8

Some Number Theory and Algorithms

Number theory has been a subject of extensive research that has continuously occupied a central position of pure mathematics since mathematics became a subject of study. Until the 1970s it remained a pure science in that its connections and applications were primarily to other branches of mathematics. When public key cryptography was discovered in the 1970s, however, number theory suddenly extended itself into a major applied subject, and a renewed interest has reenergized the subject to a new significance. Much of the security in contemporary public key cryptosystems relies on certain very difficult number theoretic problems that continue to remain recalcitrant. This chapter develops some number theoretic concepts and algorithms that, although fascinating on their own, have turned out particularly relevant and useful for public key cryptography. We begin with a discussion of the prime number theorem, which gives a quantitative description of how densely the primes are distributed among the integers. We also reveal an important periodicity in powers of (invertible) integers modulo an integer. This naturally leads us to the concepts of *orders* and *primitive roots*, which, when they exist, are integers of the highest possible order in a given modulus. Large primes and primitive roots are important ingredients for many cryptosystems. We develop some algorithms for determining whether an integer is prime, for factoring, and for the determination of primitive roots.

The Prime Number Theorem

Euclid's proof that there are infinitely many primes, although very elegant, does not tell us much about how the primes are distributed among the positive integers. A more informative but much deeper theorem is known as *the prime number theorem*; it gives a very sharp estimate of the number of primes less than any given number x.

Theorem 8.1: Prime Number Theorem

If $\pi(x)$ denotes the number of prime numbers p satisfying $p < x$, then we have

$$\pi(x) \sim \frac{x}{\ln x}$$

In other words, $\pi(x)$ is asymptotic to the ratio $x/\ln x$, meaning that the ratio $\pi(x)/(x/\ln x) \to 1$ as $x \to \infty$.

The prime number theorem was first proved in 1896 independently by Jacques Hadamard (1865–1963, a French mathematician) and Charles Jean Gustave Nicolas Baron de la Vallée Poussin (1866–1962, a Belgian mathematician). Their proofs were quite sophisticated and used complex analysis. Elementary (but difficult) proofs have later been found and the interested reader may refer to the books by Hardy and Wright [HaWr-80] or Nagell [Nag-01]. We will forgo giving a proof here. The prime number theorem shows that the integers are quite densely populated by primes. For example, if one were to attempt to factor a certain 304-digit number* n_{364} by checking for prime factors up to $\sqrt{n_{364}}$, since $\sqrt{n_{364}} > 10^{152}$, this would possibly entail checking up to $\pi(10^{152})10^{152}/\ln(10^{152}) \approx 2.86 \times 10^{149}$ divisions. Even if all the computers in the world could be programmed to work together on this task and if they each could check one trillion divisions per second, it would take many millions of life spans of our universe. There are better algorithms for factoring into primes, but the prime factorization problem has remained one of the hardest problems in computational number theory, and it is overwhelmingly believed among specialists that a truly efficient (polynomial time) algorithm probably does not exist. These guaranteed difficulties are at the heart of many effective cryptographic applications of prime numbers. The prime number theorem has many practical ramifications.

Example 8.1

(a) Use the prime number theorem to estimate the number of 50-digit prime numbers.

(b) Suppose that we were to randomly select a 50-digit odd integer (so the first digit is randomly selected from 1 to 9, the last digit from 1, 3, 5, 7, and 9, and all other 48 digits are randomly selected from 0 to 9). Estimate the probability that we select a prime number.

* RSA Security (a high-tech cryptographic security company) had offered a number of public challenges on its company Web site. One of these offered a $100,000 prize to the first person to factor a certain 304-digit number (larger prizes were available for factoring larger integers). This particular challenge remained open for several years. Such challenges actually benefit the company by helping to test the security of some of their secret codes (that rest on the infeasibility of being able to factor such large or even larger integers) against potential hackers.

Solution: Part (a): There are $\pi(10^{50}) - \pi(10^{49})$ 50-digit prime numbers. The prime number theorem estimates this number to be $10^{50}/\ln(10^{50}) - \pi(10^{49})/\ln(10^{49}) \approx 7.7996 \times 10^{47}$.

Part (b): By part (a), the proportion of 50-digit numbers that are prime is approximately $7.7996 \times 10^{47}/10^{50} \approx 1/128$. This means that if we were to randomly select odd 50-digit numbers, the chances of selecting a prime number would be about twice this number, or 1/64.

Exercise for the Reader 8.1

(a) Use the prime number theorem to estimate the number of 300-bit prime numbers (where a 300-bit string represents the binary representation of an integer as in Chapter 6, and we assume that the first bit is 1 so the number really requires specification of 299 bits).

(b) Suppose that we were to randomly select a 300-bit odd integer (so the first and last bits are set to 1, and all other 298 bits are randomly selected).* Estimate the probability that we select a prime number.

Fermat's Little Theorem

The probabilistic primality tests that we develop at the end of this chapter will help us to astoundingly improve odds that were seen in Example 8.1 of randomly generating primes of a specified size. The prototypical primality test of this sort is based on a small but important theorem of Pierre de Fermat; see Figure 8.1.† We motivate this theorem with an example

* The last bit being 1 makes the integer odd, and the first bit being 1 makes the effective length of the string 300 (since any zero bits on the left are redundant).

† After completing (the equivalent) of a bachelor's degree in mathematics, Fermat earned a law degree and followed an impressive career path as a government lawyer and council member in the city of Toulouse. He maintained his ardent interest in mathematics throughout his life and was involved in cutting-edge mathematics among the leaders in the field. His published works are relatively small in number, however, but this was due to his preference for solving new problems rather than taking the time to formally write up his work on problems that he had finished, the fact that he held another very demanding full-time job notwithstanding. Fermat often made bold scientific claims, such as finding faults or simpler approaches to the famous development of optics by René Descartes (the founder of analytic geometry). This sometimes put him at the ire of his subjects and caused him difficulties in getting his work accepted, but in most cases, Fermat's assertions later proved to be correct. In one of Fermat's notebooks he wrote that the equation $a^n + b^n = c^n$ can never have any positive integer solutions a, b, c, whenever n is an integer greater than 2 (when $n = 2$, this equation has many such integer solutions, such as $a, b, c = 3, 4, 5$). He wrote further that he had a truly remarkable fact of this result, which the margin of the notebook was too small to contain. This result, known as Fermat's last theorem, had captivated number theorists for three centuries. It was not until over three centuries after Fermat's death when a proof of this result was finally discovered and published by a Princeton mathematician, Andrew Wiles. Wiles claimed to have worked over seven years in his attic on his proof, which had been a fascination of his since childhood. His proof led to his being awarded the *Field's Medal*, which is the most coveted prize in mathematics (often called the Nobel Prize of mathematics). Apart from his prestige in number theory, Fermat is also credited as being (with Blaise Pascal) one of the founders of the subject of probability.

Figure 8.1 Pierre de Fermat (1601–1665), French mathematician.

involving modular exponentiation with a prime base and perform the exponentiation in two ways: the first method reviews the fast exponentiation algorithm (Algorithm 6.5), while the second is a much more efficient method that will initially appear to be quite mysterious. We then clear up the mystery by presenting Fermat's little theorem.

Example 8.2

Compute $2^{1452} \bmod 19$.

Solution: As was mentioned in Chapter 6, a horribly inefficient way to do this would be to first compute 2^{1452} directly, using integer arithmetic, and then convert it to its representative, modulo 19, in the set $\{0, 1, 2, \cdots, 18\}$. The first number would be so large that it would overflow on many computer systems.* We will instead perform this computation with two much more efficient schemes; the first is simply the fast modular exponentiation method (Algorithm 6.5), but with an informal notation.

Method 1: Fast Modular Exponentiation. We begin with $2^2 \equiv 4 \pmod{19}$, and continue to square both sides until the exponents exceed at least half of the desired exponent:

$$2^4 \equiv 4^2 \equiv 16$$

$$2^8 \equiv 16^2 \equiv 256 \equiv 9$$

$$2^{16} \equiv 9^2 \equiv 81 \equiv 5$$

$$2^{32} \equiv 6$$

* *Computing Note*: This should serve as a caution to students not to take for granted or to rely too much on the capabilities of computing platforms, depending on what type of system you are working on, floating point or symbolic. A floating point system only has accuracy to about 16 digits. For example, the integer 13^{20} has 23 digits, so its computation on a floating point system would only be accurate to the first 15 or so of these digits. Thus, if we took the remainder of this computation, say mod 29, we would probably get the wrong answer. Symbolic systems have much greater accuracy but usually work more slowly, but even these have limitations in the sizes of the numbers they can deal with.

$$2^{64} \equiv 17$$

$$2^{128} \equiv 4$$

$$2^{256} \equiv 16$$

$$2^{512} \equiv 9$$

$$2^{1024} \equiv 5$$

We will now be able to use the above powers to compute the desired power of 2 (mod 19). This is because of the binary expansion $1452 \sim [10110101100]$ (base 2), which is equivalent to $1452 = 1024 + 256 + 128 + 32 + 8 + 4$, as the reader can easily check. It follows that we may compute

$$\begin{aligned} 2^{1452} &= 2^{1024} \cdot 2^{256} \cdot 2^{128} \cdot 2^{32} \cdot 2^{8} \cdot 2^{4} \\ &\equiv 5 \cdot 16 \cdot 4 \cdot 6 \cdot 9 \cdot 16 \equiv 11 (\text{mod } 19) \end{aligned}$$

Method 2: At first glance, this method will appear to use a lucky trick. But we will soon give Fermat's little theorem, which will show that this trick can be easily replicated in general situations. We note that $2^{18} \equiv 1 (\text{mod } 19)$. If we apply the division algorithm to the integer division of 1452 by 18, we obtain $1452 = 80 \cdot 18 + 12$. It follows that $2^{1452} \equiv (2^{18})^{80} \cdot 2^{12} \equiv 1^{80} \cdot 11 \equiv 11 (\text{mod } 19)$. This was quite a bit less work than Method 1.

In general, the same trick used in the second method of the above example can be used to compute any power $a^e \pmod m$, provided that we can find a special exponent s (less than m) such that $a^s \equiv 1 \pmod m$. We will show that such an exponent always exists (the same will work for any a), as long as a and m are relatively prime, and show how to find it. We first deal with the case in which the modulus m is prime (as in the above example when m was 19). The following classical theorem of Pierre de Fermat shows that such a "magical" exponent is very easily found for any prime modulus.

Theorem 8.2: Fermat's Little Theorem

Suppose that p is a prime and a is an integer that is not a multiple of p, then $a^{p-1} \equiv 1 \pmod p$.

Proof: For greater clarity in this proof, we will denote elements of $\mathbb{Z}_p$ (the integers mod p) using square brackets: $[k]$ represents the set of all integers that are congruent to $k \pmod p$. We will construct a function with domain and codomain both being the set of nonzero integers mod p: $A = \{[1], [2], \cdots, [p-1]\}$, $f : A \to A$, defined by $f([x]) = [ax]$. By Proposition 2.10, this definition gives the same output no matter which representative we use of $[x]$, so it is a well-defined function on elements of A. But we still need to check that the images will never be $[0]$ (that is, so the codomain of the function can be taken to be A). Indeed, if $[ax] = [0]$, this would mean that $ax \equiv 0 (\text{mod } p)$, so that $p \mid ax$. But then Euclid's

lemma (Proposition 2.7) would imply that either $p \mid a$ or $p \mid x$. Both of these options are not possible since $[a] \neq [0]$ and since $[x] \neq [0]$.

Next we will show that f is one-to-one. Suppose that $f([x]) = f([y])$. This means that $[ax] = [ay]$ or $ax \equiv ay \pmod{p}$. By definition, this means that $p \mid (ax - ay)$ or $p \mid a(x - y)$. Euclid's lemma (Proposition 2.7) then tells us that either $p \mid a$ or $p \mid (x - y)$. Since we know the former is false, the latter must hold, which means that $x \equiv y \pmod{p}$ or $[x] = [y]$, so f is one-to-one.

Since f is a one-to-one function of the set A to itself, it follows that the images of f: $f([1])$, $f([2])$, $\cdots$, $f([p-1]) = [a]$, $[2a]$, $\cdots$, $[(p-1)a]$ are simply a relisting of the elements of A: $[1]$, $[2]$, $\cdots$, $[p-1]$, in perhaps a different order. It follows that if we multiply representatives from these to listings of the nonzero elements of $\mathbb{Z}_p$, we will get the same result (mod p):

$$1 \cdot 2 \cdot \ldots \cdot (p-1) \equiv a \cdot 2a \cdot \ldots \cdot (p-1)a \equiv a^{p-1}(1 \cdot 2 \cdot \ldots \cdot (p-1)) \pmod{p}$$

This equation implies that $p \mid [a^{p-1}(1 \cdot 2 \cdot \cdots \cdot (p-1)) - 1 \cdot 2 \cdot \cdots \cdot (p-1)]$, or $p \mid [(a^{p-1} - 1)(1 \cdot 2 \cdot \cdots \cdot (p-1))]$, and Euclid's lemma tells us that p must divide one of the factors on the right. The only possibility is that $p \mid (a^{p-1} - 1)$, so that $a^{p-1} \equiv 1 \pmod{p}$, as we wished to prove. □

In light of Example 8.1, it is now easy to see how Fermat's little theorem can help us to easily raise any integer a to any power e modulo a prime p; we simply make use of the "magic" exponent $p-1$ and use the division algorithm to write $e = q(p-1) + r$, where $0 \leq r < p-1$. It then follows that $a^e \equiv a^r \pmod{p}$.

Exercise for the Reader 8.2

Compute $18^{802} \pmod{29}$, using each of the two methods shown in Example 8.2. (In Method 2, use Fermat's little theorem.)

The Euler Phi Function

Our next theorem, which is due to Leonhard Euler,* will generalize Fermat's little theorem to work for any modulus m. The analogue of the "magic" exponent p is determined by the following very useful integer function:

* Leonhard Euler (pronounced "Oiler") entered into mathematics during one of its most exciting eras; calculus had recently been invented and the field was transforming with numerous consequences and applications. Euler's life was nothing short of phenomenal. Educated in Switzerland, he was first appointed as a professor at age 19 at the renowned St. Petersburg University in Russia, and six years later he was appointed to the Berlin Academy and became its leader. His published works were significant and touched on practically all of the fields of mathematics. He was the most prolific mathematician ever, even during the last 17 years of his life when he was completely blind (in fact, this was perhaps his most productive period). His papers were assembled into a collected works compendium that filled over 100 encyclopedia-sized tomes! His mental skills remained remarkably acute throughout his life. At age 70, for example, he could recite an entire novel, as well as stating the first and last sentences on each page, and he once settled an argument between two students whose answers differed in the 15th decimal place, by a fast computation in his head. Euler had 13 children, and he told stories about having made some of his most seminal mathematical discoveries as he was holding one child on his lap while others were playing at his feet.

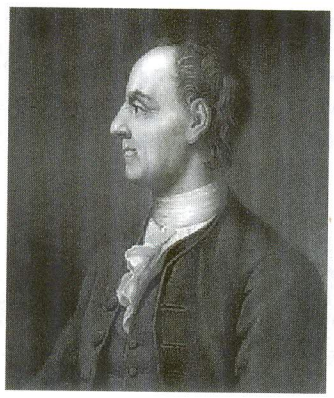

Figure 8.2 Leonhard Euler (1707–1783), Swiss mathematician.

Definition 8.1

Euler's phi function is a function defined on the set of positive integers: $\phi: \mathbb{Z}_+ \to \mathbb{Z}_+$, by $\phi(n)$ = the number of integers in the set $\{1, 2, \cdots, n\}$ that are relatively prime to n.

In the case $n = p$ is a prime number, then each of $1, 2, \ldots, p-1$, is relatively prime to p, so that $\phi(p) = p - 1$. The numbers in the set $\{1, 2, \cdots, 10\}$ that are relatively prime to 10 are 1, 3, 7, 9, so $\phi(10) = 4$. The following result makes it easy to compute $\phi(n)$ for any positive integer n, provided that we have the prime factorization of n.

Proposition 8.3

If $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_n^{\alpha_n}$, where $p_1, p_2, \cdots, p_n$ are distinct primes, and $\alpha_1, \alpha_2, \cdots, \alpha_n \in \mathbb{Z}_+$ then

$$\phi(n) = (p_1 - 1)p_1^{\alpha_1 - 1}(p_2 - 1)p_2^{\alpha_2 - 1} \cdots (p_n - 1)p_n^{\alpha_n - 1} \qquad (8.1)$$

For example, since the prime factorization of 10 is $2 \cdot 5$, Proposition 8.3 tells us that $\phi(10) = (2-1) \cdot 2^0 \cdot (5-1) \cdot 5^0 = 4$, as we had shown earlier. For another example, since the prime factorization of 378 is $3^3 \cdot 7^2$, the proposition tells us that $\phi(378) = \phi(3^3 \cdot 7^2) = (3-1) \cdot 3^2 \cdot (7-1) \cdot 7^1 = 108$. A proof of Proposition 8.3 is outlined in the exercises; see Chapter Exercises 41–43.

Exercise for the Reader 8.3

Compute the following values of Euler's phi function: $\phi(15)$, $\phi(20)$, $\phi(208)$, $\phi(2208)$.

Euler's Theorem

Notice that when $n = p$ is prime, the formula in Proposition 8.3 gives $\phi(p) = p - 1$, which was the "magic" exponent in Fermat's little theorem. Euler generalized Fermat's little theorem to work for any modulus, with his phi function continuing to play the role of the "magic" exponent.

Theorem 8.4: Euler's Theorem

Suppose that a and m are relatively prime positive integers with $m > 1$, then $a^{\phi(m)} \equiv 1 \pmod{m}$.

Exercise for the Reader 8.4

Prove Euler's theorem.

Suggestion: Mimic the proof of Fermat's little theorem, but now take the set A to be the set of all positive integers less than m that are relatively prime to m.

Example 8.3

Make use of Euler's theorem to perform each of the following tasks:

(a) Compute $18^{2551} \pmod{25}$.

(b) Find the last (one's) digit of the integer 13^{2017}.

Note: The integer in part (b) has 2246 digits.

Solution: Part (a): Since $\gcd(18, 25) = 1$ and $\phi(25) = \phi(5^2) = (5-1) \cdot 5^1 = 20$, Euler's theorem tells us that $18^{20} \equiv 1 \pmod{25}$. Using the division algorithm for the integer division of 2551 by 20 gives $2551 = 127 \cdot 20 + 11$, and consequently $18^{2551} \equiv (18^{20})^{127} \cdot 18^{11} \equiv 1 \cdot 7 \equiv 7 \pmod{25}$.

Part (b): Finding the one's digit of any number is the same as the answer we would get by converting it to an integer modulo 10. Thus, we wish to find $13^{2017} \pmod{10}$. Since $\gcd(13, 10) = 1$ and $\phi(10) = 4$, Euler's theorem tells us that $13^4 \equiv 1 \pmod{10}$. Applying the division algorithm to the given exponent divided by 4 gives $2017 = 504 \cdot 4 + 1$, hence $13^{2017} \equiv (13^4)^{504} \cdot 13^1 \equiv 1 \cdot 13 \equiv 3 \pmod{10}$.

Exercise for the Reader 8.5

(a) Compute $7^{8486} \pmod{58}$.

(b) Find the last three digits of the integer 13^{2017}.

Despite the speed and apparent magic with which Euler's theorem helps us to perform such modular exponentiations, there is one serious drawback: Evaluating $\phi(n)$ requires the prime factorization of n. As we had mentioned, this is a very hard problem for which no efficient algorithm exists (and according to the general consensus) never will. Consequently, the fast modular exponentiation algorithm generally works more quickly. Modular exponentiation has many applications; for example, it is at the core of the so-called ElGamal cryptosystem that will be studied in Chapter 9.

Two intimately related number theoretic concepts that will be useful for an assortment of public key cryptosystems are *orders* and *primitive roots*.

Modular Orders of Invertible Modular Integers

Definition 8.2

For integers $1 \leq a < n$, with a and n relatively prime, we define the **order of a relative to n** (or **order of a mod n**) to be the smallest positive exponent k for which $a^k \equiv 1 \pmod{n}$. We write this as $k = \text{ord}_n(a)$.*

The following example illustrates the different orders of elements relative to two small moduli. Note that from Euler's theorem, we know that $\text{ord}_n(a) \leq \phi(n)$.

Example 8.4

(a) Compute the orders mod n of all positive integers less than (and relatively prime to) $n = 7$.
(b) Do the same for $n = 8$.

Solution: Table 8.1 and Table 8.2 illustrate all of the powers [up to the $\phi(n)$th].

TABLE 8.1 Powers (mod 7) of Integers Relatively Prime to $n = 7$

	$a^k \pmod 7$					
	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
$a=1$	1	1	1	1	1	1
$a=2$	2	4	1	2	4	1
$a=3$	3	2	6	4	5	1
$a=4$	4	2	1	4	2	1
$a=5$	5	4	6	2	3	1
$a=6$	6	1	6	1	6	1

Note: In each row, the order is at the top of the column containing the highlighted box.

* This notation for orders is due to C. F. Gauss.

TABLE 8.2 Powers (mod 8) of Integers Relatively Prime to $n = 8$

	$a^k \pmod 8$			
	$k = 1$	$k = 2$	$k = 3$	$k = 4$
$a = 1$	1	1	1	1
$a = 3$	3	1	3	1
$a = 5$	5	1	5	1
$a = 7$	7	1	7	1

Note: In each row, the order is at the top of the column containing the highlighted box.

The next proposition collects some useful facts about orders and modular exponentiation.

Proposition 8.5

Suppose that a and $n > 1$ are relatively prime positive integers.

(a) If k is a positive integer with $a^k \equiv 1 (\text{mod } n)$, then $\text{ord}_n(a) \mid k$.
(b) $\text{ord}_n(a) \mid \phi(n)$.
(c) If i and j are nonnegative integers, then $a^i \equiv a^j (\text{mod } n)$ if, and only if, $i \equiv j \pmod{\text{ord}_n(a)}$.

Proof: Part (a): Use the division algorithm to write $k = q \cdot \text{ord}_n(a) + r$, where q and r are integers with $0 \le r < \text{ord}_n(a)$. By the assumption that $a^k \equiv 1 (\text{mod } n)$, and then using the definition of order, we may write

$$1 \equiv a^k \equiv a^{q \cdot \text{ord}_n(a) + r} \equiv (a^{\text{ord}_n(a)})^q a^r \equiv 1^q a^r \equiv a^r (\text{mod } n)$$

So we have found that $a^r \equiv 1 (\text{mod } n)$. But since $\text{ord}_n(a)$ is the smallest positive exponent with this property, we must have $r = 0$, which means that $\text{ord}_n(a) \mid k$.

Part (b): By Euler's theorem we know that $a^{\phi(m)} \equiv 1 \pmod m$. Thus, part (b) follows from part (a) with $k = \phi(n)$.

Part (c): We first assume that $a^i \equiv a^j (\text{mod } n)$ and (switching the roles of i and j if necessary) that $i \ge j$. Since $\gcd(a, n) = 1$, we know from Proposition 2.11 that the multiplicative inverse of a, a^{-1}, exists mod n. If we multiply both sides of the congruence $a^i \equiv a^j (\text{mod } n)$ by $(a^{-1})^j$, we obtain that $a^{i-j} \equiv a^{j-j} \equiv 1 (\text{mod } n)$, so the result of part (a) (with $k = i - j$) tells us that $\text{ord}_n(a) \mid i - j$; that is, $i \equiv j \pmod{\text{ord}_n(a)}$. Conversely, if $i \equiv j \pmod{\text{ord}_n(a)}$, then, assuming as before that $i \ge j$, we may write $i - j = q \cdot \text{ord}_n(a)$, or $i = j + q \cdot \text{ord}_n(a)$, for some nonnegative integer q. From this equation, we obtain

$$a^i \equiv a^{q \cdot \text{ord}_n(a) + j} \equiv (a^{\text{ord}_n(a)})^q a^j \equiv 1^q a^j \equiv a^j (\text{mod } n) \quad \square$$

Primitive Roots

Note that with regard to Table 8.1, Euler's theorem tells us [since $\phi(7) = 6$] that $a^6 \equiv 1 \pmod 7$ for each a relatively prime to 7, while in the context of Table 8.2, it says that $a^4 \equiv 1 \pmod 8$, for each a relatively prime to 8 [since $\phi(8) = 4$]. But the tables are more revealing. Table 8.1 shows that only

two elements have (maximum) order equal to Euler's exponent 6, while Table 8.2 shows that none of the elements have order equal to Euler's exponent 4. This motivates the following definition.

Definition 8.3

We say that a positive integer g that is less than n and relatively prime to n is a **primitive root** mod n if the order of g is $\phi(n)$.

From Tables 8.1 and 8.2, we see that there are two primitive roots mod 7, namely 3 and 5, but that there are no primitive roots mod 8. Notice also that the powers of the primitive roots cycle through all of the $\phi(n)$ positive integers that are less than and relatively prime to n. The following proposition shows that this is true in general.

Proposition 8.6

If g is a primitive root modulo a positive integer n, then the powers $g, g^2, g^3, \cdots, g^{\phi(n)}$ are all different (mod n), and hence this list of powers is congruent to the set of all positive integers less than n that are relatively prime to n.

Proof: If there were ever a repeated term in this sequence of powers $g, g^2, g^3, \cdots, g^{\phi(n)}$, say $g^i \equiv g^j \pmod{n}$, then part (c) of Proposition 8.5 would tell us that $i \equiv j \pmod{\text{ord}_n(g)}$. Since g is a primitive root, $\text{ord}_n(g) = \phi(n)$, this congruence is impossible if i and j are different integers in the range $\{1, 2, \cdots, \phi(n)\}$. It follows that all of the modular powers $g, g^2, g^3, \cdots, g^{\phi(n)}$ are distinct mod n. □

Exercise for the Reader 8.6

(a) As in the solution of Example 8.4, construct tables for the modular powers of all positive integers less than (and relatively prime) to $n = 4$. Use the table to find all primitive roots.

(b) Repeat the instruction of part (a) for $n = 9$.

Since some major public key cryptosystems (that will be studied in the next chapter) depend on primitive roots and orders, we would like to now address the following natural questions:

1. *Existence of Primitive Roots.* For which moduli n do primitive roots exist, and when they exist, how many will there be?
2. *Finding Primitive Roots.* When primitive roots exist mod n, how can we find them?
3. *Computing Orders.* How can we compute the order $\text{ord}_n(a)$ of an element a that is relatively prime to n?
4. *The Discrete Logarithm Problem.* If g is a primitive root mod n, and a is an integer relatively prime to n, how can we find the exponent j [unique mod $\phi(n)$ by Proposition 8.6] such that $g^j \equiv a \pmod{n}$?

We are able to give some reasonable answers to the first three of these questions. The discrete logarithm problem of the fourth question is very difficult, and no known efficient algorithms for it are known. Like the prime factorization problem, its difficulty is the basis for the security of some widely used public key systems (the ElGamal cryptosystem and digital signature scheme, and the Diffie–Hellman key exchange protocol) that will be introduced in the next chapter.

Existence of Primitive Roots

The following theorem nicely and completely answers the first question.

Theorem 8.7: Existence and Number of Primitive Roots

Suppose that n is a positive integer greater than 1. Primitive roots exist mod n if, and only if, n is of the form 2, 4, p^s or $2p^s$, where p is an odd prime and t is a positive integer. In this case, there are exactly $\phi(\phi(n))$ primitive roots mod n.

Theorem 8.7 can be proved in tandem with our next theorem on the determination of primitive roots. The proofs are rather lengthy, although not particularly difficult, so we will omit them. Interested readers may refer to any good book on number theory, such as [Ros-05].

Example 8.5

For which of the following moduli n do primitive roots exist? In cases where primitive roots exist, how many will there be (mod n)?

(a) $n = 20$
(b) $n = 59$
(c) $n = 30$
(d) $n = 1250$

Solution: The prime factorizations of these integers are $20 = 2^2 \cdot 5$, 59 (prime), $30 = 2 \cdot 3 \cdot 5$, and $1250 = 2 \cdot 5^4$. Only those of parts (b) and (d) satisfy the conditions of Theorem 8.7 and thus have primitive roots. The corresponding number of primitive roots are given by Euler's phi function (by Theorem 8.7), so that (using Proposition 8.3) 59 has $\phi(\phi(59)) = \phi(58) = \phi(2 \cdot 29) = 28$ primitive roots, and 1250 has $\phi(\phi(1250)) = \phi(\phi(2 \cdot 5^4)) = \phi(500) = \phi(2^2 \cdot 5^3) = 200$ primitive roots.

Determination of Primitive Roots

There is unfortunately no simple algorithm for determining a primitive root modulo an integer for which they are known to exist. A brute-force search of checking orders of numbers relatively prime to n can be used, but that is slow. [It can be significantly speeded up using Proposition 8.5(b), however, if the prime factorization of n is known, since the only possible orders are divisors of $\phi(n)$.]

Gauss developed a more efficient implementation of this brute-force approach for determining primitive roots modulo a prime. We will discuss Gauss's algorithm shortly; his algorithm still seems to be the best general tool for this task. We first present a theorem that reduces the problem of finding primitive roots (when they exist) to the determination of primitive roots modulo a prime.

Theorem 8.8: Determination of Primitive Roots

In each of the cases for a modulus n of Theorem 8.7 in which primitive roots exist, the following rules show how primitive roots can be found:

(a) If g is a primitive root modulo an odd prime power p^k, then g or $g + p^k$ (whichever is odd) will be a primitive root mod $2p^k$.
(b) If g is a primitive root modulo an odd prime p, then g or $g + p$ (whichever is odd) will be a primitive root mod p^2.
(c) If p is an odd prime and g is a primitive root mod p^2, then g will also be a primitive root mod any higher power p^k of p.

Before we state Gauss's algorithm for computing primitive roots modulo a prime, we will give a principle that is sometimes useful in computing orders of integers in a certain modulus. The following result gives a formula for the order of any modular power of an integer in terms of the order of the integer (mod m).

Order of Powers Formula

Proposition 8.9: Order of Powers Formula

If a, j, and $n > 1$ are positive integers, with a relatively prime to n, then

$$\text{ord}_n(a^j) = \frac{\text{ord}_n(a)}{\gcd(j, \text{ord}_n(a))} \tag{8.2}$$

We point out one immediate consequence of this proposition shows that once we have one primitive root mod n, we can easily get them all.

Corollary 8.10

If g is a primitive root (mod n), then the other primitive roots (mod n) are the same as the modular powers g^j (mod n), as j runs through all of the positive integers less than and relatively prime to $\phi(n) = \text{ord}_n(g)$.

The corollary easily follows from the proposition since with $a = g$, a primitive root used in Equation 8.2, the right side will equal $\text{ord}_n(a)$ exactly when the denominator $\gcd(j, \text{ord}_n(a))$ is 1. To illustrate, in Example 8.4, we found that $g = 3$ is a primitive root of 7. By the above corollary, all the

primitive roots (mod 7) will be given by the modular powers g^j whose exponents j are relatively prime to $\phi(7) = 6$, which are $3^1 \equiv 3, 3^5 \equiv 5$. (This is confirmed by the rest of Table 8.1.)

Proof of Proposition 8.9: Let $m = \text{ord}_n(a)$. From Proposition 8.5(a), it follows that $a^{j\ell} \equiv 1 (\text{mod } n)$ if, and only if, $j\ell$ is a multiple of m. But by definition of order, ℓ is the smallest positive integer such that $a^{j\ell} \equiv 1 (\text{mod } n)$. It thus follows that $j \cdot \text{ord}_n(a^j) = \text{lcm}(j, m)$. But [see Exercise for the Reader 2.2(c)] $\text{lcm}(j,m) = \dfrac{jm}{\gcd(j,m)}$, which, when combined with the previous equation, gives

$$j \cdot \text{ord}_n(a^j) = \frac{jm}{\gcd(j,m)} \Rightarrow \text{ord}_n(a^j) = \frac{m}{\gcd(j,m)} = \frac{\text{ord}_n(a)}{\gcd(j, \text{ord}_n(a))} \quad \square$$

Example 8.6

If possible, do each of the following:

(a) Compute $\text{ord}_{59}(7)$.
(b) Find an integer between 1 and 59 whose mod 59 order is 22.
(c) Find a primitive root of 59.
(d) Find an exponent j such that $g^j \equiv 7 (\text{mod } 59)$, where g is the primitive root that was found in part (c).

Solution: Part (a): By Proposition 8.5(b), the order of any integer (relatively prime to 59) must divide $\phi(59) = 58 = 2 \cdot 29$, so the only possible orders (apart from 1 for integers congruent to 1 mod 59) are 2, 29, and 58 (in which case we have a primitive root). So to check $\text{ord}_{59}(7)$, we need only compute at most two modular powers $7^2, 7^{29}$; if one of these is congruent to 1, the exponent will be the $\text{ord}_{59}(7)$; otherwise, this order will be 58 (and 7 would be a primitive root). Indeed, $7^2 \equiv 49$, and using fast modular exponentiation (Algorithm 6.5), $7^{29} \equiv 1 (\text{mod } 59)$, so $\text{ord}_{59}(7) = 29$.

Part (b): Since $22 \nmid 58 = \phi(59)$, we know from Proposition 8.5(a) that 22 cannot be an order mod 59.

Part (c): By Theorem 8.7, since 59 is prime, it will have $\phi(\phi(59)) = \phi(58) = 28$ primitive roots. This is nearly half of the nonzero mod 59 integers. We could do a simple brute-force search starting with $a = 2$. We would compute the powers a^2, a^{29} if neither is congruent to 1, then (as explained above) a would be a primitive root. If $a = 2$ fails to be a primitive root, we increase a by 1: $a \rightarrow a + 1 = 3$. We continue this process until we find a primitive root. Since $2^2 \equiv 4$, and (using Algorithm 6.5) $2^{29} \equiv 58 (\text{mod } 29)$, we can stop in our first attempt and declare 2 as a primitive root.

Part (d): This is a discrete logarithm question. As mentioned earlier, there are no efficient algorithms to do this. By Proposition 8.9, since $g = 2$ is a primitive root, we have

$$\text{ord}_{59}(2^j) = \frac{\overset{?}{\text{ord}_2(59)}}{\gcd(j, \underset{?}{\text{ord}_2(59)})} = \frac{58}{\gcd(j,58)}$$

Since we know from part (a) that $\text{ord}_{59}(7) = 29$, the above equation tells us that in order to have $2^j \equiv 7 \pmod{59}$, j must be even. The brute-force approach would be to continue computing modular even powers of 2: $2^2, 2^4, 2^6, \cdots \pmod{59}$ until we obtain 7. Since each term is just four times the previous term (mod 59), the numbers are small enough for the computation to be feasible:

$$2^2 \equiv 4,\ 2^4 \equiv 16,\ 2^6 \equiv 5,\ 2^8 \equiv 20,\ 2^{10} \equiv 21,$$

$$2^{12} \equiv 25,\ 2^{14} \equiv 41,\ 2^{16} \equiv 46,\ 2^{18} \equiv 7$$

Thus we have found the desired discrete logarithm to be $j = 18$.

Exercise for the Reader 8.7

(a) How many primitive roots does $n = 334$ have?
(b) What is the smallest primitive root?
(c) How many integers mod 334 have order equal to 2?

Algorithm 8.1: Gauss's Algorithm for Finding a Primitive Root Modulo a Prime **p**

The input will be an odd prime p, and output will be a primitive root $g \pmod{p}$.

Step 1. Initialize the set of noncandidates for a primitive root mod p: $N = \varnothing$. (So initially the candidates can be any integer in the range $1 < a < p$.) Select a candidate a (preferably randomly), and successively compute the powers of $a \pmod{p}$ until we first obtain 1: $a^2, a^3, \cdots, a^\ell \equiv 1$ [thus $\text{ord}_n(a) = \ell$]. If $\ell = p - 1$, then a is a primitive root, so output $g = a$, and exit the algorithm. Otherwise go to Step 2.

Step 2. Update $N \to N \cup \{a, a^2, \cdots, a^{\ell-1}\}$. Select a modular integer candidate outside of N: b (preferably randomly), and compute $m = \text{ord}_n(b)$. If $m = p - 1$, then b is a primitive root, so output $g = b$, and exit the algorithm. Otherwise, let $u = \text{lcm}(m, \ell)$, and write $u = xy$, where $x \mid m$, $y \mid \ell$, and $\gcd(x, y) = 1$. Compute the modular powers $a' \triangleq a^{\ell/y}, b' \triangleq b^{m/x} \pmod{p}$, and set $G = a'b' \pmod{p}$. G has order $xy = u$. If $u = p - 1$, then G is a primitive root, so output $g = G$, and exit the algorithm. Otherwise go to Step 3.

Step 3. Return to Step 2 with the following updates: $\ell \to u, a \to G$.

This algorithm will always terminate since in each application of Step 2, it will always be the case that $u > \ell$ (see Chapter Exercise 44 for an outline

of the proof of this fact) so that the orders of the elements found continue to increase. Let us illustrate with a small example. Of course, for primes greater than 200 or so, this algorithm should be implemented on a computing platform.

Example 8.7

Use Gauss's algorithm (Algorithm 8.1) to find a primitive root for the prime $p = 101$.

Solution: Let us begin Step 1 by choosing $a = 36$. Computing successive powers mod 101, we have $a^i \equiv 36, 84, 95, 87, 1$, so we have $\ell = \text{ord}_n(a) = 5$. Since a is not a primitive root, we move on to Step 2: We select a (nonzero) modular integer b different from the powers of a, say $b = 88$. We compute $m = \text{ord}_n(b) = 25$. Since $u = lcm(m, \ell)$, we can just take $G = m$. But since G is still not a primitive root, Step 3 tells us to go back again to Step 2, now with $a = 88$, and ℓ taken to be its order 25. We need to select b to be a nonzero modular integer different from the powers of a: $a^i \equiv 88, 68, 25, 79, 84, 19, 56, 80, 71, 87, 81, 58, 54, 5, 36, 37, 24, 92, 16, 95, 78, 97, 52, 31, 1$. We take $b = 51$ and compute $m = \text{ord}_n(b) = 100$. We have thus found a primitive root, and output $G = 51$.

Prime Number Generation

The most fundamental need in public key cryptography is the generation of large prime numbers that can serve to create new and secure keys. Here is a typical outline of a scheme for constructing a prime of a prescribed size: Say that we need a random prime of size 1000 bits. We would set the first and last bits equal to 1 (the last bit being set to one makes the number odd), and then randomly assign the remaining 998 bits. A 1000-bit number will have size approximately $\log_{10}(2^{1000}) \approx 10^{300}$, and so by the prime number theorem (Theorem 8.1), the density of primes among integers of such size is approximately $1/\ln(10^{300}) \approx 1/691$. Since we are only checking odd integers, this doubles the density of primes to be approximately 1/345. Thus, on average, it will take about 345 randomly generated odd 1000-bit numbers before we hit on a prime. On each attempt, we use a *probabilistic primality test* that will screen out probable primes. The probability that the number produced is not prime can be prescribed to be as small as we would like with such a test. Although these probabilistic probability tests cannot guarantee a prime is produced, some will allow the user to prescribe the probability that a prime is not produced to be as small as is needed for most practical purposes, say $1/10^{15}$. If it is absolutely necessary to confirm that the probable prime is actually prime, there are more expensive prime certification tests that can be applied. The analogy is similar to the public health problem of screening people for certain chronic illnesses. Usually a quick inexpensive test is given to the mass population, and for those who test positive, a more accurate (and expensive and slower) test is given to see if they really are infected. We

will focus on only the first type of probabilistic primality tests. Readers interested in learning about the prime certification tests may refer to one of the following books on algorithmic and computational number theory: [BaSh-96], [Coh-93]; see also [BrWa-99].

Initially it may seem hard to believe, but the problem of checking whether a positive integer is prime is much easier than the problem of factoring a positive integer. In fact, as mentioned in Chapter 1, it is strongly believed among number theorists that an efficient algorithm for factoring integers will never exist. Checking primality, on the other hand, can be done with deterministic algorithms that run in polynomial time.*

Fermat's Primality Test

One basic reason that primality checks can be done faster than factoring is that integers can be proved to be composite without actually producing a factorization. This phenomenon will be seen in our first simple primality test, which is based on the contrapositive of Fermat's little theorem (Theorem 8.2): If p is a prime and $1 < a < p-1$, then $a^{p-1} \equiv 1 \pmod{p}$.† The contrapositive‡ can be formulated as:

Contrapositive of Fermat's Little Theorem

If n is a positive integer and $a^{n-1} \not\equiv 1 \pmod{n}$, for some number a, $1 < a < n-1$, then n is composite.

As a simple example, we can compute (with the fast exponentiation Algorithm 6.5) that $2^{1002} \equiv 990 \pmod{1003}$, so by the contrapositive of Fermat's little theorem (with $a = 2$), this proves that 1003 is composite. This is a nice example to demonstrate how it is possible to determine that an integer is composite without actually factoring it. Since it is not clear how to choose an appropriate a to attempt to use this criterion to prove a given integer n is composite, it is best to simply make random choices and to apply the test a certain number k times. If one of these k trials results in $a^{n-1} \not\equiv 1 \pmod{n}$, then the test has proved n to be composite, and a corresponding base a for which $a^{n-1} \not\equiv 1 \pmod{n}$, is called a **witness** to the

* It has been well known by specialists since the time of Gauss that the primality check problem is easier to solve than the factoring problem and that a polynomial time deterministic algorithm for the former should exist. But it was not until 2002 when such an algorithm was discovered by a group of three computer scientists—Manindra Agrawal, Neeraj Kayal, and Nitin Saxena from the Indian Institute of Technology in Kangpur, India. The algorithm is quite simple and elegant and is named after the three or, more briefly, the *AKS test*; see [AgKaSa-04]. These three scientists have received several prestigious awards for this very important discovery that had eluded many great minds for a long period of time.

† We intentionally omitted the values $a = 1$ and $a = p - 1$, since, even if p were composite (but an odd number greater than 1), these values would always equal 1 when risen to the (even) power $p - 1$.

‡ The *contrapositive* of any logical implication of the form "If P, then Q." is the statement "If not Q, then not P." Any implication is logically equivalent to its contrapositive.

fact that n is composite. If each of the k trials results in $a^{n-1} \equiv 1 \pmod{n}$, then n is declared as probably prime. Here is a formal summary of this randomized algorithm.

Algorithm 8.2: Randomized Fermat Primality Test

Inputs: An integer $n > 3$ and a positive integer k.
Output: Either a declaration that n is composite, along with a witness integer a that satisfies $a^{n-1} \not\equiv 1 \pmod{n}$ (and thus proves that n is composite), or a declaration that n is probably prime.

Step 1. Initialize iteration counter: $i = 0$:
Step 2. Randomly* choose an integer a, $1 < a < n-1$, compute (with Algorithm 6.5) $a^{n-1} \pmod{n}$, and update the iteration counter $i \rightarrow i+1$.
Step 3. If $a^{n-1} \not\equiv 1 \pmod{n}$, then declare n is composite, and a as a witness to this fact, and exit the algorithm. Otherwise, go back to Step 2, unless $i = k$, in which case we declare n is probably prime and exit the algorithm.

We point out that even though the above algorithm and the others that follow are called primality tests, they are not capable of proving that a number is prime; they can only prove compositeness. One drawback of the Fermat primality test is that it does not come with any guarantee on the probability that any declared probable prime really is prime. (Our next primality test will come with such a guarantee, however.)

Example 8.8

Apply the Fermat primality test (Algorithm 8.2) with $k = 4$ to the following odd integers n: (a) $n = 409$, (b) $n = 721$

Solution: Part (a): $n = 409$

Step 1. Initialize the trial counter $i = 0$.
Step 2. Randomly generate a base: $a = 238$. We use Algorithm 6.5 to compute the modular power $a^{n-1} \equiv 238^{408} \equiv 1 \pmod{409}$. So 409 has passed Fermat's primality test with this value of a; since i is now 1, we repeat Step 2.
Step 2. (Second repetition) Randomly generate a base: $a = 222$. We use Algorithm 6.5 to compute the modular power $a^{n-1} \equiv 222^{408} \equiv 1 \pmod{409}$. So 409 has passed Fermat's primality test with this value of a; since i is now 2, we repeat Step 2.
Step 2. (Third repetition) Randomly generate a base: $a = 356$. We use Algorithm 6.5 to compute the modular power $a^{n-1} \equiv 356^{408} \equiv 1 \pmod{409}$. So 409 has passed Fermat's primality test with this value of a; since i is now 3, we repeat Step 2.

* We remind the reader that the computer implementation material of Chapter 1 provides schemes for computer generations of random integers in specified ranges.

Step 2. (Fourth repetition) Randomly generate a base: $a = 109$. We use Algorithm 6.5 to compute the modular power $a^{n-1} \equiv 109^{408} \equiv 1 (\text{mod } 409)$. So 409 has passed Fermat's primality test with this value of a, and since i is now 4, this is the final iteration of Step 2.

Step 3. Declare 409 as probably prime. (The reader may check that 409 is indeed prime; the test worked.)

Part (b): $n = 721$

Step 1. Initialize the trial counter $i = 0$.

Step 2. Randomly generate a base: $a = 230$. We use Algorithm 6.5 to compute the modular power $a^{n-1} \equiv 230^{720} \equiv 484 (\text{mod } 721)$.

Step 3. Since $a^{n-1} \not\equiv 1 (\text{mod } n)$, Fermat's test has proved that $n = 721$ is composite with witness $a = 230$. The reader may check this by factoring $721 = 7 \cdot 103$.

Fermat's test worked very well in the above example. For part (b), compositeness was detected in just one (random) try. Indeed, a computer calculation can quickly verify that 684 out of the 718 possible bases in the range $1 < a < 721 - 1$ would have worked as witnesses to show that 721 is composite.

Exercise for the Reader 8.8

Apply the Fermat primality test (Algorithm 8.2) with $k = 4$ to the following odd integers n:

(a) $n = 2581$
(b) $n = 1889$

Carmichael Numbers

Usually if a number n is not prime, it will succumb to Fermat's primality test for almost all values of a. Although very rare, there are extreme exceptions for which Fermat's test will detect compositeness only if the base a that is selected is an actual factor of n. (Thus the Fermat test for detecting compositeness for one of these numbers would be about as effective as trying to factor the number by randomly guessing at factors.)

Definition 8.4

A composite number $n > 1$ is called a **Carmichael number** if $a^{n-1} \equiv 1 (\text{mod } n)$, for each integer a that is relatively prime to n.

These numbers are named after the American mathematician Robert D. Carmichael (1879–1967), who introduced these numbers and some of their interesting properties. Carmichael conjectured in 1912 that there were infinitely many Carmichael numbers, but this fact was not proved

until 80 years later (see [AlGrPo-92]). The first three Carmichael numbers are 561, 1105, and 1729, and there are just 2163 Carmichael numbers that are less than 25 billion.

The Miller–Rabin Test

Apart from the existence of Carmichael numbers, another drawback of the Fermat primality test is that no matter how many iterations are used, it comes with no guaranteed confidence level of the probability that any probable primes produced will actually be prime. With a bit more work, a much more effective probabilistic primality test, known as the *Miller–Rabin primality test*,* can be developed that transcends both of these weaknesses. There will be a very quantitative performance guarantee, and there will be no analogue of Carmichael numbers for the Miller–Rabin test. As with Fermat's test, this one will depend on Fermat's little theorem, but it will hinge on the following two additional results, the second of which is a refined version of the contrapositive of Fermat's little theorem.

Lemma 8.11: Square Roots of One mod p

If p is an odd prime, then $\sqrt{1} \equiv \pm 1 \pmod{p}$; that is, modulo an odd prime, 1 has exactly two square roots, namely ± 1.

Proof: Modulo any integer n, $(\pm 1)^2 \equiv 1 (\mathrm{mod}\, n)$, so ± 1 are always (modular) square roots of 1. We need to show that modulo an odd prime p, there are no others. Indeed, if x is a square root of 1 (mod p), then $x^2 \equiv 1 \Rightarrow x^2 - 1 \equiv 0 \Rightarrow (x+1)(x-1) \equiv 0 (\mathrm{mod}\ p) \Rightarrow p \mid (x+1)(x-1)$. By Euclid's lemma (Lemma 2.7) this, in turn, implies that either $p \mid (x+1)$, or $p \mid (x-1)$; that is, $x \equiv -1$ or $x \equiv 1 (\mathrm{mod}\ p)$, as asserted. □

The test derives from the contrapositive of the following result, just as the Fermat primality test came from the contrapositive of Fermat's little theorem.

Proposition 8.12

Suppose that p is an odd prime and $1 < a < p-1$. Write $p - 1 = 2^f m$, where m is an odd integer. Then either $a^m \equiv 1 (\mathrm{mod}\ p)$ or $a^{2^j m} \equiv -1 (\mathrm{mod}\ p)$, for some j, $0 \le j < f$.

* The nonprobabilistic version of this test was discovered by Gary L. Miller in 1976 [Mil-76]. It came with an effectiveness guarantee, but the drawback was that this guarantee rested upon an unproved conjecture in number theory (the generalized Riemann hypothesis). In 1980, Michael O. Rabin ([Rab-80]) converted this algorithm into a probabilistic one that (importantly) came with the following unconditional guarantee: If the test finds a number to be composite, it is guaranteed to be composite; if the test finds the number to be "probably prime," it has at least a 75% chance of being prime. This guarantee can be improved to attain as high a percentage as one wishes, simply by (independently) iterating the test. Moreover, it has been found that the Miller–Rabin algorithm does much better, on average, than the conservative 75% guarantee level; see [DaLaPo-93].

Proof: Fermat's little theorem (Theorem 8.2) tells us that $a^{p-1} \equiv 1 (\text{mod } p)$. Since $a^{(p-1)/2} \equiv a^{2^{f-1}m}$ is a square root of $a^{p-1} (\text{mod } p)$, Lemma 8.11 allows us to conclude that $a^{2^{f-1}m} \equiv \pm 1 (\text{mod } p)$. If $a^{2^{f-1}m} \equiv -1 (\text{mod } p)$, then the second assertion holds; otherwise, we can continue to take square roots in this fashion, either verifying the second assertion of the proposition, or else getting all the way to $a^m \equiv 1 (\text{mod } p)$, which is the first assertion of the proposition. □

The contrapositive of Proposition 8.12 can be formulated as follows:

Contrapositive of Proposition 8.12

If $n > 1$ is an odd integer, with $n - 1 = 2^f m$, where m is an odd integer, and we can find an integer a, with $1 < a < n - 1$ such that $a^m \not\equiv 1 (\text{mod } n)$ and $a^{2^j m} \not\equiv -1 (\text{mod } n)$, for all j, $0 \le j < f$, then n must be composite.

As with the Fermat test, there does not seem to be a good (deterministic) method for choosing such an a to prove compositeness, so a random choice is most effective. The following implementation of the Miller–Rabin test is set up to minimize its complexity. It relies on the simple fact that if $a^{2^j m} \equiv 1 (\text{mod } n)$ for some nonnegative integer j, then $a^{2^\ell m} \equiv 1 (\text{mod } n)$, for each $\ell \ge j$.

Algorithm 8.3: The Miller–Rabin Primality Test

Inputs: An odd integer $n > 3$, suspected to be prime, and a positive integer k.
Output: Either a declaration that n is composite, along with a witness integer a that violates the contrapositive of Proposition 8.12 (and thus proves that n is composite), or a declaration that n is probably prime.

Step 1. First express $n - 1$ as $2^f m$, where m is an odd integer. Initialize the main iteration counter: $i = 0$.

Step 2. Randomly choose an integer a, $1 < a < n - 1$. Calculate (with Algorithm 6.5) $A_0 \equiv a^m (\text{mod } n)$. If $A_0 \equiv \pm 1 (\text{mod } n)$, update main iteration counter $i \to i + 1$, and move on to Step 3, otherwise enter into the following FOR loop:

FOR $j = 1$ TO $f - 2$

Compute $A_j \equiv A_{j-1}^2 (\text{mod } n)$ <this is $a^{2^j m} (\text{mod } n)$>

IF $A_j \equiv 1 (\text{mod } n)$

Declare n as composite, output the witness a, and EXIT algorithm.

ELSE IF $A_j \equiv -1 (\text{mod } n)$

Update main iteration counter $i \to i + 1$, and move on to Step 3.

END <IF>

END <FOR>

Compute $A_{f-1} \equiv A_{f-2}^2 (\text{mod } n)$ <this is $a^{(n-1)/2} (\text{mod } n)$>

IF $A_{f-1} \not\equiv -1 (\text{mod } n)$
  Declare n as composite, output the witness a, and EXIT program.
ELSE Update the main iteration counter $i \rightarrow i + 1$.
END <IF>

Step 3. If i equals k, declare n is probably prime, and exit the algorithm. Otherwise, go back to Step 2.

An integer a that proves n is composite in the Miller–Rabin primality test is called a **witness** for the compositeness of n. Observe that if the Miller–Rabin test declares n to be probably prime by using a certain integer a, then $a^{2^j m} \equiv \pm 1 (\text{mod } n)$ for some j, $0 \leq j < k$, and from this it follows (by repeated squaring) that $a^{n-1} \equiv a^{2^k m} \equiv 1 (\text{mod } n)$, so that the Fermat test would have also declared n to be probably prime. Thus the Miller–Rabin test is at least as effective as the Fermat test; in fact, it is more effective—the Computer Implementation and Exercises material at the end of this chapter will examine this phenomenon. But what is even more important and useful is that the Miller–Rabin algorithm comes with the following performance guarantee.

Theorem 8.13: Performance Guarantee for the Miller–Rabin Primality Test

If $n \geq 3$ is an odd composite number, then at most $(n-1)/4$ of the numbers in the set $\{1, 2, \cdots, n-1\}$ that are relatively prime to n will not serve as witnesses to the compositeness of n in the Miller–Rabin primality test. Thus, the probability that the Miller–Rabin test of Algorithm 8.3 with k independent iterations declares n to be probably prime is at most $(1/4)^k$.

So, for example, if we perform $k = 20$ iterations, and the Miller–Rabin test has declared that n is probably prime, the probability that this is incorrect is smaller than $(1/4)^{20} = 9.0949... \times 10^{-13}$, or less than 1 in 1 trillion! Of course, just like with the Fermat test, if a compositeness conclusion is produced, the result is 100 percent correct. Such a test can be performed very quickly on most computing platforms to produce primes of several hundred digits; the failure rate is so low that the results are reliable for most practical purposes.* The contemporary French number theorist Henri Cohen has referred to such probable primes as **industrial-grade primes**.

* The reader will have the opportunity to experiment with such examples in the Computer Implementation and Exercises material at the end of the chapter. One important fact to point out is that unless one is working on a symbolic computing platform, the default floating point arithmetic systems on most computing platforms will only be able to deal accurately with integers up to 15 digits or so. Thus, in order to effectively implement the algorithms in this chapter with larger integers, one will need to make sure that the computing platform has symbolic functionality. (Most computing platforms have this capability but not necessarily as a default mode, so some modifications in syntax might be required.) All of the applets for this book are designed to have symbolic functionality.

The Miller–Rabin Test with a Factoring Enhancement

With an additional minor observation, it is sometimes possible to get factors of n, in cases where the Miller–Rabin test finds n to be composite. Indeed, within (and using the notation of) Step 2 of Algorithm 8.3, if it is found that $A_j \equiv 1 (\text{mod } n)$ (and so n is declared composite) at a particular value of j, $1 \le j \le f-1$, this means that $a^{2^j m} \equiv 1 (\text{mod } n)$, which means that $n \mid a^{2^j m} - 1$ and $a^{2^{j-1} m} \not\equiv \pm 1 (\text{mod } n)$ (the latter being true since the FOR loop would have exited previously if this were not the case). But since $n \mid a^{2^j m} - 1 = (a^{2^{j-1} m} - 1)(a^{2^{j-1} m} + 1)$, we may conclude that n must share a nontrivial common factor with $a^{2^{j-1} m} - 1$ (because $a^{2^{j-1} m} + 1 \not\equiv n$, so n could not divide completely into this second factor). This means that gcd $(a^{2^{j-1} m} - 1, n)$ is a nontrivial factor of n. We restate this modified Miller–Rabin algorithm.

Algorithm 8.4: The Miller–Rabin Primality Test with a Factoring Enhancement

Inputs: An odd integer $n > 3$, suspected to be prime, and a positive integer k.
Output: Either a declaration that n is composite, along with either (i) a nontrivial factor of n, or (ii) a witness integer a that violates the contrapositive of Proposition 8.11 (and thus proves that n is composite), or a declaration that n is probably prime.

The algorithm is the same as Algorithm 8.3, with the following two changes:

(1) The IF statement within the FOR loop of Step 2 needs the following modification:

```
...
IF A_j ≡ 1 (mod n)
   Declare n as composite, output the nontrivial factor gcd(A_{j-1}, n),
   of n
   (computed with the Euclidean Algorithm 2.1), and EXIT algorithm
ELSE ···
```

(2) The IF statement immediately following the FOR loop of Step 2 needs the following modification:

```
IF A_{f-1} ≢ ±1(mod n)
   Declare n as composite, output the witness a, and EXIT program
ELSE IF A_{f-1} ≡ 1(mod n)
   Declare n as composite, output the nontrivial factor gcd(A_{f-2}, n),
   of n
   (computed with the Euclidean Algorithm 2.1), and EXIT algorithm
END <IF>
```

Example 8.9

Apply the enhanced Miller–Rabin primality test (Algorithm 8.4) with $k = 2$ to the following odd integers n:

(a) $n = 409$
(b) $n = 721$

Solution: Part (a): $n = 409$

Step 1. First we express $n - 1 = 408$ as $2^f \cdot m$, where $f = 3$ and $m = 51$. Initialize counter $i = 0$.

Step 2. Randomly generate a base: $a = 216$. We compute $A_0 = a^m \equiv 216^{51} \equiv 1 (\text{mod } 409)$. Since $A_0 \equiv \pm 1 (\text{mod } n)$, this trial finds n to be probably prime. Since i is now 1, we repeat Step 2.

Step 2. (Second repetition) Randomly generate a base: $a = 196$. We compute $A_0 = a^m \equiv 196^{51} \equiv 143 (\text{mod } 409)$. Since $A_0 \not\equiv \pm 1 (\text{mod } n)$, we enter into the FOR loop. For $j = 1$, we compute $A_1 \equiv A_0^2 \equiv 143^2 \equiv 408 (\text{mod } 409)$. Since $A_1 \equiv -1 (\text{mod } n)$, this trial finds n to be probably prime. Since i is now 2, this was the final iteration of Step 2.

Step 3. Declare 409 as probably prime. (The reader may check that 409 is indeed prime so the test worked.)

Part (b): $n = 721$

Step 1. First we express $n - 1 = 720$ as $2^f \cdot m$, where $f = 4$ and $m = 45$. Initialize counter $i = 0$.

Step 2. Randomly generate a base: $a = 641$. We compute $A_0 = a^m \equiv 641^{45} \equiv 64 (\text{mod } 721)$. Since $A_0 \not\equiv \pm 1 (\text{mod } n)$, we enter into the FOR loop. For $j = 1$, we compute $A_1 \equiv A_0^2 \equiv 64^2 \equiv 491 (\text{mod } 721)$. Since $A_1 \not\equiv \pm 1 (\text{mod } n)$, we next (for $j = 2$) compute $A_2 \equiv A_1^2 \equiv 491^2 \equiv 267 (\text{mod } 721)$. Since the FOR loop has completed (without finding n composite or probably prime), we do one last squaring: $A_3 \equiv A_2^2 \equiv 267^2 \equiv 631 (\text{mod } 721)$. Since $A_3 \not\equiv -1 (\text{mod } n)$, this proves that n is composite, with witness $a = 641$.

Exercise for the Reader 8.9

Apply the enhanced Miller–Rabin primality test (Algorithm 8.4) with $k = 2$ to the following odd integers n:

(a) $n = 2581$
(b) $n = 1889$

The Pollard $p - 1$ Factoring Algorithm

We close this chapter with a factoring algorithm. Although the enhanced Miller–Rabin algorithm can sometimes help with factoring, this is not its primary goal, and it is not guaranteed to do anything more than test for

primality. Although factoring is a much more difficult problem than primality testing and there are no known efficient algorithms for this problem, there are sometimes better approaches than a brute-force approach (of attempting to factor n by checking divisibility of all prime numbers $p \le \sqrt{n}$). Factoring algorithms have been designed to perform well if the composite number being factored has certain properties (that the algorithm is designed to take advantage of). We present such a method due to John Pollard [Pol-74], known now as *Pollard's $p-1$ method*. This method tends to work well if n has a prime factor p for which $p-1$ has only small prime factors. In Chapter 12, we will give another factoring algorithm based on elliptic curves that is more powerful since it requires only that n has a prime factor p for which some numbers close to p have only small prime factors. As a countermeasure to Pollard's method, the notion of *strong primes* has evolved, initially defined to be a prime p for which $p-1$ has at least one large prime factor. Since Pollard's method was announced, primes used in cryptosystems should be strong primes. The notion has been extended to include primes that are resistant to other specialized factoring algorithms. The chapter Exercises and Computer Implementation material examine some methods for generating strong primes.

As with the primality tests developed earlier, Pollard's $p-1$ factorization algorithm is based on Fermat's little theorem. We explain how and why the algorithm works and then summarize it. We assume that a composite number n (that we wish to factor) has a prime factor p for which $p-1$ has only small prime factors. It follows that we will have $p-1 \mid B!$ for a not-too-large positive integer B. For example, if $p = 26{,}951$, then $p-1 = 2 \cdot 5^2 \cdot 7^2 \cdot 11$, and the reader may check that $p-1 \mid 14!$ and that any larger but no smaller factorial will do. If we choose any base $a > 1$ ($a = 2$ is most often used), then Fermat's little theorem tells us that

$$a^{B!} \equiv a^{(p-1)q} \equiv (a^{(p-1)})^q \equiv (1)^q \equiv 1 (\text{mod } p) \Rightarrow p \mid a^{B!} - 1$$

We consider $d = \gcd(a^{B!} - 1, n)$. Now, if n has another prime factor q, it is unlikely that $q \mid a^{B!} - 1$, unless $q - 1$ also has only small prime factors. This means that d will be a nontrivial factor of n.

Algorithm 8.5: Pollard's p – 1 Factorization Algorithm

Inputs: An odd composite integer $n > 3$ and a positive integer B. An optional third input is a positive integer base a, with default value $a = 2$.
Output: Either a nontrivial factor of n or no output in case the algorithm does not find one.

Step 1. Compute $a^{B!} (\text{mod } n)$ by using fast modular exponentiation B times in the following chain:

$$a^{1!} \equiv a,\ a^{2!} \equiv (a^{1!})^2,\ a^{3!} \equiv (a^{2!})^3, \cdots a^{B!} \equiv (a^{[B-1]!})^B (\text{mod } n).$$

Step 2. Use the Euclidean algorithm (Algorithm 2.1) to compute $d = \gcd(a^{B!} - 1, n)$ (using the representative for $a^{B!}$ that was found in Step 1). If $d > 1$, output d as a nontrivial factor of n.

We now give a "small" example, although the sizes of the numbers are about the limit on what can be done with any common floating point

arithmetic computing platform. Examples with larger integers would require symbolic computing platforms and will be considered in the Computer Implementation material at the end of the chapter.

Example 8.10

Apply Pollard's $p - 1$ factorization algorithm to the integer 58,932,967.

Solution: We will apply the algorithm using $B = 12$.

Step 1. We compute $a^{B!} \pmod{n}$ by using fast modular exponentiation 12 times:

$$
\begin{aligned}
2^{1!} &\equiv 2 \\
2^{2!} &\equiv (2^{1!})^2 \equiv 4 \\
2^{3!} &\equiv (2^{2!})^3 \equiv 64 \\
2^{4!} &\equiv (2^{3!})^4 \equiv 16{,}777{,}216 \\
2^{5!} &\equiv (2^{4!})^5 \equiv 6{,}054{,}079 \\
2^{6!} &\equiv (2^{5!})^6 \equiv 56{,}169{,}321 \\
2^{7!} &\equiv (2^{6!})^7 \equiv 55{,}888{,}294 \\
2^{8!} &\equiv (2^{7!})^8 \equiv 47{,}597{,}184 \\
2^{9!} &\equiv (2^{8!})^9 \equiv 9{,}175{,}828 \\
2^{10!} &\equiv (2^{9!})^{10} \equiv 35{,}101{,}026 \\
2^{11!} &\equiv (2^{10!})^{11} \equiv 41{,}033{,}283 \\
2^{12!} &\equiv (2^{11!})^{12} \equiv 37{,}504{,}803
\end{aligned}
$$

Step 2. We use the Euclidean algorithm (Algorithm 2.1) to compute $d = \gcd(a^{B!} - 1, n) = \gcd(37{,}504{,}802; 58{,}932{,}967) = 7351$. This is the outputted nontrivial factor of n. Dividing n by this gives 8017, thereby making significant progress in factoring n.

Note that the factor that was found (7351) in the above example is a prime p (as the reader may check) and $p - 1$ factors as $2 \cdot 3 \cdot 5^2 \cdot 7^2$. This shows that the condition needed for Pollard's algorithm is indeed satisfied. Interestingly, although our theoretical explanation of the method would have suggested that we use a value of B to be at least 14, the example showed that the smaller value 12 worked. It turns out that for this example, any value of B greater than 9 would work to produce the above prime factor.

Exercise for the Reader 8.10

Apply Pollard's $p - 1$ factorization algorithm to the integer 12,637,211 using $B = 15$.

Although it is not guaranteed to produce prime factors of n, Pollard's $p - 1$ factorization algorithm is an often-used tool. In practice, one first tries a few trial divisions with smaller primes (on a computer, primes of up to 1 billion or so could be quickly checked), and then a primality test should be applied to what is left. If any composite factors remain,

Pollard's $p - 1$ factorization algorithm is repeatedly applied (perhaps by increasing the values of B). Anything remaining should be tested with a primality test, and if found to be composite, more sophisticated methods (such as the elliptic curve method that we will present in Chapter 12) can be applied. One nice feature of Pollard's algorithm is that if we try it and it does not produce a factor, the modular powers that were computed can still be used if we run the algorithm again with a larger value of B.

Chapter 8 Exercises

1. (a) Use the prime number theorem to estimate the number of primes that are less than 1 billion.
   (b) Use the prime number theorem to estimate the number of primes that lie between 1 billion and 10 billion.
   (c) Use the prime number theorem to estimate the number of primes that lie between 1 billion and 1 trillion.

2. (a) Use the prime number theorem to estimate the number of primes that are less than 1 thousand.
   (b) Use the prime number theorem to estimate the number of primes that lie between 1 thousand and 10 thousand.
   (c) Use the prime number theorem to estimate the number of primes that lie between 1 thousand and 1 million.

3. (a) Use the prime number theorem to estimate the number of 100-bit primes. This will be the number of primes between 2^{99} and 2^{100}.
   (b) If we randomly pick a 100-bit odd integer, use the result of part (a) to estimate the probability that it will be prime.
   (c) Use the prime number theorem to estimate the number of 1000-bit primes. This will be the number of primes between 2^{999} and 2^{1000}.
   (d) If we randomly pick a 1000-bit odd integer, use the result of part (c) to estimate the probability that it will be prime.

4. (a) Use the prime number theorem to estimate the number of 50-bit primes. This will be the number of primes between 2^{49} and 2^{50}.
   (b) If we randomly pick a 50-bit odd integer, use the result of part (a) to estimate the probability that it will be prime.
   (c) Use the prime number theorem to estimate the number of 5000-bit primes. This will be the number of primes between 2^{4999} and 2^{5000}.
   (d) If we randomly pick a 5000-bit odd integer, use the result of part (c) to estimate the probability that it will be prime.

5. Perform each of the following modular exponentiations, first using (i) fast modular exponentiation (Algorithm 6.5) and then (ii) Fermat's little theorem.
   (a) $2^{58} \pmod{11}$
   (b) $9^{102} \pmod{13}$
   (c) $12^{207} \pmod{23}$
   (d) $17^{1236} \pmod{47}$

6. Perform each of the following modular exponentiations, first using (i) fast modular exponentiation (Algorithm 6.5) and then (ii) Fermat's little theorem.
   (a) $3^{45} \pmod{13}$
   (b) $6^{101} \pmod{17}$
   (c) $11^{1977} \pmod{29}$
   (d) $22^{1437} \pmod{53}$

7. Compute each of the indicated values of Euler's phi function:
   (a) $\phi(60)$
   (b) $\phi(248)$
   (c) $\phi(1224)$
   (d) $\phi(9900)$

8. Compute each of the indicated values of Euler's phi function:
   (a) $\phi(50)$
   (b) $\phi(360)$
   (c) $\phi(987)$
   (d) $\phi(10{,}000)$

9. (a) Show that if n is an even positive integer, then $\phi(2n) = 2\phi(n)$.
   (b) Show that if n is an odd positive integer, then $\phi(2n) = \phi(n)$.

10. (a) Show that if n is a positive integer with $n \equiv 0 \pmod 3$, then $\phi(3n) = 3\phi(n)$.
    (b) Show that if n is a positive integer with $n \not\equiv 0 \pmod 3$, then $\phi(3n) = 2\phi(n)$.

11. Find all positive integer solutions (if any) of the following equations:
    (a) $\phi(n) = 1$
    (b) $\phi(n) = 4$
    (c) $\phi(n) = 5$
    (d) $\phi(n) = 12$

12. Find all positive integer solutions (if any) of the following equations:
    (a) $\phi(n) = 2$
    (b) $\phi(n) = 3$
    (c) $\phi(n) = 6$
    (d) $\phi(n) = 14$

13. Use Euler's theorem to compute each of the following modular exponentiations. Write each answer as an integer in $\{1, 2, \cdots, m-1\}$, if you are working mod m.
    (a) $2^{1256} \pmod{15}$
    (b) $7^{3945} \pmod{20}$
    (c) $2^{22,970} \pmod{25}$
    (d) $8^{32,149} \pmod{35}$

14. Compute each of the indicated powers, working in modular arithmetic that is specified. Write each answer as an integer in $\{1, 2, \cdots, m-1\}$, if you are working mod m.
    (a) $3^{1256} \pmod 8$
    (b) $12^{3945} \pmod{25}$
    (c) $3^{22,970} \pmod{40}$
    (d) $13^{32,149} \pmod{15}$

15. (a) As in the solution of Example 8.4, create a table of all modular powers of the modular integers a that are relatively prime with $n = 6$ (up to the $\phi(n)th$). Use the table to identify the orders of each of these modular integers as well as any primitive roots.
    (b) Repeat part (a) for $n = 12$.

16. (a) As in the solution of Example 8.4, create a table of all modular powers of the modular integers a that are relatively prime with $n = 10$ (up to the $\phi(n)th$). Use the table to identify the orders of each of these modular integers as well as any primitive roots.
    (b) Repeat part (a) for $n = 11$.

17. Compute each of the following orders, if they exist:
    (a) $\text{ord}_{10}(3)$
    (b) $\text{ord}_{21}(6)$
    (c) $\text{ord}_{304}(21)$

18. Compute each of the following orders, if they exist:
    (a) $\text{ord}_{11}(5)$
    (b) $\text{ord}_{17}(2)$
    (c) $\text{ord}_{427}(21)$

19. For each of the following integers n, do the following:
    (i) Use Theorem 8.7 to determine whether there are any primitive roots mod n; if so, how many will there be?
    (ii) If there are primitive roots, find one.
        (a) $n = 12$
        (b) $n = 13$
        (c) $n = 14$

20. For each of the following integers n, do the following:
    (i) Use Theorem 8.7 to determine whether there are any primitive roots mod n; if so, how many will there be?
    (ii) If there are primitive roots, find one.
        (a) $n = 16$
        (b) $n = 17$
        (c) $n = 18$

21. For each of the following integers n, do the following:
    (i) Determine whether there are any primitive roots mod n; if so, how many will there be?
    (ii) If there are primitive roots, find one. For parts (a) and (b) give the smallest primitive root; for parts (c) and (d), give any primitive root.

(iii) If there are primitive roots, use the one you found in (ii) to construct another.
   (a) $n = 25$
   (b) $n = 39$
   (c) $n = 31$
   (d) $n = 50$
   (e) $n = 52$
   (f) $n = 29{,}791$

22. For each of the following integers n, do the following:
   (i) Determine whether there are any primitive roots mod n; if so, how many will there be?
   (ii) If there are primitive roots, find one. For parts (a) and (b) give the smallest primitive root, for parts (c) and (d), give any primitive root.
   (iii) If there are primitive roots, use the one you found in (ii) to construct another.
      (a) $n = 17$
      (b) $n = 81$
      (c) $n = 323$
      (d) $n = 289$
      (e) $n = 4913$
      (f) $n = 162$

23. (a) Verify that $g = 3$ is a primitive root of 223.
   (b) How many integers mod 223 have order 6? If such elements exist, find one.
   (c) How many integers mod 223 have order 74? If such elements exist, find one.
   (d) How many integers mod 223 have order 10? If such elements exist, find one.

24. (a) Verify that $g = 3$ is a primitive root of 566.
   (b) How many integers mod 556 have order 12? If such elements exist, find one.
   (c) How many integers mod 556 have order 6? If such elements exist, find one.
   (d) How many integers mod 223 have order 94? If such elements exist, find one.

25. Use Gauss's algorithm (Algorithm 8.1) to find a primitive root of the following primes:
   (a) $p = 107$
   (b) $p = 211$
   (c) $p = 653$

26. Use Gauss's algorithm (Algorithm 8.1) to find a primitive root of the following primes:
   (a) $p = 127$
   (b) $p = 233$
   (c) $p = 733$

27. Apply the Fermat primality test (Algorithm 8.2) with $k = 4$ to the following odd integers:
(a) $n = 527$
(b) $n = 523$
(c) $n = 943$
(d) $n = 5963$
(e) $n = 11{,}303$
(f) $n = 1811$

28. Apply the Fermat primality test (Algorithm 8.2) with $k = 4$ to the following odd integers:
(a) $n = 449$
(b) $n = 629$
(c) $n = 1147$
(d) $n = 4559$
(e) $n = 8893$
(f) $n = 9727$

29. Apply the Miller–Rabin test (Algorithm 8.3) with $k = 4$ to each of the odd integers given in Exercise 27.

30. Apply the Miller–Rabin test (Algorithm 8.3) with $k = 4$ to each of the odd integers given in Exercise 28.

31. Apply the enhanced Miller–Rabin test (Algorithm 8.4) with $k = 4$ to each of the odd integers given in Exercise 27.

32. Apply the enhanced Miller–Rabin test (Algorithm 8.4) with $k = 4$ to each of the odd integers given in Exercise 28.

33. Apply Pollard's $p - 1$ factorization algorithm (Algorithm 8.5) to each of the following integers:
(a) $n = 7{,}781{,}707$
(b) $n = 12{,}418{,}223$
(c) $n = 47{,}486{,}269$

34. Apply Pollard's $p - 1$ factorization algorithm (Algorithm 8.5) to each of the following integers:
(a) $n = 7{,}427{,}207$
(b) $n = 8{,}468{,}039$
(c) $n = 16{,}701{,}131$

35. Determine the ones digit of $3^{100!}$.

36. Determine the ones digit of $17^{100!}$.

37. *True or False.* Indicate whether the following statement is (always) true or false. Then either give an explanation of why it is true (i.e., a proof) or a counterexample of a case where it can be false:

If p is a prime, g is a primitive root mod p, and q is a prime factor of $p - 1$, then $g^{(p-1)/q} \not\equiv 1 (\text{mod } p)$.

38. *True or False.* Indicate whether the following statement is (always) true or false. Then either give an explanation of why it is true (i.e., a proof) or a counterexample of a case where it can be false:

    If a and b are relatively prime positive integers less than a prime p, then $\text{ord}_p(ab) = \text{ord}_p(a) \cdot \text{ord}_p(b)$.

39. *True or False.* Indicate whether the following statement is (always) true or false. Then either give an explanation of why it is true (i.e., a proof) or a counterexample of a case where it can be false:

    If a and b are different integers greater than 1, $\phi(ab) = \phi(a)\phi(b)$.

40. Prove that if $n > 2$ is any integer, then $\phi(n)$ is even.

    Note: The next three exercises together will build up a proof of Proposition 8.3.

41. Prove that if p is a prime and k is a nonnegative integer, then $\phi(p^k) = p^{k-1}(p-1)$.

    Suggestion: By definition, $\phi(p^k)$ is the number of integers from the list $1, 2, \cdots, p^k$ that are relatively prime to p^k. Being relatively prime to p^k is equivalent to being relatively prime to p. Thus, the elements of this list that are not counted are precisely every pth element (the multiples of p).

42. Suppose that n and m are relatively prime positive integers.
    (a) Show that an integer a is relatively prime to nm if, and only if, it is relatively prime to both n and m.
    (b) Prove that if n and m are relatively prime positive integers, then $\phi(nm) = \phi(n)\phi(m)$.
    (c) Prove that if $n_1, n_2, \cdots, n_\ell$ are pairwise relatively prime integers, then $\phi(n_1 n_2 \cdots n_\ell) = \phi(n_1)\phi(n_2)\cdots\phi(n_\ell)$.

    Note: In number theory, functions on the positive integers that satisfy the condition of part (b) are called *multiplicative functions.*

    Suggestion: For part (b), arrange the integers $1, 2, \cdots, mn$ into the following array:

$$\begin{array}{ccccccc} 1 & 2 & \cdots & k & \cdots & m \\ m+1 & m+2 & \cdots & m+k & \cdots & 2m \\ 2m+1 & 2m+2 & \cdots & 2m+k & \cdots & 3m \\ \vdots & \vdots & & \vdots & & \vdots \\ (n-1)m+1 & (n-1)m+2 & \cdots & (n-1)m+k & \cdots & nm \end{array}$$

    By part (a), if we remove the numbers from this array that are relatively prime to either n or m, the number of integers left will be $\phi(nm)$. Use the fact that $\gcd(qm+k, m) = \gcd(k, m)$ to

conclude that either *all* of the numbers in the kth column are relatively prime to m or none will be, and conclude that the columns of relatively prime to m numbers will be those $\phi(m)$ columns corresponding to the values of k that are relatively prime to m. It suffices to show that in each of these $\phi(m)$ columns, exactly $\phi(n)$ of the n numbers will be relatively prime to n. Accomplish this by first showing that the n numbers in any such column are pairwise incongruent (mod n).

43. Prove Proposition 8.3 by using the results of the previous two exercises.

44. Suppose that $n > 1$ is a positive integer that has a primitive root.
    (a) If k is a positive integer and $k|\phi(n)$, show that the equation $x^k \equiv 1 (\mathrm{mod}\, n)$ has exactly k different solutions (mod n).
    (b) *Convergence of Gauss's Primitive Root Finding Algorithm 8.1.* Show that in Step 2 of Algorithm 8.1, we must have $u > \ell$, and hence (as pointed out after the statement of the algorithm) Algorithm 8.1 will always terminate by finding a primitive root.

Suggestions: For part (a), let g be a primitive root. If x is any modular integer that satisfies $x^k \equiv 1 (\mathrm{mod}\, n)$ then x must be relatively prime to n, so by Proposition 8.6, we must have $x \equiv g^j$, for some exponent j. Thus, $1 \equiv x^k \equiv (g^j)^k \equiv g^{jk} (\mathrm{mod}\, n)$, so by Proposition 8.5(a), we may conclude $\phi(n)|jk$, from which we obtain that j must be a multiple of $\phi(n)|k$. Since there are only k such multiples that clearly correspond to different roots of $x^k \equiv 1 (\mathrm{mod}\, n)$ the result follows.

For part (b), use the result of part (a).

Chapter 8 Computer Implementations and Exercises

Note: As mentioned in the chapter text, if your computing platform is a floating point arithmetic system, it may allow you up to only 15 or so significant digits of accuracy. Symbolic systems allow for much greater precision, being able to handle hundreds of significant digits. Some platforms allow users to choose if they wish to work in floating point or symbolic arithmetic. If you are working on such a dual-capability platform, you may wish to create two separate programs for those that might work with large integers: an ordinary version and a symbolic version (perhaps attaching a `Sym` suffix to the names of those of the latter type). In case you do not have access to a symbolic system, some particular questions may need to be skipped or modified so the numbers are of a manageable size.