Printed at the Mathematical Centre, Kruislaan 413, Amsterdam, The Netherlands.

The Mathematical Centre, founded 11th February 1946, is a non-profit institution for the promotion of pure and applied mathematics and computer science. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

MATHEMATICAL CENTRE TRACTS 154

COMPUTATIONAL METHODS IN NUMBER THEORY

PART I

edited by

H.W. LENSTRA, JR.

R. TIJDEMAN

MATHEMATISCH CENTRUM AMSTERDAM 1982

ANALYSIS AND COMPARISON OF SOME INTEGER FACTORING ALGORITHMS

by

C. POMERANCE *)

1. INTRODUCTION

Although the problem of how to efficiently factor an integer is centuries old, in recent years an extraordinary amount of interest has been focused on the issue. There are at least two reasons. First, the problem has taken on a glamorous tinge because of its connection to the public-key cryptosystem scheme of Rivest, Shamir and Adleman (see HOOGENDOORN [12]). Second, and perhaps more fundamental, with the explosive growth of computers in society, the analysis of natural algorithmic problems, including factoring, has grown into an area of its own.

It should be pointed out that unlike with the companion subject of primality testing (see LENSTRA [19], POMERANCE [24]), there have been no recent dramatic breakthroughs concerning factoring. Nevertheless, advances have been and are being made. Many of the new ideas are based on the seminal MORRISON-BRILLHART continued fraction algorithm [21]. With this algorithm composite numbers of 50 digits with no small prime factors can be factored (although this is probably about the limit of the basic algorithm).

Although the continued fraction algorithm has worked in practice, no rigorous analysis has ever been given for its running time. However, DIXON [9] recently proposed a simplified version for which a rigorous running time analysis could be provided. Dixon's algorithm is not a serious contender as a practical method for factoring. Rather, its value lies in the respectability it brings both to the continued fraction algorithm and the heuristic argument supporting the continued fraction algorithm discussed below.

Another similar algorithm, due to SCHROEPPEL [28], ingeniously replaces time consuming trial divisions required in the continued fraction algorithm and Dixon's algorithm with a procedure based on the sieve of Eratosthenes.

*) Research partially supported by an NSF grant.

To my knowledge, Schroeppel's algorithm has not proved practical.

In what follows I shall be concerned with carefully analyzing the running times of these algorithms and how certain variations affect these times. Where possible, rigorous arguments are given or sketched. At other times, an argument is presented that relies on certain explicit but unproved hypotheses. There has been a certain amount of disagreement in the literature concerning the running times of the various algorithms. For example, with

$$L(n) = \exp\{(\log n \log\log n)^{\frac{1}{2}}\}$$

(all logarithms in this paper are natural logarithms), various people have suggested that the continued fraction algorithm has a running time of $L(n)^{c+o(1)}$ on input of a composite number n, with the value of c given as $\sqrt{2}$ (SCHROEPPEL [28], MONIER [20]), 2 (KNUTH [13]), and $\sqrt{3/2}$ (WUNDERLICH [34], [35] - although Wunderlich also includes in his analysis a variation which below is called Large Prime). It is my hope that the careful analysis in this paper will settle the controversy (provided the unproved hypotheses are not controversial). On the particular issue of the continued fraction algorithm, I side with Schroeppel and Monier, that is $c = \sqrt{2}$ (with or without the Large Prime variation). Unlike prior results, the running time estimates given in this paper are two-sided, not just upper bounds.

A natural variation of the continued fraction algorithm (aspects of this idea are described in MORRISON-BRILLHART [21], Remarks 4.2, 4.6 and in KNUTH [13] p.384) is what is called below the early abort strategy. I have carefully analyzed the effect of this variation with different choices of certain parameters and have come up with a particular choice of parameters that are asymptotically optimal. It is hoped that these choices can guide factorers of finite numbers.

In addition, proposed below is a new algorithm called the quadratic sieve. It belongs to the same family as the other algorithms considered and appears to be very fast. It is based on the well known fact that the consecutive values of a quadratic polynomial (in fact, any polynomial over the integers) can be factored with a sieve. The quadratic sieve algorithm is very similar to a procedure suggested long ago by KRAITCHIK [15] (also see MORRISON-BRILLHART [21], p.199). In fact it may be fair to say that the relationship of the quadratic sieve algorithm to Kraitchik's method is analogous to the relationship of the modern continued fraction algorithm to an earlier version due to LEHMER and POWERS [18]. The quadratic sieve algorithm can also be viewed as a natural outgrowth of Schroeppel's algorithm. In preliminary work by GERVER [10], it appears to be computer practical.

Each of the algorithms discussed has a running time of the form $L(n)^{\alpha+o(1)}$ where n is the (composite) number being factored, α is a constant that depends on the algorithm and L(n) is defined above. However there is an important dichotomy between Dixon's algorithm together with its variations and the other algorithms discussed. The former can be analyzed rigorously, but are probabilistic algorithms. The latter are deterministic algorithms, but as mentioned above, the analysis contains certain leaps of faith.

By a "deterministic algorithm" we shall mean one in which the input completely pre-determines everything that follows. In contrast, by a "probabilistic algorithm" we mean one in which random numbers are employed. Such an algorithm tries to take advantage of lucky breaks that cannot be guaranteed to occur, but by the law of large numbers are expected. For a probabilistic algorithm we may talk of the probability that the running time is such and so. Indeed 100 independent implementations on the same input could lead to 100 different running times and perhaps even to different outputs (for example, different factorizations of n).

When we say a deterministic factoring algorithm has a running time of $L(n)^{\alpha+o(1)}$, we mean that the running time to factor n is precisely $L(n)^{\alpha+\theta(n)}$ where $\theta(n)$ is some function that tends to 0 as $n \to \infty$ through the composites. When we say a probabilistic factoring algorithm has a running time of $L(n)^{\alpha+o(1)}$ we mean that there is a function $\theta(n)$ which tends to 0 as $n \to \infty$ through the composites such that for every $\varepsilon > 0$, there is an M such that with probability at least $1 - \varepsilon$ the running time to factor n lies between $M^{-1}L(n)^{\alpha+\theta(n)}$ and $M \cdot L(n)^{\alpha+\theta(n)}$. That is, on most implementations of the algorithm for n the running time will not be too many times greater or less than $L(n)^{\alpha+\theta(n)}$. To emphasize this difference, when speaking of a probabilistic algorithm we shall say it has an *expected* running time of $L(n)^{\alpha+o(1)}$.

With Dixon's algorithm and its variations, probability enters the picture because the algorithms involve randomly choosing some auxiliary numbers. With the other algorithms, the auxiliary numbers are deterministically produced. In the heuristic analyses for these deterministic algorithms certain explicit hypotheses are made that essentially say that the auxiliary numbers are "pseudo-random" with respect to certain properties. Unfortunately I do not presently know how to prove these hypotheses. Thus it cannot be ruled out that at almost every instantiation of the algorithm in question the running time is much less than what is claimed - or much more.

The main results are summarized in the table. The exponents given are on the base L(n). Thus, for example, Dixon's algorithm with the Pollard-

Strassen variation has expected running time $L(n)^{\sqrt{3}+o(1)}$ and space requirements of $L(n)^{\sqrt{4/3}+o(1)}$. Reiterating what was said above, only the numbers for Dixon's algorithm and its variations have rigorous proofs.

Dixon's algorithm with the Pollard-Strassen method, the early abort strategy, and Coppersmith-Winograd elimination stands as the current champion of fully proved factoring algorithms with an expected running time of $L(n)^{\sqrt{5/2}+o(1)}$. The fastest fully proved *deterministic* factoring algorithm is the Pollard-Strassen method discussed below in Section 4. The running time is $O(n^{1/4}(\log n)^3 \log\log n \log\log\log n)$. Shanks has a deterministic factoring algorithm (see SCHOOF [27]) which, if the Extended Riemann Hypothesis holds, has a running time of $O(n^{1/5+\varepsilon})$ for every $\varepsilon > 0$. If the quadratic sieve algorithm with Coppersmith-Winograd elimination (discussed in Section 7) can be rigorously analyzed along our heuristic lines, it would be a deterministic algorithm with running time less than $L(n)^{1.0204}$ for all large composite n.

It should be pointed out that just because algorithm A has higher exponents than algorithm B does not necessarily make it the worse algorithm for all values of n. It may be that B is superior only for those $n > 10^{10^{10}}$ and for smaller n, A is the algorithm of choice. A little is said in Section 9 about the practical aspects of the algorithms considered.

It is suggested on a first reading that the proofs, especially those in Sections 2, 3 and 4, be skipped or skimmed. Many of the ideas presented are quite simple and can be understood even without knowing all of the intervening details.

This paper covers only a small fraction of the many types of factoring algorithms known. For alternative treatment of some of the algorithms presented here plus descriptions of many others, the reader's attention is called to BRENT and POLLARD [4], GUY [11], KNUTH [13], MONIER [20], SCHNORR [26], SCHOOF [27], and VOORHOEVE [33].

Among the many people who helped me with this paper I should like to acknowledge and thank Edward Azoff, John Brillhart, Rod Canfield, Andrew Odlyzko, Robert Rumely, Samuel Wagstaff, Hugh Williams, and especially Hendrik Lenstra.

Basic Algorithm	Variation	Elimination Method	Time Exponent		Space Exponent	
Dixon		Gauss	2		1	
	P-S	Gauss	1.732	$\sqrt{3}$	1.155	$\sqrt{4/3}$
	EAS	Gauss	1.732	$\sqrt{3}$	1.155	$\sqrt{4/3}$
	P-S & EAS	Gauss	1.604	$\sqrt{18/7}$	1.069	$\sqrt{8/7}$
	P-S & EAS	C-W	1.581	$\sqrt{5/2}$	1.265	$\sqrt{8/5}$
Morrison-Brillhart (continued fraction)		Gauss	1.414	$\sqrt{2}$	0.707	$\sqrt{1/2}$
	P-S	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	EAS	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	P-S & EAS	Gauss	1.134	$\sqrt{9/7}$	0.756	$\sqrt{4/7}$
	P-S & EAS	C-W	1.118	$\sqrt{5/4}$	0.894	$\sqrt{4/5}$
Schroeppel (linear sieve)		Gauss	1.5		1	
		C-W	1.248		1	
	Cull	Gauss	1.225	$\sqrt{3/2}$	0.816	$\sqrt{2/3}$
	Cull	C-W	1.117		0.895	
Quadratic Sieve		Gauss	1.061	$\sqrt{9/8}$	0.707	$\sqrt{1/2}$
		C-W	1.020		0.818	

Abbreviations: P-S, Pollard-Strassen method
EAS, Early abort strategy
C-W, Coppersmith-Winograd method

2. PRELIMINARIES AND A CONVENTION

The function $L(n) = \exp(\sqrt{\log n \log\log n})$ introduced in the previous section will often simply be denoted L. The expression $L(n)^{\alpha+o(1)}$, where α is a non-zero constant, will often be abbreviated L^{α}. With this unusual convention, seemingly absurd equations can be obtained, such as

$$2L^{\alpha} = L^{\alpha}, \quad L^{\alpha} + L^{\beta} = L^{\max\{\alpha,\beta\}}, \quad \pi(L^{\alpha}) = L^{\alpha},$$

where in the last equation π is the prime counting function. This convention serves to streamline many arguments and to have the important magnitudes

stand out. I hope it does not prove too confusing for the reader. In any event, all theorems will be stated without the convention.

A ubiquitous function from number theory that will be of use here is $\psi(x,y)$, the number of natural numbers not exceeding x free of prime factors exceeding y. We cite the following theorem from CANFIELD, ERDÖS, POMERANCE [7]:

THEOREM 2.1. *If ε is an arbitrary positive constant, then uniformly for $x \geq 10$ and $y \geq (\log x)^{1+\varepsilon}$,*

$$\psi(x,y) = x \cdot u^{-u+o(u)},$$

where $u = (\log x)/\log y$.

The assertion about uniformity means that there is a function $\theta(u)$ such that $\theta(u)/u \to 0$ as $u \to \infty$ and such that if $f(x,y)$ is defined by the equation

$$\psi(x,y) = x \cdot u^{-u+f(x,y)}$$

then $|f(x,y)| \leq \theta(u)$ for all $x \geq 10$, $y \geq (\log x)^{1+\varepsilon}$. In fact, in [7], it is shown that

$$f(x,y) = -u\,\frac{(\log\log u)-1}{(\log u)-1} + O\left(u\,\frac{(\log\log u)^2}{(\log u)^3}\right),$$

where again the error term is uniform. Actually, in [7] only the lower bound implicit in Theorem 2.1 is shown - the upper bound is found in an earlier paper of DE BRUIJN [6].

Combining Theorem 2.1 with our unusual convention, we have, for example,

$$\psi(n,L^a) = n \cdot L^{-(2a)^{-1}}.$$

In fact we shall always apply Theorem 2.1 when $y = L(x)^{a+o(1)}$, where a is a positive constant. For this restricted range of y we have a result that is more general than Theorem 2.1 that we shall occasionally need.

Let $\psi(x,y,z)$ denote the number of natural numbers not exceeding x composed solely of the primes in the interval $(z,y]$. So, for example, $\psi(x,y,1) = \psi(x,y)$.

THEOREM 2.2. *Let $\varepsilon > 0$ be arbitrary. If $u = (\log x)/\log y$ and if $z < y^{1-1/\log u}$, then*

$$\psi(x,y,z) = x \cdot u^{-u+o(u)}$$

uniformly for $(\log x)^{1+\varepsilon} < y < \exp((\log x)^{1-\varepsilon})$, $x \geq 10$.

PROOF. The upper bound follows from Theorem 2.1 since $\psi(x,y,z) \leq \psi(x,y)$. We may assume x is large. Let

$$w_1 = y^{1-2/(3\log u)}, \quad w_2 = y^{1-1/(3\log u)}.$$

Then

$$\psi(x,y,z) \geq \sum_m{}' \psi(x/m,w_1,z) \tag{2.1}$$

where the sum is over those m which are the product of $[u]$ not necessarily distinct primes in $(w_1,w_2]$. Indeed for each integer k counted by $\psi(x/m,w_1,z)$, we have $km \leq x$ and every prime in km is in $(z,w_2] \subset (z,y]$. Moreover, if $km = k'm'$ where m' is also the product of $[u]$ primes in $(w_1,w_2]$ and k' is counted by $\psi(x/m',w_1,z)$, then $m = m'$, $k = k'$.

Note that if m is the product of $[u]$ primes in $(w_1,w_2]$, then

$$w_1^{u-1} < m \leq w_2^u,$$

so that for large x

$$x^{1/\log u} > y \cdot x^{2/(3\log u)} > x/w_1^{u-1} > x/m \geq x/w_2^u = x^{1/(3\log u)}. \tag{2.2}$$

Let $u(m) = 1 + [(\log(x/m))/\log w_1]$. Since $\log w_1 \sim \log y$ as $x \to \infty$, we have from (2.2)

$$\frac{u}{\log u} \gtrsim u(m) \gtrsim \frac{u}{3\log u}. \tag{2.3}$$

Also let $w(m) = (x/m)^{1/u(m)}$. Then

$$w_1 > w(m) > w_1^{1-3(\log u)/u}. \tag{2.4}$$

Indeed, from (2.2) we have

$$w_1 = (x/m)^{(\log w_1)/\log(x/m)} > w(m)$$

$$\geq (x/m)^{(1+(\log(x/m))/\log w_1)^{-1}}$$

$$> \exp\{\log(x/m)\,\frac{\log w_1}{\log(x/m)}\,(1 - \frac{\log w_1}{\log(x/m)})\}$$

$$= w_1^{1-(\log w_1)/\log(x/m)}$$

$$> w_1^{1-(\log y)/\log(x/m)} \geq w_1^{1-(3\log u)/u}.$$

The product of any set of $u(m)$ distinct primes from $(z, w(m)]$ does not exceed $w(m)^{u(m)} = x/m$. Thus from the first inequality of (2.4) we have

$$\psi(x/m, w_1, z) \geq \psi(x/m, w(m), z) \geq \binom{\pi(w(m))-\pi(z)}{u(m)}.$$

From the second inequality in (2.4) and from the assumption $y \geq (\log x)^{1+\varepsilon}$, we have $w(m)/z \gtrsim \exp((1+\varepsilon)/3)$, so that for large x, we have $w(m)/z > e^{1/3} > 4/3$. Thus for large x, we have

$$\pi(w(m)) - \pi(z) > w(m)/(4\log w(m)),$$

so that

$$\psi(x/m, w_1, z) \geq (\frac{\pi(w(m))-\pi(z)}{u(m)})^{u(m)}$$

$$> (\frac{w(m)}{4u(m)\log w(m)})^{u(m)}$$

$$= \frac{x}{m} \cdot \exp\{-u(m)(\log u(m) + \log\log w(m) + \log 4)\} \tag{2.5}$$

$$> \frac{x}{m} \cdot \exp\{-u(m)(\log u(m) + \log\log y + \log 4)\}$$

$$= \frac{x}{m} \cdot \exp\{-u(m)(\log u - \log\log u + \log\log y + O(1))\}$$

from (2.3). Also, from the assumption $y < \exp((\log x)^{1-\varepsilon})$, we have $\log\log y < \varepsilon^{-1}\log u$. Thus from (2.3) and (2.5) there is a constant C that depends at most on ε such that

$$\psi(x/m, w_1, z) > \frac{x}{m} \cdot \exp(-Cu)$$

for all large x. Putting this estimate into (2.1) we have

$$\psi(x,y,z) > x \cdot e^{-Cu} \sum{}' m^{-1}, \tag{2.6}$$

so it remains to estimate this last sum. We have (where p represents a prime variable and we use the well-known formula for $\Sigma_{p\leq X} p^{-1}$)

$$\sum{}' m^{-1} \geq (\sum_{w_1 < p \leq w_2} p^{-1})^{[u]}/[u]!$$

$$= (\log\log w_2 - \log\log w_1 + O(e^{-\sqrt{\log y}}))^{[u]}/[u]!$$

$$= (\log((1 - \frac{1}{3\log u})/(1 - \frac{2}{3\log u})) + O(e^{-\sqrt{\log\log x}}))^{[u]}/[u]!$$

$$> (\frac{1}{4\log u})^u/[u]!$$

$$= \exp\{-u(\log u + \log\log u + O(1))\}.$$

Thus from (2.6),

$$\psi(x,y,z) \geq x \cdot \exp\{-u(\log u + \log\log u + O(1))\},$$

which completes the proof of the theorem.

REMARK. For $z = 1$, Theorem 2.2 reduces to Theorem 2.1 with $y < \exp((\log x)^{1-\varepsilon})$. Thus the above argument is an independent proof of the lower bound in Theorem 2.1 for this restricted set of y.

3. DIXON'S ALGORITHM

We shall only wish to apply factorization algorithms to composite numbers with no small prime factors. So in this and all subsequent sections we shall assume that n is composite and not divisible by any prime $p \leq L$. If n is composite, this fact can usually be quickly ascertained with at most a few pseudoprime tests (or one could use the probabilistic compositeness tests of SOLOVAY-STRASSEN [30] or RABIN [25]). To test for all prime factors $p \leq L$ by trial divisions takes L steps.

In [9], Dixon describes the following algorithm. Say we wish to find a non-trivial factorization of n. If m is an integer, denote by $Q(m)$ the least

non-negative residue of $m^2 \bmod n$. Let $a < 1$ be a positive constant to be chosen later. We perform the following steps.

1. Randomly choose a number $m \in [1,n]$ and form $Q(m)$.
2. Try to factor $Q(m)$ by trial division with the primes $p \le L^a$.
3. If $Q(m)$ completely factors in Step (2), store m, $Q(m)$, and its factorization.

These steps are repeated enough times until we have $\pi(L^a)+1$ completely factored $Q(m)$'s. To each factored $Q(m)$ we associate a vector $v(m) \in (\mathbb{Z}/2\mathbb{Z})^{\pi(L^a)}$, where the i-th coordinate of $v(m)$ is 0 or 1 depending on whether the i-th prime appears an even or an odd number of times in the prime factorization of $Q(m)$. Since we have $\pi(L^a)+1$ vectors, each with $\pi(L^a)$ coordinates, there is a linear dependency. Perform Gaussian elimination to find such a dependency. Since we are working over $\mathbb{Z}/2\mathbb{Z}$, this dependency may be expressed as

$$v(m_1) + v(m_2) + \ldots + v(m_k) = 0.$$

Thus we may compute a number X such that

$$Q(m_1)Q(m_2)\ldots Q(m_k) = X^2.$$

(Actually, we only compute the least non-negative residue $x \equiv X \bmod n$.) If y denotes the least non-negative residue of $m_1 m_2 \ldots m_k \bmod n$, then we have

$$y^2 \equiv m_1^2 m_2^2 \ldots m_k^2 \equiv Q(m_1)Q(m_2)\ldots Q(m_k) \equiv x^2 \bmod n.$$

We finally compute the greatest common factor of n and x+y. In [9], Dixon proves that if n is composite, then with probability at least 1/2, this last step will produce a non-trivial factor of n. Thus, say, if n is composite and the above procedure is repeated 10 times, then we are more than 99.9% certain of finding a non-trivial factorization of n.

Running time analysis

The most time consuming of steps (1), (2), (3) is step (2) which has a running time of L^a. Say we have to repeat steps (1), (2), (3) L^b times before we find $\pi(L^a)+1$ completely factored $Q(m)$'s. Then the running time for this stage of the algorithm is L^{a+b}. (Note that at this point, we are not sure whether $b = b(n)$ approaches a positive constant or not.) The Gaussian

elimination stage of the algorithm takes L^{3a} steps. The formation of the numbers x, y and (n,x+y) takes only L^a steps. Thus the total running time is

$$L^{\max\{a+b,3a\}}, \tag{3.1}$$

so it remains only to find b and choose a optimally. To make a long story short, it turns out that we may choose $b = a + (2a)^{-1}$. Thus the optimal choice for a is 1/2.

It is at least heuristically clear that we should choose $b = a + (2a)^{-1}$. Indeed, one might expect that the probability that $Q(m)$ completely factors with the primes $p \le L^a$ is about $n^{-1}\psi(n,L^a)$. Moreover, by Theorem 2.1 this quantity is $L^{-(2a)^{-1}}$. Thus to obtain L^a completely factored $Q(m)$'s, one should get $L^{a+(2a)^{-1}}$ values of m. The nice thing about Dixon's algorithm is that we are able to prove this heuristic guess.

Because of a variation of Dixon's algorithm studied in the next section we treat now a situation more general than is immediately required. In particular, to prove Theorem 3.2 below, only Theorem 2.1 is required, but instead we shall use the more general Theorem 2.2.

We introduce some notation. If T is a finite set, by $|T|$ we shall mean the cardinality of T. Let $T(x,y,z)$ denote the set of integers counted by $\psi(x,y,z)$. That is,

$$T(x,y,z) = \{m \le x: p|m, p \text{ prime} \Rightarrow z < p \le y\}.$$

Thus $|T(x,y,z)| = \psi(x,y,z)$. By $T_Q(x,y,z)$ we shall mean the set of residues mod n whose squares are in $T(x,y,z)$ and coprime to n. In symbols,

$$T_Q(x,y,z) = \{m \in [1,n]: (m,n) = 1,\ Q(m) \in T(x,y,z)\}.$$

Our goal is to show that in certain circumstances, $|T_Q(x,y,z)|$ and $\psi(x,y,z)$ agree up to a factor of $L^{o(1)}$. Our proof begins along the lines of Dixon's Lemma 2. What is new here is Lemma 3.2 below which depends on Theorems 2.1 and 2.2. In its place, Dixon uses a trivial estimate. By taking the middle ground of using a crude lower bound estimate for $\psi(x,y)$ in the proof of Lemma 3.2 one could obtain the upper bound estimate for the running time of Dixon's algorithm achieved by SCHNORR [26] and KNUTH [13].

LEMMA 3.1. *Suppose* $2 \le y < x \le n$ *and that all prime factors of* n *exceed* y. *Then*

$$2^{\omega(n)}\psi(x,y,z) \ge |T_Q(x,y,z)| \ge \psi(\sqrt{x},y,z)^4\Big(\sum_{T(\sqrt{x},y,z)} \tau(t)\Big)^{-2},$$

where $\omega(n)$ *denotes the number of distinct prime factors of* n *and* $\tau(t)$ *denotes the number of positive divisors of* t.

PROOF. Let Q denote the group of squares in $(\mathbb{Z}/n\mathbb{Z})^*$. Then

$$(\mathbb{Z}/n\mathbb{Z})^*/Q \simeq (\mathbb{Z}/2\mathbb{Z})^{\omega(n)}.$$

Identifying $\mathbb{Z}/2\mathbb{Z}$ with the multiplicative group $\{1,-1\}$, we have the canonical projection

$$\chi\colon (\mathbb{Z}/n\mathbb{Z})^* \to (\mathbb{Z}/2\mathbb{Z})^{\omega(n)}$$

where $\chi(t) = ((t/p_1),\dots,(t/p_{\omega(n)}))$; $p_1,\dots,p_{\omega(n)}$ being the distinct prime factors of n and (t/p_i) the Legendre symbol. Thus the function $\chi(t)$ describes $2^{\omega(n)}$ quadratic residue classes mod n.

If $t \in Q$, then by the above, $t = Q(m)$ for precisely $2^{\omega(n)}$ values of $m \in [1,n]$. Thus we have

$$|T_Q(x,y,z)| = 2^{\omega(n)}|Q \cap T(x,y,z)| \le 2^{\omega(n)}\psi(x,y,z), \tag{3.2}$$

which establishes the first inequality in the lemma.

Label the $2^{\omega(n)}$ quadratic residue classes of $(\mathbb{Z}/n\mathbb{Z})^*$ with the integers $i = 1,\dots,2^{\omega(n)}$. Let T_i denote the set of $t \in T(\sqrt{x},y,z)$ that belong to the i-th class. Let $T = \cup T_i$, so that $T = T(\sqrt{x},y,z)$. Let

$$S = \sum_T \tau(t)^{-1}, \quad S_i = \sum_{T_i} \tau(t)^{-1}, \quad S' = \sum_T \tau(t).$$

By the Cauchy-Schwarz inequality, we have

$$\psi(\sqrt{x},y,z)^2 \le SS', \tag{3.3}$$

$$S^2 \le 2^{\omega(n)} \sum_{i=1}^{2^{\omega(n)}} S_i^2. \tag{3.4}$$

If $s = tt'$ for $t,t' \in T_i$ for some i, then by the multiplicativity of Legendre symbols we have $s \in Q \cap T(x,y,z)$. Let $c(s)$ denote $\sum \tau(t)^{-1}\tau(t')^{-1}$ where the sum is over all ordered pairs t, t' where $s = tt'$ and $t,t' \in T_i$ for some i. Then $c(s) > 0$ implies $s \in Q \cap T(x,y,z)$. Moreover, since $\tau(t)^{-1}\tau(t')^{-1} \le \tau(tt')^{-1}$, we have $c(s) \le 1$ for all s. Thus from (3.2), (3.3), (3.4) we have

$$\begin{aligned} |T_Q(x,y,z)| &= 2^{\omega(n)}|Q \cap T(x,y,z)| \\ &\ge 2^{\omega(n)} \sum_{s=1}^{n} c(s) = 2^{\omega(n)} \sum_{i=1}^{2^{\omega(n)}} S_i^2 \\ &\ge S^2 \ge \psi(\sqrt{x},y,z)^4 (S')^{-2}, \end{aligned}$$

which completes the proof of the lemma.

We shall be interested in $|T_Q(x,y,z)|$ in three situations:

(i) $x = n^c$, $y = L^\alpha$, $z = 1$,

(ii) $x = n^c$, $y = L^\alpha$, $z = L^\beta$,

(iii) $x = n^c$, $y = n^c$, $z = L^\beta$,

where $0 < c \le 1$, $0 < \beta < \alpha < 1$. In addition, recall that we are assuming n is composed solely of primes $p \ge L$. Thus the inequality $L(n)^{\omega(n)} \le n$ implies $\omega(n) \le \sqrt{(\log n)/\log\log n}$, so that

$$2^{\omega(n)} = L^{o(1)}. \tag{3.5}$$

Thus from Theorem 2.1,

$$2^{\omega(n)}\psi(n^c,L^\alpha,1) = n^c L^{-c(2\alpha)^{-1}} \tag{3.6}$$

and from Theorem 2.2,

$$2^{\omega(n)}\psi(n^c,L^\alpha,L^\beta) = n^c L^{-c(2\alpha)^{-1}}. \tag{3.7}$$

In the following lemma we consider situations (i) and (ii) at once.

LEMMA 3.2. *For* $0 < c \le 1$, $0 \le \beta < \alpha$, *we have*

$$S' \overset{\text{def}}{=\!=} \sum_{T(n^{c/2},L^\alpha,L^\beta)} \tau(t) \le n^{c/2}L(n)^{-c(4\alpha)^{-1}+o(1)}.$$

PROOF. Let $T = T(n^{c/2},L^{\alpha},L^{\beta})$, let N denote a large, temporarily fixed integer, and let $\varepsilon = c/(2N)$. Then

$$S' = \sum_{T} \tau(t) = \sum_{t\in T}\sum_{d|t} 1 = \sum_{d\in T}\sum_{\substack{d'\in T\\ d'\le n^{c/2}/d}} 1$$

$$= \sum_{d\in T} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{i=1}^{N} \sum_{\substack{d\in T\\ n^{(i-1)\varepsilon}\le d\le n^{i\varepsilon}}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta}) = \sum_{i=1}^{N} S_i',$$

say. We note that

$$S_1' \le \sum_{d\le n^{\varepsilon}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$= \sum_{d\le n^{\varepsilon}} (n^{c/2}/d)(n^{c/2}/d)^{-1}\psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{d\le n^{\varepsilon}} (n^{c/2}/d)L^{-(c/2-\varepsilon)(2\alpha)^{-1}}$$

$$= n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}};$$

by Theorem 2.2. Also

$$S_N' = \sum_{\substack{d\in T\\ n^{c/2-\varepsilon}\le d}} \psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{\substack{d\in T\\ n^{c/2-\varepsilon}\le d}} n^{c/2}/d = n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}},$$

by Theorem 2.2 and partial summation. Finally, if $1 < i < N$, we have by the same techniques

$$S_i' = \sum_{\substack{d\in T\\ n^{(i-1)\varepsilon}\le d\le n^{i\varepsilon}}} (n^{c/2}/d)(n^{c/2}/d)^{-1}\psi(n^{c/2}/d,L^{\alpha},L^{\beta})$$

$$\le \sum_{\substack{d\in T\\ n^{(i-1)\varepsilon}\le d}} (n^{c/2}/d)L^{-(c/2-i\varepsilon)(2\alpha)^{-1}}$$

$$= n^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}}.$$

Thus putting together our estimates we have

$$S' \le \sum_{i=1}^{N} S_i' \le Nn^{c/2}L^{-(c/2-\varepsilon)(2\alpha)^{-1}}.$$

Since this result holds for each integer N, we have

$$S' \le n^{c/2}L^{-c(4\alpha)^{-1}},$$

which proves our lemma.

We can now prove

THEOREM 3.1. *If* $0 < c \le 1$, $0 < \beta < \alpha < 1$, *and* n *is composed solely of primes exceeding* L(n), *then*

$$|T_Q(n^c,L(n)^{\alpha},1)| = n^cL(n)^{-c(2\alpha)^{-1}+o(1)}, \tag{3.8}$$

$$|T_Q(n^c,L(n)^{\alpha},L(n)^{\beta})| = n^cL(n)^{-c(2\alpha)^{-1}+o(1)}, \tag{3.9}$$

$$|T_Q(n^c,n^c,L(n)^{\beta})| = n^cL(n)^{o(1)}. \tag{3.10}$$

PROOF. Note that Lemma 3.1 and (3.6), (3.7) immediately give the upper bounds in (3.8), (3.9). Allowing the possibility $\beta = 0$, we have

$$\psi(n^{c/2},L^{\alpha},L^{\beta}) = n^{c/2}L^{-c(4\alpha)^{-1}}$$

from Theorems 2.1 and 2.2. Thus Lemmas 3.1 and 3.2 give the lower bounds in (3.8), (3.9).

For (3.10), let T_i' denote the set of prime numbers in the interval $(L^{\beta},n^{c/2}]$ which do not divide n and belong to the i-th quadratic residue class. Thus

$$\sum_{i=1}^{2^{\omega(n)}} |T_i'| \geq \pi(n^{c/2}) - \pi(L^{\beta}) - \omega(n) \sim \frac{n^{c/2}}{\log n^{c/2}},$$

so that from (3.5) there is some i with $|T_i'| \geq n^{c/2}L^{o(1)}$. But if $p_1,p_2 \in T_i'$, then $p_1p_2 \in Q \cap T(n^c,n^c,L^{\beta})$. Such a product p_1p_2 can occur in this fashion in at most one other way, namely as p_2p_1. Thus

$$|T_Q(n^c,n^c,L^{\beta})| \geq |Q \cap T(n^c,n^c,L^{\beta})| \geq \frac{1}{2}|T_i'|^2 \geq n^cL^{o(1)}.$$

This inequality immediately gives the equality (3.10).

Recall that in the implementation of Dixon's algorithm we repeat steps (1), (2), (3) L^b times, where b is chosen so that we are virtually assured of obtaining L^a values of m for which Q(m) completely factors with the primes $p \leq L^a$. If $(m,n) > 1$, then Q(m) will not completely factor with the primes $p \leq L^a$, since $a < 1$ and n is not divisible by any prime $p \leq L$. However, very few of our randomly chosen m will satisfy $(m,n) > 1$ since it is possible to show $\phi(n)/n \geq 1 - 1/L$ where ϕ denotes Euler's function. Thus we have L^b values of m with $(m,n) = 1$.

By (3.8)

$$|T_Q(n,L^a,1)| = nL^{(-2a)^{-1}}.$$

Thus given L^b random values of $m \in [1,n]$ with $(m,n) = 1$, we expect $L^{b-(2a)^{-1}}$ of these m to fall in $T_Q(n,L^a,1)$. Thus we choose $b = a + (2a)^{-1}$. By (3.1) the running time for all of Dixon's algorithm is

$$L^{\max\{2a+(2a)^{-1},3a\}}.$$

By choosing $a = 1/2$, we thus have

THEOREM 3.2. *Dixon's algorithm has expected running time* $L(n)^{2+o(1)}$. *The space required is* $L(n)^{1+o(1)}$.

PROOF. We have already shown the assertion about the running time. To see the assertion about the space, note that the most space-consuming part of the algorithm is the construction of the matrix upon which we perform the Gaussian elimination. This matrix is $L^a \times L^a$ where we have chosen $a = 1/2$.

REMARKS. Dixon claims $L^{3\sqrt{2}}$ as an upper bound for the running time for his algorithm, but we have introduced some efficiencies into his proof. Both SCHNORR [26] and KNUTH [13] had already sharpened Dixon's proof to give a $L^{2\sqrt{2}}$ upper bound for the running time. As we have already remarked, the difference between their analysis and ours is that they used an elementary estimate for $\psi(x,y)$ rather than Theorem 2.1. Note that Theorem 3.2 is sharp - the expected running time is bounded above and below by L^2. (All of the running time estimates in this paper are sharp.)

4. VARIATIONS OF DIXON'S ALGORITHM

Already in MORRISON-BRILLHART [21], several variations of the continued fraction algorithm were proposed as a possible means of speeding things up. In this section we analyze three natural variations in the context of Dixon's algorithm. All three make sense for the continued fraction algorithm, but we shall only be able to give heuristic arguments there. However, with Dixon's algorithm we are able to give rigorous proofs. With our most elaborate combination of variations we are able to lower the exponent 2 in Theorem 3.2 to $\sqrt{5/2} \approx 1.581$.

Large Prime

This variation was suggested in MORRISON-BRILLHART [21] and is commonly used by people implementing the continued fraction algorithm (for example, see WUNDERLICH [36]). We now describe and analyze the Dixon algorithm analogue.

If the unfactored portion x of Q(m) after trial division up to y is less than y^2, then x is prime. Thus if Q(m) completely factors with the primes up to L^a, we usually find this out earlier before all the trial division steps have been executed. However, the majority of Q(m) do not completely factor with the primes up to L^a and for these Q(m) we must spend the full time (but see the "early abort strategy" below). Thus this idea cannot lower the asymptotic running time.

But say $x \in (L^a,L^{2a}]$. That is, after trial division up to L^a, the unfactored portion x is not 1, but $x \leq L^{2a}$. Thus Q(m) has a single prime factor in $(L^a,L^{2a}]$ and all other prime factors are in $(1,L^a]$. Since such Q(m)

are discovered with no extra work, why not use them?

At first glance it would seem that using these large primes makes the Gaussian elimination step more complex - instead of working with an $L^a \times L^a$ matrix, we have perhaps an $L^{2a} \times L^{2a}$ matrix. However, by the following device, called a "cull", there is very little extra cost at the elimination stage. First, the factored Q(m)'s are placed in order of their largest prime. We then pass through the list of corresponding 0 - 1 vectors v(m) exactly once. If v(m)'s last 1 is not already shared by a neighbouring v(m), this vector is completely deleted. If it is shared by a neighbour, the two vectors are added (mod 2), thus eliminating the large prime. (Of course, if there are k vectors with the same large prime, we add the first to each of the k - 1 other vectors.) After the cull we are left with 0 - 1 vectors where the coordinates corresponding to primes in $(L^a,L^{2a}]$ are all 0. We thus have reduced our large matrix to a matrix with only L^a columns. What makes this idea work is that each Q(m) has at most one prime in $(L^a,L^{2a}]$. If a Q(m) had more primes in this interval, the process would probably not be accomplished in one pass down the matrix.

To keep the space requirement and running time down, the cull procedure can be performed on the factored Q(m)'s themselves rather than with the vectors v(m). Since any Q(m) has at most $\log n/\log 2$ prime factors, even if we had as many as L^{2a} values of Q(m) involved in the cull, the space and running time would both be at most $cL^{2a}(\log n)^2 = L^{2a}$.

So we see we can cheaply integrate into our algorithm those Q(m) with a single prime in $(L^a,L^{2a}]$ and all other primes in $(1,L^a]$. The question is, how many new Q(m) are now included? We now show that unfortunately there are not many more.

THEOREM 4.1. *Assume that* n *is divisible by no prime up to* L(n). *Let* $0 < a < 1$ *and let* M *denote the number of* $m \in [1,n]$ *such that* Q(m) *is divisible by at most one prime, counting multiplicity, in the interval* $(L(n)^a,L(n)^{2a}]$ *and that all other prime factors of* Q(m) *are in* $(1,L(n)^a]$. *Then* $M = nL(n)^{-(2a)^{-1}+o(1)}$.

PROOF. We clearly have $M \geq |T_0(n,L^a,1)|$, where the notation is defined in §3. Thus by Theorem 3.1, $M \geq n \cdot L^{-(2a)^{-1}}$. On the other hand, if $t \leq n$ is divisible by at most one prime from $(L^a,L^{2a}]$ and all other primes of t are in $(1,L^a]$, then $t = Q(m)$ for at most $2^{\omega(n)}$ values of $m \in [1,n]$. To see this, one uses the argument at the beginning of the proof of Lemma 3.1, making separate cases for $(m,n) = 1$, $(m,n) > 1$. Thus, if p denotes a prime variable,

$$M \leq 2^{\omega(n)} \sum_{L^a<p\leq L^{2a}} \psi(n/p,L^a)$$

$$= 2^{\omega(n)} nL^{-(2a)^{-1}} \sum_{L^a<p\leq L^{2a}} 1/p$$

$$= 2^{\omega(n)} nL^{-(2a)^{-1}} = nL^{-(2a)^{-1}},$$

where we use Theorem 2.1 and (3.5).

We remark that the result would be the same even if we allowed Q(m) to have k primes in $(L^a,L^{ca}]$ for any fixed k and c. Following the proof of Theorem 3.2 but now using Theorem 4.1 rather than Theorem 3.1 gives us

THEOREM 4.2. *The expected running time for Dixon's algorithm with the Large Prime variation is* $L(n)^{2+o(1)}$. *The space required is* $L(n)^{1+o(1)}$.

We are not saying that the Large Prime variation is useless. To the contrary, those actually using the method (in the context of the continued fraction algorithm) find it very helpful. What we are saying is that the idea only affects the "o(1)" in the running time. In contrast, the remaining variations we shall discuss actually lower the exponent somewhat.

The Pollard-Strassen method

The naive way to see if Q(m) completely factors with the primes up to L^a is to use trial division. This is in fact the method used in Section 3. By this method it takes L^a steps to ascertain whether Q(m) so factors by this method. In [26], SCHNORR suggests using a factorization method of Pollard in place of trial division. In fact, POLLARD has two methods [22], [23] which can find the largest divisor of Q(m) consisting solely of primes $p \leq L^a$ in time $L^{a/2}$. One method, known as the Pollard-ρ method (so-called because a certain diagram in the description of the algorithm is in the shape of a "ρ"), is probabilistic and might cause some problems in a rigorous proof of the running time of Dixon's algorithm. Thus for our theoretical analysis it is preferable to use the second method which is deterministic and fully proved. This method is based on the fast Fourier transform. We present a simplified version due to STRASSEN [31], also employed by Schnorr.

Let y be a square and let $z = \sqrt{y}$. We shall show that for every integer

t, the least prime factor of (t,y!) can be found in $O(z(\log^2 z + \log\log t) \log t \log\log t \log\log\log t)$ bit operations. Since Q(m) has at most $O(\log n)$ prime factors, by an iteration of this method we can find the largest divisor of Q(m) composed solely of primes $p \le L^a$ in time $L^{a/2}$.

The idea is to write

$$y! = \prod_{j=1}^{z} (jz)!/[(j-1)z]!$$

and to compute each $(t,(jz)!/[(j-1)z]!)$; the first one of these which is larger than 1 isolates the least prime factor of (t,y!) in an interval of length z in which it can be quickly found. Consider the polynomial

$$f(x) = \prod_{i=0}^{z-1} (x-i).$$

Then $f(jz) = (jz)!/[(j-1)z]!$. Moreover by [32, Sec. 4] all of the $f(jz) \bmod t$, $j = 1,\ldots,z$, can be found in a total of $O(z \log^2 z \log t \log\log t \log\log\log t)$ bit operations (cf. [3, Sec. 4.5]). Since each

$$(t, f(jz) \bmod t) = (t,(jz)!/[(j-1)z]!)$$

can be found in $O(\log t(\log\log t)^2 \log\log\log t)$ bit operations, by [13, exercise 4.5.2.32], they all can be computed in $O(z \log t(\log\log t)^2 \log\log\log t)$ bit operations. Finally, when the first of these which exceeds 1 is found, it takes O(z) additional trial divisions of $O(\log t \log\log t \log\log\log t)$ bit operations each to find the least prime in $(t, f(jz) \bmod t)$. Thus there are a total of $O(z(\log^2 z + \log\log t)\log t \log\log t \log\log\log t)$ bit operations, as claimed.

To ascertain the effect of using the Pollard-Strassen method in place of trial division in Dixon's algorithm, recall that if we use the primes up to L^a, then we expect to get $L^{a+(2a)^{-1}}$ values of Q(m). Previously, for each (or almost all) Q(m), we took L^a steps to see if Q(m) completely factors with the primes $p \le L^a$. With the Pollard-Strassen method, it only takes $L^{a/2}$ steps. Thus the total running time is

$$L^{\max\{3a/2+(2a)^{-1},\,3a\}}.$$

Choosing $a = \sqrt{1/3}$, we have

THEOREM 4.3. *The expected running time of Dixon's algorithm with the Pollard-Strassen method is* $L(n)^{\sqrt{3}+o(1)}$. *The space required is* $L(n)^{\sqrt{4/3}+o(1)}$.

REMARK. In [26], SCHNORR obtains an upper bound of $L^{\sqrt{6}}$ for the running time of Dixon's algorithm with the Pollard-Strassen method.

The early abort strategy

For the overwhelming majority of the m chosen in Dixon's algorithm one must spend the full allotment of time, L^a for trial division, $L^{a/2}$ for the Pollard-Strassen method, only to find that Q(m) cannot be used. If there were some way to weed out most of these bad Q(m) before too much time was invested, we could speed things up. A natural idea that many have had is to abort working with Q(m) if at some pre-chosen point it does not look too likely to be composed solely of the primes below L^a. We now make this idea precise and show that we can indeed lower the exponent in the running time.

We first consider the situation where Dixon's algorithm is altered with just one go or no go decision on a Q(m). We then generalize to k decisions for a fixed but arbitrary k. Finally, we combine the early abort strategy with the Pollard-Strassen method.

If $0 < a,c,\theta < 1$, then by Dixon's algorithm with one early abort and parameters a,c,θ we mean the following variation of Dixon's algorithm. As before we try to factor the Q(m) with the primes up to L^a. However, now after trial division to $L^{\theta a}$, if the unfactored portion of Q(m) exceeds n^{1-c}, we abort the procedure with Q(m) and get the next m.

We now ask for the probability that Q(m) will be aborted at $L^{\theta a}$ and also the probability that Q(m) will not get aborted and eventually completely factor.

THEOREM 4.4. *Let* $0 < a,c,\theta < 1$ *and assume* n *is divisible by no prime* $p \le L(n)$. *Let* M_1 *denote the number of* $m \in [1,n]$ *such that* Q(m) *has a divisor* m_1 *composed solely of primes* $p \le L(n)^{\theta a}$ *and* $Q(m)/m_1 \le n^{1-c}$. *Let* M_2 *denote the number of* m *with the same property and also* Q(m) *is composed solely of primes* $p \le L(n)^a$. *Then*

$$M_1 = n \cdot L(n)^{-c(2\theta a)^{-1}+o(1)},$$

$$M_2 = n \cdot L(n)^{-c(2\theta a)^{-1}-(1-c)(2a)^{-1}+o(1)}.$$

PROOF. Using the notation from Section 3, we evidently have

$$M_1 \geq |T_Q(n^c, L^{\theta a}, 1)||T_Q(n^{1-c}, n^{1-c}, L^{\theta a})|,$$

$$M_2 \geq |T_Q(n^c, L^{\theta a}, 1)||T_Q(n^{1-c}, L^a, L^{\theta a})|.$$

Thus from Theorem 3.1 we have the lower bounds in the theorem.

We shall find the upper bounds a bit more tedious. Our first task is to estimate for a given u the number of $m \in [1,n]$ with $Q(m) = u$. If $(u,n) = 1$, we have already seen in Section 3 that this number is either 0 or $2^{\omega(n)}$. However, if $(u,n) > 1$, there is some trouble. Suppose $q > 2$ is prime, $k > 0$, $k \geq b \geq 0$ are integers and $q^b \| u$. Then it is possible to show that

$$\sum_{\substack{m\in[1,q^k] \\ m^2\equiv u(q^k)}} 1 = \begin{cases} 2q^{b/2}, & \text{if } (\frac{uq^{-b}}{q}) = 1, \ b \text{ even}, \ b < k \\ 0, & \text{if } (\frac{uq^{-b}}{q}) = -1 \\ 0, & \text{if } b \text{ odd}, \ b < k \\ q^{[b/2]}, & \text{if } b = k. \end{cases}$$

As a consequence we have

$$(4.1) \qquad \sum_{\substack{m\in[1,n] \\ Q(m)=u}} 1 = \prod_{q^k \| n} \sum_{\substack{m\in[1,q^k] \\ m^2\equiv u(q^k)}} 1 \leq \prod_{q^k \| n} 2\sqrt{(u,q^k)} = 2^{\omega(n)}\sqrt{(u,n)}.$$

To estimate M_1 from above, note that

$$M_1 = \sum_t \sum_s \sum_{\substack{m\in[1,n] \\ Q(m)=ts}} 1$$

where t runs over $T(n^{1-c}, n^{1-c}, L^{\theta a})$ and s runs over $T(n/t, L^{\theta a}, 1]$. Furthermore, since all of n's prime factors exceed L, we have $(ts,n) = (t,n)$. Thus from (4.1) we have

$$\begin{aligned} M_1 &\leq 2^{\omega(n)} \sum_t \sum_s \sqrt{(t,n)} \leq 2^{\omega(n)} \sum_{d|n} \sum_{\substack{d|t \\ t\leq n^{1-c}}} \sum_s \sqrt{d} \\ &= 2^{\omega(n)} \sum_{d|n} \sqrt{d} \sum_{\substack{d|t \\ t\leq n^{1-c}}} \psi(n/t, L^{\theta a}) \\ &\leq 2^{\omega(n)} \sum_{d|n} \sqrt{d} \sum_{\substack{d|t \\ t\leq n^{1-c}}} (n/t)L^{-c(2\theta a)^{-1}} \\ &\leq 2^{\omega(n)} \sum_{d|n} \sqrt{d}\ (n/d)(\log n)L^{-c(2\theta a)^{-1}} \\ &= nL^{-c(2\theta a)^{-1}} \sum_{d|n} 1/\sqrt{d} \\ &\leq nL^{-c(2\theta a)^{-1}} \prod_{q|n} (1 + \frac{1}{\sqrt{q}-1}) \\ &\leq nL^{-c(2\theta a)^{-1}} (1 + L^{-1/2})^{\omega(n)} \\ &= nL^{-c(2\theta a)^{-1}}, \end{aligned}$$

where we have used Theorem 2.1 and (3.5), and where q runs over the prime factors of n (which all exceed L).

Similarly we have

$$M_2 = \sum_t \sum_s \sum_{\substack{m\in[1,n] \\ Q(m)=ts}} 1$$

where t now runs over $T(n^{1-c}, L^a, L^{\theta a})$ and s runs over $T(n/t, L^{\theta a}, 1)$ as before. Now we have $(ts,n) = 1$, so that

$$(4.2) \qquad M_2 \leq 2^{\omega(n)} \sum_t \sum_s 1 = 2^{\omega(n)} \sum_t \psi(n/t, L^{\theta a}).$$

We shall find it convenient to break up this last sum. Let N be a large temporarily fixed integer and let $\varepsilon = (1-c)/N$. Let

$$T(c) = \sum_{t \in T(n^{1-c}, L^a, L^{\theta a})} \psi(n/t, L^{\theta a}).$$

Then from (4.2),

$$M_2 \le 2^{\omega(n)} T(c) = 2^{\omega(n)} + 2^{\omega(n)} \sum_{i=0}^{N-1} \{T(c+i\varepsilon) - T(c+(i+1)\varepsilon)\}$$

$$\le 2^{\omega(n)} + 2^{\omega(n)} n \sum_{i=0}^{N-1} L^{-(c+i\varepsilon)(2\theta a)^{-1}} \sum_{t > n^{1-c(i+1)\varepsilon}} 1/t$$

$$\le 2^{\omega(n)} + 2^{\omega(n)} n \sum_{i=0}^{N-1} L^{-(c+i\varepsilon)(2\theta a)^{-1}} \frac{t \in T(n^{1-c-i\varepsilon}, L^a, 1)}{- (1-c-(i+1)\varepsilon)}$$

where we use Theorem 2.1. Since the exponent of the summand is least negative when $i = 0$, we have from (3.5),

$$M_2 \le nL^{-c(2\theta a)^{-1} - (1-c-\varepsilon)(2a)^{-1}}.$$

Finally letting $N \to \infty$, so that $\varepsilon \to 0$, we have our upper bound for M_2.

THEOREM 4.5. *Dixon's algorithm with one early abort and parameters* a, c, θ *has expected running time* $L(n)^{\gamma+o(1)}$ *where*

$$\gamma = \max\{a + \theta a + c(2\theta a)^{-1} + (1-c)(2a)^{-1},\ 2a + (1-c)(2a)^{-1},\ 3a\}.$$

PROOF. We shall need, as before, L^a choices of m for which $Q(m)$ is composed solely of the primes $p \le L^a$. Every such $Q(m)$ obtained in the algorithm has the additional property that there is a divisor m_1 of $Q(m)$ composed solely of the primes $p \le L^{\theta a}$ and such that $Q(m)/m_1 \le n^{1-c}$. By Theorem 4.4, we thus should expect to use $L^{a+c(2\theta a)^{-1} + (1-c)(2a)^{-1}}$ values of m before L^a good values are found. For each m chosen we must at least do trial division to $L^{\theta a}$ which takes $L^{\theta a}$ steps. Thus the running time for trial division to $L^{\theta a}$ is

$$L^{a+\theta a+c(2\theta a)^{-1}+(1-c)(2a)^{-1}}. \tag{4.3}$$

Also by Theorem 4.4, the expected number of m for which we do not abort at $L^{\theta a}$ and continue trial division to L^a is $L^{a+(1-c)(2a)^{-1}}$. The time for each such m is L^a, so the total time for trial division to L^a is

$$L^{2a+(1-c)(2a)^{-1}}. \tag{4.4}$$

Finally the Gaussian elimination step has a running time of L^{3a}. This completes the proof of the theorem.

Thus our remaining problem is to choose the parameters a, c, θ optimally. It is not too difficult to see that the minimum expected running time is attained when the exponents in (4.3) and (4.4) are equalized. The optimal choice of parameters turns out to be

$$a = \sqrt{2/7}, \quad c = 1/7, \quad \theta = 1/2.$$

We call Dixon's algorithm with one early abort and this set of parameters simply Dixon's algorithm with one early abort. We have

THEOREM 4.6. *Dixon's algorithm with one early abort has expected running time* $L(n)^{\sqrt{7/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{8/7}+o(1)}$.

We now consider the situation for more than one early abort. Suppose

$$0 < a, c_i, \theta_i < 1; \quad c_1 + \ldots + c_k < 1; \quad \theta_1 < \ldots < \theta_k. \tag{4.5}$$

By Dixon's algorithm with k early aborts and parameters $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ we mean the following. As before, we try to factor each $Q(m)$ with the primes $p \le L^a$. However, if after trial division to $L^{\theta_i a}$ the unfactored portion of $Q(m)$ exceeds $n^{1-c_1-\ldots-c_i}$, then we abort $Q(m)$ and get the next m. Thus for trial division to be completed to L^a, our number $Q(m)$ must successfully pass k tests.

In an analogous fashion to Theorem 4.4 we can prove

THEOREM 4.7. *Say the procedure is to follow Dixon's algorithm with* k *early aborts and parameters* $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$. *For* $i = 1, \ldots, k$, *let* M_i *denote the number of* $m \in [1,n]$ *for which* $Q(m)$ *is not aborted at* $L(n)^{\theta_1 a}, \ldots, L(n)^{\theta_i a}$. *Also let* M_{k+1} *denote the number of* $m \in [1,n]$ *for which not only is* $Q(m)$ *not aborted, but* $Q(m)$ *eventually factors completely with the primes up to* $L(n)^a$. *Then*

$$M_i = n \cdot L(n)^{-c_1(2\theta_1 a)^{-1} - \ldots - c_i(2\theta_i a)^{-1} + o(1)} \quad \text{for } i = 1,\ldots,k,$$

$$M_{k+1} = n \cdot L(n)^{-c_1(2\theta_1 a)^{-1} - \ldots - c_k(2\theta_k a)^{-1} - (1-c_1-\ldots-c_k)(2a)^{-1} + o(1)}.$$

We remark that, as with Theorem 4.4, the proof of the lower bounds is relatively simple and follows easily from Theorem 3.1. The proof of the upper bounds is tedious but essentially relies on nothing more than Theorem 2.1.

For $i = 1,\ldots,k$ let

$$f_i = a + \theta_i a + c_1(2\theta_i a)^{-1} + \ldots + c_k(2\theta_k a)^{-1} + (1 - c_1 - \ldots - c_k)(2a)^{-1}$$

and let

$$f_{k+1} = 2a + (1 - c_1 - \ldots - c_k)(2a)^{-1}.$$

In the same fashion as Theorem 4.5 follows from Theorem 4.4 we have from Theorem 4.7,

THEOREM 4.8. *The expected running time for Dixon's algorithm with* k *early aborts and parameters* $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ *is*

$$L(n)^{\max\{f_1,\ldots,f_{k+1},3a\}+o(1)}.$$

So we now try to choose the parameters $a, c_1, \ldots, c_k, \theta_1, \ldots, \theta_k$ optimally. This task, which is a bit messy, can be done by induction on k. Again, as with $k = 1$, the optimal choice occurs when the functions $f_1, \ldots, f_{k+1}$ are equalized. The optimal choice is

$$a = (3 + \frac{1}{k+1})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We call Dixon's algorithm with k early aborts and this set of parameters simply Dixon's algorithm with k early aborts.

THEOREM 4.9. *The expected running time for Dixon's algorithm with* k *early aborts is* $L(n)^{\sqrt{3+1/(k+1)}+o(1)}$. *The space required is* $L(n)^{2/\sqrt{3+1/(k+1)}+o(1)}$.

If we now allow k to slowly tend to ∞ as $n \to \infty$, we call the resulting algorithm Dixon's algorithm with the early abort strategy.

THEOREM 4.10. *Dixon's algorithm with the early abort strategy has expected running time* $L(n)^{\sqrt{3}+o(1)}$. *The space required is* $L(n)^{\sqrt{4/3}+o(1)}$.

We thus find that the early abort strategy has the same effect on the running time of Dixon's algorithm as does the use of the Pollard-Strassen method to replace trial division. We now consider a combination of the two variations. That is, we use the Pollard-Strassen method instead of trial division and we use k optimally chosen early aborts. The running time analysis for this algorithm is very similar to the above. We simply replace the functions $f_1,\ldots,f_{k+1}$ with $g_1,\ldots,g_{k+1}$ where

$$g_i = f_i - \theta_i a/2, \quad i = 1,\ldots,k,$$

$$g_{k+1} = f_{k+1} - a/2.$$

However, if we use Gaussian elimination, which takes L^{3a} steps, we find that with an otherwise optimal choice of parameters, this step will dominate the running time. But there are asymptotically faster methods for finding a linear dependency than Gaussian elimination. For example, there is a recent algorithm of COPPERSMITH and WINOGRAD [8] which can find a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^K$ in time bounded by $O(K^r)$ where

$$r < 2.495548.$$

In general, we shall say that an algorithm to find a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^K$ has exponent r, if the running time for the algorithm is $K^{r+o(1)}$. Thus Gaussian elimination has exponent 3 since it runs in time $O(K^3)$ and not generally in time $O(K^{3-\varepsilon})$ for any $\varepsilon > 0$. It is evident that no elimination method has exponent $r < 2$. We shall say more about the problem of elimination in Section 8.

We now return to Dixon's algorithm with k early aborts and with the Pollard-Strassen method. If we use an elimination method with exponent $r > 5/2 + 1/(2k+2)$, the optimal choice of parameters is

$$a = (2r - 3 + \frac{k}{2k+2})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

If we use an elimination method with exponent $r \le 5/2 + 1/(2k+2)$, the optimal choice of parameters is

$$a = (3-\frac{k}{2k+2})^{-1/2}, \quad c_i = \frac{ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We have

THEOREM 4.11. *Using an elimination method with exponent r, the Pollard-Strassen method, and k optimally chosen early aborts, Dixon's algorithm has expected running time*

$$L(n)^{r(2r-3+k/(2k+2))^{-1/2}+o(1)}, \quad \text{if } r > 5/2 + 1/(2k+2),$$

$$L(n)^{(3-k/(2k+2))^{1/2}+o(1)}, \quad \text{if } r \le 5/2 + 1/(2k+2).$$

Finally, letting k tend to ∞ slowly with n, we have

THEOREM 4.12. *The expected running time for Dixon's algorithm with the Pollard-Strassen method, the early abort strategy, and an elimination method with exponent r is*

$$L(n)^{r(2r-5/2)^{-1/2}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{5/2}+o(1)}, \quad \text{if } r \le 5/2.$$

The space required is

$$L(n)^{2(2r-5/2)^{-1/2}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{8/5}+o(1)}, \quad \text{if } r \le 5/2.$$

REMARK. The algorithm of Theorem 4.12 with Coppersmith-Winograd elimination stands as the fastest factorization algorithm for which there is a rigorous proof. However, we do not advocate implementing any version of Dixon's algorithm - the interest here is purely theoretical. The remaining factorization algorithms in this paper have not been rigorously proved, but the slowest among them has a heuristic running time that is less than our champion of Theorem 4.12.

5. THE CONTINUED FRACTION ALGORITHM

Unlike the other algorithms discussed in this article, the continued fraction algorithm has actually been used to factor large numbers. For example, recently WAGSTAFF (see [5]) factored

$$\frac{3^{121}-1}{11617(3^{11}-1)},$$

a 49 digit number with no small factors, using the continued fraction algorithm.

The only difference between the continued fraction algorithm and Dixon's algorithm is in the production of the quadratic residues. In Dixon's algorithm this was done by randomly choosing values of $m \in [1,n]$ and reducing $m^2 \bmod n$, obtaining $Q(m)$. In the continued fraction algorithm, the quadratic residues are produced in a deterministic fashion. Namely, if a_i/b_i is the i-th continued fraction convergent to $\sqrt{n}$, then we let the quadratic residue Q_i be defined by

$$Q_i = a_i^2 - b_i^2 n \equiv a_i^2 \bmod n.$$

In the running of the algorithm the numbers Q_i are not computed from this definition since the numbers a_i, b_i grow geometrically. Rather there is a simple iterative procedure (described in MORRISON-BRILLHART [21]) for producing the Q_i and the $a_i \bmod n$.

From the elementary inequality (for n not a square)

$$\left|\frac{a_i}{b_i} - \sqrt{n}\right| < \frac{1}{b_i b_{i+1}},$$

it is easy to see that

$$-2\sqrt{n} < Q_i < 2\sqrt{n}. \tag{5.1}$$

Note that Q_i might very well be negative, while in Dixon's algorithm the quadratic residues $Q(m)$ are always non-negative. This minor problem is treated by using the "prime" -1 in the factorization of Q_i if $Q_i < 0$. In the final elimination step of the algorithm, when we find $Q_{i_1},\ldots,Q_{i_k}$ whose product is a square, we use the "prime" -1 an even number of times - just like every other prime.

The chief advantage that the continued fraction algorithm has over Dixon's algorithm can be found in the inequality (5.1). Since $|Q_i| < 2\sqrt{n}$, it should be easier to factor Q_i using a set of small primes than it should be for a random Q(m), which is usually much larger.

The chief difficulty in the theoretical analysis of the continued fraction algorithm lies in the deterministic nature of the Q_i. How can we be sure that the Q_i behave as ordinary numbers do in the interval $[-2\sqrt{n},2\sqrt{n}]$? In fact, if the period of the continued fraction for $\sqrt{n}$ is short, then there can only be a few different values of Q_i. (This phenomenon will usually doom the continued fraction algorithm. However a very simple device can be used in this situation: use $\sqrt{\ell n}$ rather than $\sqrt{n}$ where ℓ is a small square-free integer - see [21].)

Further complicating things is that the Q_i do not behave completely like ordinary integers. First of all, they are probably not uniformly distributed in $[-2\sqrt{n},2\sqrt{n}]$. However it is likely they are at least roughly uniformly distributed. For more on this, see KNUTH [14]. Also from the equation $a_i^2 - b_i^2 n = Q_i$, we see that if p is an odd prime and $p \nmid n$, then

$$p|Q_i \text{ implies } \left(\frac{n}{p}\right) = 1.$$

Since $(n/p) = 1$ usually for roughly one-half of the primes $p \le L^a$, we therefore might expect Q_i to less likely factor with these primes than would a random integer in the interval $[-2\sqrt{n},2\sqrt{n}]$. However, if $(n/p) = 1$, then p probably has an enhanced chance of dividing Q_i. For a random integer m, the chance that $p|m$ is $1/p$. But if $(n/p) = 1$, there is a heuristic argument that the chance $p|Q_i$ is $2/(p+1)$, or almost twice as much. To see this, we assume that the $p^2 - 1$ possible values of the pair $a_i \bmod p$, $b_i \bmod p$ (note that the 0, 0 pair cannot occur since $(a_i,b_i) = 1$) are equidistributed. But for each non-zero value of $b_i \bmod p$, there are precisely 2 values of $a_i \bmod p$ for which $p|a_i^2 - b_i^2 n$. Thus the chance that $p|Q_i$ should be $2(p-1)/(p^2-1) = 2/(p+1)$. For the prime $p = 2$, a similar analysis gives the chance 1/3 for $2|Q_i$.

In order to make a heuristic analysis of the continued fraction algorithm, we make the following

HYPOTHESIS 5.1. *There is a constant n_0 such that if $n \ge n_0$, we have the following. There is some integer $\ell \in [1,\log^2 n]$ such that the period for the continued fraction for $\sqrt{\ell n}$ is at least $n^{1/100}$. For this integer ℓ and for any a, $1/10 < a < 1$, the number of primes $p \le L(n)^a$ for which $p \nmid \ell n$ and $(\ell n/p) = 1$ is at least $\pi(L(n)^a)/3$. If a_i/b_i is the i-th convergent to $\sqrt{\ell n}$ and $Q_i = a_i^2 - b_i^2 \ell n$, then the Q_i are distributed in $[-2\sqrt{\ell n},2\sqrt{\ell n}]$ with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in $[-2\sqrt{\ell n},2\sqrt{\ell n}]$.*

For simplicity, in the following we shall always assume $\ell = 1$.

As with Dixon's algorithm we trial divide each Q_i with the primes $p \le L^a$, except now we only use those $p \le L^a$ for which $(n/p) = 1$. When we have enough completely factored Q_i, as with Dixon's algorithm we use an elimination step to assemble integers x, y with $x^2 \equiv y^2 \bmod n$. It is certainly possible for $x \equiv \pm y \bmod n$ and thus for $(x+y,n)$ to fail to be a non-trivial factor of n. We now make a heuristic assumption that this unfortunate event will not occur too often. Since we need to make similar assumptions for each of the variations of the continued fraction algorithm and for the later algorithms considered we make a more general assumption.

HYPOTHESIS 5.2. *There is a constant n_1 such that if $n \ge n_1$ and if any of the algorithms in Sections 5, 6, 7 is run long enough so that $1 + [\log^2 n]$ pairs x, y are assembled with $x^2 \equiv y^2 \bmod n$, then at least one pair will satisfy $x \not\equiv \pm y \bmod n$.*

Since a random pair x, y with $x^2 \equiv y^2 \bmod n$ satisfies $x \not\equiv \pm y \bmod n$ with probability at least 1/2 (for composite n) and since $\sum_{n=1}^{\infty} (1/2)^{\log^2 n} < \infty$, the Borel-Cantelli lemma suggests that Hypothesis 5.2 is plausible. Shanks has noticed, however, that for certain n such as $2^{60} + 2^{30} - 1$, the nature of the quadratic residues Q_i can conspire to produce many trivial pairs x, y with $x^2 \equiv y^2 \bmod n$. Thus in the case of the continued fraction algorithm and its variations, we interpret Hypothesis 5.2 to mean that some non-trivial pair x, y will be found among the first $1 + [\log^2 n]$ pairs when we work with the continued fraction expansion for $\sqrt{\ell n}$ for some $\ell \le \log^2 n$. That is, if we are having very bad luck finding a non-trivial pair x, y, we discard all calculations, choose another multiplier ℓ, and start working anew. We conjecture that this drastic procedure will prohibit Shanks' observed conspiracy, but we do not advocate its use in practice.

Thus by Hypotheses 5.1 and 5.2 we need to obtain L^a completely factored Q_i's. By Theorem 2.1 we have

$$\psi(2\sqrt{n},L^a) = \sqrt{n}L^{-(4a)^{-1}},$$

so that by Hypothesis 5.1 we must generate $L^{a+(4a)^{-1}}$ values of Q_i in order to find the requisite L^a of them which are composed solely of the primes $p \le L^a$. The time to generate a Q_i is $O(\log^2 n)$. The time used to trial divide a Q_i with the primes $p \le L^a$ for which $(n/p) = 1$ is, by Hypothesis 5.1, L^a. The Gaussian elimination step has a running time of L^{3a}. Thus the running time for the continued fraction algorithm is

$$L^{\max\{2a+(4a)^{-1},3a\}}.$$

Thus the optimal choice for a is $1/\sqrt{8}$. We have

THEOREM 5.1. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm is* $L(n)^{\sqrt{2}+o(1)}$. *The space required is* $L(n)^{1/\sqrt{2}+o(1)}$.

REMARK. This result was already known to SCHROEPPEL [28] and MONIER [20]. However in their analysis they assumed Theorem 2.1 without proof. Using an elementary lower bound estimate for $\psi(x,y)$ that is weaker than Theorem 2.1, KNUTH [13] gives L^2 as an upper bound for the running time.

We now consider the effect of the variations discussed in Section 4 on the running time of the continued fraction algorithm. For a description of the variations, we refer the reader to Section 4.

THEOREM 5.2. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with Large Prime is* $L(n)^{\sqrt{2}+o(1)}$. *The space required is* $L(n)^{1/\sqrt{2}+o(1)}$.

REMARKS. This result follows in a fashion similar to Theorem 4.2. Although the variation does not change the exponent in the running time, it has proved useful in practice. WUNDERLICH [34], [35] claims a running time of $L^{\sqrt{3/2}}$ for this variation. The reason for the difference is that Wunderlich assumes, contrary to the spirit of Theorem 4.1, that considering Q_i with at most one prime in $(L^a,L^{2a}]$ and all other primes in $(1,L^a]$ is not much different from considering Q_i with all prime factors in $(1,L^{2a}]$.

As we have argued above, we expect to use $L^{a+(4a)^{-1}}$ values of Q_i in the implementation of the continued fraction algorithm. If we replace trial division up to L^a with the Pollard-Strassen method (see Section 4), the time spent on each Q_i is $L^{a/2}$. Thus the total running time will be

$$L^{\max\{3a/2+(4a)^{-1},3a\}}.$$

Choosing $a = 1/\sqrt{6}$, we have

THEOREM 5.3. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm augmented with the Pollard-Strassen method is* $L(n)^{\sqrt{3/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/3}+o(1)}$.

REMARKS. SCHNORR [26] obtains $L^{\sqrt{3}}$ as a running time upper bound for this variation. Again, the reason for this difference is that he uses a weaker form of Theorem 2.1. SCHROEPPEL [28] achieves a running time of $L^{\sqrt{3/2}}$ for the continued fraction algorithm with the Pollard-ρ method. He assumes Theorem 2.1 without proof and also the Pollard-ρ method is assumed.

We now consider the continued fraction algorithm with k early aborts and parameters $a, c_1,\ldots,c_k, \theta_1,\ldots,\theta_k$ (which satisfy (4.5)). By this we mean that a value of Q_j is aborted if after trial division to $L^{\theta_i a}$ the unfactored portion exceeds $n^{\frac{1}{2}(1-c_1-\ldots-c_i)}$. The analysis is the same as in Section 4 except that we now take advantage of the fact that each $|Q_j| < 2\sqrt{n}$. The functions $f_1,\ldots,f_{k+1}$ are replaced by $F_1,\ldots,F_{k+1}$ where

$$F_i = a + \theta_i a + c_i(4\theta_i a)^{-1} + \ldots + c_k(4\theta_k a)^{-1} + (1 - c_1 - \ldots - c_k)(4a)^{-1}$$

for $i = 1,\ldots,k$ and

$$F_{k+1} = 2a + (1 - c_1 - \ldots - c_k)(4a)^{-1}.$$

We have

THEOREM 5.4. *Suppose* $1/10 < a < 1$, $c_1 > 0,\ldots,c_k > 0$, $c_1 + \ldots + c_k < 1$, *and* $0 < \theta_1 < \ldots < \theta_k < 1$. *Then assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with* k *early aborts and parameters* $a,c_1,\ldots,c_k, \theta_1,\ldots,\theta_k$ *is* $L(n)^{\max\{F_1,\ldots,F_{k+1},3a\}+o(1)}$.

The optimal choice of parameters occurs when the F_i are all equal. This choice is

$$a = (6+\frac{2}{k+1})^{-1/2}, \quad c_i = \frac{4ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

With this choice of parameters, we call the algorithm the continued fraction algorithm with k early aborts.

THEOREM 5.5. *Assuming Hypotheses* 5.1 *and* 5.2, *the continued fraction algorithm with* k *early aborts has running time* $L(n)^{\sqrt{3/2+1/(2k+2)}+o(1)}$. *The space required is* $L(n)^{2/\sqrt{6+2/(k+1)}+o(1)}$.

Letting $k \to \infty$ slowly with n, we call the resulting algorithm the continued fraction algorithm with the early abort strategy.

THEOREM 5.6. *Assuming Hypotheses* 5.1 *and* 5.2, *the continued fraction algorithm with the early abort strategy has running time* $L(n)^{\sqrt{3/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/3}+o(1)}$.

When we combine k early aborts with the Pollard-Strassen method, the analysis of the running time requires that we change the functions $F_1,\ldots,F_{k+1}$ to $G_1,\ldots,G_{k+1}$ where

$$G_i = F_i - \frac{1}{2}\theta_i a \qquad \text{for } i = 1,\ldots,k,$$

$$G_{k+1} = F_{k+1} - \frac{1}{2}a.$$

As in Section 4, the elimination method we use now plays a role. If we use an elimination method with exponent $r > 5/2 + 1/(2k+2)$, then an optimal choice of parameters is

$$a = (4r-5-\frac{1}{k+1})^{-1/2}, \quad c_i = \frac{2ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

If we use an elimination method with exponent $r \le 5/2 + 1/(2k+2)$, then an optimal choice of parameters is

$$a = (5+\frac{1}{k+1})^{-1/2}, \quad c_i = \frac{2ia^2}{(k+1)^2}, \quad \theta_i = \frac{i}{k+1}.$$

We have

THEOREM 5.7. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with the Pollard-Strassen method,* k *optimally chosen early aborts, and an elimination method with exponent* r *is*

$$L(n)^{r(4r-5-1/(k+1))^{-1/2}+o(1)}, \quad \text{if } r > \frac{5}{2} + \frac{1}{2k+2},$$

$$L(n)^{(5/4+1/(4k+4))^{1/2}+o(1)}, \quad \text{if } r \le \frac{5}{2} + \frac{1}{2k+2}.$$

Finally, letting $k \to \infty$ slowly with n, we have

THEOREM 5.8. *Assuming Hypotheses* 5.1 *and* 5.2, *the running time for the continued fraction algorithm with the Pollard-Strassen method, the early abort strategy, and an elimination method with exponent* r *is*

$$L(n)^{\sqrt{r^2/(4r-5)}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{5/4}+o(1)}, \quad \text{if } r \le 5/2.$$

The space required is

$$L(n)^{\sqrt{4/(4r-5)}+o(1)}, \quad \text{if } r > 5/2,$$

$$L(n)^{\sqrt{4/5}+o(1)}, \quad \text{if } r \le 5/2.$$

6. SCHROEPPEL'S LINEAR SIEVE ALGORITHM

Several years ago SCHROEPPEL [28] advanced a variation of the continued fraction algorithm that did away with continued fractions! His idea was to find another means of producing residues near $\sqrt{n}$ that had the advantage that they could collectively be factored by a sieve, much like the sieve of Eratosthenes. Unfortunately, these residues are not specifically quadratic residues. This requirement is arranged for in the elimination step. In this section I shall describe Schroeppel's algorithm and give a heuristic running time analysis. In addition, I shall describe a variation that gives an improvement in the running time.

Let $K = [\sqrt{n}]$ and consider the function

$$S(A,B) = (K+A)(K+B) - n,$$

where A, B are integers much smaller in absolute value than K. Then S(A,B) is not many times bigger than $\sqrt{n}$. Specifically, if $\alpha \le \beta$ are positive constants and

$$|A| \le L^{\alpha}, \quad |B| \le L^{\beta}, \quad A \le B,$$

then

$$|S(A,B)| \le n^{1/2}(L^{\alpha}+L^{\beta}+2) = n^{1/2}L^{\beta},$$

so that $|S(A,B)| \le n^{1/2+o(1)}$. We would like to find pairs A_i, B_i such that $\prod_{i=1}^{k} S(A_i,B_i)$ is a square, as in Dixon's algorithm and the continued fraction algorithm. But now, in addition, we want each distinct value of $A_1,\ldots,A_k$, $B_1,\ldots,B_k$ to be assumed an even number of times. Then $\prod_{i=1}^{k} (K+A_i)(K+B_i)$ is also a square. So if we can find such a fortuitous collection of pairs A_i, B_i, then we can find integers x, y such that $x^2 \equiv y^2 \bmod n$. Thus, as before, we have a good chance for (x+y,n) being a non-trivial factor of n.

To find such a collection of pairs A_i, B_i we try to factor the S(A,B) with the primes up to L^a by a sieve procedure described below. To each $S(A_0,B_0)$ composed solely of the primes $p \le L^a$, we associate a 0-1 vector with

$$V \overset{\text{def}}{=\!=} 1 + \pi(L^a) + ([L^{\alpha}]+[L^{\beta}]+1)$$

coordinates as follows. The first coordinate is 1 if $S(A_0,B_0) < 0$ and 0 otherwise. The next $\pi(L^a)$ coordinates are given by the parity of the exponents of the primes $p \le L^a$ in the prime factorization of $S(A_0,B_0)$. The last $[L^{\alpha}] + [L^{\beta}] + 1$ coordinates are all 0, except for the A_0-th and B_0-th coordinates, which are 1 (unless $A_0 = B_0$, in which case this coordinate is 0). If we can find V+1 such vectors, they must be linearly dependent. A linear dependency corresponds to a set of pairs A_i, B_i, i = 1,...,k such that

$$\prod_{i=1}^{k} S(A_i,B_i) \quad \text{and} \quad \prod_{i=1}^{k} (K+A_i)(K+B_i)$$

are both squares.

If $A = A_0$ is fixed, then $S(A_0,B)$ is a linear function in the variable B. Thus for fixed A_0, the set of $S(A_0,B)$ for $A_0 \le B \le L^{\beta}$ can be factored by a sieve. What this means is that if $p|S(A_0,B)$, then $p|S(A_0,B+kp)$ for every integer k and if p is prime and $p \nmid n$, then $p|S(A_0,B')$ implies $B' = B+kp$ for some k. Thus once we find one B with $S(A_0,B)$ divisible by p we can easily mark every $S(A_0,B)$ which is divisible by p. The time it takes to do this is essentially just the number of such multiples of p, which is $(L^{\beta}-A_0)/p$. If we repeat this procedure for each $p \le L^a$, the total time required for a fixed A_0 is

$$(L^{\beta}-A_0) \sum_{p \le L^a} 1/p \sim (L^{\beta}-A_0)\,\text{loglog}\, L^a.$$

Now summing over all $|A_0| \le L^{\alpha}$, we have the total time spent sieving is $L^{\alpha+\beta}$. That is, in time $L^{\alpha+\beta}$ we can ascertain all of the A, B such that S(A,B) is composed solely of the primes $p \le L^a$. We have ignored the problem of finding the first B such that $p|S(A_0,B)$, but this can be quickly done since it just involves solving a linear congruence mod p. In addition, we have ignored the problem of finding the exact exponent on p in the prime factorization of $S(A_0,B)$. This can be found by either sieving with the higher powers of p, by trial division by the higher powers of p on those $S(A_0,B)$ which are multiples of p, or a combination of both approaches. The running time with any of these methods does not change our above calculation of $L^{\alpha+\beta}$.

We would like to know how many completely factored values of S(A,B) we will find. We now make a reasonable assumption about the numbers S(A,B).

HYPOTHESIS 6.1. *The numbers* $|S(A,B)|$ *for* $|A| \le L(n)^{\alpha}$, $A \le B \le L(n)^{\beta}$ *are distributed with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in* $[1,n^{1/2}(L(n)^{\alpha}+L(n)^{\beta}+2)]$.

With this hypothesis, we therefore expect that the probability that S(A,B) completely factors over the primes up to L^a is

$$\psi(n^{1/2}L^{\beta},L^a)/(n^{1/2}L^{\beta}) = L^{-(4a)^{-1}}$$

by Theorem 2.1. Thus we expect a total of $L^{\alpha+\beta-(4a)^{-1}}$ completely factored values of S(A,B). However, we must produce $V+1 = L^{\max\{a,\beta\}}$ completely factored S(A,B)'s. Thus our parameters a, α, β should be chosen so that

$$\alpha + \beta - (4a)^{-1} = \max\{a,\beta\}. \tag{6.1}$$

An immediate consequence of (6.1) is that

$$\alpha + \beta \geq a + (4a)^{-1} \geq 1,$$

so that $\beta \geq 1/2$ and, a fortiori,

$$\max\{a,\beta\} \geq 1/2.$$

The value 1/2 can be attained by making the choice $a = \alpha = \beta = 1/2$, which also satisfies (6.1).

For the elimination step, we must work with a matrix of size $(V+1) \times V$ where $V = L^{\max\{a,\beta\}} \geq L^{1/2}$. Thus this step has running time at least $L^{r/2} \geq L$ if we use an elimination method with exponent r. Again the choice $a = \alpha = \beta = 1/2$ allows us to attain the lower bound. We have

THEOREM 6.1. *Assuming Hypotheses* 6.1 *and* 5.2, *the running time for Schroeppel's algorithm with an elimination method with exponent* r *is* $L(n)^{r/2+o(1)}$.

REMARK. This result was also obtained by MONIER [20] who gave separate running times of L for the sieve step and $L^{3/2}$ for the elimination step (using Gaussian elimination). Monier assumed our Theorem 2.1 without proof. SCHROEPPEL [28] claims a running time of L, but he ignores the elimination step.

Of the three speed-up variations discussed in Section 4 only Large Prime is applicable with Schroeppel's algorithm. But, for the same reason as before, this variation does not lower the exponent on the running time. In any event, this variation, as are the others, is aimed at reducing the time it takes to obtain completely factored residues. In Schroeppel's algorithm, however, the roadblock is the elimination step. A successful variation would be one that can speed up elimination.

The $(V+1) \times V$ matrix that we work with is very sparse - most entries are 0. In fact, in each row at most 2 entries among the last $[L^{\alpha}] + [L^{\beta}] + 1$ are non-zero. It would be nice to take advantage of this fact in the elimination procedure, but it is not apparent how to do this. If there were just 1 non-zero entry, rather than 2, we could proceed with a cull as described in Section 4 in connection with the Large Prime variation. We now describe an alternative method of assigning a vector to a completely factored S(A,B) that will indeed let us use a cull. To each $S(A_0,B_0)$ composed solely of the primes

$p \leq L^a$ we assign a 0 - 1 vector with

$$V' \stackrel{\text{def}}{=} 1 + \pi(L^a) + (2[L^{\alpha}] + 1) + ([L^{\alpha}] + [L^{\beta}] + 1)$$

coordinates as follows. The first $1 + \pi(L^a)$ coordinates are as before. The next $2[L^{\alpha}] + 1$ coordinates are all 0 except for the A_0-th coordinate which is 1. Similarly the last $[L^{\alpha}] + [L^{\beta}] + 1$ coordinates are all 0 except for the B_0-th coordinate which is 1.

With this different method of assigning 0 - 1 vectors to factored S(A,B)'s we can use a cull. In fact in each vector, exactly one entry among the last $[L^{\alpha}] + [L^{\beta}] + 1$ entries is non-zero. Thus we first order the rows so that the last $[L^{\alpha}] + [L^{\beta}] + 1$ columns appear in echelon form. We then make one pass down the matrix. If a B-value (that is, a non-zero entry in the last $[L^{\alpha}] + [L^{\beta}] + 1$ columns) is not repeated in a neighbouring row, then this row is culled out. If it is repeated, say k times, then the first such row is subtracted from the other k - 1 rows and is then culled out. Thus in time $L^{\max\{a,\beta\}}$ we can eliminate the last $[L^{\alpha}] + [L^{\beta}] + 1$ columns from our matrix. If we now choose a, α, β so that $\beta > \max\{a,\alpha\}$, this idea could produce a net speed-up for the algorithm. (To achieve the time $L^{\max\{a,\beta\}}$ for the cull, the 0 - 1 vectors must be treated as sparse vectors as was remarked in connection with the Large Prime variation of Dixon's algorithm.)

The running time for the sieve step will still be $L^{\alpha+\beta}$. The running time for the cull is, as we have seen, L^{β}. The running time for the elimination step in the smaller matrix is $L^{r\cdot\max\{a,\alpha\}}$. Thus the total running time for Schroeppel's algorithm with a cull is

$$L^{\max\{\alpha+\beta,ra,r\alpha\}}. \tag{6.2}$$

However, if we insist on $V' + 1 = L^{\beta}$ completely factored values of S(A,B), we do not get an improvement. That is, if we still have (as in (6.1))

$$\alpha + \beta - (4a)^{-1} \geq \beta,$$

then

$$\max\{a,\alpha\} \geq \max\{a,(4a)^{-1}\} \geq 1/2. \tag{6.3}$$

Thus from (6.2), the running time would still be at least $L^{r/2}$.

But do we really need $V'+1$ completely factored $S(A,B)$'s? After the cull procedure we actually need only $W+1$ vectors, where

$$W \stackrel{\text{def}}{=\!=} 1 + \pi(L^a) + (2[L^\alpha]+1).$$

So the question is now, how many completely factored $S(A,B)$'s do we need to be assured (or nearly assured) of having $W+1$ vectors left after the cull procedure. From (6.3) we may as well assume that $\alpha < (4a)^{-1}$, for otherwise our running time is again at least $L^{r/2}$. Thus the number of factored $S(A,B)$'s is L^γ where

$$\gamma \stackrel{\text{def}}{=\!=} \alpha + \beta - (4a)^{-1} < \beta.$$

If we have L^γ objects randomly placed in L^β cells and if we cull out one object per occupied cell, how many objects do we expect to be remaining? In fact it is $\frac{1}{2}L^{2\gamma-\beta}$ as we now see.

LEMMA 6.1. *Let* y, x *be positive integers and let* S *denote the set of functions* f: $\{1,2,\ldots,y\} \to \{1,2,\ldots,x\}$. *If* $f \in S$, *let* $Y(f) = y - |\mathrm{Rng}(f)|$, *where* $\mathrm{Rng}(f)$ *denotes the image of* f. *Then with the uniform distribution on* S, *the expected value of* Y *is*

$$y - x + x(1-x^{-1})^y$$

and the variance is

$$x(1-x^{-1})^y - x(1-2x^{-1})^y + x^2\{(1-2x^{-1})^y - (1-x^{-1})^{2y}\}.$$

In particular, if $y = x^\delta$ *where* $0 < \delta < 1$ *is fixed, then as* $x \to \infty$,

$$E(Y) = \tfrac{1}{2}x^{2\delta-1} + O(x^{\delta-1} + x^{3\delta-2}) = \sigma^2.$$

PROOF. If $f \in S$, let $X(f) = x - |\mathrm{Rng}(f)| = x - y + Y(f)$. It will be more convenient for us to compute the expectation and variance of X. Note that the variance of X is clearly the same as the variance of Y and $E(X) = x - y + E(Y)$.

Let $X_i(f) = 0$ if $i \in \mathrm{Rng}(f)$ and 1 otherwise. Then $X = X_1 + \ldots + X_x$, so that

$$E(X) = E(X_1) + \ldots + E(X_x) = xE(X_1) = x(1-x^{-1})^y$$

and our assertion about $E(Y)$ follows. Also

$$E(X^2) = \sum_{i=1}^{x}\sum_{j=1}^{x} E(X_iX_j).$$

Now $E(X_i^2) = E(X_i) = (1-x^{-1})^y$. Also if $i \neq j$, then $E(X_iX_j) = (1-2x^{-1})^y$. Thus

$$E(X^2) = x(1-x^{-1})^y + (x^2-x)(1-2x^{-1})^y$$

and the rest of our assertions follow from some simple calculations.

Applying the lemma with $y = L^\gamma$, $x = L^\beta$, $\delta = \gamma/\beta$, we thus see that after the cull procedure we almost surely have $L^{2\gamma-\beta}$ vectors left. Thus we should choose a, α, β so that $L^{2\gamma-\beta} = W+1 = L^{\max\{a,\alpha\}}$. That is,

$$2\alpha + \beta - (2a)^{-1} = \max\{a,\alpha\}.$$

Thus $2\alpha + \beta - (2a)^{-1} \geq \alpha$, so that

$$\alpha + \beta \geq (2a)^{-1}.$$

Hence the exponent in the running time, given by (6.2), satisfies

$$\max\{\alpha+\beta, ra, r\alpha\} \geq \max\{\alpha+\beta, ra\} \geq \max\{(2a)^{-1}, ra\} \geq \sqrt{r/2}.$$

Moreover, $\sqrt{r/2}$ can be attained by choosing

$$a = \alpha = 1/\sqrt{2r}, \quad \beta = (r-1)/\sqrt{2r}.$$

With this choice we have our other requisite as well, namely $\alpha < (4a)^{-1}$.

Note that the space requirement is apparently $L^{(r+2)/\sqrt{8r}}$, the size of the matrix before the cull. But, in fact, less space is required. Until now we have made the "natural" choice of fixing each A and sieving over the B's - natural because it makes sense to sieve over as long an interval as

possible. If instead we fix B and sieve over the A's then all of the factored S(A,B)'s with a given value of B are found at one time and the cull procedure can be immediately applied to this smaller set. Thus the space required is the size of the matrix after the cull, or $L^{\sqrt{2/r}}$. (This is also the size of the interval sieved.)

THEOREM 6.2. *Assuming Hypotheses* 6.1 *and* 5.2 *and that the function* f *that takes a completely factored* S(A,B) *to* B *is pseudo-random with respect to the property of Lemma* 6.1, *then the running time for Schroeppel's algorithm with a cull and with an elimination method of exponent* r *is* $L(n)^{\sqrt{r/2}+o(1)}$. *The space required is* $L(n)^{\sqrt{2/r}+o(1)}$.

REMARK. The Large Prime variation cannot be profitably used with Schroeppel's algorithm with a cull. Indeed, this strategy requires a cull of its own and a cull on one set of data destroys the ability to perform a cull on another set of data.

7. THE QUADRATIC SIEVE ALGORITHM

The advantage of Schroeppel's linear sieve algorithm over the previous algorithms is that a sieve process is substituted for trial division or the Pollard-Strassen method. The disadvantage of Schroeppel's algorithm, however, is that the residues produced are not quadratic residues. The task of producing quadratic residues is given over to the elimination step which then enters the picture as an important time bottleneck and as a large user of space. In this section I shall describe what I call the quadratic sieve algorithm - it shares the advantage of Schroeppel's algorithm, but not the disadvantage.

Briefly, the quadratic sieve algorithm can be described as the case A = B of Schroeppel's algorithm. That is, if $K = [\sqrt{n}]$, we let

$$S(A) = (K+A)^2 - n,$$

so that S(A) = S(A,A) in the notation of Section 6. Then, quite evidently, S(A) is a quadratic residue mod n. These quadratic residues, as with the continued fraction algorithm, do not lie much above $\sqrt{n}$. In fact, if $|A| \le L^b$, then

$$|S(A)| \le \sqrt{n}(2L^b + 2) = \sqrt{n}L^b.$$

What is not so obvious is that the S(A) can be factored with a sieve, but this is nevertheless the case. In fact, the consecutive values of any polynomial can be factored by a sieve-type procedure. We describe the procedure for our particular polynomial S(A). For each prime p, we find the solutions of the congruence

$$S(A) \equiv 0 \bmod p. \tag{7.1}$$

Because of the nature of S(A), (7.1) has solutions precisely when (n/p) = 1, p|n, or p = 2. Say (n/p) = 1. Then (7.1) has two solutions $A_1(p)$, $A_2(p)$. (These solutions may be found by the probabilistic method of BERLEKAMP [2] or LEHMER [17] in time bounded by $O((\log p)^c)$ or by trial and error in time bounded by $O(p(\log p)^c)$.) Then (7.1) holds if and only if

$$A \equiv A_1(p) \bmod p \quad \text{or} \quad A \equiv A_2(p) \bmod p.$$

Thus if we have a list of consecutive values of S(A), we can quickly find all of them which are multiples of p by marking every p-th one starting at some $A_1 \equiv A_1(p) \bmod p$ and by marking every p-th one starting at some $A_2 \equiv A_2(p) \bmod p$. Thus, our sieve procedure requires two passes down the list for each p for which (n/p) = 1. Divisibility by higher powers of p can be decided, as in Section 6, by either trial division, sieving by higher powers of p, or by a combination approach.

Say now we work with the primes $p \le L^a$ where $1/10 < a < 1$. As in Section 3 we assume that n is divisible by no prime $p \le L$. Say we consider the S(A) for $|A| \le L^b$. The problem of deciding the correct power of 2 in each S(A) is simple - this can be done by a sieve procedure or trial division in time L^b. So say now $2 < p \le L^a$. We solve (7.1) for each such p, finding when (n/p) = 1 the two solutions $A_1(p)$, $A_2(p)$. The time required for this step is either L^a or L^{2a} depending on whether we use a clever or stupid method (we shall assume the latter). We then use the above-described sieve to determine for each such p and each A with $|A| \le L^b$, the correct exponent on p in the prime factorization of S(A). As in Section 6, the time required is

$$\sum_{\substack{p \le L^a \\ (n/p)=1}} L^b/p = L^b,$$

provided there are at least a few small primes p for which (n/p) = 1.

The question now, as it has been in all of our previous algorithms, is how many of our residues are composed solely of the primes $p \le L^a$?

HYPOTHESIS 7.1. *There is a constant n_2, such that if $n \ge n_2$, we have the following. For any a, $1/10 < a < 1$, the number of primes $p \le L(n)^a$ for which $p \nmid n$ and $(n/p) = 1$ is at least $\pi(L(n)^a)/3$. For any $b > 1/10$, the numbers $|S(A)|$ with $|A| \le L(n)^b$ are distributed with respect to a certain fraction of them having all of their prime factors below some point as are all of the integers in $[1, \sqrt{n}(2L(n)^b + 2)]$.*

With this hypothesis and Theorem 2.1, we can indeed predict how many of the L^b values of $S(A)$ are composed solely of the primes $p \le L^a$. Namely, there are $L^{b-(4a)^{-1}}$ such values.

The elimination stage of the algorithm is exactly the same as with Dixon's algorithm or the continued fraction algorithm. We thus wish to choose b so that we have L^a factored values of $S(A)$. Thus we choose $b = a + (4a)^{-1}$. If we use an elimination algorithm with exponent r, the running time for the quadratic sieve algorithm will be

$$L^{\max\{2a, a+(4a)^{-1}, ra\}}.$$

The space requirement is L^{2a} for the matrix and $L^{a+(4a)^{-1}}$ for the sieve. However the latter requirement can be eased by breaking our long interval into smaller intervals of length L^{2a}. There is a certain start-up cost for sieving a new interval, but this is $\le L^{2a}$. Thus without changing the running time we can thus get by with a space requirement of L^{2a}. We choose

$$a = 1/\sqrt{4(r-1)}$$

and obtain

THEOREM 7.1. *Assuming Hypotheses 7.1 and 5.2, the running time for the quadratic sieve algorithm using an elimination method with exponent r is $L(n)^{r/\sqrt{4r-4}+o(1)}$. The space required is $L(n)^{1/\sqrt{r-1}+o(1)}$.*

REMARKS. The function $r/\sqrt{4r-4} = 1 + O((r-2)^2)$ as $r \to 2$. Thus this exponent is not very sensitive to changes in r. In fact the running time exponent for the quadratic sieve algorithm with Gaussian elimination is only about 4% higher than the exponent with Coppersmith-Winograd elimination. There

are several possible variations of the quadratic sieve algorithm, but none of these change the exponent in the running time. Among these are the Large Prime variation discussed in Section 4, the use of a multiplier ℓ so that ℓn is a quadratic residue for more small primes than is n, and the use of the polynomials $S_i(A) = ([i\sqrt{n}] + A)^2 - i^2 n$ for various small i.

8. ELIMINATION ALGORITHMS

In Section 4 we said that an algorithm for finding a linear dependency among a set of vectors in $(\mathbb{Z}/2\mathbb{Z})^k$ has exponent r if the running time is $k^{r+o(1)}$. A more usual notion though is that of a matrix multiplication exponent. That is, an algorithm for multiplying two k×k matrices has exponent r if the number of arithmetic operations involved is $k^{r+o(1)}$. (Note that when working over $\mathbb{Z}/2\mathbb{Z}$, the number of arithmetic operations is proportional to the running time.) In this section we show that a matrix multiplication algorithm of exponent r can be used to construct an elimination algorithm of exponent r.

But first we say a word about matrix multiplication. The naive algorithm clearly has exponent 3. About 12 years ago, Schönhage introduced a matrix multiplication algorithm with exponent smaller than 3. His method is based on a tricky way of multiplying 2×2 matrices over a non-commutative ring using only 7 multiplications and a bounded number of additions. Iterating this procedure for larger matrices gives rise to an algorithm with exponent $\log 7/\log 2 \approx 2.807$. This result began a long series of better and better algorithms. The current champions are COPPERSMITH and WINOGRAD [8] who have shown the existence of an algorithm with exponent smaller than 2.495548. For the history before this development, the reader is referred to the survey of LAZARD [16].

So say now we are working with a particular matrix multiplication algorithm M that requires M(k) arithmetic operations to multiply two general k×k matrices. Suppose S is a set of k+1 vectors each with k coordinates in $\mathbb{Z}/2\mathbb{Z}$. If $T \subset S$, let A_T denote the matrix whose rows are the vectors in T.

Suppose $m \le k$, m is a power of 2, and A is an m×k matrix. In AHO, HOPCROFT, and ULLMAN [1], Fig. 6.4, an algorithm is described to find matrices L, U, P where L is m×m unit lower triangular, U is m×k upper triangular, and P is a k×k permutation matrix, such that

$$A = LUP,$$

provided A has rank m; if A has rank < m, the algorithm will indicate this fact as well. Moreover, if M is used for all matrix multiplications, the running time will be $O(M(k))$. By embedding A in a possibly larger matrix, we can drop the restriction that m be a power of 2.

By using this algorithm on A_T for T a proper subset of S, we can find whether the vectors in T are independent or dependent. Thus with a binary search, we can find in $O(M(k)\log k)$ arithmetic steps a subset $T_0 \subset S$ and a vector $v_0 \in S - T_0$, such that T_0 is independent but $T_0 \cup \{v_0\}$ is dependent. Thus v_0 is in the span of T_0.

Say $T_0 = \{v_1,\ldots,v_m\}$. Our problem is to find $x_1,\ldots,x_m$ in $\mathbb{Z}/2\mathbb{Z}$ such that $x_1v_1 + \ldots + x_mv_m = v_0$, that is,

$$[x_1 \ldots x_m]A_{T_0} = v_0.$$

But from the decomposition $A_{T_0} = L_0U_0P_0$, which we can find from the above algorithm, this equation is easy to solve. Since P_0 is a permutation matrix, its inverse can be found in time $O(k)$ (see [1], p.239). Moreover, since U_0 is upper triangular, a solution $y_1,\ldots,y_m$ to

$$[y_1 \ldots y_m]U_0 = v_0P_0^{-1},$$

if a solution exists (and it will in our case), can be found by a back substitution technique in $O(k^2)$ arithmetic operations. We can similarly solve

$$[x_1 \ldots x_m]L_0 = [y_1 \ldots y_m].$$

In summary, we can find the linear dependency we need for our integer factorization algorithms in $O(M(k)\log k)$ arithmetic operations. Thus if $M(k) = k^{r+o(1)}$, then the running time for the corresponding dependency finding algorithm also is $k^{r+o(1)}$.

9. PRACTICAL CONSIDERATIONS

We remarked in Section 1 that an asymptotic running time analysis is by no means the last word on an algorithm. In this section I speculate on the "real world" applicability of the various ideas in this paper.

To begin, I am doubtful that a substitute for Gaussian elimination should be used. Because we are working with matrices over $\mathbb{Z}/2\mathbb{Z}$, computers

can be programmed to handle long strings of entries as a single multi-digit object when one row is subtracted from another. Thus, although Gaussian elimination requires ck^3 steps for a k×k matrix, the coefficient c can be made fairly small. Perhaps a more important problem than speeding up the elimination step is to find a convenient way of solving the space problem. In each algorithm discussed, the space constraint is caused by the problem of working with the final matrix. Perhaps the research in fast matrix multiplication can help with this problem since these ideas involve iterative procedures where only pieces of a matrix are needed in core at any given time.

I am also doubtful that the Pollard-Strassen method should be used to replace trial division in the continued fraction algorithm. However use of the Pollard-ρ method is a distinct possibility. The asymptotic time analysis should be the same. The reason we used the Pollard-Strassen method in our discussions is because it is fully proved and deterministic.

I am optimistic about the early abort strategy either by itself or supplemented with the Pollard-ρ method. Samuel Wagstaff and I did a statistical analysis of the data computed during his 70 hour factorization of

$$\frac{3^{121}-1}{(3^{11}-1)11617}$$

using the continued fraction algorithm with Large Prime. We predict that if everything were left the same except that after trial division to 100 a value of Q_i gets aborted if the unfactored portion exceeds 10^{19}, then the running time would have been only 20 hours. In further fine tuning of the early abort strategy, Wagstaff and I are currently achieving about a 10 fold speed up over the straight continued fraction algorithm with Large Prime for numbers of about 50 decimal digits.

In [36], WUNDERLICH suggests programming the continued fraction algorithm on a parallel processor such as the English ICL-DAP. In particular he suggests that the pair $a_i \bmod n$, Q_i be computed on a sequential processor and then a large batch of Q_i's be trial divided in parallel. The assumption is that the $a_i \bmod n$, Q_i generation only accounts for a small percentage of the total running time, so that it is not too important to improve this part of the program. While this is undoubtedly true asymptotically, in practice the $a_i \bmod n$, Q_i generation does take a significant amount of time, say 5 to 10%. Moreover, if the early abort strategy is utilized, then less time is spent on average with a pair $a_i \bmod n$, Q_i, so that the time spent generating the pairs may take even 30% of the total running time. There are ways to

speed up the generation of these pairs (for example, the costly reduction of a_i modulo n can be delayed so that it is done only when $i \equiv 1,2 \bmod 100$), but it may be advantageous to find a way to produce the $a_i \bmod n$, Q_i pairs in parallel. Using an idea of SHANKS [29], it is in fact possible to cheaply jump ahead in the continued fraction expansion from Q_i to Q_j where $j \approx 2i$ when i is large. Thus, an alternative to working with an exotic parallel processor might be to simultaneously implement the continued fraction algorithm on a large bank of unextraordinary computers, each sequentially working on a different interval of consecutive terms in the continued fraction expansion.

Concerning the two sieve algorithms, I feel that the quadratic sieve is superior to the linear sieve. Even so, I am not sure that the quadratic sieve is practical. The main drawback for the quadratic sieve as compared with the continued fraction algorithm appears to be the size of the quadratic residues S(A). Most of the $|S(A)|$ are about $n^{1/2+\varepsilon}$ where $\varepsilon \to 0$ as $n \to \infty$. But for a fixed n, say about 10^{50} or 10^{60}, ε may well be as large as 1/6. Thus the quadratic residues would be near $n^{2/3}$ for such n. In comparison, the continued fraction algorithm's quadratic residues Q_i always satisfy $|Q_i| < 2n^{1/2}$ and some are much smaller. A larger magnitude for the quadratic residues of course makes it less likely they will completely factor over a set of small primes. On the other hand, an advantage of the quadratic sieve algorithm over the continued fraction algorithm is that with the former almost all of the steps can be performed with single precision numbers. Indeed, when sieving, every time there is a "hit" with a prime p (or with a prime power p^j), instead of dividing p into S(A) to produce the quotient, one can instead subtract log p from log S(A) where these logs are only singly precise. If after several subtractions, the value of the logarithm left is close to 0, then S(A) has been completely factored, while if the value is below a certain bound, then S(A) has factored but for a single large prime which can be found by division. The overwhelming majority of values of S(A) are not completely factored; for them no division is necessary. GERVER [10] has recently used the quadratic sieve algorithm to factor a 47 digit composite factor of $3^{225} - 1$ taking 70 hours on an HP3000. It remains to be seen if fine tuning the algorithm can produce better results. A final remark is that it would be very easy to simultaneously use many computers with the quadratic sieve algorithm (at least for the sieving step) since each computer can sieve over its own interval.

REFERENCES

[1] AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[2] BERLEKAMP, E.R., *Factoring polynomials over large finite fields*, Math. Comp. 24 (1970), 713-735.

[3] BORODIN, A. & I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York, 1975.

[4] BRENT, R.P. & J.M. POLLARD, *Factorization of the eighth Fermat number*, Math. Comp. 36 (1981), 627-630.

[5] BRILLHART, J., D.H. LEHMER, J.L. SELFRIDGE, B. TUCKERMAN & S.S. WAGSTAFF, JR., *Factorizations of* $b^n \pm 1$ *up to High Powers*, to be published by the Amer. Math. Soc.

[6] BRUIJN, N.G. DE, *On the number of positive integers* $\leq x$ *and free of prime factors* $> y$, *II*, Nederl. Akad. Wet. Proc. Ser. A 69 = Indag. Math. 38 (1966), 239-247.

[7] CANFIELD, E.R., P. ERDÖS & C. POMERANCE, *On a problem of Oppenheim concerning "Factorisatio Numerorum"*, J. Number Theory, to appear.

[8] COPPERSMITH, D. & S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput. 11 (1982), 472-492.

[9] DIXON, J.D., *Asymptotically fast factorization of integers*, Math. Comp. 36 (1981), 255-260.

[10] GERVER, J.L., *Factoring large numbers with a quadratic sieve*, Math. Comp., to appear.

[11] GUY, R.K., *How to factor a number*, Proc. Fifth Manitoba Conf. Numer. Math., Utilitas, Winnipeg (1975), 49-89.

[12] HOOGENDOORN, P.J., *On a secure public-key cryptosystem*, this volume.

[13] KNUTH, D.E., *The Art of Computer Programming*, vol. 2 *Seminumerical Algorithms*, 2nd edition, Addison Wesley, Reading, Mass., 1981.

[14] KNUTH, D.E., *The distribution of continued fraction approximations*, to appear.

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

[16] LAZARD, D., *Sur le produit de matrices*, Gazette des Math., No. 15 (Dec. 1980), 27-48.

[17] LEHMER, D.H., *Computer technology applied to the theory of numbers*, in W.J. LeVeque, ed., MAA Stud. Math. 6 (1969), 117-151.

[18] LEHMER, D.H. & R.E. POWERS, *On factoring large numbers*, Bull. Amer. Math. Soc. 37 (1931), 770-776.

[19] LENSTRA, H.W. JR., *Primality testing*, this volume.

[20] MONIER, L., *Algorithmes de factorisation d'entiers*, Thèse de 3^{me} cycle, Orsay (1980).

[21] MORRISON, M.A. & J. BRILLHART, *A method of factoring and the factorization of* F_7, Math. Comp. 29 (1975), 183-205.

[22] POLLARD, J.M., *Theorems on factorization and primality testing*, Proc. Cambridge Phil. Soc. 76 (1974), 521-528.

[23] POLLARD, J.M., *A Monte Carlo method for factorization*, BIT 15 (1975), 331-334.

[24] POMERANCE, C., *Recent developments in primality testing*, Math. Intelligencer, 3 (1981), 97-105.

[25] RABIN, M.O., *Probabilistic algorithm for primality testing*, J. Number Theory 12 (1980), 128-138.

[26] SCHNORR, C.P., *Refined analysis and improvements on some factoring algorithms*, J. Algorithms 3 (1982), 101-127. Also see the extended abstract in S. Even and O. Kariv, eds., *Automata, Languages and Programming*, Eighth Colloquium, Acre (Akko), July 1981, Lecture Notes in Computer Science #115, Springer-Verlag, 1-15.

[27] SCHOOF, R.J., *Quadratic fields and factorization*, this volume.

[28] SCHROEPPEL, R., private correspondences dated 1977 (communicated to the author by H.C. Williams).

[29] SHANKS, D., *The infrastructure of a real quadratic field and applications*, Proc. 1972 Number Theory Conf. Boulder, Colorado, 217-224.

[30] SOLOVAY, R. & V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput. 6 (1977), 84-85; erratum: 7 (1978), 118.

[31] STRASSEN, V., *Einige Resultate über Berechnungskomplexität*, Jahresber. Deutsch. Math.-Verein 78 (1976/77), 1-8.

[32] TURK, J.W.M., *Fast arithmetic operations on numbers and polynomials*, this volume.

[33] VOORHOEVE, M., *Factorization algorithms of exponential order*, this volume.

[34] WAGSTAFF, S.S. JR. & M.C. WUNDERLICH, *A comparison of two factorization methods*, unpublished manuscript.

[35] WUNDERLICH, M.C., *A running time analysis of Brillhart's continued fraction factoring method*, in M.B. Nathanson, ed., *Number Theory Carbondale 1979*, Lecture Notes in Math. 751 (1979), 328-342.

[36] WUNDERLICH, M.C., *A report on the factorization of 2797 numbers using the continued fraction method*, unpublished manuscript.