and has his or her name entered into the record books. Because there are infinitely many prime numbers, there is always a prime larger than the current record. Looking for new primes is done somewhat systematically; rather than checking randomly, people examine numbers that have a special form. For example, in Chapter 7 we will discuss primes of the form $2^p - 1$, where p is prime; such numbers are called *Mersenne primes*. We will see that there is a special test that makes it possible to determine whether $2^p - 1$ is prime, without performing trial divisions. The largest known prime number has been a Mersenne prime for most of the past hundred years. Currently, the world record for the largest prime known is $2^{24,036,583} - 1$.

Formulas for Primes Is there a formula that generates only primes? This is another question that has interested mathematicians for many years. No polynomial in one variable has this property, as Exercise 23 demonstrates. It is also the case that no polynomial in n variables, where n is a positive integer, generates only primes (a result that is beyond the scope of this book). There are several impractical formulas that generate only primes. For example, Mills has shown that there is a constant Θ such that the function $f(n) = [\Theta^{3^n}]$ generates only primes. Here the value of Θ is known only approximately, with $\Theta \approx 1.3064$. This formula is impractical for generating primes not only because the exact value of Θ is not known, but also because to compute Θ you must know the primes that $f(n)$ generates (see [Mi47] for details).

If no useful formula can be used to generate large primes, how can they be generated? In Chapter 6, we will learn how to generate large primes using what are known as probabilistic primality tests.

Primality Proofs

If someone presents you with a positive integer n and claims that n is prime, how can you be sure that n really is prime? We already know that we can determine whether n is prime by performing trial divisions of n by the primes not exceeding $\sqrt{n}$. If n is not divisible by any of these primes, it itself is prime. Consequently, once we have determined that n is not divisible by any prime not exceeding its square root, we have produced a proof that n is prime. Such a proof is also known as a *certificate of primality*.

Unfortunately, using trial division to produce a certificate of primality is extremely inefficient. To see this, we estimate the number of bit operations used by this test. Using the prime number theorem, we can estimate the number of bit operations needed to show that an integer n is prime by trial divisions of n by all primes not exceeding $\sqrt{n}$. The prime number theorem tells us that there are approximately $\sqrt{n}/\log \sqrt{n} = 2\sqrt{n}/\log n$ primes not exceeding $\sqrt{n}$. To divide n by an integer m takes $O(\log_2 n \cdot \log_2 m)$ bit operations. Therefore, the number of bit operations needed to show that n is prime by this method is at least $(2\sqrt{n}/\log n)(c \log_2 n) = c\sqrt{n}$ (where we have ignored the $\log_2 m$ term because it is at least 1, even though it sometimes is as large as $(\log_2 n)/2$). This method of showing that an integer n is prime is very inefficient, for it is necessary not only to know all the primes not larger than $\sqrt{n}$, but to do at least a constant multiple of $\sqrt{n}$ bit operations.

To input an integer into a computer program, we input the binary digits of the integer. Consequently, the computational complexity of algorithms for determining whether an integer is prime is measured in terms of the number of binary digits in the integer. By Exercise 11 in Section 2.3 we know that a positive integer n has $[\log_2 n] + 1$ binary digits. Consequently, a big-O estimate for the computational complexity of an algorithm in terms of number of binary digits of n translates to the same big-O estimate in terms of $\log_2 n$, and vice versa. Note that the algorithm using trial divisions to determine whether an integer n is prime is exponential in terms of the number of binary digits of n, or in terms of $\log_2 n$, because $\sqrt{n} = 2^{\log_2 n/2}$. That is, this algorithm has exponential time complexity, measured in terms of the number of binary digits in n. As n gets large, an algorithm with exponential complexity quickly becomes impractical. Determining whether a number with 200 digits is prime using trial division still takes billions of years on the fastest computers.

Mathematicians have looked for efficient primality tests for many years. In particular, they have searched for an algorithm that produces a certificate of primality in polynomial time, measured in terms of the number of binary digits of the integer input. In 1975, G. L. Miller developed an algorithm that can prove that an integer is prime using $O((\log n)^5)$ bit operations, assuming the validity of a hypothesis called the generalized Riemann hypothesis. Unfortunately, the generalized Riemann hypothesis remains an open conjecture. In 1983, Leonard Adleman, Carl Pomerance, and Robert Rumely developed an algorithm that can prove an integer is prime using $(\log n)^{c \log\log\log n}$ bit operations, where c is a constant. Although their algorithm does not run in polynomial time, it runs in close to polynomial time because the function $\log\log\log n$ grows so slowly. To use their algorithm with an up-to-date PC to determine whether a 100-digit integer is prime requires just a few milliseconds, determining whether a 400-digit integer is prime requires less than a second, and determining whether a 1000-digit integer is prime takes less than an hour. (For more information about their test, see [AdPoRu83] and [Ru83].)

Until 2002, no one was able to find a polynomial time algorithm for proving that a positive integer is prime. In 2002, M. Agrawal, N. Kayal, and N. Saxena, an Indian computer science professor and two of his undergraduate students, announced that they had found an algorithm that can produce a certificate of primality for an integer n using $O((\log n)^{12})$ bit operations. Their discovery of a polynomial time algorithm for proving that a positive integer is prime surprised the mathematical community. Their announcement stated that "*PRIMES* is in *P*." Here, computer scientists denote by *PRIMES* the problem of determining whether a given integer n is prime, and P denotes the class of problems that can be solved in polynomial time. Consequently, *PRIMES* is in P means that one can determine whether n is prime using an algorithm that has computational complexity bounded by a polynomial in the number of binary digits in n, or equivalently, in $\log n$. Their proof can be found in [AgKaSa02] and can be understood by undergraduate students who have studied number theory and abstract algebra. In this paper, they also show that under the assumption of a widely believed conjecture about the density of Sophie Germain primes (primes p for which $2p + 1$ is also prime), their algorithm uses only $O((\log n)^6)$ bit operations. Other mathematicians have also improved on Agrawal, Kayal, and Saxena's result. In particular, H. Lenstra

and C. Pomerance have reduced the exponent 12 in the original estimate to $6 + \epsilon$, where ϵ is any positive real number.

It is important to note that in our discussion of primality tests, we have only addressed *deterministic* algorithms, that is, algorithms that decide with certainty whether an integer is prime. In Chapter 6, we will introduce the notion of probabilistic primality tests, that is, tests that tell us that there is a high probability, but not a certainty, that an integer is prime.

3.1 Exercises

1. Determine which of the following integers are primes.

   a) 101 c) 107 e) 113
   b) 103 d) 111 f) 121

2. Determine which of the following integers are primes.

   a) 201 c) 207 e) 213
   b) 203 d) 211 f) 221

3. Use the sieve of Eratosthenes to find all primes less than 150.

4. Use the sieve of Eratosthenes to find all primes less than 200.

5. Find all primes that are the difference of the fourth powers of two integers.

6. Show that no integer of the form $n^3 + 1$ is a prime, other than $2 = 1^3 + 1$.

7. Show that if a and n are positive integers with $n > 1$ and $a^n - 1$ is prime, then $a = 2$ and n is prime. (*Hint:* Use the identity $a^{kl} - 1 = (a^k - 1)(a^{k(l-1)} + a^{k(l-2)} + \cdots + a^k + 1)$.)

8. (This exercise constructs another proof of the infinitude of primes.) Show that the integer $Q_n = n! + 1$, where n is a positive integer, has a prime divisor greater than n. Conclude that there are infinitely many primes.

9. Can you show that there are infinitely many primes by looking at the integers $S_n = n! - 1$, where n is a positive integer?

10. Using Euclid's proof that there are infinitely many primes, show that the nth prime p_n does not exceed $2^{2^{n-1}}$ whenever n is a positive integer. Conclude that when n is a positive integer, there are at least $n + 1$ primes less than 2^{2^n}.

11. Let $Q_n = p_1 p_2 \cdots p_n + 1$, where $p_1, p_2, \ldots, p_n$ are the n smallest primes. Determine the smallest prime factor of Q_n for $n = 1, 2, 3, 4, 5$, and 6. Do you think that Q_n is prime infinitely often? (*Note:* This is an unresolved question.)

12. Show that if p_k is the kth prime, where k is a positive integer, then $p_n \leq p_1 p_2 \cdots p_{n-1} + 1$ for all integers n with $n \geq 3$.

13. Show that if the smallest prime factor p of the positive integer n exceeds $\sqrt[3]{n}$, then n/p must be prime or 1.