INPUT	OUTPUT
833816926455584605284210207 1	179424673 2038074743 * 22801763489 833816926455584605284210207 1

13.8 Factoring and Primality Testing

Input description: An integer n.

Problem description: Is n a prime number, and if not what are its factors?

Discussion: The dual problems of integer factorization and primality testing have surprisingly many applications for a problem long suspected of being only of mathematical interest. The security of the RSA public-key cryptography system (see Section 18.6 (page 641)) is based on the computational intractability of factoring large integers. As a more modest application, hash table performance typically improves when the table size is a prime number. To get this benefit, an initialization routine must identify a prime near the desired table size. Finally, prime numbers are just interesting to play with. It is no coincidence that programs to generate large primes often reside in the games directory of UNIX systems.

Factoring and primality testing are clearly related problems, although they are quite different algorithmically. There exist algorithms that can demonstrate that an integer is *composite* (i.e. , not prime) without actually giving the factors. To convince yourself of the plausibility of this, note that you can demonstrate the compositeness of any nontrivial integer whose last digit is 0, 2, 4, 5, 6, or 8 without doing the actual division.

The simplest algorithm for both of these problems is brute-force trial division. To factor n, compute the remainder of n/i for all $1 < i \leq \sqrt{n}$. The prime factorization of n will contain at least one instance of every i such that $n/i = \lfloor n/i \rfloor$, unless n is prime. Make sure you handle the multiplicities correctly, and account for any primes larger than $\sqrt{n}$.

Such algorithms can be sped up by using a precomputed table of small primes to avoid testing all possible i. Surprisingly large numbers of primes can be represented in surprisingly little space by using bit vectors (see Section 12.5 (page 385)). A bit vector of all odd numbers less than 1,000,000 fits in under 64 kilobytes. Even tighter encodings become possible by eliminating all multiples of 3 and other small primes.

Although trial division runs in $O(\sqrt{n})$ time, it is *not* a polynomial-time algorithm. The reason is that it only takes $\lg_2 n$ bits to represent n, so trial division takes time exponential in the input size. Considerably faster (but still exponential time) factoring algorithms exist, whose correctness depends upon more substantial number theory. The fastest known algorithm, the *number field sieve*, uses randomness to construct a system of congruences—the solution of which usually gives a factor of the integer. Integers with as many as 200 digits (663 bits) have been factored using this method, although such feats require enormous amounts of computation.

Randomized algorithms make it much easier to test whether an integer is prime. Fermat's little theorem states that $a^{n-1} = 1 (\text{mod } n)$ for all a not divisible by n, provided n is prime. Suppose we pick a random value $1 \leq a < n$ and compute the residue of $a^{n-1} (\text{mod } n)$. If this residue is not 1, we have just proven that n cannot be prime. Such randomized primality tests are very efficient. PGP (see Section 18.6 (page 641)) finds 300+ digit primes using hundreds of these tests in minutes, for use as cryptographic keys.

Although the primes are scattered in a seemingly random way throughout the integers, there is some regularity to their distribution. The *prime number theorem* states that the number of primes less than n (commonly denoted by $\pi(n)$) is approximately $n/\ln n$. Further, there never are large gaps between primes, so in general, one would expect to examine about $\ln n$ integers if one wanted to find the first prime larger than n. This distribution and the fast randomized primality test explain how PGP can find such large primes so quickly.

Implementations: Several general systems for computational number theory are available. PARI is capable of handling complex number-theoretic problems on arbitrary-precision integers (to be precise, limited to 80,807,123 digits on 32-bit machines), as well as reals, rationals, complex numbers, polynomials, and matrices. It is written mainly in C, with assembly code for inner loops on major architectures, and includes more than 200 special predefined mathematical functions. PARI can be used as a library, but it also possesses a calculator mode that gives instant access to all the types and functions. PARI is available at *http://pari.math.u-bordeaux.fr/*

LiDIA (*http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/*) is the acronym for the C++ *Library for Computational Number Theory*. It implements several of the modern integer factorization methods.

A Library for doing Number Theory (NTL) is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields. It is available at *http://www.shoup.net/ntl/*.

Finally, MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library) implements six different integer factorization algorithms, including the quadratic sieve. It is available at *http://www.shamus.ie/*.

Notes: Expositions on modern algorithms for factoring and primality testing include Crandall and Pomerance [CP05] and Yan [Yan03]. More general surveys of computational number theory include Bach and Shallit [BS96] and Shoup [Sho05].

In 2002, Agrawal, Kayal, and Saxena [AKS04] solved a long-standing open problem by giving the first polynomial-time deterministic algorithm to test whether an integer is composite. Their algorithm, which is surprisingly elementary for such an important result, involves a careful analysis of techniques from earlier randomized algorithms. Its existence serves as somewhat of a rebuke to researchers (like me) who shy away from classical open problems due to fear. Dietzfelbinger [Die04] provides a self-contained treatment of this result.

The Miller-Rabin [Mil76, Rab80] randomized primality testing algorithm eliminates problems with Carmichael numbers, which are composite integers that always satisfy Fermat's theorem. The best algorithms for integer factorization include the quadratic-sieve [Pom84] and the elliptic-curve methods [Len87b].

Mechanical sieving devices provided the fastest way to factor integers surprisingly far into the computing era. See [SWM95] for a fascinating account of one such device, built during World War I. Hand-cranked, it proved the primality of $2^{31} - 1$ in 15 minutes of sieving time.

An important problem in computational complexity theory is whether P = NP $\cap$ co-NP. The decision problem "is n a composite number?" used to be the best candidate for a counterexample. By exhibiting the factors of n, it is trivially in NP. It can be shown to be in co-NP, since every prime has a short proof of its primality [Pra75]. The recent proof that composite numbers testing is in P shot down this line of reasoning. For more information on complexity classes, see [GJ79, Joh90].

The integer RSA-129 was factored in eight months using over 1,600 computers. This was particularly noteworthy because in the original RSA paper [RSA78] they had originally predicted such a factorization would take 40 quadrillion years using 1970s technology. Bahr, Boehm, Franke, and Kleinjung hold the current integer factorization record, with their successful attack on the 200-digit integer RSA-200 in May 2005. This required the equivalent of 55 years of computations on a single 2.2 GHz Opteron CPU.

Related Problems: Cryptography (see page 641), high precision arithmetic (see page 423).