V
Primality and Factoring

There are many situations where one wants to know if a large number n is prime. For example, in the RSA public key cryptosystem and in various cryptosystems based on the discrete log problem in finite fields, we need to find a large "random" prime. One interpretation of what this means is to choose a large odd integer n_0 using a generator of random digits and then test n_0, $n_0 + 2$, $\ldots$ for primality until we obtain the first prime which is $\geq n_0$. A second type of use of primality testing is to determine whether an integer of a certain very special type is a prime. For example, for some large prime f we might want to know whether $2^f - 1$ is a Mersenne prime. If we're working in the field of 2^f elements, we saw that every element $\neq 0, 1$ is a generator of $\mathbf{F}_{2^f}^*$ if (and only if) $2^f - 1$ is prime (see Ex.13(a) of § II.1).

A *primality test* is a criterion for a number n *not* to be prime. If n "passes" a primality test, then it *may* be prime. If it passes a whole lot of primality tests, then it is very likely to be prime. On the other hand, if n fails any single primality test, then it is definitely composite. But that leaves us with a very difficult problem: finding the prime factors of n. In general, it is much more time-consuming to factor a large number once it is known to be composite (because it fails a primality test) than it is to find a prime number of the same order of magnitude. (This is an empirical statement, not a theorem; no assertion of this sort has been proved.) The security of the RSA cryptosystem is based on the assumption that it is much easier for someone to find two extremely large primes p and q than it is for someone else, knowing $n = pq$ but not p or q, to find the two factors in n. After discussing primality tests in §1, we shall describe three different factorization methods in §§2–5.