SIAM J. COMPUT.
Vol. 17, No. 2, April 1988





# HOW TO GENERATE FACTORED RANDOM NUMBERS*

ERIC BACH†

**Abstract.** This paper presents an efficient method for generating a random integer with known factorization. When given a positive integer $N$, the algorithm produces the prime factorization of an integer $x$ drawn uniformly from $N/2 < x \leqq N$. The expected running time is that required for $O(\log N)$ prime tests on integers less than or equal to $N$.

If there is a fast deterministic algorithm for primality testing, this is a polynomial-time process. The algorithm can also be implemented with randomized primality testing; in this case, the distribution of correctly factored outputs is uniform, and the possibility of an incorrectly factored output can in practice be disregarded.

**Key words.** factorization, primality, random variate generation

**AMS(MOS) subject classifications.** 1104, 11A51, 11Y05, 65C10

**1. Introduction.** Let $N$ be a positive number, and suppose that we want a random integer $x$ uniformly distributed on the interval $N/2 < x \leqq N$. Further suppose that we do not want to output $x$ in the usual decimal form, but rather as an explicit product of primes.

This is clearly possible if we are willing to factor $x$. However, the best known algorithms for factorization [8], [16] require roughly $O(\log x)^{\sqrt{\log x / \log\log x}}$ steps on input $x$, so this approach is out of the question if $N$ is large. In contrast, the method of this paper uses primality testing rather than factorization. Since there are efficient algorithms for determining primality [1], [10], [13], [17], the method is useful even when $N$ is so large that factorization is infeasible.

The algorithm works by assembling random primes, but it is not clear a priori with what distribution these should be selected, nor how to efficiently implement a desired distribution on the primes. Much of the paper will deal with these questions, in a rather detailed fashion. However, if one is willing to overlook these technicalities, the resulting method can be easily sketched.

It selects a factor $q$ of $x$ whose length is roughly uniformly distributed between 0 and the length of $N$, then recursively selects the factors of a number $y$ between $N/2q$ and $N/q$ and sets $x = y \cdot q$. It has now chosen $x$ with a known bias; to correct this, it flips an unfair coin to decide whether to output $x$ or repeat the whole process.

The results of this paper show not only that the distribution of $x$ is uniform, but that this is a fast algorithm. A rough measure of its running time is the number of primality tests required; this quantity has expected value and standard deviation that are both $O(\log N)$— the same as required to generate a random prime of the same size.

This estimate is the basis for a finer analysis of the running time, which uses some assumptions about primality testing. If there is a deterministic polynomial-time prime test, as proved under the Extended Riemann Hypothesis by Miller [10], then the

* Received by the editors April 25, 1983; accepted for publication (in revised form) August 6, 1985. Sections 1-8 of this article originally appeared as Chapter 2 of the author's Ph.D. dissertation, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, © 1985 by the Massachusetts Institute of Technology Press. A preliminary version of this article was presented at the 15th Annual ACM Symposium on the Theory of Computing 1983.

† Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706. This research was supported by the National Science Foundation, grants MCS-8204506 and DCR-8504485, and the Wisconsin Alumni Research Foundation.

expected number of single-precision operations needed to generate $x$ in factored form can be bounded by a polynomial in $\log N$.

If the method uses a probabilistic polynomial-time prime test such as those of Rabin [13] and Solovay and Strassen [17], a similar result holds. In this case, the distribution of correctly factored numbers is still uniform, and the possibility of producing an incompletely factored output can in practice be disregarded — all within an expected polynomial time bound.

The method has been implemented; on a medium-sized computer, it will generate a 120-digit number in about 2 minutes.

The rest of this paper is organized as follows. Section 2 gives a heuristic derivation of the algorithm, and § 3 gives a general discussion of random variate generation. Section 4 presents the algorithm in explicit form; its running time is analyzed in §§ 5–8. Finally, § 9 gives experimental results.

**2. Heuristics.** Later sections present a detailed algorithm; this one provides motivation and sketches a design based on heuristic arguments.

First, what is meant by a "random factor" of a number? If we write down all numbers between $N/2$ to $N$ in factored form, we will have an array that is roughly rectangular, because the juxtaposition of a number's factors is about as long as the number itself. If the factorizations are arranged one per line, and given in binary notation, the picture will look something like this:

$$\vdots$$

10 10 10 10011 1001110111 100001100111001
11 11 11 1011011101 10100010111000111
10 101 11010011 10111110111110101011
10111111 10000011110111000101010 1
10 10 11 1111111010011 100000111110011
100101 1101101 1100011111010101101
10 111 111 111 10010100011 11111101011
11 101 1011 1101 10111011110111001111
10 10 10 10 11101 11111 11100000000111101
11010100011 11101101001011011011

$$\vdots$$

Choosing a random factorization is equivalent to picking a row at random from this list; if the list were perfectly rectangular, we could do this by choosing a bit at random and taking the row in which it appears.

Now suppose that we wanted to get the effect of this process by choosing a prime factor $p$ first and selecting one of the remaining $N/p$ possibilities uniformly. To do this, we would pick $p$ with probability proportional to its "surface area," that is, proportional to the total number of bits occurring in all copies of $p$.

This suggests selecting the first factor $p$ with probability about $\log p/p \log N$, since $p$ occurs in about $1/p$ of the numbers, and a random bit of such a number will be in $p$ about $\log p/\log N$ of the time (ignoring repeated factors).[1]

It is instructive to see what effect this would have on the *length* of $p$. A weak form of the prime number theorem [6, Thm. 7] implies that for $0 < x < N$,

$$\sum_{p \leq x} \frac{\log p}{p \log N} \cong \frac{\log x}{\log N}.$$

[1] Unless otherwise indicated, all logarithms in this paper are to the base $e = 2.718281\ldots$

Therefore, if we chose $p$ with the proposed probability, the length of $p$ (relative to the length of $N$) would be close to that produced by a uniform distribution, since for $0 \leqq \alpha \leqq 1$, $\log p / \log N < \alpha$ with roughly the same frequency in both cases.

One can justify this uniform length heuristic in another fashion. The factorization of numbers into primes is analogous to the decomposition of permutations into disjoint cycles; for instance, one can easily prove the "prime permutation theorem": a random permutation on $n$ letters is prime (a cycle) with probability $1/n$. This analogy extends to the distribution of factor lengths: Knuth and Trabb Pardo have shown that the relative length of the $k$th largest factor of a random number has the same asymptotic distribution as the relative length of the $k$th largest cycle in a random permutation [7, § 10]. Under this analogy, our prime selection technique corresponds to a process that selects a random letter in a random permutation and outputs the cycle to which it belongs. Results on random permutations [5, p. 258] imply that the length of this cycle is uniformly distributed.

Thus, to choose $x$ uniformly with $N/2 < x \leqq N$, we might proceed as follows. Select a length $\lambda$ uniformly from $(0, \log N)$ and pick the largest prime $p$ with $\log p \leqq \lambda$. Then recursively select $y$ (the remaining bits of $x$) to satisfy $N/2p < y \leqq N/p$, and output $x$, as $p$ times the prime factorization of $y$.

If the distribution of $y$ were uniform, the probability of selecting $x$ would be about

$$\sum_{p|x} \frac{\log p}{p \log N} \cdot \frac{1}{N/p - N/2p}.$$

This is $2/N$, the correct probability for a uniform distribution, times a bias factor of

$$\frac{1}{\log N} \sum_{p|x} \log p.$$

This bias should be close to 1, and it is, provided that $x$ does not have too many repeated prime factors.

Thus, one would suspect that this method is almost right; however, a closer look at the algorithm reveals the complications listed below.

1) Merely picking the biggest prime less than some given value will not do; for one thing, the first member of a twin prime pair will be chosen less frequently than the second. A correct method must be insensitive to these local irregularities.
2) The bias factor is quite small for certain $x$, say powers of 2. This problem can be eliminated by also including prime power factors in the first step, but we must further decide how often these are chosen.
3) At the end of the algorithm, $x$ will have been chosen with a certain bias, but the recursion will not work unless all $x$'s are equally likely. The odds must be changed somehow to make the eventual output uniform.
4) Finally, we imagine selecting $y$, the rest of $x$, from $(N/2p, N/p]$ with probability $2p/N$. However, it is by no means certain, and in general not true, that there are $N/2p$ integers in this range.

Dealing with these problems requires some machinery that will be developed in the next section.

**3. Doctoring the odds.** This section discusses a general technique for using one distribution to simulate another, called the "acceptance-rejection" method [11], [15]. It requires only a little information about the distributions, a source of uniform (0,1) random real numbers, and some extra time.

This method is usually applied in situations where everything is known about the distributions. In our case, we will only know the relative probabilities involved, hence we need the following definition.

Let $(x_1, \ldots, x_n)$ be a finite set. We will say that $X$ has a finite distribution with *odds* $(p_1, \ldots, p_n)$ if $X = x_i$ with probability $p_i / \sum_{j=1}^{n} p_j$. The odds of a distribution are only defined up to a multiplicative constant; this conforms to ordinary usage, in which odds of 2:1 and 10:5 are regarded as identical.

To see how to turn one distribution into another, consider an example. Suppose we have a coin that is biased in favor of heads with odds of 2:1, and we wish to make it fair. This can be done by the following trick. Flip the coin. If it comes up tails, say "tails"; if it comes up heads, say "heads" with probability 1/2, and with probability 1/2 repeat the process.

The stopping time can be analyzed by the following "renewal" argument.The process must flip the biased coin once no matter what happens, and after this first step, it has one chance in three of being born again. Thus the expected stopping time $E(T)$ must satisfy $E(T) = 1 + E(T)/3$, so $E(T) = 3/2$. More generally, $T = t$ with probability $(2/3) \cdot (1/3)^{t-1}$; this is a geometric distribution with expected value 3/2. At each reincarnation, the process has no memory of its past, so the stopping time and the ultimate result are independent.

This example is not very useful, as it requires a fair coin to produce the effect of one; however, it points out some important features of the method. First, decisions are only made locally; after getting, say, heads, a decision can be made without knowing the other possible outcomes or even their total number. Second, only the odds matter; knowing only the relative probability of each outcome is sufficient for undoing the bias.

Here is the general version; we are given odds $(p_1, \ldots, p_n)$ but want odds $(q_1, \ldots, q_n)$. Assume that $q_i \leqq p_i$ for all $i$; we can use the following recipe:

PROCESS A: Acceptance-rejection method.
(*) Select $X$ from the original distribution.
Choose a real number $\lambda$ from the $U(0,1)$ distribution.
If $X = x_i$ and $\lambda < q_i / p_i$, output $Y = x_i$.
If not, go back to (*).

THEOREM 1. *Let $X$ have a finite distribution with odds $(p_1, \ldots, p_n)$. If $q_i \leqq p_i$ for $1 \leqq i \leqq n$, then the output of Process* A *has a finite distribution with odds $(q_1, \ldots, q_n)$. The stopping time $T$ and the output value $Y$ are independent random variables. If $P = \sum_{i=1}^{n} p_i$, $Q = \sum_{i=1}^{n} q_i$, then the stopping time is distributed geometrically with expected value $P/Q$.*

*Proof.* A direct calculation shows that the joint distribution of $T$ and $Y$ is

$$\Pr[T = t, Y = x_i] = \frac{q_i}{Q} \cdot \frac{Q}{P} \left(1 - \frac{P}{Q}\right)^{t-1}. \qquad \square$$

This technique is at the heart of the method, in two ways:

At the top level, the algorithm generates $x$, $N/2 < x \leqq N$, with probability proportional to $\log x$. It accepts $x$ with probability $\log(N/2)/\log x$, producing a uniform distribution.

To select a factor with approximately uniform length, the algorithm chooses prime powers $q$ with odds $\Delta_N(q)$ (defined below). To do this, it first chooses integers $q$ in the following way: 2 and 3 each appear with odds 1/2, 4, 5, 6, and 7 each appear with odds 1/4, and so on. It turns out that $\Delta_N(q) < 1/q$, so acceptance-rejection is used twice: first to produce the distribution $\Delta_N(q)$, and then to throw away $q$'s that are not prime powers.

**4. The complete algorithm.** This section presents an explicit program for generating factored random integers, using the language of real numbers. It assumes the uniform (0,1) distribution as a source of randomness; § 7 will show how this can be simulated by a fair coin.

For real numbers $a$ and $b$, let $\#(a,b]$ denote the number of integers $x$ satisfying $a < x \leqq b$. For prime powers $q = p^{\alpha}$, and integers $N$, let

$$\Delta_N(q) = \frac{\log p}{\log N} \cdot \frac{\#(N/2q,\, N/q)}{N}. \tag{1}$$

Note that if $[x]$ denotes the greatest integer $\leqq x$, then $\#(a/2,a] = [(a+1)/2]$; this implies the frequently used estimate

$$(a-1)/2 \leqq \#(a/2,\, a] \leqq (a+1)/2 \tag{2}$$

and also shows that for $q \leqq N$,

$$\Delta_N(q) \leqq \frac{N/q+1}{2N}.$$

The innermost part of the program selects random prime powers; using the above notation, it is defined below.

PROCESS F: Factor generator.

(*) Select a random integer $j$ with $1 \leqq j \leqq \log_2 N$.
Let $q = 2^j + r$, where $r$, $0 \leqq r < 2^j$, is chosen at random.
Choose a random real number $\lambda$ from the $U(0, 1)$ distribution.
If $q$ is a prime power, $q \leqq N$, and $\lambda < \Delta_N(q) 2^{[\log_2 q]}$, output $q$.
If not, go back to (*).

The salient features of this process are given by the following result.

THEOREM 2. *Process* F *almost surely halts*; *the number of times* (*) *is reached has a geometric distribution whose expected value is* $O(\log N)$. *It outputs a prime power* $q = p^{\alpha}$, $2 \leqq q \leqq N$, *with odds* $\Delta_N(q)$. *The stopping time and the output value are independent.*

*Proof.* The first two steps select $q$ with odds $2^{-[\log_2 q]}$, and since $2^{[\log_2 q]} \Delta_N(q) \leqq (N+q)/2N \leqq 1$ for $q \leqq N$, Theorem 1 implies that $q$ is output with the stated probability. For the stopping time estimate, since

$$P = \sum_{q=2}^{N} 2^{-[\log_2 q]} \leqq \log_2 N,$$

it will suffice to show that

$$Q = \sum_{q \leqq N} \Delta_N(q)$$

is bounded below by an absolute constant. This follows from two consequences of the prime number theorem given by Rosser and Schoenfeld [14, p. 65]:

$$\sum_{p \leqq N} \log p = N + O(N/\log N) \tag{3}$$

and

$$\sum_{p \leqq N} \log p/p = \log N + O(1). \tag{4}$$

Using (3) and (4),

$$(5)\qquad \sum_{q\leqq N}\Delta_N(q)\geqq\sum_{p\leqq N}\Delta_N(p)\geqq\sum_{p\leqq N}\frac{\log p}{2p\log N}-\sum_{p\leqq N}\frac{\log p}{2N\log N}=\frac{1}{2}+O\left(\frac{1}{\log N}\right).$$

The independence statement is a consequence of Theorem 1. □

Just as in the heuristic sketch, the factor generator is a subroutine in the main program, presented below. This process uses a "stopping value" $N_0$, which can be any convenient number.

PROCESS R: Random factorization generator.

If $N\leqq N_0$, pick a random $x$, $N/2<x\leqq N$, output the factors of $x$, and stop.

(†) Select a prime power $q=p^\alpha$, $2\leqq q\leqq N$, using Process F.
Let $N'=[N/q]$.
Call Process R recursively to choose a factored random $y$ with $N'/2<y\leqq N'$.
Let $x=y\cdot q$.

(§) Choose a real number $\lambda$ from the $U(0,1)$ distribution.
If $\lambda<\log(N/2)/\log x$, output the factors of $x$ and stop.
If not, return to (†).

The main result of this paper is the following theorem.

THEOREM 3. *Process* R *generates uniformly distributed random integers* $x$, $N/2<x\leqq N$, *in factored form.*

*Proof.* If $N\leqq N_0$, there is nothing to prove. Otherwise, note that for integers $y$, $[x]/2<y\leqq[x]$ if and only if $x/2<y\leqq x$, and so the recursive step chooses an integer $y$ uniformly with $N/2q<y\leqq N/q$. Therefore, by Theorem 2, $x$ is chosen at step (§) with probability proportional to

$$\sum_{q=p^\alpha|x}\frac{\Delta_N(q)}{\#(N/2q,N/q]}=\sum_{q=p^\alpha|x}\frac{\log p}{\log N}\cdot\frac{\#(N/2q,N/q]}{N}\cdot\frac{1}{\#(N/2q,N/q]}=\frac{\log x}{N\log N}.$$

By Theorem 1, the last part of the algorithm ensures that $x$ is output with probability $1/\#(N/2,N]$. □

**5. The distribution of factor lengths.** It was stated earlier that Process F produces a factor $q$ whose length is roughly uniformly distributed. This can be refined into the following precise statement: as $N\to\infty$, $\log q/\log N$ converges in distribution to a uniform (0,1) random variable.[2] This implies the following: if we define

$$F_N(x)=\Pr[q\leqq x \text{ in Process F}],$$

then $F_N(x)$ is close to $\log x/\log N$. Similarly, the expected values $E[\log(N/q)]$ and $E[\log^2(N/q)]$ are close to $\frac{1}{2}\log N$ and $\frac{1}{3}(\log N)^2$, respectively. The next three lemmas give upper bounds corresponding to these approximations, which are used in the next section.

LEMMA 1. *If* $N>30$, *and* $2\leqq x\leqq N$, *then*

$$F_N(x)\leqq\frac{\log x+2}{\log N-2}.$$

*Proof.* For $N>30$,

$$\sum_{p^\alpha\leqq N}\log p\leqq 1.04N$$

[2] I will not prove this as it is not needed later.

and

$$\log N - \gamma - \beta - \frac{1}{2\log N} \leqq \sum_{p \leqq N} \frac{\log p}{p} \leqq \log N,$$

where

$$\gamma = 0.577215\ldots$$

is Euler's constant and

$$\beta = \sum_{\alpha \geqq 2} \log p / p^{\alpha} = 0.755366\ldots$$

(these are (3.21), (3.24) and (3.35) from Rosser and Schoenfeld [14]). Using these inequalities, plus (1) and (2),

$$2\log N \sum_{q \leqq N} \Delta_N(q) \geqq \log N - \gamma - \beta - \frac{1}{2\log N} + \sum_{\substack{p^{\alpha} \leqq N \\ \alpha \geqq 2}} \frac{\log p}{p^{\alpha}} - 1.04 \; > \; \log N - 2.$$

Similarly

$$2\log N \sum_{q \leqq x} \Delta_N(q) \leqq \sum_{p \leqq x} \frac{\log p}{p} + \beta + \sum_{q \leqq x} \frac{\log p}{N} \leqq \log x + 2.$$

Now apply these to the formula $F_N(x) = \sum_{q \leqq x} \Delta_N(q) / \sum_{q \leqq N} \Delta_N(q)$. □

LEMMA 2. *For* $N > 30$,

$$E[\log (N/q)] \leqq \frac{\log N}{2} \cdot \frac{\log N + 4}{\log N - 2}.$$

*Proof.* The expectation can be expressed as a Stieltjes integral:

$$\int_{2^-}^{N} \log (N/x) dF_N(x).$$

Using integration by parts and Lemma 1,

$$\int_{2^-}^{N} \log (N/x) dF_N(x) = \int_{2}^{N} \frac{F_N(x)}{x} dx \leqq \int_{1}^{N} \frac{\log x + 2}{\log N - 2} \cdot \frac{dx}{x},$$

now computing the integral gives the result. □

LEMMA 3. *For* $N > 30$,

$$E \log^2 (N/q) \leqq \frac{(\log N)^2}{3} \cdot \frac{\log N + 6}{\log N - 2}.$$

*Proof.* As in the last proof, the expectation is

$$\int_{2^-}^{N} \log^2 (N/x) dF_N(x) \leqq \frac{2}{\log N - 2} \int_{1}^{N} (\log N - \log x)(2 + \log x) \frac{dx}{x}. \qquad \square$$

**6. The expected number of prime-power tests.** This section proves that the number of prime-power tests done by Process R on input $N$ has expected value and standard deviation that are both $O(\log N)$.

For every $N$, define random variables as follows. $T_N$ is the number of prime-power tests done by Process R on input $N$, and $U_N$, $V_N$, and $W_N$ count the prime-power tests done during the first call to Process F, the recursive call, and after the first return to (†), respectively.

THEOREM 4. *If* $N_0 > 10^6$, $E(T_N) = O(\log N)$.

*Proof.* Let $N > N_0$, for otherwise the theorem is true immediately. By Theorem 2, we can choose $C > 0$ so that $U_N \leqq C \log N$; we now prove by induction on $N$ that $T_N \leqq 6C \log N$.

Since $T_N = U_N + V_N + W_N$, $E(T_N) = E(U_N) + E(V_N) + E(W_N)$. By the definition of $C$ and the formula $E(X) = E_Y(E(X|Y))$ applied to $V_N$, this gives

$$E(T_N) \leqq C \log N + E_q(E(T_{[N/q]})) + \frac{\log 2}{\log N} E(T_N),$$

since the probability of renewal is at most $1 - \log(N/2)/\log N$. By induction and Lemma 2,

$$E(T_N) \leqq C \log N + 6CE(\log N/q) + \frac{\log 2}{\log N} E(T_N)$$

$$\leqq C \log N + 6C \frac{\log N}{2} \cdot \frac{\log N + 4}{\log N - 2} + \frac{\log 2}{\log N} E(T_N).$$

This implies

$$E(T_N) \leqq \frac{1}{1 - \log 2/\log N} \left(1 + 3\frac{\log N + 4}{\log N - 2}\right) C \log N,$$

and for $N > 10^6$ the coefficient of $C \log N$ is less than 6. □

The corresponding estimate for the variance is given below.

THEOREM 5. *If* $N_0 > 10^6$, $\sigma^2(T_N) = O(\log N)^2$.

*Proof.* Let $\bar{\mathrm{R}}$ denote the process obtained by replacing the top level stopping condition $\lambda < \log(N/2) / \log x$ by $\lambda < 1 - \log 2/\log N$; the recursive call uses the unaltered Process R. We can consider both processes to be defined on the same sample space; then (extending the notation in an obvious fashion)

$$\bar{T}_N = U_N + V_N + \bar{W}_N.$$

Since $U_N$, $V_N$, and $\bar{W}_N$ are independent,

$$\sigma^2(\bar{T}_N) = \sigma^2(U_N) + \sigma^2(V_N) + \sigma^2(\bar{W}_N).$$

Using the formulas $\sigma^2(X) = E(X^2) - E(X)^2$ and $E(X) = E_Y(E(X|Y))$,

$$E(\bar{T}_N^2) \leqq \sigma^2(U_N) + E(\bar{T}_N)^2 + E_q(E(T^2_{[N/q]})) + \frac{\log 2}{\log N} E(\bar{T}_N^2).$$

The proof of Theorem 4 actually shows that $E(\bar{T}_N) = O(\log N)$; we can therefore choose $D > 0$ so that $\sigma^2(U_N) + E(\bar{T}_N)^2 \leqq D(\log N)^2$. Furthermore, for any $N$, $T_N \leqq \bar{T}_N$, so

$$E(\bar{T}_N^2) \leqq D(\log N)^2 + E_q(E(\bar{T}^2_{[N/q]})) + \frac{\log 2}{\log N} E(\bar{T}_N^2).$$

An argument similar to the proof of Theorem 4, using Lemma 3, will show that $E(\bar{T}_N^2) \leqq 4D(\log N)^2$. Since

$$\sigma^2(T_N) \leqq E(T_N^2) \leqq E(\bar{T}_N^2),$$

the result follows. □

**7. A single-precision time bound.** The next two sections analyze the average number of single-precision operations needed to generate a random factorization. Any serious discussion of this must answer two questions. First, the algorithm uses real numbers, which are not finite objects; how can these be simulated? Second, one might wish to use randomized prime testing; what happens when the prime tester can make mistakes?

This section addresses the real-number issue, assuming perfect prime testing; probabilistic prime testing will be treated in the next section. In what follows, a "step" means one of the basic arithmetic operations (including comparison) on single-bit numbers, or a coin flip. All questions of addressing and space requirements will be ignored.

The following result will be used repeatedly.

LEMMA 4. *Let $T_1, T_2, \ldots$ be a sequence of random variables, and let $n$ be a positive integer-valued random variable, such that $T_i = 0$ for every $i > n$. If $E(T_i | n \geqq i) \leqq A$ and $E(n) \leqq B$, then $E(\sum_{i=1}^{n} T_i) \leqq AB$.*

*Proof.*

$$E\left(\sum_{i=1}^{n} T_i\right) = \sum_{i=1}^{\infty} E(T_i) = \sum_{i=1}^{\infty} E(T_i | n \geqq i)\Pr[n \geqq i] \leqq A \cdot \sum_{i=1}^{\infty} \Pr[n \geqq i] \leqq AB. \qquad \square$$

At several points in our algorithm we need to flip a coin with success probability $\theta$, where $\theta$ is a fixed real number between 0 and 1. This means we compare $\theta$ with a randomly generated real value $\lambda$, thus:

$$\theta = .0101010101\ldots,$$
$$\lambda = .0110010111\ldots$$

A finitary procedure with the same effect simply does the comparison bit-by-bit from the left, only generating bits as they are needed. This is clearly fast; since the bits of $\lambda$ are specified by independent coin flips, we expect to use only two of them before reaching a decision. However, it may not be convenient to generate the true bits of $\theta$ one at a time; to avoid this difficulty, we base our procedure on approximations that might be produced by some scheme like interval arithmetic.

We need the following definition. Let $\theta$, $0 \leqq \theta \leqq 1$, be a real number. A *$k$-bit approximation to $\theta$* is an integer multiple $\theta_k$ of $2^{-k}$ with $|\theta_k - \theta| \leqq 2^{-k}$ and $0 < \theta_k < 1$.

The lemma below eliminates real numbers from our algorithm; it states, in effect, that if $\theta$ is approximable by *any* polynomial-time procedure, then a biased coin flip with success probability $\theta$ takes constant time on the average.

LEMMA 5. *Let $0 \leqq \theta \leqq 1$, and assume that a $k$-bit approximation to $\theta$ can be computed in $f(k)$ steps, where $f$ is a polynomial of degree $m$ with nonnegative coefficients. Then the expected time to decide if a uniform (0,1) random variable is less than $\theta$ is at most $C_m f(1)$, where $C_m$ depends only on $m$.*

*Proof.* Let $\lambda$ be the uniform (0,1) value; we can assume it is irrational, since the set of rational numbers has measure 0. Consider the following procedure: for $k = 1, 2, 3, \ldots$, compute a $k$-bit approximation $\theta_k$ to $\theta$ and compare this to $\lambda_k$, the number formed from the first $k$ bits of $\lambda$; if $\theta_k + 2^{-k} \leqq \lambda_k$ or $\lambda_k < \theta_k - 2^{-k}$, terminate the process. The probability that no decision is reached after $k$ steps is at most $2^{1-k}$, so the expected

total time is at most a constant times

$$\sum_{k=1}^{\infty} f(k)2^{1-k} = 2\sum_{k=1}^{\infty}\sum_{j=0}^{m} a_j k^j 2^{-k} = 2\sum_{j=0}^{m} a_j \sum_{k=1}^{\infty} k^j 2^{-k},$$

where $a_0, \ldots, a_m$ are the coefficients of $f$. Polyá and Szegö [12, p. 9] give the formula (valid for $|z|<1$)

$$\sum_{k=1}^{\infty} k^j z^k = \frac{g_j(z)}{(1-z)^{j+1}},$$

where $g_j$ is a polynomial with nonnegative coefficients, satisfying $g_j(1) = j!$. The result follows by taking $z = 1/2$ and $C_m = 2^{m+2}m!$. □

LEMMA 6. *Let $p,q,N$ be integers with $2 \leqq p \leqq q \leqq N$. Then a $k$-bit approximation to*

$$\theta = \frac{\log p}{\log N} \cdot \frac{\#(N/2q, N/q]2^{[\log_2 q]}}{N}$$

*can be computed in $O(k^3 + \log\log N)$ steps.*

*Proof.* Let $p = 2^{\alpha}\cdot\varepsilon$ and $N = 2^{\beta}\cdot\eta$, where $\alpha$ and $\beta$ are integers and $1 \leqq \varepsilon, \eta < 2$. Then

$$\theta = \frac{\alpha \log 2 + \log \varepsilon}{\beta \log 2 + \log \eta} \cdot \frac{[(N+q)/2q]2^{[\log_2 q]}}{N}. \tag{6}$$

We approximate $\theta$ by using floating-point numbers to perform the computation implicit in the above expression; $k + O(1)$ bits of precision suffice to get an absolute error less than $2^{-k}$, by the following argument. First, since $0 \leqq \theta \leqq 1$, it suffices to make the relative error in the result less than $2^{-k}$. Brent [3, Thm. 6.1] shows that on the interval $1 \leqq x \leqq 2$, one can compute $\log x$ with relative error $2^{-n}$ in $O(n^3)$ steps. If we take $n = k + O(1)$, we will have enough guard bits to nullify the effect of any remaining error, since there are only a finite number of further operations. All the numbers involved have exponents that are less than $\log_2 N$, so the bound follows. □

LEMMA 7. *Let $x$ and $N$ be positive integers with $N/2 < x \leqq N$. Then a $k$-bit approximation to $\log(N/2)/\log x$ can be computed in $O(k^3 + \log\log N)$ steps.*

*Proof.* Approximate the logarithms as indicated in the proof of Lemma 6. □

LEMMA 8. *Let $q > 1$ be an integer. Then solving $p^{\alpha} = q$ for an integer $p$ and the largest possible integer $\alpha$ can be done in $O(\log q)^3(\log\log q)^2$ steps.*

*Proof.* Let $d = [\log_2 q]$; then necessarily $\alpha \leqq d$. For each such value of $\alpha$, we solve $X^{\alpha} = q$ by bisection, using 0 and $2^{[d/\alpha]+1}$ as starting values. This will find a solution or prove that none exists after $O(d/\alpha)$ evaluations of $f(X) = X^{\alpha}$. The total time is therefore at most a constant times

$$(\log q)^3 \sum 2 \leqq \alpha \leqq d \frac{\log \alpha}{\alpha} = O(\log q)^3(\log\log q)^2. \qquad \square$$

The next two results assume the Extended Riemann Hypothesis (ERH), a famous conjecture of analytic number theory. The details of this hypothesis (for which see Davenport's book [4, p. 124]) are not important here; what matters is the following consequence, first proved by Miller [10].

LEMMA 9 (ERH). *To test if an integer $p$ is prime requires $O(\log p)^5$ operations.*

*Proof.* We write $\nu_2(x)$ for the largest $e$ such that $2^e | x$, and $\mathbb{Z}_p^*$ for the multiplicative subgroup of integers modulo $p$. Then Miller's primality criterion states that $p$ is prime

if and only if every $a \in Zp^*$ satisfies

(7) $\quad a^{p-1} \equiv 1$ and for all $k < \nu_2(p-1)$, $a^{(p-1)/2^k} \equiv 1$ implies $a^{(p-1)/2^{k+1}} \equiv \pm 1$

(all congruences are interpreted modulo $p$). If $p$ is composite, there is a proper subgroup $G$ of $Zp^*$ such that (7) is violated for all $a \notin G$ [9, proof of Thm. 6]. The ERH implies that there is some number outside $G$ that is less than $2(\log p)^2$ [2, Thm. C]. Therefore $p$ is prime if and only if condition (7) holds for all positive $a \leqq 2(\log p)^2$; the result follows. □

We now make the following changes to our algorithm: prime-power testing is done as indicated in the proofs of lemmas 8 and 9, and the real number calculations are done as indicated in Lemmas 5, 6, and 7. With these modifications we have our single-precision result.

THEOREM 6 (ERH). *The expected number of single-precision operations (arithmetic, comparison, coin flips) needed by Process* R *on input N is* $O(\log N)^6$.

*Proof.* Theorem 4 and inspection of the algorithm imply that none of its steps can be executed more than $O(\log N)$ times on the average. By Lemma 4, it suffices to show that no single step of the algorithm has expected time greater than the $O(\log N)^5$ steps sufficient to test a number less than $N$ for primality. By Lemmas 5, 6, and 7, this is true for the real number comparisons. Everything else is easily estimated. □

The real point to this extravagant bound is that it is a polynomial in $\log N$. By using the prime test of Adleman, Pomerance, and Rumely [1], one can also prove an unconditional almost-polynomial time bound of $O(\log N)^{O(\log\log\log N)}$. However, much better estimates can be obtained by using probabilistic prime testing, as described in the next section.

**8. The use of probabilistic primality tests.** This section proves theorems analogous to the preceding results, assuming that a randomized prime test is used. First, a definition: call a factorization $x = p_1^{e_1} \ldots p_k^{e_k}$ *complete* if all the $p_i$'s are prime; if it uses a probabilistic prime test, Process R may output an incompletely factored number. The results in this case can be simply summarized: the distribution of completely factored numbers is still uniform, and incompletely factored numbers can be made exponentially unlikely at very little cost.

The following result is analogous to Lemma 9.

LEMMA 10. *To test if p is prime with bounded by* $4^{-n}$ *(error only being possible when p is composite) requires* $n \cdot O(\log p)^3$ *operations.*

*Proof.* Rabin [13, Thm. 1] shows that condition (7) is violated for a random $a$ with probability at most $1/4$, so choose $n$ independent random values of $a$. □

The prime tests referred to above have a very nice property; the decision is never wrong unless the input is composite. This is the key observation in the next proof.

THEOREM 7. *If the prime test used in Process* F *produces correct answers when the input is prime, then the distribution of completely factored outputs is uniform.*

*Proof.* Use induction on $N$; if $N < N_0$, this is clear. Otherwise, the prime powers produced by Process F have the same relative distribution as before, since the prime tester never makes a mistake on prime input. Since every subfactorization of a complete factorization is complete, the calculation that proves Theorem 3 is still valid. □

The order-of-magnitude bound for the average number of prime power tests still holds, if the prime test used is sufficiently accurate. Since the average number of tests increases with $N$, a constant number of the tests (7) per prime will not suffice. Instead, we choose a bound $\varepsilon(N)$ in advance, and make every prime test used by the algorithm have a chance of error at most $\varepsilon(N)$.

THEOREM 8. *Assume that the prime test used in Process* F *is correct on prime input, and has error probability at most* $\varepsilon(N)$ *on composite input. If* $\varepsilon(N) < N^{-2}$, *then the number of prime-power tests done by Process* R *on input* $N$ *has mean and standard deviation that are* $O(\log N)$, *and the probability that an incompletely factored number is produced is* $O(\varepsilon(N)\log N) = O(\log N/N^2)$.

*Proof.* For the time bound, it is only necessary to show that Lemma 1 still holds, say for $N > 10^6$. The proof of Lemma 1 amounted to a lower bound on the relative probability that $q \leqq N$ and an upper bound on the relative probability that $q \leqq x$. The lower bound still holds, and the new upper bound is at most

$$\sum_{q \leqq x} \Delta_q + \sum_{y \leqq x} \varepsilon \frac{\log y}{\log N} \cdot \frac{\#(N/2y, N/y]}{N}.$$

The second term is at most

$$\frac{1}{2\log N} \cdot \frac{1}{N^2} \sum_{y \leqq x} \left( \frac{\log y}{y} + \frac{\log y}{N} \right) \leqq \frac{1}{2\log N} \cdot \left( \frac{\log N}{N} \right)^2,$$

and this will not cause the bound to be exceeded. For the estimate relating to incorrect output, apply Lemma 4 to the random variables $X_i$ that are 1 if the $i$th prime test is incorrect, and 0 otherwise, and use the inequality $\Pr[X \geqq 1] \leqq E(X)$. □

By Lemma 10, error probability less than $N^{-2}$ can be obtained with about $\log_2 N$ tests, each using $O(\log N)^3$ steps. This will give a polynomial time bound analogous to Theorem 6.

**9. Experiments.** This section has two purposes: to show how the algorithm actually behaves in practice, and to discuss what modifications are necessary to implement it efficiently.

Call an arrival at (*) in Process F a *probe*; Theorem 4 implies that Process R requires $O(\log N)$ probes on the average. The constant implied by the "$O$" symbol can be estimated by the following heuristic argument. Typically the algorithm will produce factors whose lengths are 1/2, 1/4, 1/8, ... the length of $N$. Presumably, then, the average number of probes is close to

$$2\log_2 N\,(1 + 1/2 + 1/4 + \ldots) = 4\log_2 N,$$

since by (5), we expect Process F to use about $2\log_2 N$ probes on input $N$.

This value of $4\log_2 N$ is also the best bound provable as the stopping value $N_0 \to \infty$, and the first set of experiments were designed to see if this estimate is at all realistic when $N_0$ is small.

To do this, I coded the algorithm verbatim in $C$ (the "real" numbers have about 15 decimal digits of precision) with $N_0 = 4$. Table 1 gives statistics on the number of probes required to generate 100 random numbers for various values of $N$, together with the presumed mean value $4\log_2 N$.

It will be seen from this table that the standard deviation tends to be a bit smaller than the mean; however, I do not even have a heuristic argument to justify this observation.

To test its feasibility for large $N$, I also coded the algorithm with multiprecise arithmetic on a DEC VAX 11-780, a machine that takes about 5 microseconds to multiply two 32-bit integers. When dealing with multi-word numbers, efficiency is of primary importance; this concern led to the following version of Process F.

1) After building the random value $q$, the program first checks that $q \leqq N$, and then computes $\Delta_N(q)2^{[\log_2 q]}$ in single precision with the formula (6), as if $q$

TABLE 1
*Statistics on the number of probes.*

| $N$ | $4\log_2 N$ | Average | Maximum | Standard deviation |
|---|---|---|---|---|
| $10^2$ | 26.56 | 20.77 | 60 | 13.95 |
| $10^3$ | 39.86 | 35.59 | 218 | 34.31 |
| $10^4$ | 53.15 | 56.30 | 367 | 47.49 |
| $10^5$ | 66.44 | 71.30 | 329 | 62.24 |
| $10^6$ | 79.73 | 72.49 | 472 | 59.58 |
| $10^7$ | 93.01 | 74.61 | 346 | 63.43 |
| $10^8$ | 106.30 | 111.78 | 491 | 85.30 |
| $10^9$ | 119.59 | 120.82 | 453 | 96.78 |

were prime. Only if this exceeds the random value $\lambda$ does it subject $q$ to a prime-power test.

2) The first part of this test sees if for any small prime $p$,

$$p|q \quad \text{and} \quad p^2 \nmid q;$$

if this is true, $q$ cannot be a prime power. This sieve procedure eliminates most $q$ from further consideration, for the probability that a random number $q$ survives this test for all $p \leqq B$ is about

$$\prod_{p \leqq B} \left( \frac{p-1}{p} + \frac{1}{p^2} \right).$$

If we let

$$\alpha = \prod_p \left( 1 + \frac{1}{p(p-1)} \right) = 1.943596\ldots$$

then the survival probability is, by Mertens's theorem [6, p. 22], close to

$$\alpha \cdot \frac{e^{-\gamma}}{\log B}.$$

The program used $B = 1000$, which screens out approximately 84% of the $q$'s.

3) If $q$ passes through the sieve, it is subjected to a "pseudo" test designed to cheaply eliminate numbers that are not prime powers. This checks that $2^{q-1} \not\equiv 1$ and $\gcd(2^{q-1}-1, q) = 1$; if so, $q$ can be thrown away. The average cost of this is one modular exponentiation and one gcd computation.

4) Any number $q$ that has survived so far is tested for primality, using (7) with $a = 2, 3, 5, \ldots, 29$ (the first ten primes). There is a slight advantage to small values of $a$, since about one-third of the multiplications done by the modular exponentiation algorithm will involve a single-word quantity.

5) Only if $q$ is not declared "prime" by the above procedure does the program try to see if it is a perfect power.

(Various orderings of steps 1–5 were tried, and the one above seems to be the best.)

For the multiprecise implementation, a more realistic measure of the work done is the number of times $q$ reaches the sieve. Statistics on these values are given in Table 2, from runs that generated 50 numbers each; the last column gives the average CPU time required per random factorization.

TABLE 2
*Statistics on the number of sieve tests, and the running time in seconds.*

| $N$ | Average | Maximum | Standard deviation | Average time |
|---|---|---|---|---|
| $10^{15}$ | 29.66 | 67 | 17.70 | 1.51 |
| $10^{30}$ | 65.42 | 225 | 46.36 | 4.66 |
| $10^{60}$ | 158.92 | 387 | 95.04 | 20.52 |
| $10^{120}$ | 250.34 | 805 | 174.54 | 100.08 |

It is worth noting that for these values of $N$ the average running time is only slightly worse than $O(\log N)^2$.

Table 3 presents the mean values of four quantities related to the output values $x$: the number of prime factors of $x$, and the number of decimal digits in the largest three factors of $x$ (from the same experiments). It will be seen that the average number of prime factors grows very slowly with $N$; the observations are close to the mean values of $\log\log N + 1.03465\ldots$ predicted by prime number theory [7, p. 346]. Finally, the average lengths of the largest three factors are roughly proportional to the length of $N$; again, this is predicted by theory [7, p. 343], with constants of proportionality close to 0.62, 0.21, and 0.088, respectively.

TABLE 3
*Statistics on the prime factors.*

| $N$ | Average number | Digits in largest | 2nd largest | 3rd largest |
|---|---|---|---|---|
| $10^{15}$ | 4.48 | 10.06 | 3.76 | 1.50 |
| $10^{30}$ | 5.32 | 19.84 | 6.80 | 2.80 |
| $10^{60}$ | 6.18 | 37.36 | 13.48 | 5.94 |
| $10^{120}$ | 6.70 | 78.54 | 26.56 | 8.46 |

**Acknowledgments.** I would like to thank the following people for offering encouragement and ideas: Silvio Micali, Manuel Blum, Martin Hellman, Michael Luby, and James Demmel. I also heartily thank the seminonymous referees, whose comments improved the paper immensely.

## REFERENCES

[1] L. M. ADLEMAN, C. POMERANCE AND R. S. RUMELY, *On distinguishing prime numbers from composite numbers*, Ann. of Math., 117 (1983), pp. 173-206.
[2] E. BACH, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, MIT Press, Cambridge, 1985 (U.C. Berkeley Ph.D. dissertation, 1984).
[3] R. P. BRENT, *Fast multiple-precision evaluation of elementary functions*, J. Assoc. Comput. Mach., 23 (1976), pp. 242-251.
[4] H. DAVENPORT, *Multiplicative Number Theory*, Springer, New York, 1980.
[5] W. FELLER, *An Introduction to Probability Theory and Its Applications (Volume I)*, John Wiley, New York, 1968.
[6] A. E. INGHAM, *The Distribution of Prime Numbers*, Cambridge University Press, Cambridge, 1932.
[7] D. KNUTH AND L. TRABB PARDO, *Analysis of a simple factorization algorithm*, Theoret. Comput. Sci., 3 (1976), pp. 321-348.
[8] H. W. LENSTRA, JR., *Elliptic Curve Factorization*, Ann. of Math., 126 (1987), pp. 649-673.
[9] M. MIGNOTTE, *Tests de primalité*, Theoret. Comput. Sci., 12 (1980), pp. 109-117.

[10] G. L. MILLER, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci., 13 (1976), pp. 300–317.
[11] J. VON NEUMANN, *Various techniques used in connection with random digits*, J. Res. Nat. Bur. Standards (Applied Mathematics Series), 3 (1951), pp. 36–38.
[12] G. POLYÁ AND G. SZEGÖ, *Problems and Theorems in Analysis I*, Springer, New York, 1972.
[13] M. O. RABIN, *Probabilistic algorithm for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
[14] J. B. ROSSER AND L. SCHOENFELD, *Approximate formulas for some functions of prime numbers*, Ill. J. Math., 6 (1962), pp. 64–94.
[15] B. W. SCHMEISER, *Random variate generation: a survey*, in Simulation with Discrete Models: A State-of-the-Art View, T. I. Oren, C. M. Shub and P. F. Roth, eds., IEEE Press, New York, 1981.
[16] C. P. SCHNORR AND H. W. LENSTRA, JR., *A Monte Carlo factoring algorithm with linear storage*, Math. Comp., 43 (1984), pp. 289–311.
[17] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput., 6 (1977), pp. 84–85.