Some Integer Factorization Algorithms using Elliptic Curves

Richard P. Brent
Computer Sciences Laboratory
Australian National University

24 September 1985
Revised 10 Dec. 1985
Republished 7 Nov. 1998

Abstract

Lenstra's integer factorization algorithm is asymptotically one of the fastest known algorithms, and is also ideally suited for parallel computation. We suggest a way in which the algorithm can be speeded up by the addition of a second phase. Under some plausible assumptions, the speedup is of order $\log(p)$, where p is the factor which is found. In practice the speedup is significant. We mention some refinements which give greater speedup, an alternative way of implementing a second phase, and the connection with Pollard's "$p-1$" factorization algorithm.

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" [17]. Under some plausible assumptions Lenstra's algorithm finds a prime factor p of a large composite integer N in expected time

$$T_1(p) = \exp\left(\sqrt{(2 + o(1)) \ln p \ln \ln p}\right), \tag{1.1}$$

where "$o(1)$" means a term which tends to zero as $p \to \infty$. Previously algorithms with running time $\exp\left(\sqrt{(1 + o(1)) \ln N \ln \ln N}\right)$ were known [27]. However, since $p^2 \le N$, 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] 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 §8 suggest how large p has to be before Lenstra's algorithm is faster.

Appeared in *Australian Computer Science Communications* 8 (1986), 149–163.
Retyped, with corrections and postscript, by Frances Page at Oxford University Computing Laboratory, 1998.
Key words: integer factorization, Monte Carlo algorithm, elliptic curve algorithm, ECM, analysis of algorithms.
 rpb102 typeset using LaTeX.

After some preliminaries in §§2–4, we describe Lenstra's algorithm in §5, and outline the derivation of (1.1). In §6 and §7 we describe how Lenstra's algorithm can be speeded up by the addition of a second phase which is based on the same idea as the well-known "paradox" concerning the probability that two people at a party have the same birthday [25]. The two-phase algorithm has expected running time $O(T_1(p)/\ln p)$. In practice, for p around 10^{20}, the "birthday paradox algorithm" is about 4 times faster than Lenstra's (one-phase) algorithm. The performance of the various algorithms is compared in §8, and some refinements are mentioned in §9.

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 $a * b \bmod N$. 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 [13, 20] 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 $a * u = 1 \pmod N$. We write $u = a^{-1} \pmod N$. u can be computed by the extended GCD algorithm [13], 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 [16] this was reduced to $6 \le K \le 10$ (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 [13] 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 [13].

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 $n_1 \ge n_2 \ge \dots$. For $\alpha \ge 1$, $\beta \ge 1$, define

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

and

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

(For a precise definition of "a random integer close to M", see [14]. It is sufficient to consider integers uniformly distributed in $[1, M]$.)

Several authors have considered the function $\rho(\alpha)$, see for example [7, 9, 13, 14, 18]. It satisfies a differential-difference equation

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

and may be computed by numerical integration from

$$\rho(\alpha) = \begin{cases} 1 & \text{if } 0 \le \alpha \le 1 \\ \frac{1}{\alpha}\int_{\alpha-1}^{\alpha} \rho(t)\, dt & \text{if } \alpha > 1. \end{cases}$$

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 [14] consider $\mu(\alpha, 2)$ and by following their argument with trivial modifications we find that

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

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

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

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 and $p \ge 5$. 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"

$$y^2 = x^3 + ax + b \qquad (\text{mod } p), \tag{4.1}$$

where a and b are constants, $4a^3 + 27b^2 \ne 0$. 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 $P_1 = I$ then $P_3 := P_2$
else if $P_2 = I$ then $P_3 := P_1$
else if $(x_1, y_1) = (x_2, -y_2)$ then $P_3 := I$
else
    begin
    if $x_1 = x_2$ then $\lambda := (2y_1)^{-1}(3x_1^2 + a) \bmod p$
        else $\lambda := (x_1 - x_2)^{-1}(y_1 - y_2) \bmod p$;
    $\{\lambda$ is the gradient of the line joining P_1 and $P_2\}$
    $x_3 := (\lambda^2 - x_1 - x_2) \bmod p$;
    $y_3 := (\lambda(x_1 - x_3) - y_1) \bmod p$
    end.

It is well-known that $(G, *)$ forms an Abelian group with identity element I. Moreover, by the "Riemann hypothesis for finite fields" [12], the group order $g = |G|$ satisfies the inequality

$$|g - p - 1| \; < \; 2\sqrt{p}. \tag{4.2}$$

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 §3 will apply with $M = p$. Some results of Birch [3] suggest its plausibility. 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 §8 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 2 units of work), a squaring then requires 12 units and a nonsquaring multiplication requires 9 units of work.

The reader who is interested in learning more about the theory of elliptic curves should consult [11], [12] or [15].

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

$$y^2 = x^3 + ax + b \qquad (\bmod N) \tag{5.1}$$

(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 §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.

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 §4, and neglecting the fact that the results of §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 [10] 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 §2). Since $m = p^{1/\alpha}$, the expected work to find p is

$$W_1(\alpha) \sim c_1 p^{1/\alpha} / \rho(\alpha). \tag{5.2}$$

To minimise $W_1(\alpha)$ we differentiate the right side of (5.2) and set the result to zero, obtaining $\ln(p) = -\alpha^2 \rho'(\alpha)/\rho(\alpha)$, or (from the differential equation satisfied by ρ),

$$\ln p = \frac{\alpha \rho(\alpha - 1)}{\rho(\alpha)} . \tag{5.3}$$

In practice p is not known in advance, so it is difficult to choose α so that (5.3) is satisfied. This point is discussed in §8. For the moment assume that we know or guess an approximation to $\log(p)$, and choose α so that (5.3) holds, at least approximately. From (3.2),

$$\ln p = \alpha^2 (\ln \alpha + O(\ln \ln \alpha)) , \tag{5.4}$$

so

$$\alpha \sim \sqrt{\frac{2 \ln p}{\ln \ln p}} \tag{5.5}$$

and

$$\ln \, W_1(\alpha) \sim \frac{\rho(\alpha - 1)}{\rho(\alpha)} - \ln \rho(\alpha) \sim 2\alpha \ln \alpha \sim \sqrt{2 \ln p \ln \ln p} \, . \tag{5.6}$$

Thus

$$T_1(p) = W_1(\alpha) = \exp\left(\sqrt{(2+o(1))\ln p \ln\ln p}\right), \qquad (5.7)$$

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

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

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 "birthday paradox" 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". Let $m = p^{1/\alpha}$ be as in §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 §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 §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 §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.)

Let $H = \langle Q \rangle$ be the cyclic group generated by Q. A nice idea is to take some pseudo-random function $f\colon Q \to Q$, define $Q_0 = Q$ and $Q_{i+1} = f(Q_i)$ for $i = 0, 1, \ldots$, 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 is that we do not know how to define a suitable pseudo-random function f. Hence, we resort to the following (less efficient) algorithm.

Define $Q_1 = Q$ and

$$Q_{j+1} = \begin{cases} Q_j^2 & \text{with probability } 1/2, \\ Q_j^2 * Q & \text{with probability } 1/2, \end{cases}$$

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

$$d = \prod_{i=1}^{r-1} \prod_{j=i+1}^{r} (y_i - y_j) \qquad (\mathrm{mod}\ N) \qquad (6.1)$$

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

The probability that $p|d$ 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 $r = 23$, the probability $P_E \cong 1/2$.

In general, for $r \ll n_1$,

$$P_E = 1 - \prod_{j=1}^{r-1} (1 - j/n_1) \cong 1 - \exp\left(-\frac{r^2}{2n_1}\right), \tag{6.2}$$

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

We can obtain a good approximation to the behaviour of the "birthday paradox" algorithm by replacing the right side of (6.2) by a step function which is 1 if $r^2 > 2n_1 \ln 2$ and 0 if $r^2 \leq 2n_1 \ln 2$. Thus, a trial of the "birthday paradox" algorithm will succeed with probability approximately $\mu(\alpha, \beta)$, where β is defined by $r^2 = 2m^\beta \ln 2$, i.e.

$$\beta = \frac{2\ln r - \ln(2\ln 2)}{\ln m} \tag{6.3}$$

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

$$\rho(\alpha) + \int_0^{\alpha-1} \left\{1 - 2^{-p^{(t+\beta-\alpha)/\alpha}}\right\} \frac{\rho(t)}{\alpha - t}\, dt. \tag{6.4}$$

Computation shows that (6.3) gives an estimate of the probability of success which is within 10% of the estimate (6.4) for the values of p, α and β which are of interest, so we shall use (6.3) below (but the numerical results given in §8 were computed using (6.4)).

A worthwhile refinement is to replace d of (6.1) by

$$D = \prod_{i=1}^{r-1} \prod_{j=i+1}^{r} (x_i - x_j) \qquad (\bmod\ N). \tag{6.5}$$

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 (6.2) becomes

$$P_E \cong 1 - \exp\left(-\frac{r^2}{n_1}\right) \tag{6.6}$$

and (6.3) becomes $r^2 = m^\beta \ln 2$, i.e.

$$\beta = \frac{2\ln r - \ln\ln 2}{\ln m} \tag{6.7}$$

(6.4) still holds so long as β is defined by (6.7) instead of (6.3).

7 The use of fast polynomial evaluation

Let $P(x)$ be the polynomial with roots $x_1, \ldots, x_r$, i.e.

$$P(x) = \prod_{j=1}^{r} (x - x_j) = \sum_{j=0}^{r-1} a_j x^j \qquad (\bmod\ N) \tag{7.1}$$

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 [13] gives $M(r) = O(r^{\log_2 3})$.

The Toom-Cook algorithm [13] does not depend on the FFT, and it shows that

$$M(r) = O\left(r^{1+(c/\ln r)^{1/2}}\right) \tag{7.2}$$

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

$$M(r) = O\left(r^{1+\varepsilon}\right) \tag{7.3}$$

for some fixed ε in $(0, 1)$. Thus, using a recursive algorithm, we can evaluate the coefficients $a_0, \ldots, a_{r-1}$ of (7.1) 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 $P'(x) = \Sigma b_j x^j$.

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

$$D^2 = \prod_{j=1}^{r} P'(x_j), \tag{7.4}$$

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 $O(m) + O(r^2)$ if (6.5) is evaluated in the obvious way. To estimate the effect of this improvement, choose α as in §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

$$\frac{\rho(\alpha)}{\mu(\alpha, \beta)} = O\left(\frac{\ln \alpha}{(\alpha \ln \alpha)^{2/(1+\varepsilon)}}\right) = O\left(\frac{\ln \ln p}{(\ln p \ln \ln p)^{1/(1+\varepsilon)}}\right) . \tag{7.5}$$

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 (7.2) instead of (7.3) we obtain a speedup of order $\ln p$ in the same way.

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 §8, this occurs if $p \geq 10^{22}$ (approximately).

8 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 four 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 §5.

3. Our "birthday paradox" two-phase algorithm, as described in §6, with $\varepsilon = 1$.

4. The "birthday paradox" algorithm with $\varepsilon = 0.585$, as described in §7, with r restricted to be a power of 2.

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

Table 1: $\log_{10} W$ versus $\log_{10} p$ for Algorithms 1–4

In all cases W is measured in terms of multiplications (mod N), with one extended GCD computation counting as 8 multiplications (see §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 3 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 4 to 4.5 times faster than Algorithm 2. Algorithm 4 is slightly faster than Algorithm 3 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 3 are about 10^{19}, 10^{26} and 10^{29} respectively.

$\log_{10} p$	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 3

In Table 2 we give the optimal parameters α, β, $m = p^{1/\alpha}$, $r = (m^\beta \ln 2)^{1/2}$, T = expected number of trials (from (6.4)), m/T, w_{21} = (work for phase 2)/(work for phase 1), and S = speedup over Lenstra's algorithm, all for Algorithm 3 and several values of p.

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

Algorithm	Number of Factorizations	Observed work per factor/10^6	Expected work per factor/10^6
2	126	3.41 ± 0.30	4.17
3	100	0.74 ± 0.06	0.94

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

each factor by Algorithms 2 and 3. The agreement is reasonably good, considering the number of approximations made in the analysis. If anything the algorithms appear to perform slightly better than expected.

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 Algorithms 3 or 4) 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).

9 Further refinements

In this section we mention some further refinements which can be used to speed up the algorithms described in §§5–7. Details will appear elsewhere.

9.1 Better choice of random points

Let $e \geq 1$ be a fixed exponent, let b_i and $\overline{b}_i$ be random linear functions of i, $a_i = b_i^e$, $\overline{a}_i = \overline{b}_i^e$, and $rs \sim m^\beta$. In the birthday paradox algorithm we may compute

$$(x_i, y_i) = Q^{a_i} \qquad (i = 1, \ldots, r)$$

and

$$(\overline{x}_j, \overline{y}_j) = Q^{\overline{a}_j} \qquad (j = 1, \ldots, s)$$

and replace (6.1) by

$$d = \prod_{j=1}^{s} \prod_{i=1}^{r} (x_i - \overline{x}_j) \pmod N \tag{9.1}$$

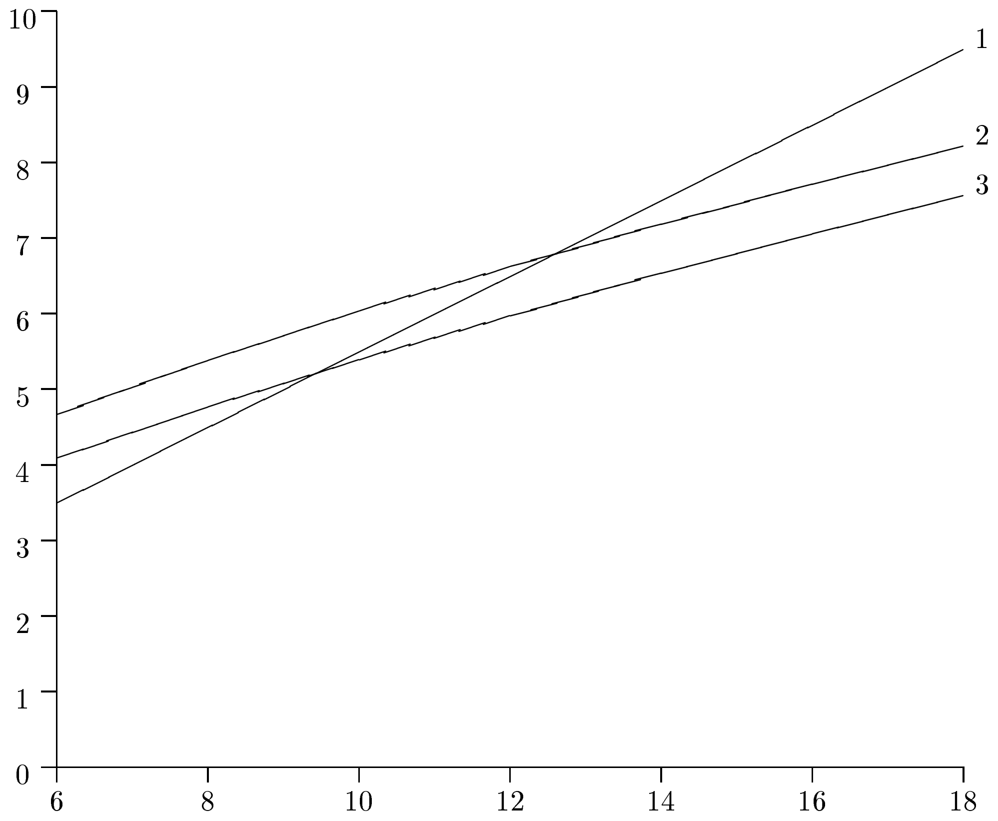

Figure 1: $\log_{10} W$ versus $\log_{10} p$ for Algorithms 1–3

Using $e > 1$ is beneficial because the number of solutions of $x^e = 1 \pmod{n_1}$ is GCD $(e, n_1 - 1)$. We take b_i and $\overline{b}_i$ to be linear functions of i so that the e-th differences of the a_i and $\overline{a}_i$ are constant, which allows the computation of $x_1, \ldots, x_r$ and $\overline{x}_1, \ldots, \overline{x}_s$ in $O((r+s)e)$ group operations. The values of $\overline{x}_j$ do not need to be stored, so storage requirements are $O(r)$ even if $s \gg r$. Moreover, by use of rational preconditioning [22, 29] it is easy to evaluate (9.1) in $(r + O(\log r))s/2$ multiplications. Using these ideas we obtain a speedup of about 6.6 over the one-phase algorithm for $p \cong 10^{20}$.

9.2 Other second phases

Our birthday paradox idea can be used as a second phase for Pollard's "$p-1$" algorithm [23]. The only change is that we work over a different group. Conversely, the conventional second phases for Pollard's "$p-1$" algorithm can be adapted to give second phases for elliptic curve algorithms, and various tricks can be used to speed them up [19]. Theoretically these algorithms give a speedup of the order $\log\log(p)$ over the one-phase algorithms, which is not as good as the $\log(p)$ speedup for the birthday paradox algorithm [6]. However, in practice, the speedups are comparable (in the range 6 to 8). We prefer the birthday paradox algorithm because it does not require a large table (or on-line generation) of primes for the second phase, so it is easier to program and has lower storage requirements.

9.3 Better choice of random elliptic curves

Montgomery [21] and Suyama [28] have shown that it is possible to choose "random" elliptic curves so that g is divisible by certain powers of 2 and/or 3. For example, we have implemented a suggestion of Suyama which ensures that g is divisible by 12. This effectively reduces p to $p/12$ in the analysis above, so gives a speedup which is very significant in practice, although not significant asymptotically.

9.4 Faster group operations

Montgomery [21] and Chudnovsky and Chudnovsky [8] have shown that the Weierstrass normal form (5.1) may not be optimal if we are interested in minimizing the number of arithmetic operations required to perform group operations. If (5.1) is replaced by

$$by^2 = x^3 + ax^2 + x \qquad (\mathrm{mod}\ N) \tag{9.2}$$

then we can dispense with the y coordinate and compute P^n in $10\log_2 n + O(1)$ multiplications (mod N), instead of about $\frac{3}{2}(\frac{11}{3} + K)\log_2 n$ multiplications (mod N), a saving of about 43% if $K = 8$.

The effect of the improvements described in §§9.3–9.4 is to speed up both the one-phase and two-phase algorithms by a factor of 3 to 4.

10 Conclusion

Lenstra's algorithm is currently the fastest known factorization algorithm for large N having a factor $p \gtrsim 10^{13}$, $\ln p/\ln N \ll 1/2$. 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 how to improve on Lenstra's algorithm by the addition of a second phase. The theoretical speedup is of order $\ln(p)$. From an asymptotic point of view this is not very impressive, but in practice it is certainly worth having and may increase the size of factors which can be found in a reasonable time by several orders of magnitude (see Figure 1 and the comments in §8).

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 by the algorithms based on elliptic curves. Other algorithms [27] may be even more effective on numbers which are the product of two roughly equal primes. This implies that the composite numbers N on which the RSA public-key cryptosystem [25, 26] 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 Brendan McKay, Andrew Odlyzko, John Pollard, Mike Robson and Hiromi Suyama for their helpful comments on a first draft of this paper.

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. R. P. Brent, *Some integer factorization algorithms using elliptic curves*, Report CMA-R32-85, Centre for Math. Analysis, Australian National University, Sept. 1985, §6.

7. N. G. de Bruijn, The asymptotic behaviour of a function occurring in the theory of primes, *J. Indian Math. Soc.* 15 (1951), 25–32.

8. D. V. Chudnovsky and G. V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, preprint, Dept. of Mathematics, Columbia Univ., July 1985.

9. 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.

10. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 4th Edition, 1960.

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

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

13. D. E. Knuth, *The Art of Computer Programming*, Vol. 2 (2nd Edition), Addison-Wesley, 1982.

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

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

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

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

18. 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.

19. P. L. Montgomery, *Speeding the Pollard methods of factorization*, preprint, System Development Corp., Santa Monica, Dec. 1983.

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

21. P. L. Montgomery, personal communication, September 1985.

22. M. Paterson and L. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, *SIAM J. Computing* 2 (1973), 60–66.

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

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

25. H. Riesel, *Prime numbers and computer methods for factorization*, Birkhauser, 1985.

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

27. R. D. Silverman, *The multiple polynomial quadratic sieve*, preprint, Mitre Corp., Bedford Mass., 1985.

28. H. Suyama, *Informal preliminary report* (8), personal communication, October 1985.

29. S. Winograd, Evaluating polynomials using rational auxiliary functions, *IBM Technical Disclosure Bulletin* 13 (1970), 1133–1135.

Postscript and historical note (added 7 November 1998)

The LaTeX source file was retyped in 1998 from the version (rpb102) which appeared in *Proceedings of the Ninth Australian Computer Science Conference*, special issue of *Australian Computer Science Communications* 8 (1986), 149–163 [submitted 24 September 1985, and in final form 10 December 1985]. No attempt has been made to update the contents, but minor typographical errors have been corrected (for example, in equations (1.1), (6.3), (6.7) and (9.2)). Some minor changes have been made for cosmetic reasons, e.g. d' was changed to D in (6.5), and some equations have been displayed more clearly using the LaTeX `\frac{...}{...}` and `\sqrt{...}` constructs – see for example (5.5)–(5.7).

A preliminary version (rpb097tr) appeared as Report CMA-R32-85, Centre for Mathematical Analysis, Australian National University, September 1985. It is more detailed but does not include the section on "further refinements" (§9 above).

For developments up to mid-1997, see:

30. R. P. Brent, Factorization of the tenth Fermat number, *Mathematics of Computation* 68 (January 1999), to appear (rpb161). A preliminary version (*Factorization of the tenth and eleventh Fermat numbers*, Technical Report TR-CS-96-02, Computer Sciences Laboratory, ANU, February 1996) is also available in electronic form (rpb161tr).

Further remarks (added 3 December 1998)

In the estimate (1.1), $T_1(p)$ is the arithmetic complexity. The bit complexity is a factor $M(N)$ larger, where $M(N)$ is the number of bit operations required to multiply integers mod N. As explained in §2, we take one multiplication mod N as the basic unit of work. In applications such as the factorization of large Fermat numbers, the factor $M(N)$ is significant.

In §4 the group operation on the elliptic curve is written as multiplication, because of the analogy with the Pollard "$p-1$" method. Nowadays the group operation is nearly always written as addition, see for example [30].

At the end of §8, we say that "the product mT is relatively insensitive to the choice of m". See [30, Table 3] for an indication of how the choice of non-optimal m changes the efficiency of the method.

Acknowledgement

I am grateful to Paul Zimmermann for his comments which prompted these "further remarks".