MATHEMATICS OF COMPUTATION, VOLUME 29, NUMBER 129
JANUARY 1975, PAGES 183–205

A Method of Factoring and the Factorization of F_7

By Michael A. Morrison* and John Brillhart

Dedicated to D. H. Lehmer on his 70th birthday

Abstract. The continued fraction method for factoring integers, which was introduced by D. H. Lehmer and R. E. Powers, is discussed along with its computer implementation. The power of the method is demonstrated by the factorization of the seventh Fermat number F_7 and other large numbers of interest.

"Quand on a à étudier un grand nombre, il faut commencer par en déterminer quelques résidus quadratiques."

M. Kraitchik

1. Introduction. The continued fraction method discussed in this paper was introduced in 1931 by D. H. Lehmer and R. E. Powers [11]. At that time, and for several decades afterwards, this method was considered by hand computers to be of little practical value because of its fallibility and so was not used. This judgment was based on the discouraging and exceedingly frustrating experience of computing for hours on a desk calculator only to find, time after time, that every combination of numbers produced, failed to factor your number (" . . . your butterfly net was empty.").

With the advent of electronic computers the practical basis for this negative judgment disappeared, since the calculations and the inhibiting, complicated data handling could now be done swiftly and automatically. Thus *several failures in a row were of no particular importance,* as long as they were followed by at least one success. That the situation had in fact changed was not recognized, however, until 1965, when the second author suggested privately that this method (even with its many failures) might well be powerful enough to factor the seventh Fermat number—a number of 39 digits which had previously withstood many factorization attempts.

In 1967 this suggestion and details of the method along with its computer implementation came to the attention of D. Knuth, who, after communicating with D. H. Lehmer and the second author, included an account of it in the second volume of his excellent series, *The Art of Computer Programming* [4]. Although it is there attributed to Legendre, this is not entirely correct, as will be shown in Section 6.

In the summer of 1970 the authors decided to use the IBM 360/91 at the UCLA Campus Computing Network to attempt the factorization of F_7 by the continued

Received July 13, 1974.

AMS (MOS) subject classifications (1970). Primary 10–04, 10A25, 10A40; Secondary 10F20.

Key words and phrases. Factorization of integers, Fermat numbers, continued fraction method.

* This research was supported in part by a National Science Foundation Graduate Traineeship.

fraction method. At that time the method had never been programmed, and there was still skepticism being expressed that it would work, especially on a number as large as F_7. It was felt by the authors, however, that the accumulation of data in the method would eventually overwhelm the number being factored, even though there might be initial failures.

After a full summer of developing the method, programming and testing, and production runs, the factorization of F_7 was obtained on the morning of September 13, 1970.

2. **The Method.** Let $N > 1$ be an odd, composite integer. In rough outline the method is then the following:

Step A. Expand $\sqrt{N}$, or $\sqrt{kN}$ for some suitably chosen integer $k \geqslant 1$, into a simple continued fraction

$$\sqrt{kN} = [q_0, q_1, \cdots, q_{n-1}, (\sqrt{kN} + P_n)/Q_n]$$

to some point $n = n_0$. For each value of n, $1 \leqslant n \leqslant n_0$, the familiar identity

$$(1) \qquad A_{n-1}^2 - kNB_{n-1}^2 = (-1)^n Q_n,$$

where A_n/B_n is the nth convergent, implies the congruence

$$(2) \qquad A_{n-1}^2 \equiv (-1)^n Q_n \pmod{N}.$$

We shall speak of the pair of positive integers (A_{n-1}, Q_n) in this congruence as an "$A - Q$ pair".

Remark 2.1. The value of n_0 is initially large enough to produce the number of $A - Q$ pairs estimated to be sufficient for the method to succeed.

Step B. Find among the set of $A - Q$ pairs generated in Step A certain subsets, called "S-sets", each having the property that the signed product $\Pi_i(-1)^i Q_i$ of its Q_i's is a square. If no S-set can be found, return to Step A to expand $\sqrt{kN}$ further.

Step C. Each S-set found in Step B gives rise to the congruence

$$(3) \qquad A^2 \equiv \prod_i A_{i-1}^2 \equiv \prod_i (-1)^i Q_i = Q^2 \pmod{N},$$

where $1 \leqslant A < N$. Compute the A and Q of (3) and the GCD$(A - Q, N) = D$ for the S-sets produced in Step B. If $1 < D < N$ for some S-set, the method succeeds and D is a nontrivial factor of N. Otherwise, return to Step A.

Remark 2.2. Observe that Q^2 in (3) is not reduced (mod N).

3. **The Method in Detail.** In this section, Steps A, B, and C outlined above will be explained in enough detail to enable one to write a factoring program using this method. The majority of ideas concerned with writing a fast, efficient program will be presented in Sections 4 and 5.

Step A. Expand $\sqrt{kN}$ into a simple continued fraction by the following algorithm (note Example 3.1):

(i) Set $A_{-2} = 0, A_{-1} = 1, Q_{-1} = kN, r_{-1} = g, P_0 = 0, Q_0 = 1$, and $g = [\sqrt{kN}]$, where the bracket indicates the greatest integer.

(ii) Use (4) below to generate q_n and r_n for $n \geqslant 0$.

(iii) Use (5) to compute A_n (mod N) for $n \geqslant 0$. (Note that it is not necessary to compute B_n in this algorithm.)

(iv) Use (6) to generate $g + P_{n+1}$ for $n \geqslant 0$.

(v) Use (7) to produce Q_{n+1} for $n \geqslant 0$. (For hand computation see Remark 3.7.)

(vi) Increase n by 1 and return to (ii).

$$g + P_n = q_n Q_n + r_n, \text{ where } 0 \leqslant r_n < Q_n, \tag{4}$$

$$A_n \equiv q_n A_{n-1} + A_{n-2} \pmod{N}, \tag{5}$$

$$g + P_{n+1} = 2g - r_n, \tag{6}$$

$$Q_{n+1} = Q_{n-1} + q_n(r_n - r_{n-1}). \tag{7}$$

Example 3.1. Let $N = 13290059$ and $k = 1$. (See [11, p. 773].) Then $g = 3645$. The following table contains selected results from the expansion of $\sqrt{kN}$ up to $n = 52$:

TABLE 1

n	$g + P_n$	Q_n	q_n	r_n	A_{n-1} (mod N)	Q_n factored
-1	---	13290059	---	3645	0	---
0	3645	1	3645	0	1	---
1	7290	4034	1	3256	3645	$2 \cdot 2017$
2	4034	3257	1	777	3646	3257
3	6513	1555	4	293	7291	$5 \cdot 311$
4	6997	1321	5	392	32810	1321
5	6898	2050	3	748	171341	$2 \cdot 5^2 \cdot 41$
10	6318	1333	4	986	6700527	$31 \cdot 43$
22	4779	4633	1	146	5235158	$41 \cdot 113$
23	7144	226	31	138	1914221	$2 \cdot 113$
26	5622	3286	1	2336	11455708	$2 \cdot 31 \cdot 53$
31	6248	5650	1	598	1895246	$2 \cdot 5^2 \cdot 113$
40	6576	4558	1	2018	3213960	$2 \cdot 43 \cdot 53$
52	7273	25	290	23	2467124	5^2

Remarks. 3.1. By definition $q_n = [(\sqrt{kN} + P_n)/Q_n]$, which is easily seen to be identical to $[(g + P_n)/Q_n]$, where the bracket indicates the greatest integer. This suggests that the algorithm for the continued fraction expansion be arranged so that the binomial $g + P_n$ is used instead of P_n.

3.2. The integers P_n and Q_n always lie within the following bounds: $0 \leqslant P_n < \sqrt{kN}$ and $0 < Q_n < 2\sqrt{kN}$ for $n \geqslant 0$.

3.3. The fact that Q_n satisfies $0 < Q_n < 2\sqrt{kN}$ can be used as a running check on the arithmetic of the algorithm, since an error will most likely cause Q_n to eventually fall outside these bounds.

3.4. One method of calculating g is the following modification of the Newton-Raphson recursion: With an initial estimate $x_0 > \sqrt{kN}$ (which can be calculated using the square root of the leading part of kN), successively compute $x_{n+1} = [(x_n^2 + kN)/2x_n]$ for $n \geqslant 0$, where the bracket indicates the greatest integer. When $x_{n+1} - x_n \geqslant 0$, then $g = x_{n+1}$.

3.5. The continued fraction expansion of $\sqrt{kN}$ is always periodic, because of the bounds on P_n and Q_n. In those cases where the period of $\sqrt{N}$ is too short for the method to succeed, it is necessary to expand $\sqrt{kN}$ for some $k > 1$. For example, the Fermat numbers $F_m = 2^{2^m} + 1$, $m \geqslant 1$, require such a multiplier, since $F_m = [g, \overline{2g}]$, where $g = 2^{2^{m-1}}$. More will be said about multipliers in Remarks 4.5, 4.7, and 5.3.

3.6. Observe that the congruences (2), (3), and (5), as well as the computations in Step C, involve only N, not kN, even when a multiplier $k > 1$ is being used. Also observe that Q_n is already reduced (mod N), since k is always small in comparison with N and thus $0 < Q_n < 2\sqrt{kN} < N$.

3.7. Although formula (8) below requires a division and is thus not as good as (7) for rapid, automatic calculation, it does make hand computation more reliable, since the division must be exact.

$$Q_{n+1} = (kN - P_{n+1}^2)/Q_n \quad \text{for } n \geqslant 0. \tag{8}$$

3.8. It may be possible to factor N directly, if Q_n is a square and n is even. For then (1) can be written as $kNB_{n-1}^2 \mp A_{n-1}^2 - (\sqrt{Q_n})^2$, and the $\mathrm{GCD}(A_{n-1} - \sqrt{Q_n}, N)$ may yield a factor of N. A special case of this is when $Q_n = 1$, which occurs only at the end of a period. (For most numbers the period length of the expansion of $\sqrt{kN}$ is approximately $\sqrt{kN}$.)

Example 3.2. In the expansion of $\sqrt{13290059}$ shown in Table 1, $Q_{52} = 25$ and the $\mathrm{GCD}(A_{51} - \sqrt{Q_{52}}, N) = \mathrm{GCD}(2467119, 13290059) = 4261$.

Example 3.3. Let $N = 209$ and $k = 1$. In the expansion of $\sqrt{209}$, $A_7 = 153$ and $Q_8 = 1$. Thus $153^2 \equiv 1 \pmod{209}$, which yields the factorization $209 = 11 \cdot 19$.

Step B. This phase of the method is twofold: namely, determine if any S-sets exist in the set of $A - Q$ pairs generated in Step A and find some of them when they do. As it happens, a simple procedure can be devised which will solve both of these problems simultaneously. It requires, however, that the Q_n's involved have been completely factored.

For the present we set aside the question of factoring the Q_n's (this is dealt with in Section 4), only mentioning here that not every Q_n generated in Step A is completely factored, since the present method works much more rapidly if the Q_n's with large prime divisors are not used.

Suppose, then, that we have a set of $A - Q$ pairs in which each Q_n has been completely factored. Let F be the set of these Q_n's and let f be the cardinality of F. It is clear that when multiplying Q_n's from F to form a square, those primes which divide *some* Q_n to an odd power ("odd-power" primes) must be given special consideration. To do this efficiently, we first introduce binary "exponent" vectors and devise a procedure for working with them. To record our work, each exponent vector is assigned a companion "history" vector.

Let the Q_n's in F be given a definite ordering. Let the odd-power primes dividing the members of F also be given a definite ordering, say, $p_1, p_2, \cdots, p_r$ (this is usually derived from the ordering of F). With the ith element of F (say it is Q_n) associate the *signed* "exponent" vector $e_i = (\alpha_0, \alpha_1, \cdots, \alpha_r)$, where

$$\alpha_0 = \begin{cases} 1, & \text{if } n \text{ is odd,} \\ 0, & \text{otherwise,} \end{cases}$$

and for $1 \leqslant j \leqslant r$,

$$\alpha_j = \begin{cases} 1, & \text{if } p_j \text{ divides } Q_n \text{ to an odd power,} \\ 0, & \text{otherwise.} \end{cases}$$

Note that the sign bit α_0 corresponds to the sign $(-1)^n$ in Eq. (2) and is found from the subscript n of Q_n and not from the index i of the ordering of F.

For each e_i, the companion "history" vector is $h_i = (\beta_1, \beta_2, \cdots, \beta_f)$, where for $1 \leqslant m \leqslant f$

$$\beta_m = \begin{cases} 1, & \text{if } m = i, \\ 0, & \text{otherwise.} \end{cases}$$

Example 3.4. Using the data of Table 1, let $F = \{Q_3 = 5 \cdot 311, Q_5 = 2 \cdot 5^2 \cdot 41, Q_{22} = 41 \cdot 113\}$ and let the elements of F be ordered as listed. Then $f = 3$ and $r = 5$. Let $p_1 = 5, p_2 = 311, p_3 = 2, p_4 = 41$, and $p_5 = 113$. The exponent and history vectors are then:

$$\begin{aligned} e_1 &= (1, 1, 1, 0, 0, 0) \quad \text{and} \quad h_1 = (1, 0, 0), \\ e_2 &= (1, 0, 0, 1, 1, 0) \quad \text{and} \quad h_2 = (0, 1, 0), \\ e_3 &= (0, 0, 0, 0, 1, 1) \quad \text{and} \quad h_3 = (0, 0, 1). \end{aligned}$$

Note in e_2 that $\alpha_0 = 1$, since the sign is $(-1)^5$, and $\alpha_1 = 0$, since $p_1 = 5$ divides Q_5 to an even power.

Given these associations, it is obvious that a signed exponent vector can also be associated in the same way with the product of two Q_n's from F, and that this vector will merely be the vector sum of the exponent vectors associated with these Q_n's, the sum being computed in the $r+1$ dimensional vector space Z_2^{r+1} over Z_2, the integers (mod 2). Furthermore, that these particular Q_n's *were* multiplied can be "recorded" by also adding their two companion history vectors in the vector space Z_2^f.

Example 3.5. Using the vectors of Example 3.4, it is clear that $(1, 0, 0, 1, 0, 1) = (1, 0, 0, 1, 1, 0) + (0, 0, 0, 0, 1, 1) = e_2 + e_3$ represents the square-free part of $(-Q_5)(Q_{22}) = -5^2 \cdot 2 \cdot 41^2 \cdot 113$. (Note the order.) The history vector associated with this product is $(0, 1, 1) = (0, 1, 0) + (0, 0, 1) = h_2 + h_3$, the sum being computed in Z_2^3.

Suppose now that F contains all the Q_n's belonging to some S-set. Then the set of exponent vectors associated with F contains a subset whose sum is the *zero* vector, since this is the vector associated with a (positive) square. Thus the existence of an S-set among the $A - Q$ pairs under consideration is equivalent to the set of exponent vectors being linearly dependent in Z_2^{r+1}.

The following reduction procedure, which is the forward part of Gaussian elimination (carried out from right to left), will determine whether the set of exponent vectors is linearly dependent in Z_2^{r+1}. Note that the effect of step (iii) (b) is to record which vectors have been combined.

In describing this procedure, the phrase "rightmost 1" will refer to the 1 farthest to the right in an exponent (not history) vector. For example, the rightmost 1 in $e = (1, 0, 0, 1, 0, \underline{1}, 0, 0)$ has been underlined. The components of the exponent vectors are numbered 0 to r from left to right.

Reduction Procedure

(i) Set $j = r$.

(ii) Find the "pivot" vector e_i of smallest subscript whose rightmost 1 is in the jth component. If none exists, go to (iv).

(iii) (a) Replace every vector e_m, $i < m \leqslant f$, whose rightmost 1 is in the jth component, by the sum $e_i + e_m$, computed in Z_2^{r+1}.

(b) Whenever e_m is replaced by $e_i + e_m$, replace h_m by $h_i + h_m$, computed in Z_2^f.

(iv) Reduce j by 1. If $j \geqslant 0$, return to (ii). Otherwise, stop.

If upon the completion of the above procedure some vector, say e_s, is zero, then an S-set exists. For each such S-set, we say that an S-congruence, $A^2 \equiv Q^2 \pmod{N}$, is produced. The actual $A - Q$ pairs involved are easily determined from the history vector h_s.

Example 3.6. For hand computation, each exponent vector and its companion history vector may be placed side by side to form a row of an $f \times (r + 1 + f)$ matrix. Using the information from Table 1, let $F = \{Q_5, Q_{10}, Q_{22}, Q_{23}, Q_{26}, Q_{31}, Q_{40}\}$. Suppose F has been ordered as listed, and let the order of the primes be as below. (Note a column for 5 is not used.) Then the initial matrix would be:

	Sign	2	41	31	43	113	53	1	2	3	4	5	6	7
e_1	1	1	1	0	0	0	0	1	0	0	0	0	0	0
e_2	0	0	0	1	1	0	0	0	1	0	0	0	0	0
e_3	0	0	1	0	0	1	0	0	0	1	0	0	0	0
e_4	1	1	0	0	0	1	0	0	0	0	1	0	0	0
e_5	0	1	0	1	0	0	1	0	0	0	0	1	0	0
e_6	1	1	0	0	0	1	0	0	0	0	0	0	1	0
e_7	0	1	0	0	1	0	1	0	0	0	0	0	0	1

Reducing the above matrix in the manner described earlier yields:

	Sign	2	41	31	43	113	53	1	2	3	4	5	6	7
	1	1	1	0	0	0	0	1	0	0	0	0	0	0
	0	0	0	1	1	0	0	0	1	0	0	0	0	0
	0	0	1	0	0	1	0	0	0	1	0	0	0	0
**	0	0	0	0	0	0	0	1	0	1	1	0	0	0
	0	1	0	1	0	0	1	0	0	0	0	1	0	0
**	0	0	0	0	0	0	0	1	0	1	0	0	1	0
**	0	0	0	0	0	0	0	0	1	0	0	1	0	1

The three starred rows in the reduced matrix represent S-congruences. The A's and Q's of these congruences will be computed in Step C below.

Remarks. 3.9. Care must be taken that only those vectors (rows) are combined whose rightmost 1's are in the component (column) being examined. Thus, for example, in the reduced matrix it is wrong to combine rows 1 and 3 (assuming that the third column—that under 41—is being processed).

3.10. For reasons of speed, which will be discussed further in Remark 5.11, the procedure for processing the exponent vectors was carried out from right to left, rather than the more customary left to right.

3.11. In a binary computer, vector addition (mod 2) is equivalent to the operation "exclusive-or".

3.12. Sometimes the form of N provides an "$A - Q$ pair" which can be input to the program. For example, if $N = F_m$ is a Fermat number, then $(2^{2^{m-1}})^2 \equiv -1 \pmod N$. Or, if N divides the Fibonacci number U_{2n+1}, then the identity $U_{2n+1} = U_{n+1}^2 + U_n^2$ yields $U_{n+1}^2 \equiv -U_n^2 \pmod N$.

Step C. Since this step is directed toward the calculation of the GCD$(A - Q, N)$, it is sufficient to know both A and Q (mod N).

By virtue of its definition in (3), $A \pmod N$ may be computed by simply forming the product of the appropriate A_i's, reducing $\pmod N$ after each multiplication.

The value of $Q \pmod N$ may, of course, be found *directly* by first computing the product Q^2, taking a square root, and then reducing the result $\pmod N$. (Note that the reduction cannot be done before the square root is taken.) This direct approach, however, makes use of modular arithmetic only once—the final reduction. It also requires that the square root of an extremely large number be calculated, which is a time-consuming process even on a fast computer.

In contrast, the indirect approach outlined below makes full use of modular reduction, takes advantage of the "overlap" of the Q_i's, and quickly produces $\bar{Q}$, the least positive remainder of $Q \pmod N$. For convenience, let the Q_i of the particular S-set be renumbered $Q_1, Q_2, \cdots, Q_s$, $s \geqslant 2$. The letters $I, \bar{Q}, R$, and X represent variables, while the arrow indicates replacement.

Square Root Procedure

```
(i) $2 \longrightarrow I,\ 1 \longrightarrow \bar{Q},\ Q_1 \longrightarrow R$
(ii) $\mathrm{GCD}(R, Q_I) \longrightarrow X$
(iii) $X\bar{Q} \pmod N \longrightarrow \bar{Q}$
(iv) $(R/X)(Q_I/X) \longrightarrow R$
(v) $I + 1 \longrightarrow I$
(vi) IF $I \leqslant s$ GO TO (ii)
(vii) $\sqrt{R} \longrightarrow X$
(viii) $X\bar{Q} \pmod N \longrightarrow \bar{Q}$
```

The value of R in step (vii) above is relatively small. For this reason ordinary methods will quickly produce the square root required (see Remark 3.4).

The actual GCD calculations in this part are straightforward and present no difficulty. On a binary computer they can even be performed without division, as noted in Knuth [4, p. 297].

Example 3.7. Using the history vector in row 7 of the reduced matrix in Example 3.6, we have the following S-congruence:

$$(6700527 \cdot 11455708 \cdot 3213960)^2 \equiv (2 \cdot 31 \cdot 43 \cdot 53)^2 \pmod N$$

or $141298^2 \equiv 141298^2 \pmod N$. This represents one of two types of failures which can occur.

Using the history vector in row 6, we have

$$(171341 \cdot 5235158 \cdot 1895246)^2 \equiv (2 \cdot 5^2 \cdot 41 \cdot 113)^2 \pmod N$$

or $13058409^2 \equiv 231650^2 \pmod N$. But the $\mathrm{GCD}(12826759, 13290059) = 1$ and the method fails.

Using the history vector in row 4, we have

$$(171341 \cdot 5235158 \cdot 1914221)^2 \equiv (2 \cdot 5 \cdot 41 \cdot 113)^2 \pmod N$$

or $1469504^2 \equiv 46330^2 \pmod N$. This time the $\mathrm{GCD}(1423174, 13290059) = 4261$ and $N = 3119 \cdot 4261$.

Remarks. 3.13. It should be mentioned that multiplying two S-congruences, each of which has failed to factor N, will produce another S-congruence which will also fail to factor N.

3.14. Although not evident from Example 3.7, it seems that fewer failures are encountered if those S-congruences corresponding to zero vectors of largest subscript are tested first. This is equivalent in the matrix formulation to trying those at the "bottom" of the matrix first.

4. Factoring Q_n.

As was mentioned earlier, it is faster to ignore Q_n's containing large prime divisors than it is to completely factor every Q_n generated in Step A. This is not really surprising, since the true worth of any Q_n is based on whether or not we can *find* an S-set to which it belongs, and when a large prime divisor p is involved, there is little chance it will appear to an even power. Thus we must discover at least two Q_n's having p as a divisor before there is any possibility of finding S-sets containing such Q_n's. However, it is unlikely that the continued fraction algorithm will produce two such numbers in a reasonable amount of time.

Having made an *a priori* decision, then, as to when a prime shall be considered "too large", we proceed by attempting to factor the Q_n's using only primes less than this predetermined value. In our original program, written to factor F_7, we adopted this simple strategy, using in Step B only those Q_n's which completely factored over the given set of primes, called the "factor base".

The following theorem is of great practical importance, since it enables one to exclude about one half of the primes which might otherwise be included in the factor base.

THEOREM. *If in the continued fraction expansion of* $\sqrt{kN}$ *an odd prime* p *divides* Q_n, $n \geqslant 1$, *then the value of the Legendre symbol* (kN/p) *is* 0 *or* 1.

Proof. Suppose $n \geqslant 1$ and $p \mid Q_n$. Then Eq. (1) implies that $A_{n-1}^2 \equiv kNB_{n-1}^2 \pmod p$. But p cannot divide B_{n-1}, since it is known that $\mathrm{GCD}(A_{n-1}, B_{n-1}) = 1$. Thus $(A_{n-1}/B_{n-1})^2 \equiv kN \pmod p$ and kN is a quadratic residue of p. Q.E.D.

The factor base can now be chosen by selecting a certain number of the smallest possible odd primes p for which $(kN/p) = 0$ or 1. In addition, the prime 2 is always included in the factor base. (In selecting these primes, one should, of course, check that no p divides N.)

A refinement of the factor base approach, which effectively cuts the total running time by almost one half, has been used in later versions of our programs. It is based on the fact that after *discovering* the second largest prime divisor of a Q_n, the factorization is essentially completed. It is possible to identify the second largest prime divisor whenever, after having removed all prime divisors of Q_n which belong to the factor base, the remaining cofactor is less than p_λ^2 (where p_λ denotes the largest prime in the factor base).

Since p_λ^2 is quite large (even for p_λ as small as, say, 503), it becomes necessary to introduce an "upper bound" (UB) so that essentially worthless factorizations (those with large prime divisors) can be recognized and ignored as before. Thus in the refined approach, a Q_n is passed to Step B only if either (1) it completely factors over the factor base, or (2) all of its prime factors, except the largest, lie in the factor base, and the largest is less than UB.

The advantage of this modification is that a much smaller factor base can be used and thus the set of factored Q_n's can be produced with considerably less dividing (see Remark 7.2). Regardless of which of these factor base techniques is used, when a "reasonable" number of the Q_n's have been factored, the $A - Q$ pairs obtained are processed in the manner described in Steps B and C.

Remarks. 4.1. Determining the optimal values for the number FB of primes in the factor base and the upper bound UB seems mainly to be a matter of experience. Currently, we are using the values listed in Table 2.

TABLE 2

Number of digits in N	FB	UB
$\leqslant 20$	60	3000
21–23	150	10000
24, 25	200	14400
26–28	300	22500
29, 30	400	29000
31, 32	450	36000
33, 34	500	36000
35, 36	550	36000
37, 38	600	44000
39, 40	650	53000
41–46	700–1000	63000

4.2. The factoring of the Q_n's is time-consuming, requiring better than 90% of the total running time for most numbers. A slight increase in speed may be obtained by discarding those Q_n's which still remain larger than some predetermined value (such as 10^{15}), after a certain number of the primes in the factor base have been tested (say one half).

4.3. The Legendre symbol is evaluated as usual by the quadratic reciprocity law and the formula $(2/p) = (-1)^{(p^2-1)/8}$. On a binary computer the symbol's evaluation can be carried out rapidly in a way similar to the binary GCD method in Knuth [4, p. 297].

4.4. It appears from experience that most of the primes in the factor base do divide some Q_n. Thus, it seems unlikely that there are other conditions which could be used to reduce the factor base further. (Note that the primes dividing k should be included in the factor base. For example, in the expansion of $\sqrt{257F_7}$, the prime 257 divides $Q_{8018} = 2^4 \cdot 3 \cdot 7 \cdot 43 \cdot 257 \cdot 503 \cdot 4733 \cdot 5303 \cdot 9431$ as well as many other Q_n.)

4.5. A multiplier k may be chosen in such a way that many of the small primes lie in the factor base. This seems to be advantageous, even though in doing so k may have to be a two or three digit number. For larger k, the advantage of having numerous small primes in the factor base must be balanced against the resulting increase in the size of the Q_n's. (See Remark 5.3.)

4.6. In the interest of maximum output, several inconclusive experiments have been conducted in which only certain Q_n's were selected for factoring. Such strategies have included considering only Q_n's which were smaller than a fixed amount, say $\sqrt{kN}/10^3$; or Q_n's which were divisible by 24 or 30; or, as suggested privately by R. Schroeppel, only Q_n's for which q_n exceeds a fixed value (as high as 300 for large N). There is considerable need for further experimenting here.

4.7. If several k's are used for the same N, the complete set of $A - Q$ pairs obtained can still be processed in Step B. (See Remark 3.6.) In general, of course, a single value of k should be used, since otherwise more factored Q_n's would be required to produce an S-set.

5. **Program Details.** It was decided early in our work that two programs should be written in order to have an economical set-up which would run easily in the time-sharing system at UCLA. The first program, RESIDUE, would generate the $A - Q$ pairs and factor the Q_n's, while the second program, ANSWER, would process the resulting information.

The alternative was a single program which would factor a Q_n and then process the $A - Q$ pair immediately. Such a program would continue to require more memory space the longer it ran, thereby proving to be both expensive and difficult to operate.

The following comments give a description of each program's capabilities as well as a more technical discussion of various time-saving ideas. (The major input parameters are also given.) It should be pointed out that both RESIDUE and ANSWER are PL/1 programs using machine-language subroutines for multi-precise arithmetic computations, factoring the Q_n's, and vector manipulation.

RESIDUE. This program accepts as input:

- the number N to be factored ($\leqslant 46$ digits)
- integers G and H (if known) such that $G^2 \equiv -H^2 \pmod{N}$
- a multiplier k, $0 \leqslant k < 2^{31}$ (see Remark 5.3)
- the number (FB) of primes desired in the factor base
- an upper bound (UB) (see Section 4)
- the number (LIM) of factored Q_n's desired (see Remark 5.5)
- an upper limit (QL) on the subscript n (see Remark 5.6)
- restart values (when used) $n, A_{n-1}, Q_{n-1}, A_n, g + P_n, Q_n, q_n, r_n$ (see Remark 5.7).

In addition to its main function of generating $A - Q$ pairs whose Q_n's have been completely factored, RESIDUE prints both input and restart data, tests N to determine whether it is composite or pseudo-prime (see [1]), checks restart values, and attempts to factor N when it recognizes that some Q_n is a square.

Remarks. 5.1. When computing q_n, three subtractions of Q_n from $g + P_n$ were tried before division was resorted to. This was based upon the fact that approximately 41% of the partial quotients in a simple continued fraction expansion are 1, while about 17% are 2 and 9% are 3. (See [9, p. 122].) Since multi-precise division is significantly slower than subtraction, this approach produces the expansion more rapidly.

5.2. On the IBM 360/91 a fixed-point divide requires 36–37 cycles, while a (double-precision) floating-point divide takes at most twelve cycles. (One cycle equals sixty nanoseconds.) For this reason, floating-point arithmetic was used to factor the Q_n's. For each prime p in the factor base (the primes were stored in memory in floating format), it required only one floating divide to check whether p divided Q_n if $Q_n < 2^{55}$, and even though the remainder had to be computed, the overall result was a divisibility test performed in less than one half the time required by fixed-point operations. Notice that two fixed-point divides would have been necessary for $Q_n > 2^{31}$, with three divides needed for $Q_n > 2^{62}$. On the average the floating-point programming was capable of about 800,000 divisibility tests per second.

5.3. If $k = 0$ is input to the program, then RESIDUE chooses its own multiplier in the range $1 \leqslant k \leqslant 97$ according to a strategy slightly more complicated than the following: for each k in the range which allows either 3 or 5 to be in the factor base, determine the number of primes $p_i \leqslant 31$ such that the Legendre symbol $(kN/p_i) = 0$ or 1. Choose as the multiplier that k which allows the largest number of such primes. If several k's allow this maximal number, compute $\Sigma(1/p_i)$ for each, where the sum is over those primes in the factor base which are $\leqslant 31$. Pick the smallest k having the largest sum.

5.4. The recommended values for factor base size (FB) and the upper bound parameter (UB), which are listed in Table 2, represent several years experience and a considerable amount of experimentation. Nevertheless, they are only at best a compromise to cover a large range of numbers and seldom represent optimal values for a particular N.

5.5. When LIM $= 0$ is input to the program (the recommended procedure), RESIDUE terminates itself when the number of factored Q_n's exceeds the appropriate value of LIM in Table 3. This dynamic limit is recomputed each time a new Q_n is factored. Table 3 contains empirical formulas for predicting when sufficient information exists to factor N by means of an S-congruence. These formulas are designed to be used with the values of UB listed in Table 2. The results to date have been fairly satisfying. If, however, it happens that there is not sufficient data to factor N, then additional $A - Q$ pairs (with Q_n factored) are obtained, 50 or 100 at a time.

5.6. The purpose of the input parameter QL may not be readily apparent. It is

TABLE 3

Number of digits in N	Dynamic LIM
$\leqslant 30$	.80(FB + Y)
31–34	.82(FB + Y)
$\geqslant 35$	.84(FB + Y)

Y = current number of factored Q_n's with their *largest* prime divisors lying outside the factor base,

FB = number of primes in the factor base.

possible that, in the time allotted, RESIDUE might not be able to obtain the required number of factored Q_n's. In such a case, the operating system would terminate the program and no restart values would be printed, necessitating that the program be rerun if N cannot be factored with the data at hand. To avoid this, RESIDUE is designed to terminate (with restart data printed) whenever the subscript n exceeds QL. In practice, then, the value of QL is determined by the speed of the particular computer and the allotted running time.

5.7. Whenever restart values are entered, RESIDUE verifies them by the following four checks performed in sequence:

(i) Is $A_{n-1}^2 \equiv (-1)^n Q_n \pmod{N}$?

(ii) Does $Q_{n-1} = (kN - P_n^2)/Q_n$?

(iii) Does $g + P_n = q_n Q_n + r_n$?

(iv) Is $A_n^2 \equiv (-1)^{n+1} Q_{n+1} \pmod{N}$?

(To find Q_{n+1} use (7), after first computing g and $r_{n-1} (= g - P_n$ from (6)).)

5.8. The output from RESIDUE for each $A - Q$ pair (for which Q_n was factored) was designed to fit on two cards: the first contained n, A_{n-1}, and Q_n; the second contained n and the odd-power primes (up to fifteen in number) dividing Q_n.

ANSWER. This program accepts as input:

– the number N to be factored ($\leqslant 46$ digits)

– integers G and H (if known) such that $G^2 \equiv -H^2 \pmod{N}$

– the total number (QTOT) of $A - Q$ pairs to be input (Note: QTOT $= f$)

– an upper bound (PTOT) on the total number of distinct primes in the factorizations (usually FB + Y)

– the (card) data output by RESIDUE (see Remark 5.8).

In addition to deciding whether any S-congruences exist (and attempting to factor N if they do), ANSWER prints the input data (exclusive of the $A - Q$ pairs) and performs a pseudo-prime test on any discovered factors of N. In the event that there are composite factors of N, ANSWER continues to process any remaining S-congruences in an attempt to *completely* factor N.

Remarks. 5.9. ANSWER constructs six arrays in memory: two arrays of multiprecise numbers (one for the A_{n-1}'s and one for the Q_n's), two arrays of bit vectors (one for exponent vectors and one for their associated history vectors), a table of primes, and a table of pointers. All six arrays are constructed simultaneously as the $A - Q$ pairs and the factorizations of the Q_n's are input.

5.10. The table of primes mentioned in Remark 5.9 is constructed and used as follows: The first prime of the first factorization is placed in the first position of the prime table and the first bit of the first exponent vector is set to 1 (recall that the sign is placed in the zeroth bit). Subsequently, any prime p of a particular factorization is compared with the primes $p_1, p_2, \cdots, p_m$ already in the prime table. If p equals some p_j, then the jth bit of the corresponding exponent vector is set to 1. Otherwise, p becomes p_{m+1} and the $(m + 1)$st bit is set to 1. All the vector arrays are "zeroed out" initially.

5.11. The main reason the reduction procedure of Step B is performed from right to left on the exponent vectors is that there will be less combining of vectors than if the operation proceeded from left to right. This is a result of the construction of the prime table which tends to place the small primes in the early part of the table. They are thus represented by the left components of the exponent vectors, while the large primes tend to be represented on the right. Hence, vectors which may have small primes in common will be excluded from any mutual combining very quickly if their largest primes do not agree.

5.12. The pointer table mentioned in Remark 5.9 enables the procedure discussed in Step B to be done swiftly with only occasional scanning of the (rather sparse) exponent vectors. To each exponent vector there corresponds an entry in the pointer table–its pointer (see Remark 5.13). The value of this pointer indicates the vector component containing the rightmost 1. Two pointers agree if and only if their corresponding exponent vectors have their rightmost 1's in the same component.

In using the pointer table, a scan pointer is first established. Initially, this corresponds to component r. Beginning with the first pointer in the table, each entry in the pointer table is compared with the scan pointer. If a match does not exist, then no exponent vector has a rightmost 1 in that component. In such a case, the scan pointer is reduced so that it points to the next component to the left and the process is repeated until all components have been examined.

If, on the other hand, a match occurs, the first match establishes the "pivot" vector. This vector is exclusive-ored (component-wise addition in Z_2) *into* those exponent vectors corresponding to subsequent matches with the scan pointer. Thus, only this pivot vector will retain its rightmost 1 in the component being considered. When the pivot vector has been combined with another vector, it is necessary to locate the new rightmost 1 in the new vector and update its pointer. (It is during this operation that zero vectors are found.)

When no further matches with the scan pointer exist, it is reduced so that it points to the next component to the left, and the entire process is repeated until all components have been examined.

5.13. *Pointer design.* Assume the computer being programmed has a 32 bit (4 byte) word. Suppose each exponent vector begins on a full word boundary. Let this be the 0th word of the vector. Assuming the bits of each word are numbered 0 to 31 (left to right), it is possible to uniquely identify the rightmost 1 of any exponent vector in terms of its word number and its bit number; e.g., given the vector

$$\overset{\textit{Word } 0}{10010000000000010100000100000111} \qquad \overset{\textit{Word } 1}{0000010\overbrace{\cdots}^{26}0}.$$

The rightmost 1 has word number 1, bit number 5.

Let each entry in the pointer table occupy two consecutive bytes (or a full word if the machine lacks half-word capability). The left byte contains the word number, the right byte (in its five most significant bits) the bit number. For the vector above the pointer would be

Left byte	*Right byte*
00000001	00101000.

When constructing an exponent vector, each time you advance one component to the right, the addition of 8 to a register containing the pointer will correctly update it.

5.14. ANSWER, as presently written, requires large amounts of core as indicated by Table 4. However, as indicated in Table 7, it requires very little running time. RESIDUE, on the other hand, seldom needs more than 140K.

TABLE 4

Number of digits in N	Average core for ANSWER (in K)
20	150
21–25	220
26, 27	280
28–30	360
31, 32	440
33, 34	500
35, 36	750
37, 38	880

By sacrificing speed, ANSWER may be tailored for machines with limited core. It is not necessary, for example, to store the $A - Q$ pairs internally. They may be placed on disk or tape in such a way that it is possible to locate any desired pair rather simply. Also, it is not necessary that the array of history vectors (when considered as a matrix) be rectangular–lower triangular is sufficient.

The output of RESIDUE may be scanned before it is passed to ANSWER.

During such an intermediate step, a factorization is flagged if its largest prime is unmatched and lies outside the factor base. It will then be ignored by ANSWER. For most N, 25% or more of the factored $A - Q$ pairs can be discarded on any given run of ANSWER. Of course, the factorization of any Q_n, which is completed within the factor base, would not be flagged.

If a scan step is used, the values of UB in Table 2 can be increased in order to take fullest advantage of possible matches without increasing core requirements.

Finally, the exponent and history vectors may be stored in a compact format and fully expanded only when they are to be combined.

5.15. As an option, ANSWER also has the capability of verifying the congruence $A_{n-1}^2 \equiv (-1)^n Q_n \pmod{N}$ for each $A - Q$ pair input. To date this check has never caught the IBM 360 in error.

6. **Related Factoring Methods.** The factoring method discussed in the preceding sections is based on a combination of ideas due to Legendre and Kraitchik. It is the purpose of this section to consider these ideas and illustrate their relationship to the method at hand.

(a) Legendre [7] wrote Eq. (1) as $kNB_{n-1}^2 = A_{n-1}^2 - (-1)^n Q_n$. The right side of this equation can be written as $x^2 \pm ay^2$, where $x = A_{n-1}$ and $ay^2 = Q_n$, "a" being square-free. Thus, if p is a prime dividing N, it must have a linear form associated with divisors of $x^2 \pm ay^2$. For example, if kNB_{n-1}^2 can be expressed as $x^2 - 2y^2$, then p must have one of the forms $8m \pm 1$.

By combining enough linear forms Legendre built a sieve which excluded many of the possible divisors of N. A good enough sieve can be used to find a factor of N by merely trying (as possible divisors) those numbers which survive the sieve. When N is small, a sieve may even be able to establish primality by excluding all possible factors $< \sqrt{N}$.

The factoring method of Legendre can, therefore, be described as a direct search technique which uses a sieve to create a sequence of trial divisors. As such, it may fail to find a large prime factor of N.

In contrast, the method of this paper does not use a direct search, since no sequence of trial divisors is created. In fact, the real power of the method lies in its "indifference" to the relative size of the prime factors of N. It is thus probably not correct to refer to the method of this paper as that of Legendre, even though both depend on the continued fraction expansion of $\sqrt{kN}$ (cf. [4, p. 351]).

It is important to note, however, that Legendre's method and other sieving techniques are often quite effective in factoring rather large integers (see [1, p. 88]). For example, it was by this method that D. H. Lehmer, G. D. Johnson, and the second author factored $2^{101} - 1$ on the IBM 704 (see [4, p. 354]).

Many devices have been constructed to assist in making the use of sieves more automatic and reliable. The stencils of D. N. Lehmer and the Hollerith card version of J. D. Elder [13] are of great value in hand computation. (The booklet accompanying Lehmer's stencils and Elder's sieve cards contains an excellent resume of factoring methods.)

Over the last forty-five years, D. H. Lehmer and his associates have built various powerful machines to carry out the sieving process automatically, rapidly, and accurately (see [8], [10], [12]). A new shift-register sieve, SRS-181, capable of processing 20,000,000 values per second, is presently being built at Berkeley and is expected to be operational by the end of 1974.

(b) The factoring methods of Kraitchik [5] do not use continued fractions. Instead, he obtains quadratic residues of N by rather ad hoc methods in which the expressions $\lambda N - x^2$ or $N - \lambda x^2$ are completely factored for certain choices of λ and x. For example [5, p. 27], if $N = (10^{23} - 1)/9$ and $\lambda = 1$, then

$$N - 105408657045^2 = 2 \cdot 11^2 \cdot 13 \cdot 59^2 \cdot 71^2 \cdot 107 \cdot 131 \cdot 163$$

or

$$105408657045^2 \equiv - 2 \cdot 11^2 \cdot 13 \cdot 59^2 \cdot 71^2 \cdot 107 \cdot 131 \cdot 163 \pmod{N},$$

which implies $- 2 \cdot 13 \cdot 107 \cdot 131 \cdot 163$ is a quadratic residue of N.

The residues found in this way are then employed either to set up a sieve, as in Legendre's method, or to create "cycles" (Kraitchik's terminology), that is, to select certain congruences, $x_i^2 \equiv R_i \pmod{N}$, whose product will yield a square on the right side (possibly with some cancelling). For example [6, p. 201], if $N = 721 \cdot 2^{28} + 1 = 193541963777$, then he finds the congruences

$$439935^2 \equiv 2^8 \cdot 7^2 \cdot 67 \quad \text{and} \quad 1609^2 \cdot 7^2 \cdot 67 \equiv 449490^2 \pmod{N}.$$

Multiplying these and cancelling $7^2 \cdot 67$ gives $707855415^2 \equiv 7191840^2 \pmod{N}$. Thus the $\mathrm{GCD}(700663575, N) = 9342181$ and $N = 20717 \cdot 9342181$. (Readers of Kraitchik's works should beware of numerical errors.)

Remarks. 6.1. It should be pointed out that when cycles are used, it is not necessary to set up a sieve as in Legendre's method. This is a great advantage, since sieves demand considerable care in their construction and use.

Even though the use of cycles is a major part of the present method, it is not correct to attribute this method to Kraitchik, since he did not use continued fractions to obtain quadratic residues of N, as in (2).

6.2. Kraitchik uses the multiplier λ as we do to gain some control over which primes can divide $\lambda N - x^2$ (cf. Remarks 4.5 and 5.3).

6.3. When N is expressed as $x^2 - y^2$, a nontrivial representation infallibly gives a factorization of N. Unfortunately, this representation is usually discovered by sieving, and sieving, at present, does not compete with the method of this paper. At this time, the only known possible rival to the present *general* method is that due to Shanks [17]. However, Shanks' method has not yet been programmed in machine language, so an accurate comparison cannot be made.

7. **Numerical Results.** *Factoring* F_7. In 1905, Morehead [14] and Western [18] each proved that

$$F_7 = 2^{128} + 1 = 340282366920938463463374607431768211457$$

is composite. They used the well-known theorem of Proth [16] which states that $F_m = 2^{2^m} + 1$ is prime if and only if $3^{(F_m - 1)/2} \equiv -1 \pmod{F_m}$, $m \geqslant 1$.

In our attempt to factor F_7 it was first necessary to choose a multiplier $k > 1$, both to produce an expansion with a long period and to allow small primes to be in the factor base. The choice $k = 257$ was made only after some experimenting with other values, such as 17, 3617, 22697, and 1516609494. Each was compared with 257 on the basis of how many of the first 5000 Q_n's could be completely factored over a factor base of the first 2700 "acceptable" primes.

From the first 1,330,000 Q_n's of the expansion of $\sqrt{257F_7}$, 2059 complete factorizations were obtained. On the average, the program processed 250 Q_n's per second and yielded one completely factored Q_n about every three seconds. After the program was run for about ninety minutes over a period of seven weeks, the accumulated data was processed by ANSWER using 1504K bytes of memory. The first four S-congruences failed to factor F_7. The factorization of F_7 (see [15]), which is the first entry of Table 6, was found using the congruence:

$$\begin{aligned} &233503648380835852177232143618227956476^2 \\ &\qquad \equiv 25186478145728041297312271934852021 2223^2 \pmod{F_7}. \end{aligned}$$

Although in its current form the factoring program could now probably factor F_7 in about fifty minutes (using a small factor base and an upper bound), the prospects of using it to factor F_8, a number of seventy-eight digits, are not very bright, since the size of each Q_n would be about that of F_7.

Remarks. 7.1. In the expansion of $\sqrt{257F_7}$, the even Q_n's were automatically divisible by 8. This is a result of Eq. (1), which states that $A_{n-1}^2 - 257F_7B_{n-1}^2 = (-1)^n Q_n$, and the fact that the $\mathrm{GCD}(A_{n-1}, B_{n-1}) = 1$. For if Q_n is even, then both A_{n-1} and B_{n-1} must be odd. Thus, the equation taken (mod 8) implies that $8 \mid Q_n$.

7.2. Table 5 contains some statistics, derived from the expansion of $\sqrt{257F_7}$, which strikingly illustrate the increased rate at which factored Q_n's can be produced when a small factor base is used and the largest prime divisor of a factored Q_n is not required to be in the factor base. (Note that 52183 was the largest prime in any factored Q_n. See Section 4, Paragraph 2.)

Other Results. With the factorization of F_7 completed, the original programs, and later revisions, were used to factor other numbers of interest. These are mainly of two types:

(1) $a^n \pm 1$, or one of its composite, primitive factors,

(2) U_n or V_n, or one of their composite, primitive factors. Here U_n denotes the nth Fibonacci number and V_n denotes the nth term of the associated Lucas sequence (see Jarden [3]).

Forty-two factorizations (including F_7), which were completed by the method of this paper, are given in Table 6. In each case the factorization accomplished consisted of finding the two largest (nonalgebraic) prime factors.

TABLE 5

$\%Q_n$ indicates the percentage of factored Q_n (out of a total of 2059) whose 2nd largest prime divisor is less than the BOUND.

$\%P$ indicates the percentage of primes in the factor base (out of a total of 2700) less than the BOUND.

BOUND	$\%Q_n$	$\%P$
8000	43.90	17.78
9000	47.94	19.85
10000	52.45	22.11
11000	56.14	24.33
12000	59.64	26.00
13000	63.14	28.11
14000	66.39	30.07
15000	69.74	32.30
20000	80.91	42.33
25000	88.00	51.96
30000	93.35	60.59
40000	98.45	79.26
52183	100.00	100.00

The forms of the numbers in entries 4 and 10 of Table 6 arise from the Aurifeuillian factorizations:

$$6^{12n+6} + 1 = (6^{4n+2} + 1)(6^{4n+2} - 6^{3n+2} + 3 \cdot 6^{2n+1} - 6^{n+1} + 1) \cdot (6^{4n+2} + 6^{3n+2} + 3 \cdot 6^{2n+1} + 6^{n+1} + 1)$$

and

$$12^{6n+3} + 1 = (12^{2n+1} + 1)(12^{2n+1} - 6 \cdot 12^n + 1)(12^{2n+1} + 6 \cdot 12^n + 1).$$

In Table 6, any algebraic (see [1, p. 87]) factors are placed before the colon, while an asterisk following a factor indicates it was first discovered by either D. H. Lehmer, Emma Lehmer, and J. L. Selfridge, or by Bryant Tuckerman at the IBM Research Center, Yorktown Heights, New York. These factors are included here with their kind permission.

Remark 7.3. Although the most effective strategy for choosing a multiplier seems to be rather elusive, the following three examples clearly illustrate the importance of the multiplier k.

1. The composite thirty-one digit cofactor N of V_{273} (entry 34 of Table 6)

factored in about seventy seconds with a multiplier of $k = 1$. Here $(N/p) = 1$ for seventeen out of the twenty-four odd primes less than one hundred as shown below:

$$N = 18957795045078266679704795920 81.$$

The factor base included 3, 5, 7, 11, 13, 17, 19, 29, 31, 37, 41, 53, 59, 61, 71, 73, and 79.

TABLE 6

1. $2^{128} + 1 = 59649589127497217 \cdot 5704689200685129054721$

2. $5^{77} - 1 = 2^{2} \cdot 19531 \cdot 12207031 : 527093491^{*} \cdot 8090594434231 \cdot 162715052426691233701$

3. $5^{93} + 1 = 2 \cdot 3^{2} \cdot 7 \cdot 1303 \cdot 21207101 \cdot 28086211607 : 258065887^{*} \cdot 75005167927$
   $\cdot 533456714907222004663 69$

4. $6^{46} + 6^{35} + 3 \cdot 6^{23} + 6^{12} + 1 = 97 : 23027140435639321 \cdot 2792195192301416 41$

5. $7^{53} + 1 = 2^{3} : 107 \cdot 345449549^{*} \cdot 35416476134069 \cdot 58902316970027001503$

6. $7^{75} + 1 = 2^{3} \cdot 11 \cdot 43 \cdot 191 \cdot 6568801 \cdot 79787519018560501 : 151 \cdot 6005492312551$
   $\cdot 702137028919988 8801$

7. $11^{41} + 1 = 2^{2} \cdot 3 : 711628063^{*} \cdot 1216150172449 \cdot 47937852368006 0338823$

8. $11^{55} - 1 = 2 \cdot 5^{2} \cdot 3221 \cdot 15797 \cdot 1806113 : 25301 \cdot 39161 \cdot 643170158708221$
   $\cdot 645654335737185721$

9. $11^{60} + 1 = 2 \cdot 41 \cdot 7321 \cdot 10657 \cdot 20113 \cdot 1120648576818041 : 52020741601$
   $\cdot 405899996710177424529 61$

10. $12^{33} - 6 \cdot 12^{16} + 1 = 1657 : 5690162377645219 \cdot 4350447692666281 9$

11. $12^{37} + 1 = 13 : 5250079^{*} \cdot 4150805645839 \cdot 30023720899326796981$

12. $12^{38} + 1 = 5 \cdot 29 : 1977673^{*} \cdot 176477034940417 \cdot 2016864235215616489$

13. $12^{70} + 1 = 5^{2} \cdot 29 \cdot 673 \cdot 85403261 \cdot 13156924369 : 71874601^{*} \cdot 10365509281^{*}$
    $\cdot 1612092376073761 \cdot 529845566468895012 1$

14. $U_{173} = : 1639343785721 \cdot 389678749007629271532733$

15. $U_{197} = : 15761 \cdot 25795969 \cdot 227150265697 \cdot 717185107125886549$

16. $U_{229} = : 457 \cdot 2749 \cdot 40487201 \cdot 132605449901 \cdot 47831560297620361798553$

17. $U_{243} = 2 \cdot 17 \cdot 53 \cdot 109 \cdot 2269 \cdot 4373 \cdot 19441 : 448607550257 \cdot 1600041112430640307056 1$

18. $U_{249} = 2 \cdot 99194853094755497 : 1033043205255409 \cdot 238122152840097877 69$

19. $U_{279} = 2 \cdot 17 \cdot 557 \cdot 2417 \cdot 4531100550901 : 11717 \cdot 594960058508093 \cdot 6279830532252706321$

20. $U_{375} = 2 \cdot 5^{3} \cdot 61 \cdot 3001 \cdot 230686501 \cdot 15841416796404570 0001 : 9001 \cdot 169501$
    $\cdot 415101054555 01 \cdot 990629340694465350 1$

21. $V_{152} = 47 : 562766385967 \cdot 2206456200865197103$

22. $V_{169} = 521 : 596107814364089 \cdot 671040394220849329$

23. $V_{176} = 2207 : 1409 \cdot 6086461133983 \cdot 319702847642258783$

24. $V_{179} = : 359 \cdot 1066737847220321 \cdot 66932254279484647441$

25. $V_{181} = : 97379 \cdot 21373261504197751 \cdot 32242356485644069$

26. $V_{184} = 47 : 367 \cdot 37309023160481 \cdot 441720958100381917103$

27. $V_{191} = : 22921 \cdot 395586472506832921 \cdot 910257559954057439$

28. $V_{201} = 2^{2} \cdot 4021 \cdot 24994118449 : 2686039424221 \cdot 940094299967491$

29. $V_{209} = 199 \cdot 9349 : 419 \cdot 20669776469 \cdot 2959707364050967146316591$

*See Section 7.

30. $V_{246} = 2 \cdot 3^2 \cdot 163 \cdot 800483 \cdot 350207569 : 67031206681 \cdot 4672450542188230967 1121$

31. $V_{249} = 2^2 \cdot 221806434537978679 : 499 \cdot 430849126348 51 \cdot 572087591261946589$

32. $V_{250} = 3 \cdot 41 \cdot 401 \cdot 570601 : 1353439001 \cdot 5465167948001 \cdot 848175747705896380 01$

33. $V_{264} = 2 \cdot 47 \cdot 1103 \cdot 52337681992411201 : 893844775132847 \cdot 306871863078979598 3$

34. $V_{273} = 2^2 \cdot 29 \cdot 79 \cdot 211 \cdot 521 \cdot 859 \cdot 689667151970161 : 1836084445651 \cdot 1032512153239041931$

35. $V_{280} = 47 \cdot 1601 \cdot 3041 \cdot 10745088481 : 6135922241 \cdot 164154312001 \cdot 1326451946603465248 1$

36. $V_{290} = 3 \cdot 41 \cdot 347 \cdot 1270083883 : 5801 \cdot 52201 \cdot 96281 \cdot 6854280100961 \cdot 37296197227456649716 1$

37. $V_{291} = 2^2 \cdot 3299 \cdot 566785575021415 79 : 5496409 \cdot 320657355925861 \cdot 4959318126280687189$

38. $V_{294} = 2 \cdot 3^2 \cdot 83 \cdot 281 \cdot 1427 \cdot 5881 \cdot 61025309469041 : 587 \cdot 1150184101339307$
$\cdot 190773791763188929$

39. $V_{297} = 2^2 \cdot 19 \cdot 199 \cdot 991 \cdot 2179 \cdot 5779 \cdot 9901 \cdot 1513909 : 220862269 \cdot 1369471729429$
$\cdot 137096217949680001$

40. $V_{303} = 2^2 \cdot 809 \cdot 7879 \cdot 201062946718741 : 77569 \cdot 3334819 \cdot 42669355669$
$\cdot 3720204334901306428 9$

41. $V_{318} = 2 \cdot 3^2 \cdot 1483 \cdot 2969 \cdot 1076012367720403 : 14627 \cdot 346656889 \cdot 57157491464963$
$\cdot 116171668216510969$

42. $V_{342} = 2 \cdot 3^2 \cdot 227 \cdot 26449 \cdot 29134601 \cdot 212067587 : 683 \cdot 20521 \cdot 47881 \cdot 6368731219987307$
$\cdot 324968740886536921$

2. The Fibonacci number $U_{173} = 638817435613190341905763972389505493$ required more than 800 seconds to factor with $k = 1$ (see entry 14 in Table 6). A later test-run using the program-selected multiplier $k = 2$ showed that the number could have been factored in less than 200 seconds.

3. Using multipliers of comparable size, entry 27 of Table 6 required 1016 seconds to factor, while entry 29 (approximately the same size) needed only 365 seconds.

8. General Remarks. 8.1. The factor programs described in this paper no longer exist at UCLA. The latest versions closely approximate a single stage program in their operation and are now running at the Department of Mathematical Sciences, Northern Illinois University, DeKalb, Illinois. By means of JCL, control is passed back and forth between RESIDUE and ANSWER until in most cases N is factored. In their present forms these programs are suitable for general use at a computer center, especially if a reasonable limit on the size of N is established in order to avoid excessive use of both time and core. The power of this factoring package is evidenced in some part by the information in Table 7 (these figures are based on a comparatively small number of factorizations).

8.2. Any method which could consistently produce quadratic residues of N (see (2)) considerably smaller than $2\sqrt{N}$ would be of great interest, since the size of the residues effectively determines the practical limits for this approach.

8.3. For some reason that is not entirely clear, composite numbers with several prime divisors seem to factor much more quickly than those of comparable size with only two prime divisors. The fact that these extra prime divisors tend to produce factor bases containing primes slightly larger than normal does not seem to fully account for the phenomenon.

TABLE 7

Average Factorization Times (secs.)

Number of digits in N	IBM 360/65 RESIDUE	IBM 360/65 ANSWER	IBM 360/65 TOTAL	IBM 360/91 TOTAL
16	6.0	2.0	8.0	2.0
17–18	7.5	2.0	9.5	2.5
19–20	19.0	3.0	22.0	5.5
21–22	43.5	5.5	49.0	12.5
23–24	65.5	14.5*	80.0	20.0
25	134.5	15.5*	150.0	37.5
26	275.0	19.0*	294.0	72.0
27	326.5	19.5*	346.0	82.0
28	364.0	20.0*	384.0	88.0
29	585.0	25.0*	610.0	140.0

* Assumes an average of 1.5 runs of ANSWER (cf. 8.1)

8.4. It does not appear that either prior knowledge of the form of the factors of N or knowledge that N has no factors below a certain limit can be used in any way to speed up the continued fraction factoring method.

8.5. It can happen, as observed in [11, p. 771], that N and Q_n can have a factor in common. Such a factor must also divide P_n and P_{n+1}. For example, in the expansion of $\sqrt{209}$, $P_4 = P_5 = Q_4 = 11$. However, in some expansions such as $\sqrt{2813}$, the $\mathrm{GCD}(N, Q_n) = 1$ for every n. Whether or not such an approach is practical in trying to factor a large N has not been investigated, as far as we know.

8.6. It is unfortunate that there does not appear to be any practical approach to finding S-sets which does not require the complete factorization of some collection of Q_n's. If such a technique did exist, it would no doubt greatly speed up the present method.

8.7. It is very important to realize that once S-sets begin to appear, increasing the number of factored Q_n's by as little as 50 tends to produce a large increase in the number of S-sets.

8.8. Having about seven S-congruences is usually sufficient to factor N. The method seldom seems to succeed, however, when there is only one such congruence, and there are examples where it has failed with as many as 25 S-congruences.

9. **Acknowledgments.** The authors would like to express their gratitude to David Cantor of UCLA for the use of his multiple-precision subroutines and for his interest and general support of this work. They would also like to express their gratitude to D. H. and Emma Lehmer for their many helpful ideas and suggestions and to J. M. Pollard

for his insightful comments on an earlier version of this paper. Finally, they would like to thank the directors of the UCLA Campus Computing Network for providing the computer time to carry out this project.

Department of Mathematics
University of California
Los Angeles, California 90024

Department of Mathematics
University of Arizona
Tucson, Arizona 85721

1. J. BRILLHART & J. L. SELFRIDGE, "Some factorizations of $2^n \pm 1$ and related results," *Math. Comp.*, v. 21, 1967, pp. 87–96; Corrigendum, *ibid.*, v. 21, 1967, p. 751. MR 37 #131.

2. J. BRILLHART, D. H. LEHMER & J. L. SELFRIDGE, "New primality criteria and factorizations of $2^m \pm 1$," *Math. Comp.* (To appear.)

3. D. JARDEN, *Recurring Sequences*, 2nd ed., Riveon Lematematika, Jerusalem, 1966, pp. 40–59. MR 33 #5548.

4. D. KNUTH, *The Art of Computer Programming*, Vol. 2: *Semi-Numerical Algorithms*, Addison-Wesley, Reading, Mass., 1969. MR 44 #3531.

5. M. KRAITCHIK, *Recherches sur la théorie des nombres.* Tome II, Gauthier-Villars, Paris, 1929.

6. M. KRAITCHIK, *Théorie des nombres.* Tome II, Gauthier-Villars, Paris, 1926, pp. 195–208.

7. A. M. LEGENDRE, *Théorie des nombres.* Tome I, 3rd ed., Paris, 1830, pp. 334–341; Also under the title, *Zahlentheorie*, translated by H. Maser, Teubner, Leipzig, 1893, pp. 329–336.

8. D. H. LEHMER, "A photo electric number sieve," *Amer. Math. Monthly*, v. 40, 1933, pp. 401–406.

9. D. H. LEHMER, "Computer technology applied to the theory of numbers," *Studies in Number Theory*, Math. Assoc. Amer., distributed by Prentice-Hall, Englewood Cliffs, N. J., 1969, pp. 117–151. MR 40 #84.

10. D. H. LEHMER, "An announcement concerning the delay line sieve DLS-127," *Math. Comp.*, v. 20, 1966, pp. 645–646.

11. D. H. LEHMER & R. E. POWERS, "On factoring large numbers," *Bull. Amer. Math. Soc.*, v. 37, 1931, pp. 770–776.

12. D. N. LEHMER, "Hunting big game in the theory of numbers," *Scripta Math.*, 1933, pp. 229–235.

13. D. N. LEHMER, *Factor Stencils*, rev. and extended by J. D. Elder, Carnegie Inst. of Washington, Washington, 1939. MR 1 #133.

14. J. C. MOREHEAD, "Note on Fermat's numbers," *Bull. Amer. Math. Soc.*, v. 11, 1905, pp. 543–545.

15. M. A. MORRISON & J. BRILLHART, "The factorization of F_7," *Bull. Amer. Math. Soc.*, v. 77, 1971, p. 264. MR 42 #3012.

16. F. PROTH, *Comptes Rendus*, Paris, v. 87, 1878, p. 374.

17. D. SHANKS, "Class number, a theory of factorization, and genera," *Proc. Sympos. Pure Math.*, v. 20, Amer. Math. Soc., Providence, R.I., 1971, pp. 415–440. MR 47 #4932.

18. A. E. WESTERN, "Note on Fermat's numbers and the converse of Fermat's theorem," *Proc. London Math. Soc.*, v. 3, 1905, xxi–xxii.