MATHEMATICS OF COMPUTATION
VOLUME 48, NUMBER 177
JANUARY 1987, PAGES 243-264

Speeding the Pollard and Elliptic Curve Methods of Factorization

By Peter L. Montgomery

To Daniel Shanks on his 70th birthday

Abstract. Since 1974, several algorithms have been developed that attempt to factor a large number N by doing extensive computations modulo N and occasionally taking GCDs with N. These began with Pollard's $p - 1$ and Monte Carlo methods. More recently, Williams published a $p + 1$ method, and Lenstra discovered an elliptic curve method (ECM). We present ways to speed all of these. One improvement uses two tables during the second phases of $p \pm 1$ and ECM, looking for a match. Polynomial preconditioning lets us search a fixed table of size n with $n/2 + o(n)$ multiplications. A parametrization of elliptic curves lets Step 1 of ECM compute the x-coordinate of nP from that of P in about $9.3 \log_2 n$ multiplications for arbitrary P.

1. Introduction. In 1974 and 1975, J. M. Pollard introduced two algorithms that are remarkable for their ability to locate most factors of moderate size (up to about 12 digits) of huge numbers, with many larger successes. Previously, using trial division, the practical limit was about 8 digits. H. C. Williams and H. W. Lenstra, Jr. have since announced related methods. The author is routinely finding factors of around 17 digits with Lenstra's method.

We let $\lfloor x \rfloor$ and $\lceil x \rceil$ designate the greatest integer not exceeding x and the least integer not less than x, respectively. The greatest common divisor (GCD) of two integers m and n is designated by $\mathrm{GCD}(m, n)$. This GCD is said to be *nontrivial* if it is not equal to 1. The number of primes less than or equal to n is designated by $\pi(n)$. The notation $\varphi(n)$ designates Euler's totient function. If a/b and c/d are rational numbers and N is an integer, the notation $a/b \equiv c/d \bmod N$ means $ad \equiv bc \bmod N$ and $\mathrm{GCD}(bd, N) = 1$.

The notation $(x_1, \ldots, x_n) \leftarrow (e_1, \ldots, e_n)$ designates a parallel assignment statement. The x_i must be distinct variables. To execute it, evaluate all expressions on the right. Then assign the value of each e_i to the corresponding x_i.

Define the Fibonacci numbers F_n and Lucas numbers L_n for $n \geq 0$ by

$$\begin{cases} F_0 = 0, & F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n, \\ L_0 = 2, & L_1 = 1, \quad L_{n+2} = L_{n+1} + L_n. \end{cases} \tag{1.1}$$

Suppose N is an odd composite number to be factored, and p is an unknown prime factor of N. Each algorithm does extensive computations modulo N and occasionally takes a GCD with N, hoping thereby to find p. Although p is

Received December 16, 1985; revised July 15, 1986.
1980 *Mathematics Subject Classification*. Primary 10A25.
Key words and phrases. Factorization, polynomial evaluation, elliptic curves, Lucas functions.

0025-5718/87 $1.00 + $.25 per page

unknown, the computations are also taking place modulo p; a GCD "succeeds" when an intermediate result is zero modulo p but nonzero modulo N.

Pollard observed that the cost of each GCD with N can be reduced essentially to that of a multiplication modulo N, since

(1.2) $p \mid \mathrm{GCD}(xy \bmod N, N)$ if and only if $p \mid \mathrm{GCD}(x, N)$ or $p \mid \mathrm{GCD}(y, N)$.

By repeatedly using this equation, one can trade 100 (say) GCDs with N for 99 multiplications modulo N and one GCD with N. This may cause multiple factors of N to appear at once, but that danger can be overcome by backtracking when a nontrivial GCD is found. Therefore, it is convenient to merge the multiplications modulo N and the GCDs with N into one count when comparing versions of an algorithm.

The Monte Carlo method [22] iterates a function modulo N while looking for a duplicate modulo p; it takes $O(\sqrt{p})$ comparisons and function evaluations, and quickly locates factors under 10 digits. By preconditioning the coefficients of a cubic polynomial, we reduce the cost of each comparison (i.e., GCD) to two-thirds the cost of a multiplication modulo N, for a 10% gain in speed. The cost of each comparison drops asymptotically to half a multiplication modulo N using higher-degree polynomials.

The $p - 1$ [21], $p + 1$ [33], and Elliptic Curve (ECM) [15], [16] methods each operate in an Abelian group G; the choice of G distinguishes the methods. In each case, although the elements of G are defined modulo p where p is not explicitly known, the elements can be computed via arithmetic modulo N. For example, the $p - 1$ method uses the multiplicative group of nonzero elements of $\mathrm{GF}(p)$, and these computations can be done in the ring $\mathbf{Z}_N$ of integers modulo N. Each method selects an element $a \in G$ and computes $b = a^R$ where R is a positive integer divisible by all small primes. Then it assumes $b^s = 1$ where s is not too large; the problem is to find s. The usual search technique uses one group operation to compute each successive b^s from the previous such value, but most such group operations can be eliminated by selecting an integer w near the square root of our search limit and writing $s = vw - u$, where $0 \leqslant u < w$. Then $b^s = 1$ if and only if $b^{vw} = b^u$. The values of $b^{vw} = (b^w)^v$ and b^u can be obtained through table look-ups. If each group operation requires one multiplication, this cuts the search time almost in half. By instead testing $\mathrm{GCD}(f(vw) - f(u), N)$ for suitable f, we can test multiple values of s at once, cutting the search time almost in half again.

When factoring a large integer, the best general approach may be to use trial division to find small prime factors, apply Pollard-like algorithms to find prime factors of moderate size, and apply more sophisticated algorithms [9], [10], [12], [20], [24], [25], [28], [30] if the cofactor is not prime and not too large.

Section 10 describes the implementation of these algorithms and their use in obtaining new factors of Fibonacci and Lucas numbers.

2. The Monte Carlo Method of Factoring. Let F be a function from a finite set S to itself. Select $x_0 \in S$. Define

$$x_{i+1} = F(x_i) \qquad (i \geqslant 0). \tag{2.1}$$

Since S is finite, there exist $m \geqslant 0$ and $n \geqslant 1$ such that $x_{m+n} = x_m$. By (2.1)

$$x_{i+kn} = x_i \qquad (i \geqslant m \text{ and } k \geqslant 1).$$

Following [3], we call the least such n and m the *period* and the *length of the nonperiodic segment* of the sequence x_i. If F is selected randomly, then [13, Exercise 3.1-12]

$$\mathrm{E}(m) = \mathrm{E}(n-1) = (\pi|S|/8)^{1/2} + O(1). \tag{2.2}$$

Floyd [13, Exercise 3.1-6] observed that one can find an instance where $x_i = x_j$ and $i \neq j$ by testing whether $x_{2i} = x_i$ for $i = 1, 2, 3, \ldots$. Indeed, i can be the least nonzero multiple of n not less than m, so $i \leqslant m + n$. Each iteration of Floyd's algorithm requires three evaluations of F to replace (x_i, x_{2i}) by (x_{i+1}, x_{2i+2}), and one comparison of two elements of S.

Pollard's Monte Carlo method [22] of factoring an integer N computes

$$x_{i+1} \equiv F(x_i) \bmod N, \tag{2.3}$$

where F is a suitable polynomial function of degree at least 2 and x_0 is arbitrary. Unless otherwise stated, we will assume

$$F(x) = x^2 + c \qquad (c \neq 0, -2). \tag{2.4}$$

If p is a prime factor of N, then (2.3) holds with N replaced by p, so (2.1) applies with $S = \{0, 1, \ldots, p-1\}$. Although the sequence has been defined modulo N, we can think of it as being defined modulo p. We can apply Floyd's algorithm, searching for i such that $\mathrm{GCD}(x_{2i} - x_i, N) > 1$. When $x_{2i} \equiv x_i \bmod p$, we will discover the factor p of N (unless multiple factors of N appear at once). By (2.2), this will usually occur in $O(\sqrt{p})$ iterations if F behaves like a random function.

3. Brent's Improvement to Monte Carlo. The Monte Carlo algorithm spends 75% of its time evaluating F. Although we usually cannot afford to store all values of x_i, the cost of the algorithm would drop by 25% if these values did not need to be recomputed.

In 1980, Brent [3] published a variation that computes each x_i only once and uses $O(\log N)$ storage. He computes $\mathrm{GCD}(x_i - x_j, N)$ for $j = 1, 3, 7, 15, \ldots$ and $3(j+1)/2 \leqslant i \leqslant 2j + 1$. As in Pollard's version of the algorithm, the cost of each GCD is essentially the cost of one multiplication modulo N. Brent shows that his method is 24% faster than the original algorithm, on average.

3.1. *Reducing the Cost of a GCD in Monte Carlo.* Equation (1.2) reduces the cost of each GCD to the cost of a multiplication modulo N. We can further reduce that cost, and speed Brent's version of the Monte Carlo algorithm by 14%.

Let *test* x_i *against* x_j mean to test whether $\mathrm{GCD}(x_i - x_j, N)$ is nontrivial, perhaps using (1.2). Brent's method tests x_i against x_j where j is fixed as i varies over several consecutive integers. For example, Brent tests x_i against x_{63} for $96 \leqslant i \leqslant 127$. We could instead test x_i against x_j for $i = 98, 101, 104, \ldots, 128$ and $63 \leqslant j \leqslant 65$. The latter scheme will uncover a nontrivial GCD whenever Brent's scheme uncovers one, possibly two function evaluations later. Although the new scheme makes 33 comparisons rather than 32, we claim these comparisons collectively require fewer than 32 multiplications modulo N.

Let T be a set of integers modulo N, not necessarily distinct (here $T = \{x_{63}, x_{64}, x_{65}\}$). Let g be the polynomial defined by

$$g(x) = \prod_{t \in T} (x - t) \bmod N. \tag{3.1.1}$$

If p is prime, then $p \mid \mathrm{GCD}(g(x), N)$ if and only if $p \mid \mathrm{GCD}(x - t, N)$ for some $t \in T$. When $|T| = 3$, three multiplications modulo N suffice to compute the coefficients of g. Another multiplication lets us write g as

$$g(x) = (x + \alpha_1)(x^2 + \alpha_2) + \alpha_3 = (x + \alpha_1)(F(x) - c + \alpha_2) + \alpha_3$$

for some constants $\alpha_1, \alpha_2, \alpha_3$. Set $\alpha_4 = \alpha_2 - c$ to obtain

$$g(x_i) = (x_i + \alpha_1)(x_{i+1} + \alpha_4) + \alpha_3.$$

Since x_{i+1} can be assumed known (except x_{128} and x_{129}), each block of three comparisons requires only one multiplication modulo N to evaluate $g(x_i)$, and one more for the GCD. The preconditioning used four multiplications modulo N, so we have found a way to do the 33 comparisons using $4 + 2 \cdot 11 + 2 = 28$ multiplications modulo N. As the number of consecutive terms being compared against one x_j grows, the asymptotic cost of a comparison drops to two-thirds of the cost of a multiplication modulo N. Since Brent's method uses between one-third and one-half as many comparisons as function evaluations, its overall cost drops by a fraction between $1/12$ and $1/9$.

This construction can be generalized. Winograd [2, pp. 192–194], [35] shows how to precondition a monic polynomial g of degree $2^k - 1$ so one can evaluate $g(x)$ in $2^{k-1} - 1$ multiplications if $x, x^2, x^4, \ldots, x^{2^{k-1}}$ are known. The preconditioning uses only addition, subtraction, and multiplication, so it can be done modulo N. Write

$$g(x) = g_1(x)(x^{2^{k-1}} + \alpha) + g_2(x), \tag{3.1.2}$$

where g_1 and g_2 are monic polynomials of degree $2^{k-1} - 1$ and α is a constant. Apply the scheme recursively to g_1 and g_2. The result follows by induction on k.

Define

$$F^0(x) = x, \qquad F^j(x) = F(F^{j-1}(x)), \qquad j > 0.$$

Then F^j is a monic polynomial of degree 2^j, and $F^j(x_i) = x_{i+j}$. Winograd's construction is equally valid if we replace $x^{2^{k-1}}$ by $F^{k-1}(x)$ in (3.1.2). Using $k = 3$, one can do seven comparisons at the cost of four multiplications modulo N (plus preconditioning), cutting between $3/28$ and $1/7$ of the time from Brent's method. As $k \to \infty$, we cut Brent's time about 14%. However, it is of little benefit to use high values of k, because ECM soon becomes superior.

If $F(x) = x^2 + c$ and $k = 2$, then we can precondition with three multiplications modulo N if $T = \{t_1, t_2, t_3\}$ and $F(t_1)$ is known. Let

$$\alpha_1 = t_2 + t_3, \quad \alpha_2 = \alpha_1 + t_1, \quad \alpha_3 = \alpha_1 t_1 + t_2 t_3 - c, \quad \alpha_4 = \alpha_1(F(t_1) + \alpha_3).$$

Then

$$(x - t_1)(x - t_2)(x - t_3) = (x - \alpha_2)(F(x) + \alpha_3) + \alpha_4.$$

3.2. *When* $F(x) \neq x^2 + c$. If $\deg F > 2$, the above analysis fails in two ways. We may no longer have x^2 and other monic polynomials of degrees $2, 4, 8, \ldots$ already evaluated. However, as (10.2.1) illustrates, we may be able to change F slightly so that these are available.

The more important difference is the cost of evaluating F. In Section 10.2, we will need between 20 and 40 multiplications modulo N to evaluate F at a point, so the relative costs of a comparison and of a function evaluation will change significantly. Even if we halve the cost of a comparison, the improvement in algorithm performance will be slight. Instead, we should invest in more comparisons in hopes the algorithm will terminate earlier and will therefore require fewer function evaluations.

When $\deg F > 2$, Brent [4] modifies his method to test x_i against x_j for $j < i \leqslant 2j + 1$ and $j = 0, 1, 3, 7, 15, \ldots$. In the worst case, where $x_{j+2} = x_0$ for some $j = 2^k - 1$, this version of Brent's method requires almost three times the minimum number of function evaluations before discovering $x_{3j+3} = x_{2j+1}$.

Sedgewick and Szymanski [27] present a method that can be applied when the cost of evaluating F is high. They maintain a table, holding up to M pairs (j, x_j), and a look-up function telling whether a given x is equal to x_j for some pair (j, x_j) in the table. We can represent the table by a monic polynomial g of degree M, with table look-up corresponding to testing $\mathrm{GCD}(g(x), N)$. They provide worst-cost estimates dependent upon the costs of evaluating F, of table look-up, and of changing table contents (preconditioning). They say M should be even but always include $(0, x_0)$ in the table; we can omit that pair without affecting asymptotic cost, and use a polynomial of maximum degree $M - 1$.

A generalization of Brent's scheme [25, p. 296] selects a positive integer h and a ratio $r > 1$. Select an increasing sequence $\{a_k\}_{k=0}^{\infty}$ for which $\lim a_{k+1}/a_k = r$. For $k \geqslant h$, test

$$x_i \text{ against } x_j \qquad (j = a_{k-1}, a_{k-2}, \ldots, a_{k-h};\ a_{k-1} < i \leqslant a_k).$$

As in [3], this algorithm always finds the minimum period, not a multiple thereof.

When the length of the nonperiodic segment is large but the period is small ($x_j = x_{j+1}$ where $j = a_k + 1$), this scheme can require r times the minimum number of function evaluations. When the period is large but the length of the nonperiodic segment is small ($x_j = x_0$ where $j = a_k - a_{k-h} + 1$), it can require $1 + r/(r^h - 1)$ times the minimum number of function evaluations. To minimize worst-case performance, solve $r = 1 + r/(r^h - 1)$ for r. Call the result r_{worst}.

An analysis similar to that in [3, Section 4] shows that we need

$$I_h(r) = \frac{(r^{2h} - r^h + 1)(r - 1)}{2r^h(r^h - 1)\ln r} + \frac{1}{2}$$

times the minimum number of function evaluations, on average. Let r_{avg} be the value of r minimizing $I_h(r)$, for a fixed h. Table 1 shows the approximate values of r_{avg} and r_{worst} for $h \leqslant 5$.

TABLE 1
Values of r_{avg} and r_{worst} for low h

h	r_{avg}	r_{worst}	$I_h(r_{avg})$	$I_h(r_{worst})$
1	2.4771	2.6180	1.5367	1.5390
2	1.8281	1.8019	1.2740	1.2743
3	1.5874	1.5590	1.1885	1.1890
4	1.4607	1.4384	1.1455	1.1459
5	1.3817	1.3650	1.1193	1.1196

The values for $I_h(r_{avg})$ are approximately $1 + 0.6/h$. Let the cost of each test be T function evaluations. The major cost of the algorithm is $(1 + Th)I_h(r_{avg})$ times the minimum number of function evaluations. This is minimized when $h \approx \sqrt{0.6/T}$. The application in Subsection 10.2 requires about 20 multiplications per function evaluation, and each test will take about 2/3 multiplication (tentatively assuming one or more cubic polynomials is used), so $T \approx 1/30$. We find $h \approx \sqrt{18}$, or $h = 4$. The expected cost is $(1 + 4/30)(1.1455) \approx 1.298$ times the minimum number of function evaluations. It is convenient to use instead $h = 3$ and $a_k = F_k$, the kth Fibonacci number, so $r = (\sqrt{5} + 1)/2 \approx 1.618$. The expected cost is then $(1 + 3/30)(1.1890) \approx 1.308$ times the minimum, a 1% degradation. The corresponding figure for Brent's method (with $T = 1/20$, $h = 1$, $r = 2$) is $(1.05)(1.5820) \approx 1.661$.

3.3. *Description of Algorithms.* Algorithm MCF tries to factor a composite integer N by this technique, using $F(x) = G(x^2)$ for some polynomial G. It works in conjunction with algorithms TEST (given an integer T, test whether $\mathrm{GCD}(T, N) > 1$) and CHEK (compute cofactor and backtrack when a divisor of N is found). Global constant c_{max} is one more than the maximum number of times to apply (1.2) before taking a GCD. Global array C has subscripts 1 to c_{max}, and is used to save recent arguments to TEST for possible backtracking. Global variable c is an integer between 0 and c_{max}, telling how many elements of C are in use. Inputs to MCF are N, G, and x_0. All arithmetic involving t_1, t_2, w, x, x_0, y, z, α_1, α_2, α_3, and α_4 is modulo N.

ALGORITHM MCF
$z \leftarrow G(x_0^2)$, $y \leftarrow z^2$, $(c, e, f, i, t_1, t_2, x) \leftarrow (0, 1, 1, 1, z, x_0, G(y))$
repeat {precondition again}
  {e and $f = i$ are consecutive Fibonacci numbers}
  $\{z = x_f,\ t_1 = x_e,\ t_2 = x_{f-e},\ x = x_{i+1},\ y = z^2\}$
  $\alpha_1 \leftarrow t_1 + t_2$, $\alpha_2 \leftarrow \alpha_1 + z$, $\alpha_3 \leftarrow \alpha_1 z + t_1 t_2$, $\alpha_4 \leftarrow \alpha_1(y + \alpha_3)$
  $(e, f, t_1, t_2) \leftarrow (f, e + f, z, t_1)$
  repeat {main loop, executed up to $f - e$ times}
    $\{x = x_{i+1},\ e \leqslant i < f\}$
    $w \leftarrow x - \alpha_2$, $y \leftarrow x^2$
    call TEST $((y + \alpha_3)w + \alpha_4)$
    {argument to TEST equals $(x - t_1)(x - t_2)(x - z)$}
    $x \leftarrow G(y)$, $i \leftarrow i + 1$
  until $i = f$ **or** $N = 1$
  $z \leftarrow w + \alpha_2$ {recover last x}
until $N = 1$ {or unsuccessful termination}. □

```
ALGORITHM TEST (T)
  c ← c + 1
  C_c ← T mod N
  if c = c_max then call CHEK. □

ALGORITHM CHEK
  if c > 0 and GCD(∏_{i=1}^{c} C_i mod N, N) > 1 then
    {backtrack, compute cofactor, check for multiple or repeated factors}
    repeat
      d ← N, k ← 0
      for i := 1 step 1 to c do
        if GCD(C_i, N) > 1 then
          if GCD(C_i, d) > 1 then
            d ← GCD(C_i, d)
          end if
          k ← k + 1, C_k ← C_i
        end if
      end for
      if k > 0 then
        N ← N/d, c ← k
        output d {this need not be prime, but we tried}
      end if
    until k = 0
    if N = 1 then
      {nothing to do}
    else if N passes a primality test then
      output N
      N ← 1
    else {cofactor is composite}
      reduce intermediate results modulo N
    end if
  end if
  c ← 0. □
```

4. The $p-1$ Method of Factoring. The $p-1$ method of factoring selects an integer a coprime to N (but not $a = \pm 1$). Step 1 of the algorithm computes $b \equiv a^R \bmod N$, where $R > 0$ is divisible by all prime powers below a bound B_1. This takes $O(\log R) = O(B_1)$ operations modulo N using standard methods of exponentiation [13, p. 441ff.]. It is unnecessary to compute R explicitly. If $p-1 \mid R$, then $p \mid b-1$ by Fermat's theorem, so $p \mid \mathrm{GCD}(b-1, N)$. Let $d = \mathrm{GCD}(b-1, N)$. If $1 < d < N$, then d is a proper factor of N. If $d = N$, then B_1 is too high; reduce the value of R and retry, or try a different value of a.

Step 1 of the algorithm fails if $d = 1$. One strategy abandons the algorithm. Another increases B_1. Usually one selects $B_2 \gg B_1$ and assumes $p - 1 = Qs$, where $Q \mid R$, and where s is a prime between B_1 and B_2. The problem is to find s, and

hence p. If

$$T_s \equiv b^s - 1 \equiv a^{Rs} - 1 \bmod N, \tag{4.1}$$

then $p|T_s$ by Fermat's theorem, since $p - 1|Rs$.

The standard continuation of the $p - 1$ method separately tests each prime s between B_1 and B_2 to see if $\mathrm{GCD}(T_s, N) > 1$. Let s_j denote the jth prime. Since the difference $s_{j+1} - s_j$ between two consecutive primes is known to be small (it seems not to exceed $O((\log s_j)^2)$), the values of $b^{s_{j+1}-s_j} \bmod N$ can be stored in a table. Then $b^{s_{j+1}} \equiv b^{s_j} b^{s_{j+1}-s_j} \bmod N$ for $j > \pi(B_1) + 1$. The standard continuation requires

$$O\left((\log B_2)^2\right) + O\left(\log s_{\pi(B_1)+1}\right) + 2\left(\pi(B_2) - \pi(B_1)\right)$$

multiplications modulo N and GCDs with N. If $B_2 \gg B_1$, this is approximately $2\pi(B_2)$.

Pollard also suggested a Fast Fourier Transform (FFT) continuation to the $p - 1$ method. The FFT continuation partitions the interval (B_1, B_2) into several smaller intervals, each of length w. The coefficients of

$$h(x) = \prod (x - b^u) \bmod N \qquad (0 \leqslant u < w \text{ and } \mathrm{GCD}(u, w) = 1)$$

can be computed with $O(\varphi(w)(\log \varphi(w))^2)$ operations modulo N, by recursively writing the polynomial as a product of two monic polynomials of degrees as close as possible and using fast algorithms for polynomial multiplication [1, p. 269]. A polynomial of degree n can be evaluated at n successive terms of a geometric progression in $O(n \log n)$ steps [1, Exercise 8.27], [21, p. 523]. Let $v_1 = \lceil B_1/w \rceil$ and $v_2 = \lceil B_2/w \rceil$. Use this to evaluate $\mathrm{GCD}(h(b^{vw}), N)$ for $v_1 \leqslant v \leqslant v_2$. If $s = vw - u$ where $0 \leqslant u < w$ and $v_1 < v \leqslant v_2$, then any divisor of $\mathrm{GCD}(b^s - 1, N)$ will divide $\mathrm{GCD}(b^{vw} - b^u, N)$ and hence $\mathrm{GCD}(h(b^{vw}), N)$. Selecting $w \approx \sqrt{B_2}$ gives good asymptotic performance, but Pollard expressed uncertainty as to when this becomes practical.

4.1. *Reducing the Cost of the Standard Continuation.* By mixing ideas in Pollard's two continuations of the $p - 1$ algorithm, one can improve the performance of the standard continuation. As in the FFT continuation, pick an integer w near $\sqrt{B_2}$. Let $v_1 = \lceil B_1/w \rceil$ and $v_2 = \lceil B_2/w \rceil$. Precompute the values of $b^u \bmod N$ for $0 \leqslant u < w$ and the values of $b^{vw} \bmod N$ for $v_1 \leqslant v \leqslant v_2$. This investment takes $O(\sqrt{B_2})$ multiplications modulo N. For each prime $s = vw - u$ between B_1 and B_2, replace (4.1) by

$$T_s \equiv b^{vw} - b^u \bmod N. \tag{4.1.1}$$

Then $\mathrm{GCD}(T_s, N) = \mathrm{GCD}(b^s - 1, N)$. This resembles Shanks's "baby steps" and "giant steps" [28, p. 419]. The advantage over (4.1) is that no new multiplication appears in (4.1.1). The $2(\pi(B_2) - \pi(B_1)) + o(\pi(B_2))$ multiplications modulo N and GCDs with N required by the standard continuation have been replaced by $\pi(B_2) - \pi(B_1) + O(\sqrt{B_2})$ such computations. For large B_1 and B_2, this is a 50% improvement.

On the other hand, memory requirements have grown from $O((\log B_2)^2)$ to $O(\sqrt{B_2})$ values modulo N. We can offset some of this increase by storing b^u only where $\mathrm{GCD}(u, w) = 1$. If the primes are processed in ascending order, then the values of b^{vw} can be computed as needed and need not be stored. Pollard [23] points out that we can reduce memory requirements further by using a lower value of w.

One can do almost 50% better by testing two primes at once. Change (4.1.1) to

$$T_s \equiv f(vw) - f(u) \bmod N, \tag{4.1.2}$$

where $f(n) = b^{n^2} \bmod N$ (other choices for f will be presented later). Then

1. Values of $f(n)$ can be efficiently evaluated for successive n in an arithmetic progression.

2. If $p - 1 = Qs$ where $Q|R$, then $f(m) \equiv f(n) \bmod p$ whenever $m \equiv \pm n \bmod s$.

Property 1 holds if $f(n) \equiv b^{g(n)} \bmod N$ for any integer-valued polynomial function g. Suppose $\deg g = d$ and we need $f(x)$, $f(x + h)$, $f(x + 2h), \ldots$. Define

$$\begin{cases} g_d(n) = g(n), \\ g_i(n) = g_{i+1}(n + h) - g_{i+1}(n) \qquad (i = d - 1, d - 2, \ldots, 0). \end{cases} \tag{4.1.3}$$

Then $\deg g_i = i$ for $0 \leqslant i \leqslant d$. We keep track of $b^{g_i(n)} \bmod N$ for $0 \leqslant i \leqslant d$, as n ranges over $x, x + h, \ldots$. Since g_0 is constant, only d multiplications modulo N are required when replacing n by $n + h$. This is a variation of an algorithm [13, p. 469] for tabulating the values of a polynomial at successive terms of an arithmetic progression, using only addition after the first few steps.

Property 2 explains the choice $g(n) = n^2$. The statement is easily verified, since $s|m \pm n|m^2 - n^2$ implies $p|b^s - 1|b^{m^2} - b^{n^2}$.

To utilize these properties, we find pairs (v, u) such that every prime within the interval (B_1, B_2) is represented as $vw \pm u$ for some pair (v, u) in our collection. We intend to compute the corresponding values of $\mathrm{GCD}(f(vw) - f(u), N)$ after obtaining $f(vw)$ and $f(u)$ through table look-ups. Since our table sizes will be small, we need restrictions on u and v. The restrictions might be

$$\begin{cases} |u| \leqslant u_{\max}, \\ v_1 = \lfloor B_1/w \rfloor \leqslant v \leqslant \lceil B_2/w \rceil = v_2, \end{cases}$$

where $u_{\max} \geqslant w/2$ is selected in advance. The time to build the tables of $f(vw)$ and $f(u)$ will be $O(v_2 - v_1) + O(u_{\max})$. Provided both $v_2 - v_1$ and $u_{\max}$ are much less than $\pi(B_2) - \pi(B_1)$, the primary cost of the algorithm will be proportional to the number of pairs (v, u) required, so we want both $vw \pm u$ to be primes as often as possible.

Before stating an algorithm for obtaining the (v, u) pairs, we give a numerical example. Consider $B_1 = 100$, $B_2 = 200$, $w = 30$, $u_{\max} = 30$. The 21 primes between 100 and 200 are 120 ± 19, 120 ± 17, 120 ± 11, 120 ± 7, 150 ± 1, 180 ± 17, 180 ± 13, 180 ± 1, 107, 157, 173, 191, 199. The last five primes can each be paired with a composite $vw \pm u$, giving us 13 pairs. After the tables of values of $f(vw)$ and $f(u)$ have been built, only 13 more multiplications modulo N and GCDs with N are required.

4.2. *A Pairing Algorithm.* These pairings were obtained in a straightforward way. If $s_i = vw - u$ and $s_j = vw + u$ are two primes between B_1 and B_2, then $s_i + s_j = 2vw$ is a multiple of $2w$. Conversely, if $s_i + s_j$ is a multiple of $2w$, then $s_i = vw - u$ and $s_j = vw + u$ for some v and u. Therefore, it suffices to look at residue classes

modulo $2w$. We loop through each pair of complementary residue classes, pairing each as yet unpaired prime with the least possible mate (if any) from the other class. We know $B_1/w \leqslant v \leqslant B_2/w$, but must verify $0 \leqslant u \leqslant u_{\max}$.

We would like the pairs (v, u) to be output in ascending order by v, so only one value of $f(vw)$ need be stored at a time. Instead, we will assume storage is available for $2L - 1$ successive values of $f(vw)$, where $L > \lceil u_{\max}/w \rceil$. Algorithm PAIR uses a separate queue for each residue class modulo $2w$. A *queue* is a data structure for which all insertions are made at one end (the *rear*) and all deletions are made at the other end (the *front*). If $|q| < w$, then queue Q_q contains (in ascending order) the values of a such that $2aw + q$ is prime but no pair (v, u) with $2aw + q = vw \pm u$ has been output. All members of Q_q are between $a_{\min}$ and $a_{\min} + L - 1$ inclusive. Algorithm PAIR requires a table of primes in ascending order. The time required by the algorithm is $O(\pi(B_2) - \pi(B_1)) + O((B_2 - B_1)/(L - \lceil u_{\max}/w \rceil)) + O(u_{\max})$.

ALGORITHM PAIR

$a_{\min} \leftarrow \lfloor (B_1 + w)/2w \rfloor$
set Q_q to empty state for each q satisfying $|q| < w$
while more primes between B_1 and B_2 **do**
  get next prime s, where $B_1 < s < B_2$
  $a \leftarrow \lfloor (s + w)/2w \rfloor$
  while $a \geqslant a_{\min} + L$ **do** {if storage for tables of $f(vw)$ exceeded}
    $a_{\min} \leftarrow a_{\min} + L - \lceil u_{\max}/w \rceil$
    for all q satisfying $|q| < w$ **do**
      for all $a' \in Q_q$ with $a' < a_{\min}$ **do**
        remove a' from Q_q
        output the pair $(2a', |q|)$
      end for
    end for
  end while
  $q \leftarrow s - 2aw$ {so $s = 2aw + q$}
  {look for mate in Q_{-q}, or save in Q_q}
  repeat
    if Q_{-q} is nonempty **then**
      remove the front element a' of Q_{-q}
      {we have found two primes $2aw + q$ and $2a'w - q$}
      $u \leftarrow w(a - a') + q$
      if $u > u_{\max}$ **then**
        output the pair $(2a', |q|)$
      else
        output the pair $(a + a', u)$
      end if
    else
      insert a at the rear of Q_q
      $u \leftarrow 0$ {force exit from loop}
    end if
  until $u \leqslant u_{\max}$
end while {end of loop over primes}

for all q satisfying $|q| < w$ **do** {empty the queues}
  while Q_q is nonempty **do**
    remove front element a' of Q_q
    output the pair $(2a', |q|)$
  end while
end for. □

The queues can be implemented as linked lists. It is a property of the algorithm that Q_q and Q_{-q} will not both be nonempty at once. If all prime divisors of $2w$ are below B_1, then Q_q will always be empty if $\mathrm{GCD}(q, 2w) > 1$. Therefore, the total size of the queues will never exceed $L\varphi(2w)/2$. This can be reduced to $n\varphi(2w)/2$ where $u = \lceil u_{\max}/w \rceil$ by modifying the algorithm to remove a' from Q_q before inserting a in Q_q if $a' \leqslant a - n$, thereby assuring Q_q will never have over n members. Another implementation of the queues keeps a bit pattern for each Q_q, in which bit b is set to 1 if and only if $a_{\min} + b \in Q_q$, for $0 \leqslant b < L$. The bit patterns must be shifted when $a_{\min}$ advances.

Each pair (v, u) output by Algorithm PAIR satisfies

$$2a_{\min} \leqslant v \leqslant 2(a_{\min} + L - 1) \quad \text{and} \quad 0 \leqslant u \leqslant u_{\max}. \tag{4.2.1}$$

It is straightforward to extend the algorithm to maintain tables of $f(u)$ and $f(vw)$ for all u and v satisfying (4.2.1). The operation "output the pair (v, u)" will translate into "**call** TEST($f(vw) - f(u)$); **if** $N = 1$ **then return**." Here TEST is as described in Subsection 3.3. Also initialize $c \leftarrow 0$ at the start of PAIR, and call CHEK at its end. If several values of N are to be factored using the same B_1, B_2, w, L, and $u_{\max}$, then Algorithm PAIR need be run only once.

Table 2 shows the number of pairs output by this algorithm when run with $B_1 = 10^5$, $B_2 = 10^6$ for various values of $u_{\max}$ and w. Since there are 68,906 primes in this interval, a lower bound on the number of pairs is 34,453.

We seem to do better if $2w$ is divisible by several low primes. This is reasonable since if $vw - u$ is prime, then no prime dividing $2w$ can divide $vw + u$, increasing the chance the latter is prime.

TABLE 2
Number of pairs generated by Algorithm PAIR

w	1000	1024	1155	1800	2310
$\varphi(w)$	400	512	480	480	480
$u_{\max} = \lfloor w/2 \rfloor$	62183	63605	56150	59072	56257
$u_{\max} = w$	57189	59323	50185	53317	50246
$u_{\max} = 2w$	50916	53276	44417	47038	44437
$u_{\max} = 3w$	47140	49356	41642	43695	41623
$u_{\max} = 4w$	44784	46799	39925	41657	40062
$u_{\max} = 6w$	41740	43507	38142	39408	38228
$u_{\max} = 8w$	40047	41602	37216	38168	37324
$u_{\max} = 12w$	38213	39386	36288	36974	36446
$u_{\max} = 16w$	37282	38238	35843	36379	36067
$u_{\max} = 24w$	36448	37141	35492	35933	35828

5. Lucas Functions. Let P be an element of a commutative ring with identity (normally the integers modulo N). For each integer n, define the Lucas functions $U_n = U_n(P)$ and $V_n = V_n(P)$ by

$$U_0 = 0, \quad U_1 = 1, \quad U_{n+1} = PU_n - U_{n-1},$$
$$V_0 = 2, \quad V_1 = P, \quad V_{n+1} = PV_n - V_{n-1}.$$

Also define $\Delta = \Delta(P) = P^2 - 4$. If $x^2 - Px + 1 = (x - \alpha)(x - \beta)$, and if division by $\alpha - \beta$ is allowed, then

$$U_n(P) = (\alpha^n - \beta^n)/(\alpha - \beta), \quad V_n(P) = \alpha^n + \beta^n, \quad \Delta(P) = (\alpha - \beta)^2.$$

These functions satisfy many identities [14], [33] (the argument P will be omitted when it is clear from the context):

$$U_{-n} = -U_n, \quad V_{-n} = V_n, \quad U_{2n} = U_nV_n, \quad V_{2n} = V_n^2 - 2, \quad V_n^2 - \Delta U_n^2 = 4,$$

$$2U_{m+n} = U_mV_n + V_mU_n, \quad 2V_{m+n} = V_mV_n + \Delta U_mU_n, \tag{5.1}$$

$$V_{m+n} = V_mV_n - V_{m-n}, \tag{5.2}$$
$$V_{mn}(P) = V_m(V_n(P)),$$

$$(V_{m+n} - 2)(V_{m-n} - 2) = (V_m - V_n)^2. \tag{5.3}$$

THEOREM 1. *If p is an odd prime and P is an integer and the Legendre symbol $(\Delta(P)/p) = \varepsilon \neq 0$, then $V_m(P) \equiv V_n(P) \bmod p$ whenever $p - \varepsilon | m \pm n$.*

Proof. If $m = 0$ or $n = 0$, see [14, p. 423]. Otherwise use Eq. (5.3). □

One can compute $U_n(P)$ and $V_n(P)$ from n and P with $O(\log n)$ operations [18], [33].

In Subsection 4.1, we used $f(n) = b^{n^2} \bmod N$. Another acceptable selection is

$$f(n) = b^n + b^{-n} \equiv V_n(b + b^{-1}) \bmod N.$$

This is well defined since $\mathrm{GCD}(b, N) = 1$. This selection of f requires only one multiplication modulo N to compute each successive value of $f(vw)$ or $f(u)$, by (5.2). It also leads to compatibility with the $p + 1$ method of factoring.

6. The $p + 1$ Method of Factoring. Williams's $p + 1$ method of factoring [33] assumes N is an integer to be factored and p is an unknown prime factor of N for which $p + 1$ has only small prime factors. Pick an integer P_0 other than $0, \pm 1, \pm 2$. As in the $p - 1$ method, pick bounds $0 \ll B_1 \ll B_2$. Compute $P' \equiv V_R(P_0) \bmod N$, where $R > 0$ is divisible by all prime powers below B_1. If $\mathrm{GCD}(P' - 2, N) = 1$, then Step 1 of the $p + 1$ method has been unsuccessful.

The hope is that $P_0 = \alpha + \alpha^{-1}$ for some $\alpha \in \mathrm{GF}(p^2) - \mathrm{GF}(p)$; this will hold if $\Delta(P_0) = P_0^2 - 4$ is a quadratic nonresidue modulo p. In that case, α and α^{-1} will be algebraic conjugates, implying $\alpha\bar{\alpha} = 1$. The multiplicative subgroup of $\mathrm{GF}(p^2)$ satisfying this equation has order $p + 1$, so this method succeeds if $\Delta(P_0)$ is a quadratic nonresidue and $p + 1$ is highly composite.

Williams gives a continuation similar to the standard continuation of the $p - 1$ method and requiring about $5(\pi(B_2) - \pi(B_1))$ multiplications modulo N and GCDs with N. Instead, by Theorem 1, if $\varepsilon = (\Delta(P_0)/p)$, and $p - \varepsilon = Qs$ where $Q | R$, then

$$V_m(P') \equiv V_{mR}(P_0) \equiv V_{nR}(P_0) \equiv V_n(P') \bmod p$$

whenever s divides $m \pm n$. We can now apply the methods of Subsection 4.1, using $f(n) = V_n(P') \bmod N$.

If the $p-1$ and $p+1$ algorithms are run on the same N, we hope $\varepsilon = -1$ in the $p+1$ method, but that condition seems impossible to check beforehand. Should $\varepsilon = +1$, then the $p+1$ method finds p when the $p-1$ method would have succeeded. Observe that the algorithm may be a $p-1$ method for some primes dividing N and a $p+1$ method for others; it is really a $p-(\Delta/p)$ method. Therefore [33, p. 229], the $p+1$ algorithm must normally be tried using multiple values of P_0, although Algorithm PAIR need be done only once. If little is known about the factors of N, I suggest $P_0 \equiv 2/7 \bmod N$, so that $\Delta \equiv -192/49 \bmod N$ and $(\Delta/p) = (-3/p)$ for all $p > 7$. Then $p-(\Delta/p)$ will be divisible by 6. This should find p if $3|p-1$ when $p-1$ is highly composite, or if $3|p+1$ when $p+1$ is highly composite. An alternate selection is $P_0 \equiv 6/5 \bmod N$, in which case $\Delta \equiv -64/25 \bmod N$, and $p-(\Delta/p)$ is divisible by 4. In practice, one can try $p+1$ once or twice before switching to ECM.

7. Elliptic Curves. Let K be a field of characteristic other than 2 or 3. An elliptic curve over K is the set of points $(x, y) \in K \times K$ satisfying the Weierstrass equation

$$y^2 = x^3 + Ax + B, \tag{7.1}$$

where $A, B \in K$ and $4A^3 + 27B^2 \neq 0$. These points plus a point at infinity form an Abelian group when addition is suitably defined [32, p. 181]. The identity element of the group is the point at infinity, and the negative of (x, y) is $(x, -y)$.

Four cases arise when adding two points. If either is the identity, then their sum is the other point. If the points are negatives of one another, then their sum is the identity. If $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two points on the curve where $x_1 \neq x_2$, and neither is the identity, then their sum is $P_3 = (x_3, y_3)$ where

$$m = (y_2 - y_1)/(x_2 - x_1), \qquad x_3 = m^2 - x_1 - x_2,$$
$$y_3 = m(x_1 - x_3) - y_1 = m(x_2 - x_3) - y_2.$$

Here m is the slope of a straight line passing through P_1, P_2, and $-P_3$. The remaining case occurs when $x_1 = x_2$ but $y_1 \neq -y_2$. By (7.1), this implies $y_1 = y_2$, so the points are identical; one can use $m = (3x_1^2 + A)/(2y_1)$ (the slope of the tangent line), with the above equations for x_3 and y_3.

If $K = \mathrm{GF}(p)$ where p is prime, then the order of the group is between $p + 1 - 2\sqrt{p}$ and $p + 1 + 2\sqrt{p}$ [32, p. 187] but varies with A and B. A major strength of ECM is that different curves are unlikely to have equal group orders, so ECM can be repeated (with a different curve) when it fails.

D. V. Chudnovsky and G. V. Chudnovsky [8, Section 4] give several alternative parametrizations of elliptic curves.

8. The Elliptic Curve Method of Factoring. Lenstra's elliptic curve method of factoring [15], [16] notes that although the ring of integers modulo N is not a field, unless N is prime, the same algebraic operations used to add two points over a field may be used in the ring until a noninvertible denominator is encountered. At that time, we will usually get a factor of N. He begins with a random elliptic curve and a random point $P_0 = (x_0, y_0)$ on the curve. We hope $N = pq$ where the order of the

group modulo p (and hence the order of P_0) has only small prime divisors. So compute $P' = RP_0 = (x_1, y_1)$, where R is divisible by all primes below our bound B_1. If all denominators are invertible, then Step 1 of ECM has been unsuccessful.

Lenstra does not suggest a continuation. The methods of Subsection 4.1 are applicable if we set $f(n) = x_n \bmod N$, where $nP' = (x_n, y_n)$. With this selection, the costs of Step 2 in the $p \pm 1$ methods and ECM are essentially equal.

9. Other Continuations to $p \pm 1$ and Elliptic Curve Methods.

9.1. *Avoiding a Table of Primes.* Pollard's FFT continuation to the $p - 1$ method can be viewed as follows. Let

$$\begin{cases} S_1 = \{b^u \bmod N: 0 \leqslant u < w \text{ and } \mathrm{GCD}(u, w) = 1\}, \\ S_2 = \{b^{vw} \bmod N: \lfloor B_1/w \rfloor < v \leqslant \lceil B_2/w \rceil\}. \end{cases} \tag{9.1.1}$$

Then check if $\mathrm{GCD}(s_1 - s_2, N) > 1$ for some $s_1 \in S_1$ and $s_2 \in S_2$. The algorithm of Subsection 4.1 can be viewed in this perspective if it is changed to test *all* differences $f(vw) - f(u)$, not just those for which $vw + u$ or $vw - u$ is prime and (v, u) is in our list. Define

$$\begin{cases} S_1 = \{f(u) \bmod N: 0 \leqslant u \leqslant w/2 \text{ and } \mathrm{GCD}(u, w) = 1\}, \\ S_2 = \{f(vw) \bmod N: \lceil B_1/w - \tfrac{1}{2} \rceil \leqslant v \leqslant \lfloor B_2/w + \tfrac{1}{2} \rfloor\}. \end{cases} \tag{9.1.2}$$

The two sets called S_2 have comparable sizes, but S_1 in (9.1.2) is only half as large as S_1 in (9.1.1).

Sets S_1 and S_2 in (9.1.2) have the form $S_1 = f(D_1) \bmod N$ and $S_2 = f(D_2) \bmod N$, where D_1 and D_2 are sets of integers, and where every prime between B_1 and B_2 divides some nonzero element of $D_1 \pm D_2$. In (9.1.2), D_2 is an arithmetic progression, and D_1 is almost one but has selected terms omitted. We can reduce $|D_1||D_2|$ (the potential number of comparisons) while keeping both $|D_1|$ and $|D_2|$ (and hence total memory) small, by using two sieves rather than one. Select two coprime moduli w_1 and w_2 such that $w_1 w_2 \ll B_2$ is divisible by many low primes. Set $v_1 = \lceil B_1/w_2 - w_1/2 \rceil$ and $v_2 = \lfloor B_2/w_2 + w_1/2 \rfloor$. Let

$$\begin{cases} D_1 = \{uw_1: 0 \leqslant u \leqslant w_2/2 \text{ and } \mathrm{GCD}(u, w_2) = 1\}, \\ D_2 = \{vw_2: v_1 \leqslant v \leqslant v_2 \text{ and } \mathrm{GCD}(v, w_1) = 1\}. \end{cases} \tag{9.1.3}$$

Another interesting choice is

$$D_1 = \{2^{jm}: 0 \leqslant j \leqslant 2J\}, \qquad D_2 = \{2^{Jm}3^{km}: 0 \leqslant k \leqslant K\}$$

for positive integers m, J and K where $(J + 1)(K + 1) > (B_2 - 1)/2$. These $|D_1|$ and $|D_2|$ are much larger than those in (9.1.3), since no sieving has been done, but any odd prime s will divide some nonzero element of $D_1 \pm D_2$ if $s - 1 < B_2 \mathrm{GCD}((s - 1)/2, m)$, not merely if $s < B_2$. Many primes greater than B_2 will qualify, even some greater than mB_2. We are applying a miniature $p - 1$ algorithm in our search for s. Here, D_1 and D_2 are geometric rather than arithmetic progressions, so (4.1.3) does not apply, but all members of $f(D_1)$ and $f(D_2)$ can be evaluated with $2m(J + K)$ multiplications modulo N if $f(n) = V_n(P')$.

Brent [5] suggests a "birthday paradox" continuation, in which $D_1 = D_2$. Its elements are selected randomly, and one hopes for duplicates modulo s.

Define the polynomials ($i = 1, 2$)

$$h_i(x) = \prod (x - s_i) \bmod N \qquad (s_i \in S_i).$$

We need the resultant of h_1 and h_2 (or of $h_1 h_2$ and its derivative). Schwartz [26, pp. 705–707] gives an asymptotically fast resultant algorithm. There is a danger that multiple factors of N will appear at once if a single GCD with N is done at the end of the resultant computation.

Another possibility is to evaluate $\mathrm{GCD}(h_1(s_2), N)$ for each $s_2 \in S_2$. The FFT continuation uses this approach, taking advantage of the fact that S_2 in (9.1.1) is a geometric progression, a property not shared by S_2 in (9.1.2). But a polynomial of degree at most $n - 1$ can be evaluated at n points in $O(n(\log n)^2)$ steps using other FFT algorithms [1, Chapter 8]. It remains to be determined whether this extra factor of $\log n$ will offset the reduced size of S_1 when B_1 and B_2 are in the range of interest.

Alternatively, we can precondition the coefficients of h_1, as in Subsection 3.1, so it can be evaluated at an arbitrary $s_2 \in S_2$ with about $\frac{1}{2} \deg h_1$ multiplications modulo N, for a total cost of about $\frac{1}{2} \deg h_1 \deg h_2 = \frac{1}{2} |S_1||S_2|$ multiplications. Another way to precondition using only rational operations appears in [1, Exercise 12.36]; add the condition $r_u = m - l + 1$ to its description.

9.2. *Using* $f(n) = V_{g(n)} \bmod N$. If f satisfies the second condition of Subsection 4.1 (with $p - 1 = Qs$ replaced by an appropriate group equation), then so does F where $F(n) = f(g(n))$ and where g is an odd or even polynomial function with integer coefficients. This is because if s divides either $vw \pm u$, then s will divide one of $g(vw) \pm g(u)$. The advantage to using F is that a prime $s > B_2$ may divide $g(vw) \pm g(u)$ even though s divides neither $vw \pm u$. This increases our chances of finding s and hence p, at the cost of increased time to compute values of $F(vw)$ and $F(u)$. If $\deg g > 1$, then our s might divide $F(vw) - F(u)$ when $vw \pm u$ are both composite, so this becomes more attractive when using one of the methods in Subsection 9.1. I suggest g be chosen so $g(x) \pm g(y)$ have many algebraic factors, such as $g(x) = x^n$ where n has many divisors.

This approach requires evaluating f at points $x, x + h, x + 2h, \ldots$ of an arithmetic progression. For ECM, we can compute $g(n)P$ as in (4.1.3) (unless we are using the alternative parametrization of Subsection 10.3.1). For the $p - 1$ method of factoring, successive values of $V_{g(n)}$ can be found by separately computing $b^{g(n)} \bmod N$ and $b^{-g(n)} \bmod N$ in this manner. For the $p + 1$ method, let $d = \deg g$. Define g_i for $0 \leqslant i \leqslant d$ by (4.1.3). If we know $V_{g_i(n)}/2 \bmod N$ and $U_{g_i(n)}/2 \bmod N$ for $0 \leqslant i \leqslant d$, then $5d$ multiplications modulo N suffice to replace n by $n + h$, by (5.1). Another method defines

$$\Delta' = (\Delta - \lambda^2)/4\lambda^2 \bmod N, \qquad W_k = (V_k + \lambda U_k)/2 \bmod N,$$

where λ is arbitrary except that $\mathrm{GCD}(\lambda, N) = 1$. Then

$$V_k = W_k + W_{-k} \bmod N, \qquad \lambda U_k = W_k - W_{-k} \bmod N,$$

$$W_{j+k} = W_j W_k + \Delta' \lambda^2 U_j U_k \bmod N, \qquad W_{-j-k} = W_{-j} W_{-k} + \Delta' \lambda^2 U_j U_k \bmod N.$$

For $0 \leqslant i \leqslant d$, keep track of $W_{\pm g_i(n)} \bmod N$ and $c_i U_{g_i(n)} \bmod N$ where $c_i \equiv \lambda \Delta' \bmod N$ if i is even and $c_i \equiv \lambda \bmod N$ if i is odd. Then $\lfloor 3.5d \rfloor$ multiplications modulo N suffice to replace n by $n + h$.

9.3. *Can We Pair More than Two Primes?* If Algorithm PAIR is replaced by an algorithm which generates triplets of primes, and if f is changed accordingly, then the algorithm in Subsection 4.1 will be speeded up 33%. I have been unable to do this.

If $f(x) - f(y)$ divides $f(xn) - f(yn)$ whenever x, y, and n are integers, then

$$\mathrm{GCD}(f(x_1x_2) - f(x_1y_2) - f(y_1x_2) + f(y_1y_2), N)$$

can be tested instead of separately testing

$$\mathrm{GCD}(f(x_1) - f(y_1), N) \quad \text{and} \quad \mathrm{GCD}(f(x_2) - f(y_2), N).$$

If the pairs of primes can themselves be paired for this computation, then the number of GCDs with N will drop by half. This seems to require evaluating f at too many points to be worthwhile.

10. Implementations. The author is preparing a table of factorizations [7] of Fibonacci numbers F_n for odd $n \leqslant 999$, and of Lucas numbers L_n for $n \leqslant 500$. Those tables were the primary testing grounds for these algorithms.

If $p > 5$ is an odd primitive prime factor of F_n or L_n (meaning $p|F_n$ (respectively $p|L_n$) but $p \nmid F_k$ and $p \nmid L_k$ if $0 < k < n$), then $p \equiv (5/p) \equiv (p/5) \bmod 2n$. If $5|n$, then $p \equiv 1 \bmod 2n$. Otherwise, we merely know $p \equiv \pm 1 \bmod 2n$.

Let N be a composite cofactor of a Fibonacci or Lucas number. The author found approximately 180 new factors of such N between 1983 and 1985, using these algorithms on primarily a VAX 11/780 and a CDC 7600. The Monte Carlo method was tried on approximately half of the entries, usually for about 30,000 function evaluations. Meanwhile either $p - 1$ or $p + 1$ was run against the same input number. Monte Carlo was less productive than $p \pm 1$, and was abandoned once ECM was implemented. The elliptic curve and $p + 1$ methods were run on all composite entries in the tables. The programs used a variation of the algorithm in [17]* for arithmetic modulo N.

10.1. *Implementations of $p \pm 1$.* The $p - 1$ and $p + 1$ methods were tried on each N, using increasing limits and various seeds. Williams [33] and Naur [19] had previously tried these methods. Most runs were made before ECM was discovered.

The first and last runs of $p + 1$ used $P_0 \equiv 23/11 \bmod N$. Since $\Delta = P_0^2 - 4 \equiv 45/121 \bmod N$ is 5 times a rational square, this will locate a factor p of N if $p - (5/p)$ is highly composite. However $p - (5/p)$ is known *a priori* to be divisible by $2n$.

The last $p + 1$ run used $B_1 = 2{,}000{,}000$ and $B_2 = 100{,}000{,}000$. Equation (9.1.3) was used with $w_1 = 221$ and $w_2 = 2310$. It used $f(n) = V_{g(n)}(P')$, where

$$g(n) = V_6(n) = n^6 - 6n^4 + 9n^2 - 2.$$

We wrote h_1 (of degree $\frac{1}{2}\varphi(w_2) = 240$) as a product of 16 monic polynomials each of degree 15, and expanded each of the latter using Winograd's scheme (3.1.2). It took 21 multiplications modulo N to compute each value of $f(vw_2)$, and an additional 130 multiplications modulo N and one GCD with N to evaluate $\mathrm{GCD}(h_1(f(vw_2)), N)$ if $\mathrm{GCD}(v, 221) = 1$.

*Robert Baillie has kindly pointed out an error in [17]. In the fifth line of Section 2 on p. 520, change "modulo R" to "modulo b". Also change "R" to "b" in the first statement within the **for** loop on p. 520.

The use of $f(n) = V_{g(n)}$ rather than $f(n) = V_n$ occasionally produces surprises, such as the factor 124,205,327,610,431 of $3^{281} - 1$ despite $B_2 = 2{,}000{,}000$, because 21,488,179 (the only large divisor of $p - 1$) also divides $V_3(2310 \cdot 516) + V_3(221 \cdot 733)$. Such successes seem rare, and did not occur during the final $p + 1$ run on the Fibonacci and Lucas numbers.

The largest factors found by $p \pm 1$ were

$$142{,}240{,}444{,}249{,}423{,}907{,}190{,}721 \text{ of } F_{537},$$
$$6{,}563{,}589{,}514{,}883{,}537{,}474{,}323{,}387 \text{ of } L_{442},$$
$$619{,}802{,}607{,}259{,}514{,}583{,}330{,}235{,}693{,}729 \text{ of } F_{971},$$

for which

$$p - 1 = 2^6 \cdot 3 \cdot 5 \cdot 7^2 \cdot 23 \cdot 179 \cdot 1693 \cdot 6311 \cdot 68741623 \qquad (F_{537}),$$
$$p + 1 = 2^2 \cdot 13 \cdot 17 \cdot 47 \cdot 2459 \cdot 69029 \cdot 255877 \cdot 3637223 \qquad (L_{442}),$$
$$p - 1 = 2^5 \cdot 3 \cdot 13 \cdot 23 \cdot 971 \cdot 25801 \cdot 689851 \cdot 1089469 \cdot 1146793 \qquad (F_{971}).$$

The 30-digit factor of F_{971} was found in Step 1. Silverman found a 26-digit factor of L_{431} via $p - 1$.

10.2. *Implementation of Monte Carlo.* Brent and Pollard [4] recommend $F(x) = x^m + 1$ if the factors of N are known to be congruent to 1 modulo m. They demonstrate by finding the previously unknown factor 1,238,926,361,552,897 of $2^{256} + 1$ using $m = 1024$. Use of this polynomial seems to reduce the expected number of function evaluations before finding a prime p by $(\mathrm{GCD}(p - 1, m) - 1)^{1/2}$. Gold and Sattler [11] report a similar result, without the second "-1" term.

When $5|n$, we used $F(x) = x^m + 1$. The selection of m was sometimes $120n$, sometimes $5040n$, and sometimes $16 \cdot 9 \cdot 5 \cdot 7 \cdot 11 \cdot 13n$.

For the case where $5 \nmid n$, consider $F(x) = V_m(x) + c$ where $2n|m$. Let p be a prime satisfying $p \equiv \pm 1 \bmod 2n$. An argument resembling that in [4] suggests that this reduces the expected number of function evaluations before finding p by a factor of

$$((\mathrm{GCD}(p - 1, m) + \mathrm{GCD}(p + 1, m) - 2)/2)^{1/2} \geqslant ((2n + 2 - 2)/2)^{1/2} = \sqrt{n}.$$

To utilize Algorithm MCF of Subsection 3.2, we can rewrite F as

$$F(x) = V_{m/2}(V_2(x)) + c = V_{m/2}(x^2 - 2) + c = G(x^2), \tag{10.2.1}$$

where $G(x) = V_{m/2}(x - 2) + c$. Actually, we used $F(x) = V_{m/2}(x^2)$, with m as in the $5|n$ case. Algorithm CFRC of [18] was used (along with the factorization of $m/2$) to find a fast way to evaluate $F(x)$.

The author found only ten Fibonacci and Lucas factors with this algorithm before abandoning it in favor of ECM. The largest such factor was 17,672,296,363,133,261 of F_{893}.

10.3. *Effectiveness of ECM.* The author ran ECM several times, using approximately 50 total curves for each N. In Step 2, $f(n)$ is the x-coordinate of nP.

ECM found over 100 factors missed by other methods (albeit using considerably more computer time). The largest Fibonacci and Lucas factors found by ECM were

$$2{,}442{,}882{,}935{,}400{,}038{,}849{,}127{,}521 \text{ of } F_{517},$$
$$10{,}245{,}029{,}712{,}795{,}120{,}034{,}405{,}043 \text{ of } L_{386},$$
$$12{,}158{,}771{,}296{,}959{,}377{,}863{,}294{,}133 \text{ of } F_{563},$$
$$5{,}890{,}430{,}821{,}204{,}665{,}088{,}535{,}469{,}913 \text{ of } F_{869}.$$

The factor of L_{386} was found with $B_1 = 10^6$; for the others, B_1 was between 50000 and 100000. In each case, B_2 was about 60 times as large.

The author subsequently implemented the parametrization in Subsection 10.3.1, and tackled twenty-five composite entries of the form $12^n \pm 1$ from [6]. Previous ECM runs, primarily by Silverman, had removed their small factors. Using approximately 80 curves per number, with B_1 varying from 100000 to 500000, I found eleven new factors of 20 to 24 digits, but was disappointed to find no larger ones. After two partially factored entries were completed by Silverman using the methods of [30], seventeen composite entries of 76 to 136 digits remained. Silverman's runs revealed I had missed a 22-digit factor of $12^{73} + 6 \cdot 12^{36} + 1$.

The author then returned to the Lucas numbers, using the parametrization in Subsection 10.3.1. He found three huge factors:

$$1{,}090{,}414{,}335{,}383{,}168{,}463{,}561{,}145{,}167{,}623 \text{ of } L_{412},$$
$$5{,}373{,}430{,}329{,}122{,}468{,}821{,}883{,}671{,}012{,}169 \text{ of } L_{482},$$
$$227{,}693{,}725{,}298{,}545{,}340{,}302{,}283{,}668{,}318{,}476{,}481 \text{ of } L_{464}.$$

Both 31-digit factors were found using $B_1 = 175000$. The 36-digit factor was found using $B_1 = 225000$. In each case, B_2 was 40 times as large as B_1.

10.3.1. *Elliptic Curve Parametrization.* The author's original implementation of ECM used affine coordinates, as in (7.1). Adding two points takes 2 multiplications, 6 additions, and 1 division when the points are distinct, and slightly more when they are equal. Each division requires a multiplication and an inversion. When working over several curves, the program used a scheme similar to (1.2) to do all the inversions at once, since $(1/x) = y(1/xy)$ and $(1/y) = x(1/xy)$. This reduces the asymptotic cost of an inversion to that of 3 multiplications. If one uses a method requiring $\log_2 n$ duplications of points and $0.25 \log_2 n$ additions or subtractions of points when computing nP from P, then the asymptotic cost of this method is about $(7 + 6(0.25)) \log_2 n = 8.5 \log_2 n$ multiplications per curve. However, one must run several curves at once to achieve this performance. Furthermore, this inversion algorithm is not suitable for parallel or distributed processing.

The author later discovered an alternative parametrization that requires no inversions during Step 1, once the necessary constants have been computed. It resembles (4.18i) and (4.18ii) in [8] and uses the equation

$$By^2 = x^3 + Ax^2 + x \tag{10.3.1.1}$$

for some A and B.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on the curve, with $x_1 \neq x_2$ and $x_1 x_2 \neq 0$. Then $P_1 + P_2 = (x_3, y_3)$ satisfies

$$x_3 = B\left[(y_1 - y_2)/(x_1 - x_2)\right]^2 - A - x_1 - x_2,$$
$$\begin{aligned} x_3(x_1 - x_2)^2 &= B(y_1 - y_2)^2 - (A + x_1 + x_2)(x_1 - x_2)^2 \\ &= -2By_1y_2 + x_1x_2(x_1 + x_2 + 2A) + x_1 + x_2 \\ &= B(x_2y_1 - x_1y_2)^2/x_1x_2. \end{aligned}$$

Similarly, $P_1 - P_2 = (x_4, y_4)$ satisfies

$$x_4(x_1 - x_2)^2 = B(x_2y_1 + x_1y_2)^2/x_1x_2.$$

Multiply these equations and use (10.3.1.1) to obtain

$$x_3x_4(x_1 - x_2)^2 = (x_1x_2 - 1)^2$$

after division by $(x_1 - x_2)^2$. This equation remains valid if $x_1x_2 = 0$. If $P_1 = P_2$, a similar derivation yields

$$4x_1x_3(x_1^2 + Ax_1 + 1) = (x_1^2 - 1)^2.$$

These equations reference only the x_i, not the y_i. Fortunately, ECM does not require us to compute the y_i.

Let P be an arbitrary point on the curve, and let the x-coordinate of nP be the rational number X_n/Z_n. From the ratios $(X_{m-n}: Z_{m-n})$, $(X_m: Z_m)$, and $(X_n: Z_n)$, one can compute the ratio $(X_{m+n}: Z_{m+n})$ via the addition formula

$$X_{m+n} \leftarrow Z_{m-n}(X_mX_n - Z_mZ_n)^2, \qquad Z_{m+n} \leftarrow X_{m-n}(X_mZ_n - Z_mX_n)^2$$

if $mP \neq nP$, and via the duplication formula

$$X_{2n} \leftarrow (X_n^2 - Z_n^2)^2, \qquad Z_{2n} \leftarrow 4X_nZ_n(X_n^2 + AX_nZ_n + Z_n^2)$$

if $m = n$. The addition formula is valid everywhere if we allow $\mathrm{GCD}(X_n, Z_n, N)$ to exceed 1. Once that condition occurs, it will persist, so we can periodically test $\mathrm{GCD}(Z_n, N)$.

The addition formula seems to require 8 multiplications and the duplication formula to require 6 multiplications. The costs drop if we store the ratios $(X_m: Z_m: X_m + Z_m: X_m - Z_m)$ and rewrite the formulae as (the right sides of the addition formula have been multiplied by 4)

$$X_{m+n} \leftarrow Z_{m-n}[(X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n)]^2,$$
$$Z_{m+n} \leftarrow X_{m-n}[(X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n)]^2,$$

and

$$4X_nZ_n = (X_n + Z_n)^2 - (X_n - Z_n)^2, \qquad X_{2n} \leftarrow (X_n + Z_n)^2(X_n - Z_n)^2,$$
$$Z_{2n} \leftarrow (4X_nZ_n)((X_n - Z_n)^2 + ((A + 2)/4)(4X_nZ_n)).$$

Note that we can precompute $(A + 2)/4$. These formulae require 6 multiplications and 4 additions to add two points whose difference is known, and 5 multiplications and 4 additions to duplicate a point.

Using the binary method, we can compute nP from P with $11 \log_2 n$ multiplications and $8 \log_2 n$ additions, by repeatedly computing either $(2mP, (2m + 1)P)$ or $((2m + 1)P, (2m + 2)P)$ from $(mP, (m + 1)P)$. If one starts with $X_1 = 2$ and $Z_1 = 1$, then this cost reduces to $9 \log_2 n$ multiplications and $9 \log_2 n$ additions.

We can do almost as well for arbitrary X_1 and Z_1 by noticing that these equations functionally resemble (5.2). The methods of [18] may be used to evaluate nP from P with about $1.55 \log_2 n$ addition or duplication steps, which corresponds to about $9.3 \log_2 n$ multiplications and $6.2 \log_2 n$ additions. In practice, both this method and the binary method (with $X_1/Z_1 = 2$) use about 130,000 multiplications for Step 1 to reach 10,000. The binary method has a simpler control structure and a greater percentage of squarings (44% vs. 34%) but requires 45% more additions. In the binary method, 11% of the multiplications can be replaced by additions if $(A + 2)/4$ is sufficiently small.

One can use this parametrization during Step 1 and the Weierstrass parametrization during Step 2, by arbitrarily setting $y = 1$ at the end of Step 1, using (10.3.1.1) to compute B, and applying a linear transformation to obtain (7.1).

10.3.2. *Selection of Elliptic Curves and Initial Points.* ECM lets one pick which curve to use. Naturally, one prefers a curve whose group order has some known prime divisors, since the group order is more likely to be highly composite. When using the Weierstrass equation (7.1), each linear factor $x - x_0$ of $x^3 + Ax + B$ corresponds to a point $(x_0, 0)$ of order 2 on the curve. If the cubic has three linear factors, then the group will have a subgroup isomorphic to $\mathbf{Z}_2 \times \mathbf{Z}_2$. Therefore, the group modulo each prime divisor of N will have order divisible by 4. For example, one can select three distinct squares s_1^2, s_2^2, and s_3^2, and use the point

$$(x_1, y_1) = \left((s_1^2 + s_2^2 + s_3^2)/3,\ s_1 s_2 s_3\right)$$

on the curve

$$\left(x + s_1^2 - x_1\right)\left(x + s_2^2 - x_1\right)\left(x + s_3^2 - x_1\right) = y^2.$$

When using (10.3.1.1), it is desirable to use $B = A + 2$ so that the point $(1, 1)$ will have order 4. We also desire $A = k + 1/k$ for some k, so that there will be three points of order 2. We can achieve both of these (and hence have a group order divisible by 8) providing we can select $x_1 = X_1/Z_1$ where $(k + x_1)(k + 1/x_1)$ is a perfect square. This will hold if $k = (x_1^2 - m^2)/(x_1(m^2 - 1))$ for some m. To prevent degenerate cases such as division by zero, and to ensure that our starting point is not in the known subgroup, one needs

$$mx_1\left(x_1^2 - 1\right)\left(m^2 - 1\right)\left(x_1^2 - m^2\right)\left(x_1^2 - m^4\right) \neq 0.$$

In particular, we can select $x_1 = 2$ and $m = 3, 4, 5, \ldots$.

When -1 is a quadratic residue, we can obtain a curve whose group order is divisible by 16 if we do not insist that $x_1 = 2$. The point (x, y) has order 4 if $x^2 + 2kx + 1 = 0$ and $(1 + k)y^2 = (1 - k)x^2$. Such a rational point exists if -1 and $k^2 - 1$ are quadratic residues. The latter condition holds if $(x_1^2 - 1)(x_1^2 - m^4)$ is a perfect square. One nontrivial solution is $x_1 = m^2 + 2$ where $m = (t^2 - 3)/2t$ for $t = 4, 5, 6, \ldots$.

Let p be a prime which does not divide $B(A + 2)(A - 2)$. Suyama [31] observes that the order of the group associated with (10.3.1.1) modulo p will always be divisible by 4. If $B(A + 2)$ is a quadratic residue, then the point $(1, \sqrt{(A + 2)/B})$ has order 4. If $B(A - 2)$ is a quadratic residue, then the point $(-1, \sqrt{(A - 2)/B})$ has order 4. If $(A + 2)(A - 2)$ is a quadratic residue, then the cubic has three linear factors, and again there is a subgroup of order 4.

Suyama next notes that if

$$A = (-3a^4 - 6a^2 + 1)/4a^3, \qquad B = (a^2 - 1)^2/4ab^2,$$

where $a, b \in \mathbf{Q}$ and $ab(a^2 - 1)(9a^2 - 1) \neq 0$, then the point (a, b) has order 3, implying the group order is divisible by 12. It remains to select x_1; we require that

$$4a^3x_1^3 - (3a^4 + 6a^2 - 1)x_1^2 + 4a^3x_1$$

be a perfect square. Suyama suggests $x_1 = 3a/4$, where $9 - 6a^2$ is a perfect square (e.g., $a = 6u/(u^2 + 6)$ where u is rational). Alternative initial points are $x_1 = a^3$ where $4a^2 + 5$ is a perfect square, and $x_1 = (3a^2 + 1)/4a$, where $3a^2 + 1$ is a perfect square.

There will be a torsion group of order 12 over $\mathbf{Q}$ if $(1 - a)(1 + 3a)$, and hence $B(A + 2)$ is a perfect square. Set $a = (u^2 - 4u - 12)/(u^2 + 12u - 12)$; then both $3a^2 + 1$ and $(1 - a)(1 + 3a)$ will be perfect squares whenever $u^3 - 12u$ is a perfect square (e.g., $u = 4$, 54, 49/4, 2166/625, 14884/1089). Avoid $u = 0$, -2, 6 since they lead to degenerate cases. The explicit torsion group seems to give a 50% chance that $B(A + 2)$, $B(A - 2)$ and $(A - 2)(A + 2)$ will all be quadratic residues (ensuring the group order is divisible by 24), compared with a 25% chance if we know nothing about A and B.

Acknowledgments. D. H. Lehmer provided me with a copy of his multiprecision arithmetic package. H. C. Williams and Robert Silverman verified the primality of many of the cofactors found. Andrew Odlyzko sent me [8] and [15]. My colleagues gave their support and encouragement while this work was in progress. System Development Corporation provided extensive, otherwise idle, computer time, and a Versatec printer plotter. Mike Mitchell and Richard Milton spent many hours running jobs on a CDC 7600. Without the Engineering and Mathematical Sciences Library at the University of California, Los Angeles, this research could not have been accomplished.

System Development Corporation
2525 Colorado Avenue
Santa Monica, California 90406

1. ALFRED V. AHO, JOHN E. HOPCROFT & JEFFREY D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

2. SARA BAASE, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, Mass., 1983.

3. RICHARD P. BRENT, "An improved Monte Carlo factorization algorithm," *BIT*, v. 20, 1980, pp. 176–184.

4. RICHARD P. BRENT & JOHN M. POLLARD, "Factorization of the eighth Fermat number," *Math. Comp.*, v. 36, 1981, pp. 627–630.

5. R. P. BRENT, "Some integer factorization algorithms using elliptic curves," presented to Australian Computer Science Conference, ACSC-9, 1986.

6. JOHN BRILLHART, D. H. LEHMER, J. L. SELFRIDGE, BRYANT TUCKERMAN & S. S. WAGSTAFF, JR., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ Up to High Powers*, Contemp. Math., vol. 22, Amer. Math. Soc., Providence, R. I., 1983.

7. JOHN BRILLHART, PETER L. MONTGOMERY & ROBERT D. SILVERMAN, "Tables of Fibonacci and Lucas factorizations." (In preparation.)

8. D. V. CHUDNOVSKY & G. V. CHUDNOVSKY, *Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorization Tests*, Research report RC 11262 (#50739), IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., 1985.

9. JAMES A. DAVIS & DIANE B. HOLDRIDGE, *Most Wanted Factorizations Using the Quadratic Sieve*, Sandia report SAND84-1658, August, 1984.

10. JOSEPH L. GERVER, "Factoring large numbers with a quadratic sieve," *Math. Comp.*, v. 41, 1983, pp. 287–294.

11. R. GOLD & J. SATTLER, "Modifikationen des Pollard-Algorithmus," *Computing*, v. 30, 1983, pp. 77–89.

12. RICHARD K. GUY, "How to factor a number," *Congressus Numerantium XVI*, Proc. Fifth Manitoba Conf. on Numerical Math., Winnipeg, 1976, pp. 49–89.

13. DONALD E. KNUTH, *The Art of Computer Programming, Vol. II, Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

14. D. H. LEHMER, "An extended theory of Lucas' functions," *Ann. of Math.* (2), v. 31, 1930, pp. 419–448.

15. H. W. LENSTRA, JR., "Elliptic curve factorization," announcement of February 14, 1985.

16. H. W. LENSTRA, JR., "Factoring integers with elliptic curves." (To appear.)

17. PETER L. MONTGOMERY, "Modular multiplication without trial division," *Math. Comp.*, v. 44, 1985, pp. 519–521.

18. PETER L. MONTGOMERY, "Evaluation of $V_n(P, 1)$ via Lucas chains," *Fibonacci Quart.* (Submitted.)

19. THORKIL NAUR, "New integer factorizations," *Math. Comp.*, v. 41, 1983, pp. 687–695.

20. M. A. MORRISON & J. BRILLHART, "A method of factoring and the factorization of F_7," *Math. Comp.*, v. 29, 1975, pp. 183–208.

21. J. M. POLLARD, "Theorems on factorization and primality testing," *Proc. Cambridge Philos. Soc.*, v. 76, 1974, pp. 521–528.

22. J. M. POLLARD, "A Monte Carlo method for factorization," *BIT*, v. 15, 1975, pp. 331–334.

23. J. M. POLLARD, private communication.

24. HANS RIESEL, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser Progress in Mathematics, vol. 57, Boston, 1985.

25. C. P. SCHNORR & H. W. LENSTRA, JR., "A Monte Carlo factoring algorithm with linear storage," *Math. Comp.*, v. 43, 1984, pp. 289–311.

26. J. T. SCHWARTZ, "Fast probabilistic algorithms for verification of polynomial identities," *J. Assoc. Comput. Mach.*, v. 27, 1980, pp. 701–717.

27. R. SEDGEWICK & T. G. SZYMANSKI, *The Complexity of Finding Periods*, Proc. Eleventh Annual ACM Symposium on Theory of Computing, ACM, New York, 1979, pp. 74–80.

28. DANIEL SHANKS, *Class Number, a Theory of Factorization, and Genera*, Proc. Sympos. Pure Math., vol. 20, Amer. Math. Soc., Providence, R. I., 1971, pp. 415–440.

29. ROBERT D. SILVERMAN, private communication.

30. ROBERT D. SILVERMAN, "The multiple polynomial quadratic sieve," *Math. Comp.*, v. 48, 1987, pp. 329–339.

31. HIROMI SUYAMA, "Informal preliminary report (8)," 25 Oct. 1985.

32. JOHN T. TATE, "The arithmetic of elliptic curves," *Invent. Math.*, v. 23, 1974, pp. 179–206.

33. H. C. WILLIAMS, "A $p + 1$ method of factoring," *Math. Comp.*, v. 39, 1982, pp. 225–234.

34. H. C. WILLIAMS, private communication.

35. S. WINOGRAD, "Evaluating polynomials using rational auxiliary functions," *IBM Technical Disclosure Bulletin*, v. 13, 1970, pp. 1133–1135.