Integer Factorization using the Quadratic Sieve

Chad Seibert*
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN 56567
seib0060@morris.umn.edu

March 16, 2011

Abstract

We give a light introduction to integer factorization using the quadratic sieve. Although it is not the fastest known factorization algorithm, it provides a stepping stone for understanding the general number field sieve, the asymptotically fastest known algorithm. We explain the algorithm in detail and work out its complexity and give some empirical results.

1 Introduction

When we want to factor a large number N, we have several options. One of the most important is the idea of primality testing. There are algorithms that provably determine whether a number is prime, e.g. the AKS primality test. If the number is composite, we have to try something else.
Trial division is a factoring method where we take each prime less than $\sqrt{N}$ and see if it is a factor. This algorithm is extremely slow for large numbers N, such as a RSA modulus. This is already an enormous calculation if N is thirty digits long and it would take much longer than the life of the universe for number much larger than it.

2 Difference of Squares

A factorization method that has been around since the era of Pierre de Fermat is *Fermat's factorization algorithm*. Let N be a positive, odd integer. Every odd number can be represented as the difference of two squares: $N = a^2 - b^2$ for a and b greater than zero. Put another way, we have $N = (a-b)(a+b)$. We can now recast this as the problem of finding a and b. Rearranging, we obtain $a^2 - N = b^2$. For each $a > \lceil\sqrt{N}\rceil$, we compute $a^2 - N$ and check to see if the result is a perfect square. If it is, we have found a and b and the factors are simply $a - b$ and $a + b$. Notice that if $a = b$, then $a - b$ is zero and we haven't actually found a solution. Furthermore, if $a \equiv \pm b \mod N$, we still don't have a solution. Therefore, we forbade these solutions. Thus, we have the following theorem:

Theorem 2.1. *If there exist a and b such that $a^2 - N = b^2$ and $a \not\equiv \pm b \mod N$, then $\gcd(a \pm b, N)$ are its factors.*

Let us give an example of such a factorization. Let $N = 6969$ and $a = \lceil\sqrt{6969}\rceil = 84$. We compute $84^2 - 6969 = 87$, which is not a perfect square. We increment a and try again. We obtain $85^2 - 6969 = 256$, which is a perfect square. Therefore, $a = 85$ and $b = 16$. Since $85 \neq \pm 16$, we can use the result of Theorem 2.1. We obtain factors 69 and 101 by computing $\gcd(85 \pm 16, 6969)$.
Now, we ask a simple question: how well does this perform? We can see that if the factors aren't near the square root of N, we will have to iterate through many a to find such a b. But even if we found such a and b, how likely is it that $a \not\equiv \pm b \mod N$? The following lemma, taken from [6] provides us with that answer.

Lemma 2.2. *If N has at least two different odd prime factors, then more than half of the solutions to $a^2 \equiv b^2 \mod N$ with $\gcd(ab, N) = 1$ satisfy (2.1).*

Proof. For an odd prime power p^u, the congruence $y^2 \equiv 1 \mod p^u$ has exactly two solutions. Since N is divisible by at least two different odd primes, the congruence $y^2 \equiv 1 \mod N$ has at least four solutions. Label these y as $y_1, y_2, \ldots, y_s$ where $y_1 = 1, y_2 = -1$. Then a complete enumeration of all the pairs of residues a, b modulo N that are coprime to N and i take the values of $1, \ldots, s$. Two of these pairs have $i = 1, 2$ and $s - 2$ (out of s) of these pairs have $i = 3, \ldots, s$. The latter pairs satisfy (2.1) and since $s \geq 4$, we are done. □

This shows that if we do find such a and b, then it is likely that they do lead to a proper factorization of N. Therefore, the main question is how do we find such a and b. To motivate our answer, let us consider another example of Fermat's factorization algorithm. This time, let $N = 301$ and $a = 18$. We perform the following computations:

a	$a^2 - N$
18	23
19	60
20	99
21	140
22	183
23	228
24	275
25	324
26	375
27	428
28	483
29	540

We see that there are no perfect squares in the second column. How long will it take to find one? It could take quite a while, especially if the number is a *semi-prime*, the product of two prime numbers. It turns out that we have enough information to find a factorization. Although we have don't have a perfect square in the second column, we do have a product that is: $60 * 540 = 180^2$. The significance of this is that instead of $a^2 = b^2 \mod N$, we have $(ac)^2 = (bd)^2 \mod N$ for some $a^2 - N = b'$ and $c^2 - N = d'$. We can factor the number, giving

$$19^2 = 60 \mod 301$$

$$29^2 = 540 \mod 301$$

$$19^2 * 29^2 = 180^2 \mod 301.$$

Thus $a = 19 * 29$ and $b = 180$. Finding the factors gives

$$\gcd(19 * 29 - 180, 301) = 7$$

and

$$\gcd(19 * 29 + 180, 301) = 43.$$

We now have a new strategy for factoring numbers. We have a sequence of integers $a^2 - N$ for $a \geq \lceil \sqrt{N} \rceil$. We would like to pick out a subsequence that when multiplied together yields a square. However, we don't want to try all subsequences, there is an exponential amount of them. How long does the sequence have to be in order to guarantee that there is such a subsequence? How do we find it? The answers to these questions require the notion of *smooth numbers*.

We say a number m is B-smooth if all prime factors of m are less than or equal to B. We first note that if such a number in our sequence is not smooth, then it is unlikely to be a part of our subsequence. Let p be a large prime dividing some element m in the sequence. Then there must be another element m' in the sequence that has p as a divisor. Since p is large, multiples of p will be harder and harder to find. Hence, it is profitable to not include them in our search, and if possible, ignore them entirely. Now, we present the following lemma [6].

Lemma 2.3. *If $m_1, m_2, \ldots, m_k$ are positive B-smooth integers, and if $k > \pi(B)$ (where $\pi(B)$ denotes the number of primes in the interval $[1, B]$), then some non-empty subsequence of (m_i) has product which is a perfect square.*

Proof. For a B-smooth number m, we define its *exponent vector* $v(m)$. If m has the prime factorization

$$m = \Pi_{i=1}^{\pi(B)} p_i^{v_i}$$

where p_i is the ith prime number and each exponent v_i is a nonnegative integer, then $v(m) = (v_1, v_2, \ldots v_{\pi(b)})$. Then a subsequence $m_{i_1}, \ldots m_{i_t}$ has product a square if and only if $v(m_{i_1}) + \ldots + v(m_{i_t})$ has all even entries. That is, if and only if this sum of vectors is the zero vector mod 2. Now, the vector space $\mathbb{F}_2^{\pi(B)}$, where $\mathbb{F}_2$ is the finite field with 2 elements, has dimension $\pi(B)$. And we have $k > \pi(B)$ vectors. So this sequence of vectors is linearly dependent in this vector space. However, a linear dependency when the field of scalars is $\mathbb{F}_2$ is exactly the same as a subsequence sum being the zero vector. □

Later on, we will find the notion of exponent vectors a useful one. Exponent vectors are a concise way of storing the factorization of a B-smooth number. They are also quite sparse in nature and we can store them in a concise manner, saving on memory requirements.

3 A First Attempt

We now know enough information to devise a simple algorithm to factor numbers. We are given a number N which satisfies the following requirements:

1. The number is composite.

2. The number has no prime factors up to its logarithm base ten. It is more efficient to divide these out via the use of some other algorithm(trial division, elliptic curve method, Pollard's rho, etc) before we start.

3. The number is not a power. In order to use the lemma, we need two different odd primes to divide N. This is not the case if N is a power [6].

Then a reasonable first attempt, given what we know now, might be the following:

1. Compute parameter B and find the primes in $[2, B]$

2. Compute $x^2 - N$ and see if it is B-smooth for $x \geq \lceil \sqrt{N} \rceil$. Do this until we have $\pi(B)$ smooth numbers

3. Using linear algebra, form a matrix M with columns comprised of the exponent vectors of each smooth number. Find a linear dependency by computing the null-space.

4. Construct a, b in a manner corresponding to Theorem 2.1. If $a = \pm b \mod N$, then start over with step 1.

There are many gaps in this algorithm. In particular, how do we determine whether a number is B-smooth (which we discuss in Section 4) and how do we choose the parameter B (which we will discuss in Section 5)?

4 Smooth Numbers

4.1 Determining Smooth Numbers

In order to determine whether a given number N is B-smooth, we need to determine all prime numbers in the range $(2, B)$. We could naively use trial division on each number and determine the primes this way. However, we can do much better than this. We will use a classic algorithm - the sieve of Eratosthenes.
This algorithm requires $N - 1$ numbers to be stored in memory. To store all the primes less than 10^{40} requires $2^{132.887}$ bits of memory. The largest computing array has less than 2^{60} bits of memory. Therefore, there is no chance of even storing all these primes. The algorithm, however, is very useful for determining all the prime numbers less than say one billion. See algorithm (1).
To determine whether a given number is B-smooth, we can divide N by each prime p in $(2, B)$. We can use the sieve of Eratosthenes to determine all the primes in the range $(2, B)$. The prime p may have multiplicity greater than one, so we need to repetitively divide N by p. More specifically, we have

4.2 Distribution of B-Smooth Numbers

We now possess the methods for finding B-smooth numbers, but a natural question is wondering how often do the occur? This is a difficult question, and only heuristic analysis have given satisfactory answers. If B is too small, we will have a hard time finding any smooth numbers. If B is too large, then we will have too many smooth numbers and finding

Algorithm 1 The Sieve of Eratosthenes

```
 1: for $i \in [2, B]$ do
 2:     $a[i]$ is unmarked
 3: end for
 4: for $i \in [2, \sqrt{B}]$ do
 5:     if $i$ is unmarked then
 6:         for each multiple $j$ of $i \in [i + 1, B]$ do
 7:             Mark $a[j]$
 8:         end for
 9:     end if
10: end for
11: return All unmarked $a[i]$
```

Algorithm 2 IsSmooth: Determines if a number N is B-smooth. The set P_B is the set primes less than B.

```
1: for $i \in P_B$ do
2:     while $N \mod i = 0$ do
3:         $N \leftarrow N \mod i$
4:     end while
5: end for
6: if $N = 1$ then
7:     return Is Smooth
8: else
9:     return Not Smooth
10: end if
```

Algorithm 3 FactorOut(N, p): Removes all p from N.

```
1: while $N \mod p = 0$ do
2:     $N \leftarrow N \mod p$
3: end while
4: return $N$
```

a linear dependency will prove difficult. Heuristic analysis found in [3] show that the upper bound of

$$L(n) = e^{\frac{1}{2}ln(N)ln(ln(N))}$$

gives an optimal bound for B. We will use $B = \lceil L(n)^{1/2} \rceil$, also derived in [3], as our B value.

5 Smooth Number Generation

Given a smoothness bound B, we wish to quickly find a set of B-smooth numbers. We first need to find all the primes numbers from 2 to B; this is easily done using the sieve of Eratosthenes. Then, we need determine which primes could possibly divide any numbers in our sieving region. Then we use a modified sieve of Eratosthenes to determine which numbers are smooth.

5.1 Quadratic Residues

First, we need to find a range of numbers to search for smooth numbers in. Computing $x^2 \mod N$ is equivalent to finding $x^2 - N$ in the range $N < x^2 < 2n$ or equivalently $x \in [\sqrt{N}, \sqrt{2n}]$[1].
The concept of quadratic residues is important for us to determine which primes in our factor base we should consider. A integer p is a *quadratic residue* modulo N if there exists an x such that $x^2 \equiv q \mod N$. We consider three cases.

1. First, no such x exists. Then p cannot divide any values of $x^2 - N$ for $x \in [\sqrt{N}, \sqrt{2n}]$. Suppose $p \mid x^2 - N$. Then $x^2 - N = 0 \mod p$ and $x^2 = N \mod p$. Then N is

a quadratic residue modulo p. The contrapositive states that if N is not a quadratic residue mod p, then $p \nmid x^2 \mod N$.

2. Second, x exists and is zero. Then $0^2 = N \mod p$, implying that $p \mid N$. We are done, since the factors are p and N/p.

3. Third, $0 < s < p$. Suppose $p \mid x^2 - N$. Then $x^2 = N \mod p$ or equivalently $x^2 = s^2 \mod p$. Now $x^2 - s^2 = 0 \mod p$ or $(x+s)(x-s) = 0 \mod p$. Since $\mathbb{F}_p$ is a field, either $x - s = 0 \mod p$ or $x + s = 0 \mod p$. This implies $x = \pm s \mod p$. The only two solutions will be $s^2 = (-s)^2 = N \mod p$.

We define the *Legendre symbol* as

$$\left(\frac{a}{p}\right) = \begin{cases} 1 \text{ if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod{p} \\ -1 \text{ if } a \text{ is a quadratic non-residue modulo } p \\ 0 \text{ if } a \equiv 0 \pmod{p}. \end{cases}$$

To compute the Legendre symbol we give the following equivalent definition,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 \text{ if } a^{(p-1)/2} \equiv 1 \pmod{p} \\ -1 \text{ if } a^{(p-1)/2} \equiv -1 \pmod{p} \\ 0 \text{ if } a \equiv 0 \pmod{p}. \end{cases}$$

This give us a way of determining whether a prime in our factor base could possible divide a number $x \in [\sqrt{N}, \sqrt{2n}]$. We simply check to see if $a^{(p-1)/2} = 1$ and add these p to the factor base.

5.2 Generating Smooth Numbers Rapidly

Now, given a range $[a, b] \subset [\sqrt{N}, \sqrt{2n}]$, we wish to determine which $x \in [a, b]$ are candidates for being smooth numbers. For a given p in our factor base, we note that the first multiple of p in the range $[a, b]$ is $\lceil a/p \rceil p$. Given a s such that $s^2 = N \mod p$, the set of possible smooth numbers containing p as a factor is [1]

$$\begin{aligned} x \in \; & \{\lceil a/p\rceil p - s, \lceil a/p\rceil p + s, \lceil a/p\rceil p + p - s, \lceil a/p\rceil p + p + s, \lceil a/p\rceil p + 2p - s, & (1) \\ & \lceil a/p\rceil p + 2p + s, \ldots\}. & (2) \end{aligned}$$

This provides immense savings in computational power over the naive approach. Now, we can use a modified version of the sieve of Eratosthenes. Since we only care if an integer contains the prime divisors in our factor base, rather than divide each number by each prime in $[2, \sqrt{N}]$, we will divide each number by only those in our factor base. Furthermore, for each prime in the factor base, we will only divide those numbers in the above equation by p. Like the sieve of Eratosthenes, we need to repetitively divide the number in question by p.

Once we have performed this step for each prime in the factor base, we need only to look for those numbers in $[a, b]$ which as a result of repeated division are now one. These represent numbers smooth with respect to the factor base. That is to say, their prime factorizations contain only primes in the factor base. We call these numbers *relations*.

6 An Algorithm for Sieving

We now have all the information necessary to devise an algorithm for sieving smooth numbers. First, we determine a smoothness bound using equation INSERT HERE. Second, use the sieve of Eratosthenes to determine all the prime numbers less than our bound and add them to a list. Third, we calculate the Legendre symbol for each prime in the list and add it to the factor base if the calculation results in one. Fourth, we create large intervals in the range $(\sqrt{N}, \sqrt{2n})$. Each interval can be given to a separate processor to sieve over. We perform the next three steps for each interval. Fifth, we calculate $x^2 - N$ for each $x \in [a, b]$. Sixth, for each prime in the factor base, we find s such that $s^2 = N \mod p$. For each x in (1), we repeatedly divide x by the prime p until $x \nmid p$. Seventh, for each $x \in [a, b]$, if $x = 1$, then add x to the list of relations. Union all lists from all computers into a single list. In pseudo-code, we have the following

7 Computing the Null Space and Finding the Factors

7.1 Review of Null Spaces

Suppose we have a set of vectors $v \in V$ with $n = |V|$ and we ask the question if whether there exist constants a_i, $0 \leq i \leq n$, such that

$$\sum_{i=1}^{n} a_i v_i = \mathbf{0}.$$

This is a classic problem in linear algebra, known as finding the null space of a matrix. First, we form a matrix such that each column is a vector in V. Then, we perform Gaussian elimination on this matrix. We determine which columns are zero vectors; such columns must be linear combinations of other columns. More about Gaussian elimination and null-spaces can be found in [2].

7.2 Finding the Null Space

At this point, we have a set of smooth numbers in the range $[\sqrt{N}, \sqrt{2N}]$. Now we need to determine a subset of these that, when multiplied together, yield a perfect square. To do this, we transform the multiplication problem into an addition problem modulo two. We first find the exponent vectors of each relation. Then the problem of finding a subset of relations that multiply to a perfect square becomes the problem of finding a subset of relations that add to the zero vector. This is the classic problem of finding the null space of a matrix. We form a matrix with each exponent vector as a column. The rows represent which numbers have a given prime factor. Hence the matrix will be size $|F| \times |R|$, where $|F|$ is the size of the factor base and $|R|$ is the number of relations. There are many basic algorithms for finding the null space; for small matrices, Gaussian elimination works well. For larger matrices (greater than 100000×100000), though, we need to use a more efficient algorithm, block Lanczos. This algorithm was designed specifically to find the null space of large matrices. There are even variants for finding the null space modulo two and it has

Algorithm 4 The Sieving Phase

```
1:  $B \leftarrow \lceil L(n)^{1/2} \rceil$
2:  $p_B \leftarrow SieveOfEratosthenes(B)$
3:  **for** $p \in p_B$ **do**
4:      $z \leftarrow N^{(p-1)/2} \mod p$
5:      **if** $z = 1$ **then**
6:          $F \leftarrow F \cup \{p\}$
7:      **end if**
8:  **end for**
9:  Let $I$ contain several large intervals $[a, b]$, one for each processor
10: **for** $[a, b] \in I$ **do**
11:     **for** $x \in [a, b]$ **do**
12:         $R[x - a] \leftarrow x^2 - N$
13:     **end for**
14:     **for** $p \in F$ **do**
15:         Find $s$ satisfying $s^2 = N \mod p$.
16:         $x \leftarrow \lceil a/p \rceil p$
17:         **while** $x - s \leq b$ **do**
18:             **if** $a < x - s < b$ **then**
19:                 $R[x - a] \leftarrow FactorOut(x - s, p)$
20:             **end if**
21:             **if** $a < x + s < b$ **then**
22:                 $R[x + a] \leftarrow FactorOut(x + s, p)$
23:             **end if**
24:             $x \leftarrow x + p$
25:         **end while**
26:     **end for**
27:     **for** $x \in [a, b]$ **do**
28:         **if** $R[x] = 1$ **then**
29:             $L \leftarrow L \cup \{x\}$
30:         **end if**
31:     **end for**
32: **end for**
33: Union all lists $L$ from all processors
```

become the algorithm of choice for this task. The details of its implementation are outside the scope of this paper, however, we direct the interested reader to [4].

7.3 Obtaining Factors

After computing the null space, we have a set S such that

$$r_{s_1} + r_{s_2} + \ldots + r_{s_N} = \mathbf{0} \mod 2$$

for $s_i \in S$. We now know that multiplying these integers will yield a perfect square. Now, let

$$r_{s_1} + r_{s_2} + \ldots + r_{s_N} = \mathbf{s}$$

when computed in the integers. We know that s is a perfect square, since there are no odd exponents in the prime factorization of s. Equivalently, if we have $r_{s_1} = x_i^2 \mod N$, then

$$x_{s_1}^2 x_{s_2}^2 \ldots x_{s_N}^2 = (x_{s_1} x_{s_2} \ldots x_{s_N})^2 = s \mod N.$$

We let $r^2 = s \mod N$. If

$$r \neq \pm x_{s_1} x_{s_2} \ldots x_{s_N}$$

then the factors are

$$\gcd(x_{s_1} x_{s_2} \ldots x_{s_N} - r, N)$$

and

$$\gcd(x_{s_1} x_{s_2} \ldots x_{s_N} + r, N).$$

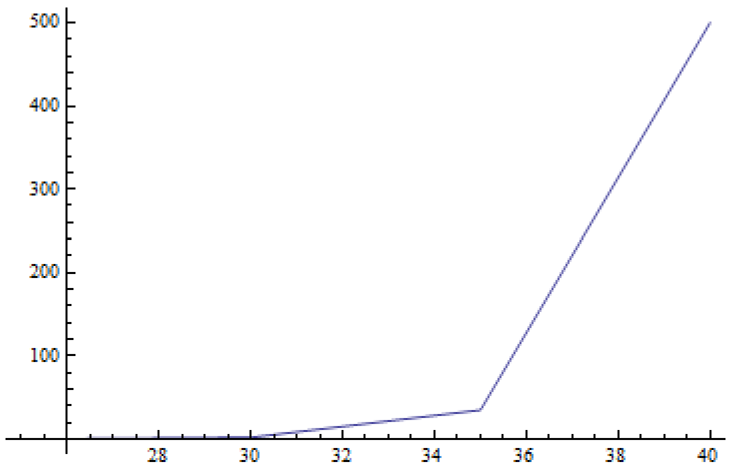
In pseudo-code, we have

8 Computational Results

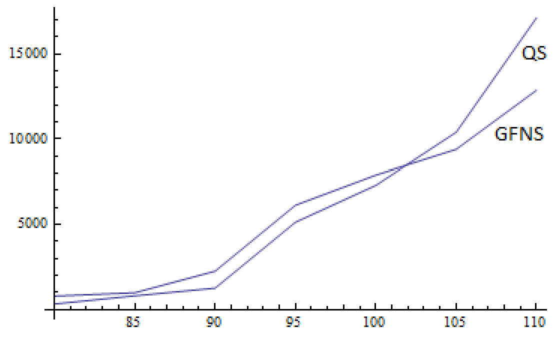
We present two major computational results regarding the complexity of the quadratic sieve. We first demonstrate an implementation of the quadratic sieve created by the authors. Then we show that after the number is 100 digits or longer, the *general number field sieve* is faster for arbitrary numbers. The general number field sieve is the asymptotically fastest known algorithm for factoring number longer than 100 digits in length.
Our first result is demonstrating our implementation of the quadratic sieve. It is a straight forward implementation of algorithms 4 and 5. It is only suitable for numbers in the range of 20 to 40 digits. This is mostly due to the fact that B-smooth numbers become increasingly rare the larger N becomes. There are ways around this, but these methods are outside the scope of this paper. The following graph demonstrates the speed of our program as a function of the number digits.

Algorithm 5 Recovers the factors from a set of relations

```
1:  Form matrix $M$ from the exponent vectors of $R$
2:  Reduce matrix $M$ modulo 2
3:  Find null space of matrix $M$
4:  for $\mathbf{v} \in \mathcal{N}$ do
5:      $\mathbf{g} \leftarrow \mathbf{0}$
6:      $r \leftarrow 1$
7:      for $i \in 1, 2, \ldots |L|$ do
8:          if $v_i = 1$ then
9:              Get $(x_i, g_i)$ from $L$
10:             $\mathbf{g} \leftarrow \mathbf{g} + x_{s_i}$
11:             $t \leftarrow tx_i$
12:         end if
13:     end for
14:     $\mathbf{h} = \frac{1}{2}\mathbf{g}$
15:     Let $r$ be the integer formed from exponent vector $\mathbf{h}$
16:     if $r^2 \neq \pm t^2 \mod N$ then
17:         return $p = \gcd(r - t, N), q = \gcd(r + t, N)$
18:     end if
19: end for
```



As can be seen, our program works well for number in this range. Extrapolating the exponential curve shows that it is, however, not capable of numbers greater that 45 digits in length.
Our second result shows that the general number field sieve is faster for numbers greater that 100 digits. Since this is outside the range of our program, we use a program written by Jason Papadopoulos called msieve [5]. The program is capable of factoring numbers using either the quadratic sieve or the general number field sieve. We created a group of semi-primes in the range of 80 to 110 digits in multiples of 5. Each group had 10 numbers in it. We used a local computer lab to factor each number using both the quadratic sieve and the general number field sieve. The results are presented in the following plot.



As we can see, at around 95 digits in length, either algorithm would suffice, but numbers greater than 100 digits, the general number field sieve generally performs faster. This confirms the empirical results found by many other groups.

9 Conclusion

The quadratic sieve is one of several algorithms used to factor large integers. For numbers in the range of 40 to 100 digits, it is the fastest known algorithm. For numbers larger than 100 digits, the general number field sieve tends to perform faster.
We have given a thorough introduction to the quadratic sieve. There are many, many optimizations to make it faster. For example, instead of just sieving over the polynomial $x^2 - N$, we can sieve over $(ax + b)^2 - N$. This speeds up the factorization considerable by not relying on the polynomial $x^2 - N$ to generate all the relations. With this optimization, we can distribute the factorization over a large network of computers by giving each one a polynomial.
We demonstrated a simple factorization program and showed that it can be used to factor numbers in the range of 30 to 50 digits in length. It was not optimized enough to larger numbers, although.
We tested a widely known factorization program known as msieve. It can factor numbers using the quadratic sieve or the general number field sieve. We showed that for this particular program, the general number field sieve starts to factor number faster than the quadratic sieve at around 100 digits. This is consistent with empirical results found by other authors.

References

[1] Gregory V. Bard. The quadratic sieve. In *Algebraic Cryptanalysis*, pages 159–183. Springer US, 2009.

[2] David R. Hill Bernard Kolman. *Elementary Linear Algebra with Applications*. Pearson Prentice Hall, ninth edition, 2008.

[3] Stephani Lee Garrett. On the quadratic sieve. Master's thesis, THE UNIVERSITY OF NORTH CAROLINA AT GREENSBORO, 2008.

[4] Peter L. Montgomery. A block lanczos algorithm for finding dependencies over gf(2). In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'95, pages 106–120, Berlin, Heidelberg, 1995. Springer-Verlag.

[5] Jason Papadopoulos. Integer factorization source code, March 2011.

[6] Carl Pomerance. Smooth numbers and the quadratic sieve. In *Buhler and Stevenhagen 2007*. University Press, 2005.