta^2)$ time. What is the running time of these three methods under this more reasonable cost measure for the elementary arithmetic operations?

31-4 Quadratic residues

Let p be an odd prime. A number $a \in Z_p^*$ is a ***quadratic residue*** modulo p, if the equation $x^2 = a \pmod{p}$ has a solution for the unknown x.

a. Show that there are exactly $(p - 1)/2$ quadratic residues, modulo p.

b. If p is prime, we define the ***Legendre symbol*** $\left(\frac{a}{p}\right)$, for $a \in \mathbb{Z}_p^*$, to be 1 if a is a quadratic residue, modulo p, and -1 otherwise. Prove that if $a \in \mathbb{Z}_p^*$, then

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p} .$$

Give an efficient algorithm that determines whether a given number a is a quadratic residue, modulo p. Analyze the efficiency of your algorithm.

c. Prove that if p is a prime of the form $4k + 3$ and a is a quadratic residue in $\mathbb{Z}_p^*$, then $a^{k+1} \bmod p$ is a square root of a, modulo p. How much time is required to find the square root of a quadratic residue a, modulo p?

d. Describe an efficient randomized algorithm for finding a nonquadratic residue, modulo an arbitrary prime p, that is, a member of $\mathbb{Z}_p^*$ that is not a quadratic residue. How many arithmetic operations does your algorithm require on average?

Chapter notes

Knuth [260] contains a good discussion of algorithms for finding the greatest common divisor, as well as other basic number-theoretic algorithms. Dixon [121] gives an overview of factorization and primality testing. Bach [33], Riesel [378], and Bach and Shallit [34] provide overviews of the basics of computational number theory; Shoup [411] provides a more recent survey. The conference proceedings edited by Pomerance [362] contains several excellent survey articles.

Knuth [260] discusses the origin of Euclid's algorithm. It appears in Book 7, Propositions 1 and 2, of the Greek mathematician Euclid's *Elements*, which was written around 300 B.C.E. Euclid's description may have been derived from an algorithm due to Eudoxus around 375 B.C.E. Euclid's algorithm may hold the honor of being the oldest nontrivial algorithm, rivaled only by an algorithm for multiplication known to the ancient Egyptians. Shallit [407] chronicles the history of the analysis of Euclid's algorithm.

Knuth attributes a special case of the Chinese remainder theorem (Theorem 31.27) to the Chinese mathematician Sun-Tsŭ, who lived sometime between 200 B.C.E. and 200 C.E.—the date is quite uncertain. The same special case was given by the Greek mathematician Nichomachus around 100 C.E. It was generalized by Qin Jiushao in 1247. The Chinese remainder theorem was finally stated and proved in its full generality by L. Euler in 1734.

The randomized primality-testing algorithm presented here is due to Miller [327] and Rabin [373] and is the fastest randomized primality-testing algorithm known,

to within constant factors. The proof of Theorem 31.40 is a slight adaptation of one suggested by Bach [32]. A proof of a stronger result for MILLER-RABIN was given by Monier [332, 333]. For many years primality-testing was the classic example of a problem where randomization appeared to be necessary to obtain an efficient (polynomial-time) algorithm. In 2002, however, Agrawal, Kayal, and Saxena [4] surprised everyone with their deterministic polynomial-time primality-testing algorithm. Until then, the fastest deterministic primality testing algorithm known, due to Cohen and Lenstra [97], ran in $(\lg n)^{O(\lg \lg \lg n)}$ time on input n, which is just slightly superpolynomial. Nonetheless, for practical purposes, randomized primality-testing algorithms remain more efficient and are generally preferred.

Beauchemin, Brassard, Crépeau, Goutier, and Pomerance [40] nicely discuss the problem of finding large "random" primes.

The concept of a public-key cryptosystem is due to Diffie and Hellman [115]. The RSA cryptosystem was proposed in 1977 by Rivest, Shamir, and Adleman [380]. Since then, the field of cryptography has blossomed. Our understanding of the RSA cryptosystem has deepened, and modern implementations use significant refinements of the basic techniques presented here. In addition, many new techniques have been developed for proving cryptosystems to be secure. For example, Goldwasser and Micali [190] show that randomization can be an effective tool in the design of secure public-key encryption schemes. For signature schemes, Goldwasser, Micali, and Rivest [191] present a digital-signature scheme for which every conceivable type of forgery is provably as difficult as factoring. Katz and Lindell [253] provide an overview of modern cryptography.

The best algorithms for factoring large numbers have a running time that grows roughly exponentially with the cube root of the length of the number n to be factored. The general number-field sieve factoring algorithm (as developed by Buhler, Lenstra, and Pomerance [77] as an extension of the ideas in the number-field sieve factoring algorithm by Pollard [360] and Lenstra et al. [295] and refined by Coppersmith [102] and others) is perhaps the most efficient such algorithm in general for large inputs. Although it is difficult to give a rigorous analysis of this algorithm, under reasonable assumptions we can derive a running-time estimate of $L(1/3, n)^{1.902+o(1)}$, where $L(\alpha, n) = e^{(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}}$.

The elliptic-curve method due to Lenstra [296] may be more effective for some inputs than the number-field sieve method, since it can find a small prime factor p quite quickly. With this method, the time to find p is estimated to be $L(1/2, p)^{\sqrt{2}+o(1)}$.