The Multiple Polynomial Quadratic Sieve

By Robert D. Silverman

For Daniel Shanks on the occasion of his 70th birthday

Abstract. A modification, due to Peter Montgomery, of Pomerance's Quadratic Sieve for factoring large integers is discussed along with its implementation. Using it, allows factorization with over an order of magnitude less sieving than the basic algorithm. It enables one to factor numbers in the 60-digit range in about a day, using a large minicomputer. The algorithm has features which make it well adapted to parallel implementation.

1. Introduction. The basic quadratic sieve algorithm has origins which date back to M. Kraitchik, but was first explicitly stated and analyzed by C. Pomerance. The only two reported implementations of this algorithm were done by J. Gerver at Rutgers and J. Davis and D. Holdridge at Sandia National Laboratories. The latter used a Cray-1 to factor some numbers in the 60-70 digit range from the 'Cunningham Project'. They also used a Cray XMP to factor the then 'MOST WANTED' number $(10^{71} - 1)/9$ in 9.5 hours. While some of the arguments that follow have been detailed by Pomerance elsewhere [9], we present ours in a manner which is more oriented towards implementation of the algorithm. The basic algorithm depends on constructing a solution to the following equation, where N is the number you wish to factor:

$$(1) \qquad A^2 \equiv B^2 \bmod N.$$

If $A \not\equiv B$ and $A \not\equiv -B \bmod N$, then $(A + B, N)$ and $(A - B, N)$ are proper factors of N.

This version of the quadratic sieve generates a set of quadratic residues of N using the following single polynomial:

$$(2) \qquad Q(x) = \left(x + [\sqrt{N}]\right)^2 - N \equiv H^2 \bmod N.$$

It follows immediately that if a prime $p \mid Q(x)$, then $p \mid Q(x + kp)$ for all $k \in \mathbf{Z}$. Thus, the values of the polynomial may be factored with a sieve, once one solves $Q(x) \equiv 0 \bmod p$. This may be solved by any one of a number of available algorithms [5, p. 437]. The potential divisors p of $Q(x)$ are exactly those primes for which the Legendre symbol $(N/p) = 1$ and the unit -1 is needed to hold the sign.

The algorithm now proceeds as follows:

(i) Select a factor base $FB = \{ p_i \mid (N/p_i) = 1, p_i \text{ prime}, i = 1, \dots, F \}$ for some appropriate value of F, and $p_0 = 1$ for the sign.

(ii) Solve the quadratic equation $Q(x) \equiv 0 \bmod p_i$ for all $p_i \in FB$. There will be two roots r_1 and r_2 for each p_i.

(iii) Initialize a sieve array to zero over the interval $[-M, M]$ for some appropriate M.

(iv) For all $p_i \in FB$ add the value of $[\log(p_i)]$ to the sieve array at locations r_1, $r_1 \pm p_i, r_1 \pm 2p_i \ldots$ and $r_2, r_2 \pm p_i, r_2 \pm 2p_i \ldots$.

(v) The value of $Q(x)$ will be approximately $M\sqrt{N}$ over $[-M, M]$, so compare each sieve location with $[\log(N)/2 + \log(M)]$. Fully factored residues will have their corresponding sieve value close to this value. For these, construct the exact factorization via division. Factorizations are found so infrequently that the time to do this division is negligible. One need not check all the primes in the factor base in doing this division. If x is the location in the sieve array, one need only compute $R \equiv x \bmod p_i$. Only if R equals one of the two roots do we go ahead and do the multi-precise division;

$$(3) \qquad\qquad Q(x) = \prod_{i=0}^{F} p_i^{a_i}, \qquad p_i \in FB.$$

Let v_j be the corresponding vector of exponents $[\alpha_{j1} \; \alpha_{j2} \; \alpha_{j3} \cdots \alpha_{jF}]$ with $H_j^2 = Q(x)$.

(vi) Collect $F + 1$ factorizations. One then finds a set of residues whose product is a square via Gaussian elimination over GF(2) on the matrix formed by reducing all of the $v_j \bmod 2$. This creates a linear dependency $\bmod 2$ on the exponents, and the product of the vectors in that dependency forms a square. It is then trivial to construct an instance of congruence (1).

The chief difficulty with this approach is that one must obtain approximately as many fully factored residues as there are primes in the factor base. In order to obtain enough factorizations, M must be very large, and the residues grow linearly in size with M.

A way around this problem was formally suggested by Peter Montgomery: Simply use many polynomials to generate the residues and sieve each polynomial over a much smaller interval. Utilizing multiple polynomials enables one to keep the sieve interval small, and hence makes the residues easier to factor. We have implemented this approach and found that it is quite effective. It allows one to find enough factored residues using less than one-tenth the total sieve length in the single polynomial version. We also show that the cost of changing polynomials is small.

In fact, the implementation of Sandia was done in two stages. Their first version used a single polynomial. A later version used multiple polynomials, but in a disguised rather than explicit form. This latter version created implicit polynomials from subsequences of the main sieve which were divisible by a large prime q lying outside the factor base. They called their method "special q's". Montgomery's suggested polynomials are somewhat better, however, because they yield residues which are smaller on average and are less clumsy to implement.

In Section 2 we shall discuss how to select the coefficients of the polynomials. Section 3 will show how to compute those coefficients efficiently. Section 4 will discuss the basic steps of the algorithm. Section 5 will discuss selection of the

algorithm's input parameters. Section 6 will present some numerical results. In Section 7 we discuss its parallel implementation and finally, in Section 8, we compare our algorithm with other methods.

2. Selection of Coefficients. Select $Q(x) = Ax^2 + Bx + C$. In order to make $Q(x)$ generate quadratic residues, it is required that $B^2 - 4AC = N$. Since this latter expression is congruent to 0 or 1 mod 4, it means that if $N \equiv 3 \bmod 4$ one must premultiply it by a small constant k, so $kN \equiv 1 \bmod 4$. This is also a good thing to do, in general, because it usually allows one to find a factor base which is much richer in small primes. We will later discuss a function that may be used to evaluate a multiplier. A suggestion by Pomerance [8] allows one to do away with requiring $kN \equiv 1 \bmod 4$: Simply take $2B$ as the middle coefficient rather than B. However, we have not yet implemented this suggestion. It would be advantageous to keep the value of $Q(x)$ small, in some appropriate sense, over $[-M, M]$. There are several obvious ways of doing this, e.g.,

$$
\begin{aligned}
&\text{(a)}\quad \text{Minimize } \sup |Q(x)| \text{ over } [-M, M]\\
\text{or}\\
&\text{(b)}\quad \text{Minimize } \int_{-M}^{M} |Q(x)|\, dx\\
\text{or}\\
&\text{(c)}\quad \text{Minimize } \int_{-M}^{M} Q^2(x)\, dx\\
\text{subject to}\\
&\text{(d)}\quad B^2 - 4AC = kN \quad \text{and} \quad A, B, C \in \mathbf{Z}.
\end{aligned}
$$

(4)

It is easily seen that (4a) and (4b) are essentially equivalent. The length of the base of the parabola is $2M$ and its area will be directly proportional to its height. Relaxing the integer constraints and solving each of the above Lagrange multiplier problems, one finds that they all yield essentially the same result. The exact answer for (4a) is

$$
\begin{aligned}
A &= W_1\sqrt{kN}\,/M,\\
B &= 0,\\
C &= W_2 M\sqrt{kN}\,,
\end{aligned}
$$

(5)

where

$$
W_1 = \sqrt{2}\,/2 \quad \text{and} \quad W_2 = -1/2\sqrt{2}\,.
$$

The only difference among the results of the different minimization problems is that the constants W_1 and W_2 change very slightly. W_1 ranges from .70 to .75 and W_2 ranges from −.3 to −.35 with $W_1 W_2 = -\frac{1}{4}$.

The maximum value of $Q(x)$ over $[-M, M]$ is $M\sqrt{kN}\,/2\sqrt{2}$, a factor of $\sqrt{8}$ improvement over (2) and the 'special q' polynomials at Sandia.

$B = 0$ follows immediately from symmetry considerations, but (4a) and (4c) give similar results for A and C because the constraint $B^2 - 4AC = kN$ is very binding on the shape of $Q(x)$. The simplest way to see this is to realize that at the roots its slope will be $\pm\sqrt{kN}$. We would really like to flatten the parabola, but the constraint on the discriminant means that the curve must have a certain 'steepness'. Thus, we can do little more than translate the parabola up and down.

3. Computation of Coefficients. A simple method for selecting A, B, and C comes from a method for finding modular square roots quickly. To satisfy (4d) one must have

$$(6) \qquad\qquad B^2 \equiv kN \bmod 4A.$$

Let $A = D^2$, $(D/kN) = 1$, $D \equiv 3 \bmod 4$ and $A \approx \sqrt{kN/2}/M$. It is desirable that D be prime because if a prime in the factor base divides A, then $Q(x) \equiv 0 \bmod p$ has only one root and the probability that $p \,|\, Q(x)$ over $[-M, M]$ drops from $2/p$ to $1/p$. It is sufficient for practical purposes that D be only a probable prime. Alternatively, one may select D to be the product of primes not in the factor base, but its factorization must be known to solve (6). Our implementation took D to be a probable prime. To find the coefficients, compute

$$(7a) \qquad\qquad h_0 \equiv (kN)^{(D-3)/4} \bmod D,$$

$$(7b) \qquad\qquad h_1 \equiv kNh_0 \equiv (kN)^{(D+1)/4} \bmod D.$$

Then

$$(8) \qquad h_1^2 \equiv kN(kN)^{(D-1)/2} \bmod D \equiv kN \bmod D \quad \text{since } (D/kN) = 1.$$

Let

$$(9) \qquad\qquad h_2 \equiv (2h_1)^{-1} \left[\frac{kN - h_1^2}{D} \right] \bmod D.$$

One now has

$$(10) \qquad\qquad B \equiv h_1 + h_2 D \bmod A$$

and

$$(11) \qquad\qquad B^2 \equiv h_1^2 + 2h_1 h_2 D + h_2^2 D^2 \equiv kN \bmod A.$$

Since B must be odd, if it is even subtract it from A.

The value of $(2h_1)^{-1} \bmod D$ is easily obtained since $h_0 \equiv h_1^{-1} \bmod D$ has already been computed. One also has

$$(12) \qquad\qquad C = \left[\frac{B^2 - kN}{4A} \right].$$

It is not necessary, in practice, to actually compute (12) since the value of C is not really needed. It may be used, however, as a check on the other computations. Compute and save the value of $1/2D \bmod kN$ for later. This will enable us to quickly compute $Q(x)$ when we find a factorization.

As a result of the way D was chosen, one also has

$$(13) \qquad\qquad Q(x) \equiv H^2 \equiv \left(\frac{2Ax + B}{2D} \right)^2 \bmod kN.$$

Some care must be taken: If x lies between the real roots of $Q(x)$, then $Q(x)$ is negative, and one must subtract its value from kN.

The cost of finding the coefficients is dominated by the probable prime and residue tests on D, the computation of h_1 and $1/2h_1 \bmod D$ and $1/2D \bmod kN$. Nevertheless, the total arithmetic to be done is small.

Finally, the roots of $Q(x) \bmod p_i$, $p_i \in FB$, are

$$(14) \qquad\qquad (-B \pm \sqrt{kN})(2A)^{-1} \bmod p_i,$$

since $B^2 - 4AC$ is invariant.

Most of the cost of changing polynomials occurs in computing (14). The cost of computing (14) is dominated by the computation of $(1/2A) \bmod p_i$, which must be done for all primes in the factor base. Even with an efficient algorithm for doing this, such as the extended Euclidean algorithm, one must typically do this thousands of times when changing polynomials. On a SUN-3/75, for a 60-digit number with a factor base of 3000 primes, it takes .9 seconds to compute the coefficients and 2.6 seconds to compute all the roots.

4. Description of the Algorithm. We outline here the basic steps of the algorithm.

(i) Select a multiplier k such that $kN \equiv 1 \bmod 4$ and kN is rich in small quadratic residues. We prefer $kN \equiv 1 \bmod 8$, since $2 \in FB$ only under this condition.

(ii) Select the size of the factor base F, the length of the sieve interval $2M + 1$, and a large prime tolerance T. Suggestions for the values of these parameters are given below.

(iii) Compute a test value

$$\left[\log\left(\frac{M\sqrt{kN/2}}{p\max^T} \right) \right],$$

where $p\max$ is the largest prime in the factor base. If $T \leqslant 2$, then whenever a value in the sieve exceeds this value, the corresponding value of $Q(x)$ will be fully factored. If $T > 2$, then $Q(x)$ may not be fully factored. However, these partial factorizations can also be useful, as we shall see later.

(iv) Compute the factor base FB and $\sqrt{kN} \bmod p_i$ for all $p_i \in FB$. Compute $[\log(p_i)]$ for all $p_i \in FB$.

> **While** (not enough factorizations)
> **begin**
> Generate coefficients for the polynomial.
> Solve $Q(x) \equiv 0 \bmod p_i$ for all $p_i \in FB$.
> Do the sieving.
> Scan the sieve array: if any value exceeds the test value
> > **begin**
> > Compute $Q(x)$ from (13) and try to find its factorization via division.
> > Save the value of H and the exponent vector from the factorization of $Q(x)$.
> > **end**
> **end**

(v) *Large Prime Procedure.* Many of the factorizations found in (iv) will not be complete over the factor base. However, it has been noted that if $Q(x)$ is factored as

$$(15) \qquad\qquad Q(x) = \prod_i p_i^{a_i} L, \quad \text{where } L > 1,$$

then, whenever we find two or more instances of the same value of L occurring, we may multiply the corresponding instances of (15) together. This yields a factor of L^2 on the right-hand side and we may save this result for the matrix reduction step.

Note that it is not necessary that L be prime: We only need two or more to match. We search for matches simply by sorting all instances of (15) using the value of L as the key. We refer to (15) as a *large prime* factorization. The value of T in (ii) allows one to control the size of L. We choose to keep all L's whose value lies below $p\max^{T}$, where $p\max$ is the largest prime in the factor base. This can have the effect of more than halving the run time.

Let FF be the number of residues fully factored over the factor base and let FT be the total number of factorizations with F as the size of the factor base. Then

$$(16) \qquad R = \frac{FT}{F + FT - FF}.$$

Use this to determine when enough factorizations have occurred [11]. A dependency can almost always be found by taking $R = .96$. In practice it is not necessary to collect $F + 1$ factorizations. Usually, having about $.9F$ rows in the matrix is sufficient to find a dependency, and taking $R = .96$ achieves this.

(vi) *Matrix Reduction*. Finally, collect all of the factorizations found and reduce the matrix over GF(2). For each linear dependency S,

$$(17) \qquad \begin{aligned} P_1 &\equiv \prod_j H_j \bmod kN \quad \text{for } j \in S, \\ P_2 &\equiv \prod_i p_i^{\sum_j V_{ji}/2} \bmod kN \quad \text{for all } p_i \in FB \text{ and } j \in S. \end{aligned}$$

If $P_1 \not\equiv P_2$ and $P_1 \not\equiv -P_2 \bmod kN$, then $(P_1 + P_2, kN)$ and $(P_1 - P_2, kN)$ will be nontrivial factors of N.

(vii) *Some Coding Considerations*. Initializing the sieve array when changing polynomials does take some time. Since the sieve is an array of bytes, one should equivalence the sieve array to a full word array of the machine before doing the initialization. One can also combine the array initialization with sieving the smallest primes by selecting an appropriate value with which to initialize the sieve.

5. Selection of Input Parameters. The author has found, after many trials, that the following combination of parameters works well (K means 1000):

TABLE 1

Number of Digits	Factor Base Size	M	T	Median VAX/780 Run Time
24	100	5K	1.5	15 sec
30	200	25K	1.5	80 sec
36	400	25K	1.75	400 sec
42	900	50K	2.0	1800 sec
48	1200	100K	2.0	8100 sec
54	2000	250K	2.2	27600 sec
60	3000	350K	2.4	97200 sec
66	4500	500K	2.6	360000 sec

The multiplier may be evaluated using the following modified version of the Knuth-Schroeppel function:

$$f(k, N) = \sum_i g(p_i, kN) \log(p) - \tfrac{1}{2}\log(k) \quad \text{for all } p_i \in FB,$$

(18)

$$g(p, kN) = 2/p \quad \text{if } p \nmid k, \qquad g(2, kN) = 2 \quad \text{if } N \equiv 1 \bmod 8,$$
$$= 1/p \quad \text{if } p \mid k, \qquad\qquad\quad = 0 \quad \text{otherwise.}$$

It was observed in [7] that one might not want to keep all of the large prime factorizations that were found. For this algorithm, the time spent doing the division is significant for small numbers. A large T means the algorithm spends a lot of time constructing factorizations that might not be of much use. This is because the probability of finding a match between very large primes is small.

As numbers get larger, the percent of time spent doing the factorizations drops sharply. Since the cost of handling the large primes is small, it is advantageous to keep as many as possible. For reasons of programming convenience and ease of data management we restricted our program to single-precision integers on our machine (32 bits). The value of $p\max^T$ is usually larger than 32 bits, however, for numbers greater than 54 digits. The reason for this is to allow a margin for round-off error, since we only work with integer approximations to $\log(p)$. Also, since sieving with respect to prime powers is expensive, we do not do it. Instead, multiple instances of a given prime factor are found when constructing the factorizations. Selecting a larger value of T allows for more multiple prime factors.

Also, because of differences in machine architecture, it is desirable to break up the sieve interval into pieces. The optimal size usually depends on the amount of cache memory available. When sieving, one would like to keep global memory references to a minimum and for that purpose should make each piece of the sieve interval small enough to fit in cache.

Depending on the quality of the multiplier k and the corresponding value of the modified Knuth-Schroeppel function, one often sees variations in run time of up to a factor of 2.5 for numbers of a given size. Thus, the above times are only approximate and represent typical times that we observed in our computations.

6. Some Numerical Results. We present here several factorizations from the MOST/MORE wanted tables of the Cunningham project [1], along with several others of interest (U and V are respectively Fibonacci and Lucas numbers) and the factorization times. These were all obtained with a SUN-3/75. Typically, about 15 to 20 percent of the total run time was spent computing coefficients of the polynomials and finding their roots, although the larger numbers took up to 30 percent. The processing of the large primes from (15), the matrix reduction step, and the computation of congruence (1) took only a small fraction of the total run time. This post-processing depends only on the size of the factor base and not on the number being factored. For a factor base of 3000 primes, processing the large primes took about 10 minutes, reducing the matrix about 20 minutes, and computing congruence (1) about 1 minute. A factor base of 1000 primes took only about 1 minute total for all three steps.

The designation $a, b +$ or $a, b -$ means $a^b + 1$ or $a^b - 1$, respectively, while $2,446L$ is part of the Aurefeullian factorization of $2^{446} + 1$ [1]. The designation Pxx means a prime number of xx digits. Those factorizations marked with * were done using a parallel implementation, described below. The time given for these factorizations is the total time summed over all machines.

<div align="center">

TABLE 2

FACTORIZATION TIMES

</div>

Number	Size	Factors	TIME (hrs)
V298	45	271293387891105049.P28	0.25
U507	53	173408891952128923997973.P27	2.50
U615	54	1846858344247612322281.P33	2.33
12,67−	54	17577834702049211.P38	3.10
6,91−	56	48215910563832798697.P37	4.10
5,83 +	56	4029666108840585686296627.P31	3.75
6,86 +	56	2914764989376043020733.P35	6.50
2,224 +	58	167773885276849215533569.P35	11.5
3,131 +	60	114742271896804438572098194909.P30	10.0
2,446L	60	52016435676012089.P43	14.2
2,263−	62	120226360536848498498024035943.P36	15.2
11,62 +	63	4311672901046383796549.P41	22.2*
2,239 +	66	32605142983704221670173899.P41	42.1
7,79−	66	913242407367610843676812931.P40	37.5
6,94 +	69	1029538544148223697293. 30585762365533687252981.P25	81.0
10,73 +	70	10826684964539959837294043117.P42	87.0*
2,272 +	74	335631827046798245410603730138717057.P38	255.0*
6,128 +	75	2339340566463317436161. 2983028405608735541756929.P29	315.0*
2,269 +	81	424255915796187428893811.P57	1265*

7. Parallel Implementation. The algorithm lends itself readily to parallel implementation and we have already done so. A central processor selects the value of D from Section 3, and passes it to a satellite processor. The satellite computes the polynomial roots, performs the sieving, and returns any factorization back to the central processor. Since all of the computations on the satellites are done independently of one another, and since the satellites need only communicate with the central machine, it is possible to obtain the maximum theoretical utilization of all the processors. In our implementation we have achieved having N satellites give an N-fold speedup. A further advantage of having multiple satellites is that of a greatly enhanced real memory. The amount of swapping and paging is reduced to virtually nil, and we have far fewer cache misses.

8. Comparison With Other Methods.

A. *Continued Fraction Algorithm.* The CFRAC algorithm of Morrison and Brill-hart has been a champion among factoring algorithms until the QS implementation by Davis et al. It had been thought that the crossover point of QS with CFRAC was between 50 and 70 digits. Until now, no one has ever programmed both methods on the same machine. We have considerable experience with a VAX version of CFRAC that uses Pomerance's early abort strategy and we present a comparison of the two methods:
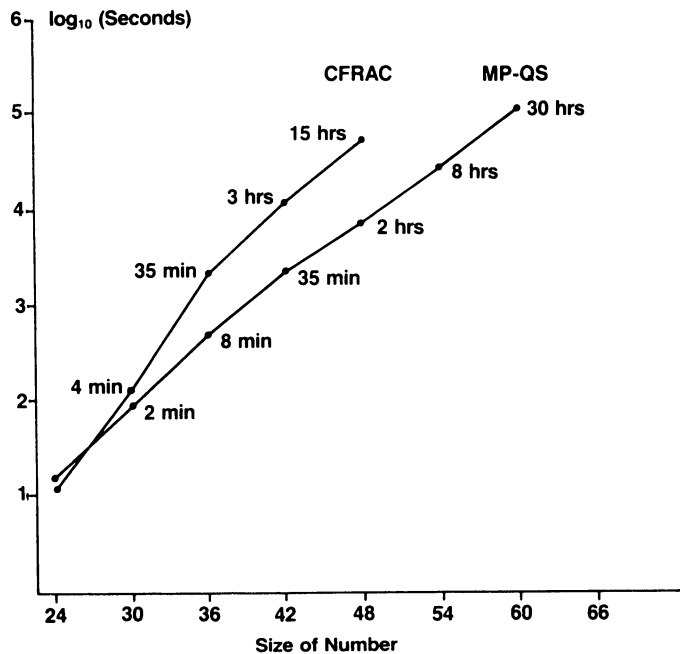
The multiple polynomial version of QS is significantly faster, unless the numbers are very small and those numbers take an insignificant time to factor anyway.

When one selects a value of M sufficiently large so that our algorithm only uses one polynomial, the run time increases dramatically. A crossover point with CFRAC appears around 40 digits.

B. *Single Polynomial* QS. We present here data, taken from [2], along with our own data, which shows the total number of residues examined by the variations of QS. These data represent typical values for the total number of residues sieved by both our method and the single polynomial and 'special q' versions of Davis et al. The values given are typical for numbers of a given size, but the columns do not represent factorizations of the same numbers.

TABLE 3

Size	MP-QS/VAX	SPECIAL-Q/CRAY	BASIC-QS/CRAY
52	8.0E + 8	1.0E + 9	9.0E + 9
53	4.0E + 8	—	2.0E + 10
55	5.0E + 8	—	1.4E + 10
58	1.0E + 9	4.3E + 9	2.7E + 10
60	2.1E + 9	2.2E + 10	9.0E + 10
63	1.0E + 9	1.2E + 10	—

One can clearly see a dramatic improvement over the basic algorithm in both the 'special q' and our versions. Our version performed somewhat better than the 'special q' version for several reasons:

(i) We used multipliers while Sandia did not.

(ii) We used the large prime variation while Sandia did not.

(iii) We changed polynomials as frequently as possible. Sandia could have obtained better performance by using more 'special q's'.

(iv) The pipeline architecture of the Cray makes sieving extremely efficient. Sieving on the Cray is relatively much faster than the VAX, even considering the average difference in machine speeds. Thus, it is profitable to do more sieving on each polynomial.

(v) Our polynomials generated residues which were smaller.

These results show that it is desirable to change polynomials as frequently as possible. The exact amount of time one should spend doing this, relative to doing the sieving, will be machine-dependent and may require experimentation. By changing polynomials frequently one gains at least 10 digits over the basic algorithm, and the new polynomials we have presented gain about 1 additional digit.

The MITRE Corporation
Burlington Road
Bedford, Massachusetts 01730

1. J. BRILLHART, D. H. LEHMER, J. L. SELFRIDGE, B. 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.

2. J. DAVIS & D. HOLDRIDGE, *Factorization Using the Quadratic Sieve*, Sandia Report #SAND 83-1346, 1983.

3. J. DAVIS & D. HOLDRIDGE, "Status report on factoring," *Advances in Cryptology* (T. Beth, N. Cot, and I. Ingemarrson, eds.), Lecture Notes in Comput. Sci., vol. 209, Springer-Verlag, Berlin and New York, 1985, pp. 183–215.

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

5. D. KNUTH, *The Art of Computer Programming*, Vol. 2, *Semi-Numerical Algorithms*, Addison-Wesley, Reading, Mass., 1969.

6. P. MONTGOMERY, personal communication.

7. M. MORRISON & J. BRILLHART, "A method of factoring and the factorization of $F7$," *Math. Comp.*, v. 29, 1975, pp. 183–205.

8. C. POMERANCE, personal communication.

9. C. POMERANCE, "The quadratic sieve factoring algorithm," *Advances in Cryptology* (T. Beth, N. Cot, and I. Ingemarrson, eds.), Lecture Notes in Comput. Sci., vol. 209, Springer-Verlag, Berlin and New York, 1985, pp. 169–182.

10. C. POMERANCE, "Analysis and comparison of some integer factoring algorithms," in *Computational Methods in Number Theory* (H. Lenstra and R. Tijdeman, eds.), 1982, pp. 89–141.

11. C. POMERANCE & S. S. WAGSTAFF, JR., "Implementation of the continued fraction integer algorithm," *Congress. Numer.*, v. 37, 1983, pp. 99–118.