SOME INTEGER FACTORIZATION ALGORITHMS USING ELLIPTIC CURVES

Richard P. Brent*

CMA-R32-85

September 1985

* Permanent Address:
Computer Sciences Laboratory
Research School of Physical Sciences
Australian National University
GPO Box 4
Canberra, ACT 2601
Australia

AMS MOS 68C25, 10H15, 14K07, 65C05, 68E99

Centre for Mathematical Analysis
Hanna Neumann Building
Australian National University
GPO Box 4
Canberra, ACT 2601
Australia

ABSTRACT

Lenstra's integer factorization algorithm is asymptotically one of the fastest known algorithms, and is also ideally suited for parallel computation. We suggest and analyse two different ways in which Lenstra's algorithm can be speeded up by the addition of a second phase. The first way is related to the Pollard "p-1" algorithm, while the second way is related to the Brent-Pollard "rho" algorithm. Under some plausible assumptions, the speedups over Lenstra's single-phase algorithm are of order lnln(p) and ln(p) respectively, where p is the factor which is found. In practice, for $p \cong 10^{20}$, the speedups are about 3 and 4.5 respectively. The results of some numerical experiments confirm our predictions.

1. Introduction

Recently H. W. Lenstra Jr. proposed a new integer factorization algorithm, which we shall call "Lenstra's algorithm" or the "one-phase elliptic curve algorithm" [2,15]. Under some plausible assumptions Lenstra's algorithm finds a prime factor p of a large composite integer N in expected time

$$T_1(p) = \exp((2\ln(p)\ln\ln(p))^{1/2}(1 + o(1)), \qquad (1.1)$$

where "o(1)" means a term which tends to zero as $p \to \infty$. Previously algorithms with running time $\exp((\ln(N)\ \ln\ln(N))^{1/2}(1 + o(1))$ were known. However, since $p^2 \leq N$, Lenstra's algorithm is comparable in the worst case and often much better, since it often happens that $2\ln(p) \ll \ln(N)$.

The Brent-Pollard "rho" algorithm [5,19] is similar to Lenstra's algorithm in that its expected running time depends on p, in fact it is of order $p^{1/2}$. Asymptotically $T_1(p) \ll p^{1/2}$, but because of the overheads associated with Lenstra's algorithm we expect the "rho" algorithm to be faster if p is sufficiently small. The results of Section 9 suggest how large p has to be before Lenstra's algorithm is faster.

After some preliminaries in Sections 2 to 4, we describe Lenstra's algorithm in Section 5, and outline the derivation of (1.1). Then, in Sections 6 and 7, we describe two different ways in which Lenstra's algorithm can be speeded up by the addition of a "second phase" after Lenstra's "first phase". In Section 6 we describe the "p-1 two-phase algorithm", so called because of its analogy with Pollard's two-phase "p-1" algorithm [18]. In Section 7 we describe the "birthday paradox two-phase algorithm", whose name comes from the fact that it is based on the same idea as the well-known "paradox" concerning the probability that two people at a party have the same birthday [20]. The two-phase algorithms have expected running times $O(T_1(p)/\ln\ln(p))$ and $O(T_1(p)/\ln(p))$ respectively. In practice, for p around

10^{20}, the "p-1 two-phase algorithm" is about 3 times faster than Lenstra's (single-phase) algorithm, and the "birthday parodox algorithm" is about 4 times faster. The performance of the various algorithms is compared in Section 9.

Our conclusion is that, although (1.1) is not a polynomial in ln(p), it is a sufficiently slowly growing function that factors $p \cong 10^{20}$ are accessible at the expense of a few hours of computer time, and we can forsee that p around 10^{50} may be accessible in a few years time. This has interesting implications for the security of the RSA public-key cryptosystem [21], which depends on the assumed difficulty of factoring large integers.

2. Our unit of work

The factorization algorithms which we consider use arithmetic operations modulo N, where N is the number to be factorized. We are interested in the case that N is large (typically 50 to 200 decimal digits) so multiple-precision operations are involved. As our basic unit of work (or time) we take one multiplication modulo N (often just called "a multiplication" below). More precisely, given integers a, b in [0, N), our unit of work is the cost of computing a*b mod N. Because N is assumed to be large, we can simplify the analysis by ignoring the cost of additions mod N or of multiplications/divisions by small (i.e. single-precision) integers, so long as the total number of such operations is not much greater than the number of multiplications mod N. See [11, 17] for implementation hints.

In some of the algorithms considered below it is necessary to compute inverses modulo N, i.e. given an integer a in (0, N), compute u in (0, N) such that a*u = 1 (mod N). We write $u = a^{-1} \pmod N$. u can be computed by the extended GCD algorithm [11], which finds integers u and v such that au + Nv = g, where g is the GCD of a and N. We can always assume that g = 1, for otherwise g is a nontrivial factor of N, and the factorization algorithm can terminate !

Suppose that the computation of $a^{-1} \pmod N$ by the extended GCD algorithm takes the same time as K multiplications (mod N). Our first implementation gave $K \simeq 30$, but by using Lehmer's method [14] this was reduced to $6 \leq K \leq 10$ (the precise value depending on the size of N). It turns out that most computations of $a^{-1} \pmod N$ can be avoided at the expense of about 8 multiplications (mod N), so we shall assume that K = 8.

Some of the algorithms require the computation of large powers (mod N), i.e. given a in [0, N) and $b \gg 1$, we have to compute $a^b \pmod N$. We shall assume that this is done by the "binary" algorithm [11] which requires between $\log_2 b$ and $2.\log_2 b$ multiplications (mod N) - on average say $(3/2)\log_2 b$ multiplications (of which about $\log_2 b$ are squarings). The constant 3/2 could be reduced slightly by use of the "power tree" or other sophisticated powering algorithms [11].

3. Prime factors of random integers

In order to predict the expected running time of Lenstra's algorithm and our extensions of it, we need some results on the distribution of prime factors of random integers. Consider a random integer close to M, with prime factors $n_1 \geq n_2 \geq \ldots$ For $\alpha \geq 1$, define

$$\rho(\alpha) = \lim_{M \to \infty} \text{Prob}\ (n_1 < M^{1/\alpha}),$$

and for $\alpha - 1 \geq \beta \geq 1$, define

$$\mu(\alpha, \beta) = \lim_{M \to \infty} \text{Prob}\ (n_2 < M^{1/\alpha} \text{ and } n_1 < M^{\beta/\alpha}).$$

(For a precise definition of "a random integer close to M", see [12]. It is actually sufficient to consider integers uniformly distributed in [1, M].)

Several authors have considered the function $\rho(\alpha)$, see for example [6,7,11,12,16]. It satisfies a differential-difference equation

$$\alpha\rho'(\alpha) + \rho(\alpha - 1) = 0$$

and may be computed by numerical integration from

$$\rho(\alpha) = \begin{cases} 1 & \text{if } 1 \leq \alpha \leq 2 \\ \frac{1}{\alpha}\int_{\alpha-1}^{\alpha}\rho(t)dt & \text{if } \alpha > 2. \end{cases}$$

We shall need the asymptotic results

$$\ln \rho(\alpha) = -\alpha(\ln\alpha + \ln\ln\alpha - 1) + o(\alpha) \tag{3.1}$$

and

$$\rho(\alpha - 1)/\rho(\alpha) = \alpha(\ln\alpha + O(\ln\ln\alpha)) \tag{3.2}$$

as $\alpha \to \infty$.

The function $\mu(\alpha, \beta)$ is not so well known, but is crucial for the analysis of the two-phase algorithms. Knuth and Trabb Pardo [12] consider $\mu(\alpha, 2)$ and by following their argument with trivial modifications we find that

$$\mu(\alpha, \beta) = \rho(\alpha) + \int_{\alpha-\beta}^{\alpha-1} \frac{\rho(t)dt}{\alpha - t} . \tag{3.3}$$

When comparing the two-phase and one-phase algorithms the ratio $\rho(\alpha)/\mu(\alpha, \beta)$ is of interest, and we shall need the bound

$$\rho(\alpha)/\mu(\alpha, \beta) = O(\ln\alpha(\alpha\ln\alpha)^{-\beta}) \tag{3.4}$$

as $\alpha \to \infty$, for fixed $\beta > 1$.

4. The group of an elliptic curve (mod p)

In this section we consider operations mod p rather than mod N, and assume that p is a prime, $p \geq 5$. When applying the results of this section to factorization, p is an (unknown) prime factor of N, so we have to work mod N rather than mod p.

Let S be the set of points (x, y) lying on the "elliptic curve"

$$y^2 = x^3 + ax + b \pmod{p}, \tag{4.1}$$

where a and b are constants, $4a^3 + 27b^2 \not\equiv 0$. Let

$$G = S \cup \{I\},$$

where I is the "point at infinity" and may be thought of as $(0, \infty)$. Lenstra's algorithm is based on the fact that there is a natural way to define an Abelian group on G. Geometrically, if P_1 and $P_2 \in G$, we define $P_3 = P_1 * P_2$ by taking P_3 to be the reflection in the x-axis of the point

Q which lies on the elliptic curve (4.1) and is collinear with P_1 and P_2. Algebraically, suppose $P_i = (x_i, y_i)$ for i = 1, 2, 3. Then P_3 is defined by:

```
if P1 = I then P3 := P2
else if P2 = I then P3 := P1
else if (x1, y1) = (x2, -y2) then P3 := I
else
  begin
  if x1 = x2 then λ := (2y1)^-1 (3x1^2 + a) mod p
             else λ := (x1 - x2)^-1 (y1 - y2) mod p;
  { λ is the gradient of the line joining P1 and P2 }
  x3 := (λ^2 - x1 - x2) mod p;
  y3 := (λ(x1 - x3) - y1) mod p
  end.
```

It is well-known that (G, *) forms an Abelian group with identity element I. Moreover, by the "Riemann hypothesis for finite fields" [10], the group order $g = |G|$ satisfies the inequality

$$|g - p - 1| < 2\sqrt{p} . \qquad (4.2)$$

Lenstra's heuristic hypothesis is that, if a and b are chosen at random, then g will be essentially random in that the results of Section 3 will apply with M = p. Some results of Birch [3] make this plausible. Nevertheless, the divisibility properties of g are not quite what would be expected for a randomly chosen integer near p, e.g. the probability that g is even is asymptotically 2/3 rather than 1/2. We shall accept Lenstra's hypothesis as we have no other way to predict the performance of his algorithm. Empirical results described in Section 9 indicate that the algorithms do perform roughly as predicted.

Note that the computation of $P_1 * P_2$ requires (3 + K) units of work if $P_1 \neq P_2$, and (4 + K) units of work if $P_1 = P_2$. (Squaring is harder than multiplication !) If we represent P_i as $(x_i/z_i, y_i/z_i)$ then the algorithm

given above for the computation of $P_1 * P_2$ can be modified to avoid GCD computations; assuming that $z_1 = z_2$ (which can usually be ensured at the expense of two units of work), a squaring then requires 11 units and a nonsquaring multiplication requires 9 units of work. Possibly some clever reorganisation of the computation would reduce these numbers slightly.

The reader who is interested in learning more about the theory of elliptic curves should consult [9], [10] or [13].

5. Lenstra's algorithm

The idea of Lenstra's algorithm is to perform a sequence of pseudo-random trials, where each trial uses a randomly chosen elliptic curve and has a nonzero probability of finding a factor of N. Let m and m" be parameters whose choice will be discussed later. To perform a trial, first choose $P = (x, y)$ and a at random. This defines an elliptic curve

$$y^2 = x^3 + ax + b \qquad (\text{mod } N)$$

(In practice it is sufficient for a to be a single-precision random integer, which reduces the cost of operations in G; also, there is no need to check if GCD $(N, 4a^3 + 27b^2) \neq 1$ as this is extremely unlikely unless N has a small prime factor.) Next compute $Q = P^E$, where E is a product of primes less than m,

$$E = \prod_{p_i \text{ prime},\ p_i < m} p_i^{e_i},$$

where

$$e_i = \lfloor \ln(m")/\ln(p_i) \rfloor .$$

Actually, E is not computed. Instead, Q is computed by repeated operations of the form $P := P^k$, where $k = p_i^{e_i}$ is a prime power less than m", and the operations on P are performed in the group G defined in Section 4, with one important difference. The difference is that, because a prime factor p of N is not known, all arithmetic operations are performed modulo N rather than modulo p. Consequently, elements of G are not represented uniquely.

Suppose initially that $m'' = N$. If we are lucky, all prime factors of $g = |G|$ will be less than m, so $g|E$ and $P^E = I$ in the group G. This will be detected because an attempt to compute $t^{-1} \pmod N$ will fail because GCD (N, t) > 1. In this case the trial succeeds. (It may, rarely, find the trivial factor N if all prime factors of N are found simultaneously, but we neglect this possibility.)

Making the heuristic assumption mentioned in Section 4, and neglecting the fact that the results of Section 3 only apply in the limit as M (or p) $\to \infty$, the probability that a trial succeeds in finding the prime factor p of N is just $\rho(\alpha)$, where $\alpha = \ln(p)/\ln(m)$.

In practice we choose $m'' = m$ rather than $m'' = N$, because this significantly reduces the cost of a trial without significantly reducing the probability of success. Assuming $m'' = m$, well known results on the distribution of primes [8] give $\ln(E) \sim m$, so the work per trial is approximately $c_1 m$, where $c_1 = (\frac{11}{3} + K)\frac{3}{2\ln 2}$. Here c_1 is the product of the average work required to perform a multiplication in G times the constant $\frac{3}{2\ln 2}$ which arises from our use of the binary algorithm for computing powers (see Section 2). Since $m = p^{1/\alpha}$, the expected work to find p is

$$W_1(\alpha) \sim c_1 p^{1/\alpha}/\rho(\alpha) \quad . \tag{5.1}$$

To minimize $W_1(\alpha)$ we differentiate the right side of (5.1) and set the result to zero, obtaining $\ln(p) = -\alpha^2 \rho'(\alpha)/\rho(\alpha)$, or (from the differential equation satisfied by ρ),

$$\ln(p) = \alpha\rho(\alpha - 1)/\rho(\alpha) \quad . \tag{5.2}$$

In practice p is not known in advance, so it is difficult to choose α so that (5.2) is satisfied. This point is discussed in Section 9. For the moment assume that we know or guess an approximation to ln(p), and choose α so that (5.2) holds, at least approximately. From (3.2),

$$\ln(p) = \alpha^2(\ln\alpha + O(\ln\ln\alpha)) \tag{5.3}$$

so

$$\alpha \sim (2\ln(p)/\ln\ln(p))^{1/2} \tag{5.4}$$

and

$$\ln W_1(\alpha) \sim \frac{\rho(\alpha - 1)}{\rho(\alpha)} - \ln\rho(\alpha) \sim 2\alpha\ln\alpha \tag{5.5}$$

$$\sim (2\ln(p)\ln\ln(p))^{1/2} \tag{5.6}$$

Thus

$$T_1(p) = W_1(\alpha) = \exp((2\ln(p)\ln\ln(p))^{1/2}(1 + o(1)) \tag{5.7}$$

as stated in Section 1. It may be informative to write (5.7) as

$$T_1(p) = W_1(\alpha) = p^{2/\alpha + o(1/\alpha)}, \tag{5.8}$$

so $2/\alpha$ is roughly the exponent which is to be compared with 1 for the method of trial division or 1/2 for the Brent-Pollard "rho" method. For $10^{10} < p < 10^{30}$, α is in the interval (3.2, 5.0).

6. The "p-1" two-phase algorithm

In this section we show how to increase the probability of success of a trial of Lenstra's algorithm by the addition of a "second phase". The idea is the same as for Pollard's two-phase "p-1" algorithm [18], the only difference is that we work in the group G of an elliptic curve instead of in the multiplicative group of integers modulo p.

Let $m = p^{1/\alpha}$ be as in Section 5, and $m' = m^\beta > m$. Let g be the order of the random group G for a trial of Lenstra's algorithm, and suppose that g has prime factors $n_1 \geq n_2 \geq \ldots$ Then, making the same assumptions as in Section 5, the probability that $n_1 < m'$ and $n_2 < m$ is $\mu(\alpha, \beta)$, where μ is defined by (3.3). Suppose we perform a trial of Lenstra's algorithm, computing $Q = P^E$ as described in Section 5. With probability $\mu(\alpha, \beta) - \rho(\alpha)$ we have $m \leq n_1 < m'$ and $n_2 < m$, in which case the trial fails because $Q \neq I$, but $Q^{n_1} = I$ in G. (As in Section 5, g should really be the order of P in G rather than the order of G, but this difference is unimportant and will be neglected.)

The idea of the "p-1" two-phase algorithm is to compute Q^k for each prime k in [m, m'). If, for some such k, $Q^k = I$ in G, then the factor p of N will be detected (during a GCD computation, as for Lenstra's algorithm). We can save work by precomputing a table of Q^j for all (or most) j which occur as differences $p_{i+1} - p_i$ for prime $p_i < m'$. For example, if $m' \leq 10^6$ then it is sufficient to compute Q^j for even $j \leq 114$ (see [20]). Then $Q^{p_{i+1}} = Q^{p_i} * Q^{p_{i+1} - p_i}$ can be computed with just one multiplication in G, i.e. with cost $c_2 = 3 + K$ multiplications (mod N).

There are approximately $(m' - m)/\ln(m')$ primes in [m, m'), so the cost of one trial of the "p-1" two-phase algorithm is about $c_1 m + c_2(m'-m)/\ln(m')$, and the expected cost of finding the factor p of N is

$$W_2(\alpha, \beta) \sim (c_1 m + c_2(m'-m)/\ln(m'))/\mu(\alpha, \beta) \ . \qquad (6.1)$$

The optimal choice of α and β to minimize $W_2(\alpha, \beta)$ is more difficult than the choice of α for Lenstra's algorithm. However, we can obtain an order of magnitude estimate of the improvement due to use of the second phase by supposing that α is chosen as for Lenstra's algorithm (defining $m = p^{1/\alpha}$), and then m' is chosen so that the work for each phase is about equal, i.e. $c_1 m \cong c_2(m' - m)/\ln(m')$. Thus $m' = p^{\beta/\alpha} \cong (c_1/c_2)p^{1/\alpha}\ln(p^{\beta/\alpha})$, and from (5.3) we have

$$\beta = 1 + \frac{1}{\alpha} + o(\frac{1}{\alpha}) \ . \qquad (6.2)$$

Now, from (3.2) and (3.3), $\mu(\alpha, 1 + \frac{1}{\alpha}) \geq \rho(\alpha) + \frac{\rho(\alpha - 1)}{\alpha + 1} \sim \rho(\alpha)\ln\alpha$, so

$$\begin{aligned} W_2(\alpha, \beta)/W_1(\alpha) &= O(\rho(\alpha)/\mu(\alpha, \beta)) \\ &= O(1/\ln\alpha) = O(1/\ln\ln(p)) \end{aligned} \qquad (6.3)$$

In other words, we obtain a (theoretical) speedup of at least order lnln(p). Clearly the speedup is at most of order ln(p), for without decreasing the probability of success of a trial we could just perform Lenstra's algorithm with m replaced by m', which increases the cost of a trial by a factor $O(\ln(m')) = O(\ln(p))$. In fact, a more detailed analysis shows that our lower bound lnln(p) gives the correct order of magnitude of the speedup. In practice, the speedup is about 3 (see Section 9).

7. The "birthday paradox" two-phase algorithm

The "birthbday paradox" algorithm, like the "p-1" two-phase algorithm, is intended to find a factor p of N if $n_1 < m'$ and $n_2 < m$, where $n_1 \geq n_2 \geq \ldots$ are the prime factors of $g = |G|$ and G is a randomly chosen group, as in Lenstra's algorithm. We assume that $m < m' = m^{\beta} \leq m^2$, so $1 < \beta \leq 2$.

As in Section 6, let $Q \neq I$ be the result of an unsuccessful trial of Lenstra's algorithm, and suppose that $Q^{n_1} = I$ in G for some (unknown) $n_1 < m'$. Let $H = \langle Q \rangle$ be the cyclic group generated by Q. A nice idea is to take some pseudo-random function $f: Q \to Q$, define $Q_1 = Q$ and $Q_{i+1} = f(Q_i)$ for $i = 1, 2, \ldots$, and generate $Q_1, Q_2, \ldots$ until $Q_{2i} = Q_i$ in G. As in the Brent-Pollard "rho" algorithm [5], we expect this to take $O(\sqrt{n_1})$ steps. The only flaw in this nice idea (which would allow us to take $\beta = 2$ in the analysis below) is that we do not know how to define a suitable pseudo-random function f. Hence, we resort to the following (less efficient) algorithm. r is a parameter whose choice is discussed later.

Define $Q_1 = Q$ and

$$Q_{j+1} = \begin{cases} Q_j * Q_k & \text{, where } k = k(j) \text{ is a randomly} \\ & \text{chosen integer in } [1, j], \end{cases}$$

for $j = 1, 2, \ldots, r-1$, so $Q_1, \ldots, Q_r$ are essentially random points in H and are generated at the expense of O(r) group operations. Suppose $Q_j = (x_j, y_j)$ and let

$$d = \prod_{i=1}^{r-1} \prod_{j=i+1}^{r} (y_i - y_j) \pmod{N} \qquad (7.1)$$

If, for some $i < j \leq r$, $Q_i = Q_j$ in G, then $p \mid (y_i - y_j)$ so $p \mid d$ and we can find the factor p of N by computing GCD (N, d). (We can not find i and j by the algorithm used in the Brent-Pollard "rho" method because $Q_i = Q_j$ does not imply that $Q_{i+1} = Q_{j+1}$.)

The probability P_E that $p \mid d$ is the same as the probability that at least two out of r people have the same birthday (on a planet with n_1 days in a year). For example, if $n_1 = 365$ and $r = 23$, then $P_E \cong 1/2$.

In general, for $r << n_1$,

$$P_E = 1 - \prod_{j=1}^{r-1} (1 - j/n_1) \cong 1 - \exp(-\frac{r^2}{2n_1}), \qquad (7.2)$$

so we see that $P_E \cong 1/2$ if $r \gtrsim (2(\ln 2)n_1)^{1/2}$.

We can obtain a good approximation to the behaviour of the "birthday paradox" algorithm by replacing the right side of (7.2) by a step function which is 1 if $r^2 > 2(\ln 2)n_1$ and 0 if $r^2 \leq 2(\ln 2)n_1$. Thus, a trial of the "birthday paradox" algorithm will succeed with probability approximately $\mu(\alpha, \beta)$, where β is defined by $r^2 = 2(\ln 2)m^{\beta}$, i.e.

$$\beta = 2\ln(r)/\ln(2(\ln 2)m) \qquad (7.3)$$

and $\mu(\alpha, \beta)$ is as in Section 3. A more precise expression for the probability of success is

$$\rho(\alpha) + \int_1^{\alpha-1} \{1 - 2^{-p^{(t + \beta - \alpha)/\alpha}}\} \frac{\rho(t)}{\alpha - t}\, dt . \qquad (7.4)$$

Computation shows that (7.3) gives an estimate of the probability of success which is within 10 percent of the estimate (7.4) for the values of p, α and β which are of interest.

A worthwhile refinement is to compute $y_j^2 \pmod N$ for $j = 1, \ldots, r$ (at the cost of r units of work), and replace d of (7.1) by

$$d' = \prod_{i=1}^{r-1} \prod_{j=i+1}^{r} (y_i^2 - y_j^2) \pmod N . \qquad (7.5)$$

Since $(x_j, -y_j)$ is the inverse of (x_j, y_j) in H, this refinement effectively "folds" H by identifying each point in H with its inverse. The effect is that (7.2) becomes

$$P_E \cong 1 - \exp(-\frac{r^2}{n_1}) \qquad (7.6)$$

and (7.3) becomes $r^2 = (\ln 2)m^{\beta}$, i.e.

$$\beta = 2\ln(r)/\ln((\ln 2)m) . \qquad (7.7)$$

(7.4) still holds so long as β is defined by (7.7) instead of (7.3).

Comparing the "birthday paradox" algorithm with the "p-1" two-phase algorithm, we see that they give essentially the same probability of success per trial, for the same p, α and β. Thus, we prefer the algorithm

which costs least per trial. The cost of the "birthday paradox" algorithm is $r^2/2 + O(r) = m^\beta(\ln 2)/2 + O(m^{\beta/2})$ multiplications per trial (using (7.7)), while the cost of the "p-1" two-phase algorithm is $c_2 m^\beta/\ln(m^\beta)(1 + o(1))$ per trial, where $c_2 = 3 + K$ is the cost of a group operation. (For both algorithms we have omitted the cost $c_1 m$ of the first phase, which is the same in each case.) Thus, the "birthday paradox" algorithm is preferable if $\ln(m) < 2c_2/(\beta \ln 2)$, i.e. if $m \lesssim 10^{11}$ (using K = 8 and a typical value of $\beta = 1.25$). However, $m \ll 10^{11}$ for $p < 10^{50}$ (see Section 9), so the "birthday paradox" algorithm is always preferable to the "p-1" two-phase algorithm in practice.

Our comparison indicates that the "p-1" two-phase algorithm is preferable to the "birthday paradox" algorithm if p is sufficiently large ($p \gg 10^{50}$). However, this is not the case, because for such large p (and hence large r) the cost of the second phase of the "birthday paradox" algorithm can be reduced from $O(r^2)$ to $O(r^{1+\varepsilon})$, for any $\varepsilon > 0$. We discuss this theoretical improvement in the next section.

8. The use of fast polynomial evaluation

Let P(x) be the polynomial with roots $y_1^2, \ldots, y_r^2$, i.e.

$$P(x) = \prod_{j=1}^{r} (x - y_j^2) = \sum_{j=0}^{r-1} a_j x^j \pmod N \qquad (8.1)$$

and let M(r) be the work necessary to multiply two polynomials of degree r, obtaining a product of degree 2r. As usual, we assume that all arithmetic operations are performed modulo N, where N is the number which we are trying to factorize.

Because a suitable root of unity (mod N) is not known, we are unable to use algorithms based on the FFT [1]. However, it is still possible to reduce M(r) below the obvious $O(r^2)$ bound. For example, binary splitting and the use of Karatsuba's idea gives $M(r) = O(r^{\log_2 3})$ (see [11]).

The Toom-Cook algorithm [11] does not depend on the FFT, and it shows that

$$M(r) = O(r^{1 + (c/\ln(r))^{1/2}}) \tag{8.2}$$

as $r \to \infty$, for some positive constant c. However, the Toom-Cook algorithm is impractical, so let us just assume that we use a polynomial multiplication algorithm which has

$$M(r) = O(r^{1 + \varepsilon}) \tag{8.3}$$

for some fixed ε in (0, 1). Thus, using a recursive algorithm, we can evaluate the coefficients $a_0, \ldots, a_{r-1}$ of (8.1) in O(M(r)) multiplications, and it is then easy to obtain the coefficients $b_j = (j + 1)a_{j+1}$ in the formal derivative $P'(x) = \sum b_j x^j$.

Using fast polynomial evaluation techniques [4], we can now evaluate P'(x) at r points in time O(M(r)). However,

$$d'^2 = \prod_{j=1}^{r} P'(y_j^2) \pmod N, \tag{8.4}$$

so we can evaluate d'^2 and then GCD (N, d'^2).

Thus, we can perform the "birthday paradox" algorithm in time $O(m) + O(r^{1 + \varepsilon})$ per trial, instead of the $O(m) + O(r^2)$ assumed in Section 7. To estimate the effect of this improvement, choose α as in Section 5 and $\beta = 2/(1 + \varepsilon)$ so that each phase of the "birthday paradox" algorithm takes about the same time. From (3.4) we have

$$\begin{aligned} \rho(\alpha)/\mu(\alpha, \beta) &= O(\ln\alpha/(\alpha\ln\alpha)^{2/(1+\varepsilon)}) \\ &= O(\ln\ln(p)/(\ln(p)\ln\ln(p))^{1/(1+\varepsilon)}) . \end{aligned} \tag{8.5}$$

Thus, for any $\varepsilon' > \varepsilon$, we have a speedup of at least order $(\ln(p))^{1/(1+\varepsilon')}$ over Lenstra's algorithm. If we use (8.2) instead of (8.3) we obtain a speedup of order ln(p) in the same way. This should be compared with the speedup of order lnln(p) for the "p-1" two-phase algorithm.

Unfortunately the constants involved in the "O" estimates make the use of "fast" polynomial multiplication and evaluation techniques of little value unless r is quite large. If r is a power of 2 and binary splitting is used, so $\varepsilon = \log_2 3 - 1 \cong 0.585$ above, we estimate that d'^2 can be evaluated in

$8r^{1+\varepsilon}$ + O(r) time units, compared to $r^2/2$ + O(r) for the obvious algorithm. Thus, the "fast" technique may actually be faster if $r \geq 2^{10}$. From the results of Section 9, this occurs if $p \gtrsim 10^{22}$.

9. Optimal choice of parameters

In Table 1 we give the results of a numerical calculation of the expected work W required to find a prime factor p of a large integer N, using five different algorithms:

1. The Brent-Pollard "rho" algorithm [5], which may be considered as a benchmark.
2. Lenstra's one-phase elliptic curve algorithm, as described in Section 5.
3. Our "p-1" two-phase elliptic curve algorithm, as described in Section 6.
4. Our "birthday paradox" two-phase algorithm, as described in Section 7, with $\varepsilon = 1$. (W is computed from (7.4) and (7.7).)
5. The "birthday paradox" algorithm with $\varepsilon = 0.585$, as described in Section 8, with r restricted to be a power of 2.

In all cases W is measured in terms of multiplications (mod N), with one extended GCD computation counting as 8 multiplications (see Section 2). The parameters α and β were chosen to minimize the expected value of W for each algorithm (using numerical minimization if necessary). The results are illustrated in Figure 1.

From Table 1 we see that Algorithm 4 is better than Algorithm 1 for $p \gtrsim 10^{10}$, while Algorithm 2 is better than Algorithm 1 for $p \gtrsim 10^{13}$. Algorithm 3 is about 3 times faster than Algorithm 2, while Algorithm 4 is 4 to 4.5 times faster than Algorithm 2. Algorithm 5 is slightly faster than Algorithm 4 if $p \gtrsim 10^{22}$.

The differences between the algorithms appear more marked if we consider how large a factor p we can expect to find in a given time. Suppose that we can devote 10^{10} units of work to the factorization. Then, by interpolation in Table 1 (or from Figure 1), we see that the upper bounds on p for Algorithms 1, 2 and 4 are about 10^{19}, 10^{26} and 10^{29} respectively.

$\log_{10}(p)$	Alg. 1	Alg. 2	Alg. 3	Alg. 4	Alg. 5
6	3.49	4.67	4.24	4.09	4.26
8	4.49	5.38	4.94	4.76	4.91
10	5.49	6.03	5.57	5.39	5.53
12	6.49	6.62	6.16	5.97	6.07
14	7.49	7.18	6.71	6.53	6.60
16	8.49	7.71	7.23	7.05	7.12
18	9.49	8.21	7.73	7.56	7.59
20	10.49	8.69	8.21	8.04	8.05
30	15.49	10.85	10.35	10.22	10.14
40	20.49	12.74	12.22	12.11	11.97
50	25.49	14.44	13.91	13.82	13.62

Table 1: $\log_{10}W$ versus $\log_{10}p$ for various algorithms

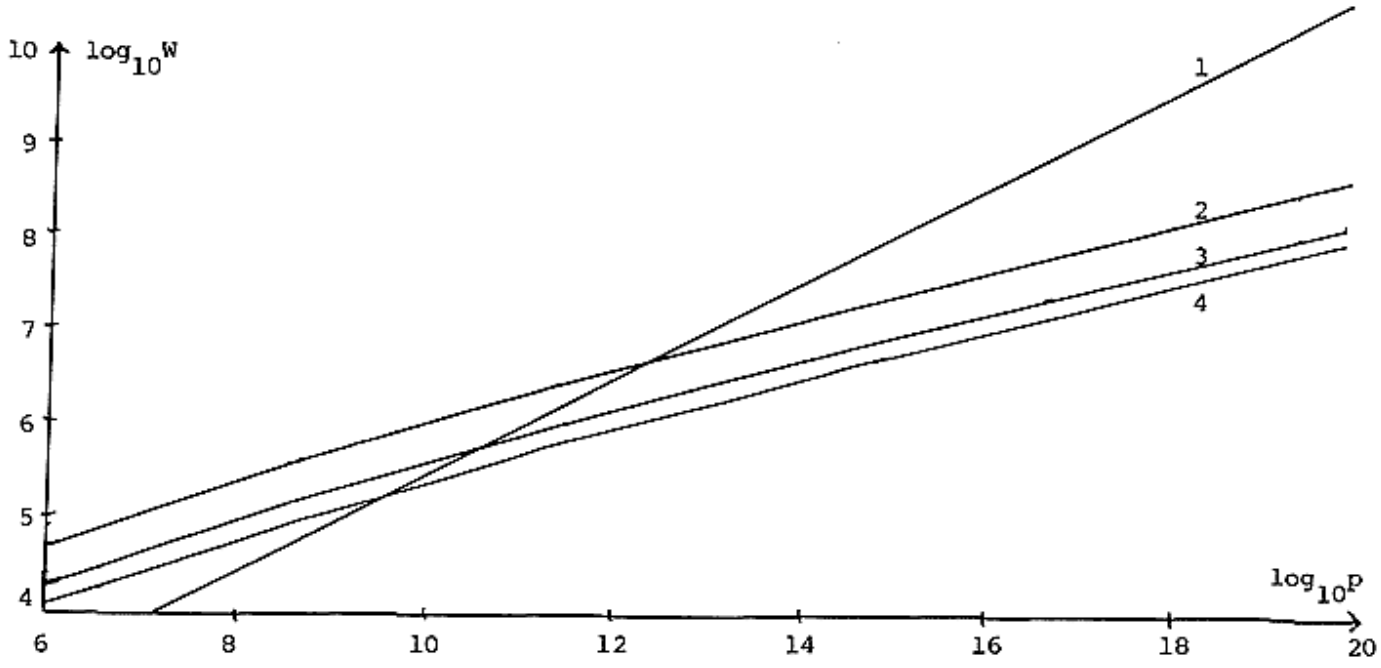

Figure 1: $\log_{10}W$ versus $\log_{10}p$ for Algorithms 1-4

In Table 2 we give the optimal parameters α, β, $m = p^{1/\alpha}$, $r = ((\ln 2)m^{\beta})^{1/2}$, T = expected number of trials (from (7.4)), m/T, w_{21} = (work for phase 2)/(work for phase 1), and S = speedup over Lenstra's algorithm, all for Algorithm 4 and several values of p.

$\log_{10}p$	α	β	m	r	T	w_{21}	m/T	S
10	3.72	1.56	484	104	12.1	0.64	40	4.37
20	4.65	1.35	19970	669	147.5	0.47	135	4.46
30	5.36	1.27	397600	2939	1141	0.44	348	4.32

Table 2: Optimal parameters for Algorithm 4

In order to check that the algorithms perform as predicted, we factored several large N with smallest prime factor $p \cong 10^{12}$. In Table 3 we give the observed and expected work to find each factor by Algorithms 2, 3 and 4. The agreement is reasonably good, considering the number

Algorithm	Number of factorizations	Observed work$_6$ per factor/10^6	Expected work$_6$ per factor/10^6
2	59	3.03 ± 0.38	4.17
3	50	1.26 ± 0.17	1.44
4	117	0.71 ± 0.06	0.94

Table 3: Observed versus expected work per factor for Algorithms 2-4

of approximations made in the analysis and the unproved hypothesis which is discussed in Section 4. The algorithms actually perform slightly better than expected, but their relative performance is approximately as predicted. In particular, Algorithm 4 is faster than Algorithm 2 by a factor of more than 4.

In practice we do not know p in advance, so it is difficult to choose the optimal parameters α, β etc. There are several approaches to this problem. If we are willing to devote a certain amount of time to the attempt to factorize N, and intend to give up if we are unsuccessful after the given amount of time, then we may estimate how large a factor p we are likely to find (using Table 1 or Figure 1) and then choose the optimal parameters for this "worst case" p. Another approach is to start with a

small value of m and increase m as the number of trials T increases. From Table 2, it is reasonable to take $m/T \cong 135$ if we expect to find a prime factor $p \cong 10^{20}$. Once m has been chosen, we may choose r (for Algorithm 4) or m' (for Algorithm 3) so that w_{21} (the ratio of the work for phase 2 to the work for phase 1) has a moderate value. From Table 2, $w_{21} \cong 0.5$ is reasonable. In practice these "ad hoc" strategies work well because the total work required by the algorithms is not very sensitive to the choice of their parameters (e.g. if m is chosen too small then T will be larger than expected, but the product mT is relatively insensitive to the choice of m).

10. Conclusion

Lenstra's algorithm is currently the fastest known factorization algorithm for large N having a factor $p << \sqrt{N}$, $p \gtrsim 10^{13}$. It is also ideally suited to parallel computation, since the factorization process involves a number of independent trials which can be performed in parallel.

We have described two algorithms which improve on Lenstra's algorithm by the addition of a second phase. The theoretical speedups are of order lnln(p) (for the "p-1" two-phase algorithm) and ln(p) (for the "birthday paradox" two-phase algorithm). From an asymptotic point of view this is not very impressive, but in practice a speedup of 3 to 5 is certainly worth having and may increase the p which can be found in a reasonable time by several orders of magnitude (see Figure 1 and the comments in Section 9). The "p-1" two-phase algorithm might be preferred to the "birthday paradox" algorithm because of its lower storage requirements, but provided storage is not a limitation we recommend the "birthday paradox" algorithm. Its storage requirement is $O(r(\log_2 N))$ bits, and from Table 2 we have $r < 3000$ if $p < 10^{30}$.

Given increasing circuit speeds and increasing use of parallelism, it is reasonable to predict that 10^{14} multiplications might be devoted to factorizing a number in the not-too-far-distant future. (There are about

3×10^{13} microseconds in a year.) Thus, from Table 1, it will be feasible to find prime factors p with up to about 50 decimal digits. This implies that the composite numbers N on which the RSA public-key cryptosystem [20, 21] is based should have at least 100 decimal digits if the cryptosystem is to be reasonably secure.

11. Acknowledgements

I wish to thank Sam Wagstaff, Jr. for introducing me to Lenstra's algorithm, and Mike Robson and Brendan McKay for their helpful comments (especially concerning the refinement mentioned in Section 7). Computations were performed on a VAX 11/750 computer which was funded in part by the ARGS. It is also a pleasure to acknowledge the research environment provided by the Centre for Mathematical Analysis at the Australian National University.

12. References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. E. Bach, *Lenstra's Algorithm for Factoring with Elliptic Curves (exposé)*, Computer Science Dept., Univ. of Wisconsin, Madison, Feb. 1985.
3. B. J. Birch, How the number of points of an elliptic curve over a fixed prime field varies, *J. London Math. Soc.* 43 (1968), 57-60.
4. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, 1975.
5. R. P. Brent, An improved Monte Carlo factorization algorithm, *BIT* 20 (1980), 176-184.
6. N. G. de Bruijn, The asymptotic behaviour of a function occurring in the theory of primes, *J. Indian Math. Soc.* 15 (1951), 25-32.
7. K. Dickman, On the frequency of numbers containing prime factors of a certain relative magnitude, *Ark. Mat., Astronomi och Fysik* 22A, 10 (1930), 1-14.

8. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford Univ. Press, 4th edition, 1960.

9. K. F. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Springer-Verlag, 1982, Ch. 18.

10. J-R. Joly, Équations et variétés algébriques sur un corps fini, *L'Enseignement Mathématique* 19 (1973), 1-117.

11. D. E. Knuth, *The Art of Computer Programming*, Vol. 2 (2nd edition), Addison Wesley, 1981.

12. D. E. Knuth and L. Trabb Pardo, Analysis of a simple factorization algorithm, *Theoretical Computer Science* 3 (1976), 321-348.

13. S. Lang, *Elliptic Curves - Diophantine Analysis*, Springer-Verlag, 1978.

14. D. H. Lehmer, Euclid's algorithm for large numbers, *Amer. Math. Monthly* 45 (1938), 227-233.

15. H. W. Lenstra, Jr., *Elliptic Curve Factorization*, personal communication *via* S. Wagstaff Jr., Feb. 1985.

16. J. van de Lune and E. Wattel, On the numerical solution of a differential-difference equation arising in analytic number theory, *Math. Comp.* 23 (1969), 417-421.

17. P. L. Montgomery, Modular multiplication without trial division, *Math. Comp.* 44 (1985), 519-521.

18. J. M. Pollard, Theorems in factorization and primality testing, *Proc. Cambridge Philos. Soc.* 76 (1974), 521-528.

19. J. M. Pollard, A Monte Carlo method for factorization, *BIT* 15 (1975), 331-334.

20. H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhauser, 1985.

21. R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* 21 (1978), 120-126.