Integer Factorization

Fredrik Lilkaer, Philip Sköld
DD2440 - Advanced Algorithms

November 12, 2013

ABSTRACT
This report gives an overview of implementing algorithms for factorizing big integers. Several advanced algorithms for integer factorization is implemented, discussed and analyzed, among them the Quadratic Sieve Algorithm. The implementation can efficiently factor integers containing up to 44 digits.

Royal Institute of Technology
School of Computer Science and Communication

CONTENTS

1 Introduction
- 1.1 KATTIS Problem Specification
- 1.2 Basic Notation
- 1.3 Primality Tests
- 1.4 GMP - GNU Multiprecision Library
- 1.5 Conventions

2 Factorization Algorithms
- 2.1 Trial Division
- 2.2 The "Fill"-method
- 2.3 Pollard Rho, ρ-method
- 2.4 Pollard Rho, Brent's method
- 2.5 Quadratic Sieve

3 Implementation Details
- 3.1 Known Primes
- 3.2 Perfect Powers
- 3.3 Quadratic Sieve (QS) optimizations

4 Performance
- 4.1 Number Suites
- 4.2 Results

5 Conclusion

1 INTRODUCTION

The Fundamental Theorem of Arithmetic states that each number $N \in \mathbb{N}, N \geq 2$ can be expressed as a *unique* product of primes [1]. This product is the *factorization of* N, and the prime numbers in it are factors of N. The problem of finding the factorization of N is referred to as integer factorization, and no efficient algorithm for factorizing big integers exists today. The most notable exploit to the apparent difficulty of integer factorization is encryption, specifically RSA-encryption which is very widely used.

This report addresses the problem of integer factorization and presents several different well known algorithms of varying complexity and efficiency(chapter 2. A comparizon of the different algorithms is made in chapter 4. The report is part of an assignment in the computer science course, DD2440 - Advanced Algorithms, at the Royal Institute of Technology and the results are judged by the KATTIS System.[2]

In this chapter, basic notation, and algorithms that are commonly used in integer facorization are presented.

1.1 KATTIS Problem Specification

The KATTIS input consists of an integer $N > 1$ with *at most 100 bits*. There are exactly 100 numbers that should be factorized and the total time limit is $15s$. For each number the output should be either the factorization, or the string "fail". Finally, the RAM limit is 64MB.

1.2 Basic Notation

1.2.1 Modular Arithmetic

Integer congruence relationships are very commonly used in integer factorization. Two integers a and b congruent modulo N if $a - b$ is a multiple of N . In this paper, this is written as:

$$a \equiv b \pmod{N}$$

If we operate on numbers that are congruent modulo N, we say that those number are in the cyclic group $\mathbb{Z}/N\mathbb{Z}$.

1.2.2 GCD - Greatest Common Divisor

The greatest divisor of two integers a and b is the largest number that both numbers evenly(without remainer).

1.3 Primality Tests

When factorizing integers, it helps to have routines that can determine whether a number, N, is a prime or not. This check is referred to as a *primality test*. One of the fastest algorithms known today is *AKS Primality Test* which has a running time of $\mathcal{O}(\log^6(N))$ for general integers[3]. However, for practical applications *Pseudo primality tests* are used, which are probibalistic but faster. The rate at which pseudo primality tests fail is generally very low.

The Naive Implementation for primality testing is to try dividing N with all integers, $2 \leq d \geq \sqrt{N}$. This is not a pseudo primality test, and while it does indeed determine whether N is a prime or not it has a very slow running time of $\mathcal{O}(\sqrt{N})$.

The Miller-Rabin pseudo primality test is a more efficient, probibalistic, algorithm which is probably one of the most commonly used primality tests today, and the one used in this implementation. It relies on the fact that if N is prime, then $x^k \equiv 1 \pmod{N}$ where $k > 0, x \in 1, 2, ..., N-1$. If N is a composite number, then $x^k \not\equiv 1 \pmod{N}$, with a probability of $(s-1)/2$ where s is the number of times you run the procedure[4]. According to Rivest et. al[4, p. 974], choosing $s = 50$ will be enough for "almost any imaginable application".

Carmichael Numbers are numbers that satisfy the congruence $x^{m-1} \equiv 1 \pmod{m}$ and these are the only composite numbers that can pass the basic Miller-Rabin; they are the so-called "nonwitnesses". However, these are extremely rare and more complicated versions of Miller-Rabin removes this small defect[4].

1.4 GMP - GNU Multiprecision Library

Integer Factorization generally involve very big integers; larger than what can be stored in c++ standard primitive types. Because of this, libraries for arbitrary precision must be used, GMP (GNU Multiprecicion Library) being one of the most popular. GMP also provides arithmetic functions for signed integer arithmetic, including common algorithms mentioned above. The implementation in this report uses GMP for high-precision arithmetic, and also utilizes some of the basic functions used in integer factorization such as GCD and pseudo primality tests.

1.5 Conventions

The rest of the paper will concern integer factorization. For clarity the following definitions will be used throughout, unless otherwise specified.

- $\hat{N} \in \mathbb{N}$ is an integer that should be factorized
  The input consist of several such integers, each of which is a composite or prime.
- $N \in \mathbb{N}, N \equiv 1 \pmod{2}$ is an odd integer.
  It is trivial to remove factors of 2 *from a number* $\hat{N}$*, which reduces the problem of factorizing* $\hat{N}$*.*
- p is a prime number.

2 FACTORIZATION ALGORITHMS

In this chapter, 4 algorithms for integer factorization are presented: A naive Trial Division method, two variants of the Pollard Rho algorithm and the Quadratic Sieve which is more advanced.

2.1 Trial Division

Trial Division is the naive algorithm for integer and works by testing if $p|N$ for all primes $p \leq \left\lfloor \sqrt{N} \right\rfloor$. This procedure is guaranteed to find all factors of N and might work well if N contains only very small factors.

2.2 The "Fill"-method

For faster factorization, there exists many more efficient algorithms than Trial Division. However, in general, these algorithms find *a factor* to N, not all of them. To make use of this, the algorithms are run repeatedly and everytime a factor, p, is found the problem is divided(literally) so that we have a new "N" to factorize.

Since all the found factors are stored in a vector the algorithm is called *Fill*, as it's filled with the prime-factors of N.

2.3 Pollard Rho, ρ-method

The Pollard Rho Method, or "Pollard's Rho Heuristic", is a more efficient algorithm presented by John Pollard, 1975 [5]. It tries to find an integer, ϕ so that $\gcd(\phi, N) < 1$, i,e we have a common divisor of both ϕ and N. To increase the probability of finding a ϕ, Pollard's heuristic relies on the birthday paradox which states that in a the expected probability of finding two random numbers $x_1 \equiv x_2 \pmod N$ exceeds $\frac{1}{2}$ after $\sqrt{2 \ln 2}\sqrt{p}$ randomly chosen elements $\pmod N$[6]. This is usually done by deterministically generating pseudo-random sequences, S in $\mathbb{Z}/N\mathbb{Z}$, with the recurrence (or a similar one) $x_{n+1} = x_n^2 + a \pmod N$ and some x_0, a. When we find two elements that are congruent modulo p, then $\gcd(x_2 - x_1, N) < 1$:

$$p|kp \text{ and } p|N$$

$$x_1 \equiv x_2 \pmod p \iff x_1 - x_2 = kp$$

$$\implies \gcd(kp, N) < 1 \implies \gcd(x_1 - x_2, N) < 1$$

One way to demonstrate this is with an illustration in the shape of the letter ρ, which is why the algorithm has its name. Consider we have $N = 1387, x_{n+1} = x_n^2 - 1 \pmod N$ and $x_0 = 2$, shown in figure 1: after 3 iterations we find a factor, 19, of N:

x_1	x_2	$\gcd(x_1 - x_2, N)$
2	2	-
3	8	1
8	1194	1
63	177	**19**
1194	-	-

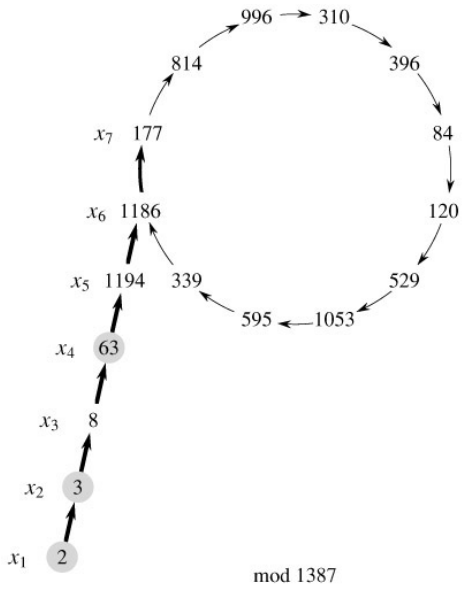

Figure 1: Pollard's rho algorithm example from Introduction to Algorithms[4].

To find a factor, $p|N$, we simply continue calculating x_n and x_{2n} as illustrated in figure 1, until we have $\gcd(x_n - x_{2n}, N) < 1$:

2.4 Pollard Rho, Brent's method

An improvement of Pollard's origininal ρ-method was introduced by Brent [7]. Instead of computing $\gcd(x_n - x_{2n}, N)$ for many n(i.e going through the recurrence), only one value x_n is calculated. If n is a power of $k = 2$ then save $y = x_n$, and then calculate $\gcd y - x_n, N$ instead. This means that periodic sequences can be detected faster (another generating element, x_0, have to be used in this case).

2.5 Quadratic Sieve

The QS is the second fastest general purpose integer factorization algorithm, next to the General Number Field Sieve (GNFS)[8]. For numbers less than 10^{100} QS is widely considered the fastest integer factorization algorithm.

In the KATTIS test suite, the input is restricted to integers up to 2^{100}. This paper will not address the GNFS, as the performance is not needed by the test suite utilized.

2.5.1 Fermat Factorisation

The heart of QS is the Fermat Factorisation based on finding a congruence of squares.

$$a^2 \equiv b^2 \pmod N$$

If such a relation exists, it is possible to factor N using the conjugate rule in $\mathbb{Z}_N$ to $a^2 - b^2 \equiv (a+b)(a-b) \equiv 0 \pmod N$. This yields $(a+b)(a-b) = kN$. $a \equiv \pm b$ renders the equation to $0 = 0$, so we further impose requirements that

$$a \not\equiv b \pmod N$$

The purpose is to eliminate a trivial factorization giving $k = 0$ if $N \neq 0$, as we also require $k \geq 1$. If $(a+b), (a-b) \nmid k$ we have $(a+b)|N \vee (a-b)|N$. Computing $f_1 = \gcd(a+b, N), f_2 = \gcd(a-b, N)$ will yield candidate factors. If $f_i \notin \{1, N\}$, a factorization for N is found!

The naive Fermat factorization method tries to find such congruences by picking an a at random at calculate $a^2 \pmod N$ and checking if it is a square modulo N.

2.5.2 Finding congruences

The QS improves on the congruence finding by generating multiple candidate a:s and trying to multiply different a_i together to a square modulo N.

Observation 1 All numbers can be uniquely represented as a product of prime powers (fundamental theorem of arithmetic).

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_1 k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}$$

We define the exponent vector to be the vector of prime powers.
Observation 2 Multiplication of two numbers n_1 and n_2 can be done by added the corresponding exponent vectors.
Observation 3 A square is a number with all prime powers even. If all prime powers are even we can create two equal numbers with every prime exponent divided by two that will multiply to the original n.
Observation 4 To know whether or not a number is a square all that is needed is the $GF(2)$[1] representation of the exponent vector, the exponent vector with all powers mod 2.
Observation 5 The product of two squares is a square. Adding even exponents will result in even exponents.

Define a polynomial $Q(x) = a^2 - N$ such that $a = x + \lceil\sqrt{N}\rceil$. Observe $(a(x))^2 - Q(x) = (a(x))^2 - ((a(x))^2 - N) = N$, so if we can find an a such that $((a(x))^2 - N)$ is a square we can factorize N by the conjugate rule.

Pick a smoothness bound B.

$$B = ce^{0.5\sqrt{\log N \log\log N}}$$

Generate a factor base $\mathbb{F}$ such that

$$p \in \mathbb{F} \iff p \leq B \wedge p \text{ is prime} \wedge N^{(p-1)/2} = 1$$

Generate a set $\mathbb{Y}$ of $y_i = Q(x_i)$ that can be factored by the factors in $\mathbb{F}$[2] write it as an exponent vector in $GF(2)^{|\mathbb{F}|}$. Generate an $|\mathbb{Y}|$ by $|\mathbb{Y}|$ identity tracking matrix. Perform Gauss-Jordan row reduction on the matrix containing the exponent vectors in attempt to create a zero row while performing the same row operations on the tracking matrix. If we can find a row that is zero in the exponent matrix and nonzero in the tracking matrix it implies a solution that contains a non-empty set of y_i that multiplies to a square (zero exponent vector). Finding y_i:s such that $|\mathbb{Y}| > |\mathbb{F}|$ guarantees that we will have linear dependencies in the exponent matrix, but they might be trivial, so we aim to have $|\mathbb{Y}| > |\mathbb{F}| + 10$ to make the probability of finding a nontrivial solution high.

[1] The Galois Field of two elements, the smallest finite field. The only contained elements are the additive and the multiplicative identities (0 and 1).
[2] Such elements are called B-smooth.

2.5.3 Sieving for smooth numbers

A procedure to find smooth y_i, that can be factorized over $\mathbb{F}$ and fit in the exponent matrix is required. The naive approach would be to generate a set of y_i and for each $p \in \mathbb{F}$ try to divide each candidate y_i by p. For each y_i all factors are saved. If y_i ever reaches 1 then the original y_i is a smooth number, we know it's factorization and it can go into the exponent matrix. The correctness of the method is trivial to realize, however it does not provide sufficient performance. We will look at performance optimizations of the sieving in section 3.3.

3 IMPLEMENTATION DETAILS

This chapter addresses important implementation details and optimizations that were used, focusing on QS as it is the most complex algorithm of the ones included in this report.

3.1 Known Primes

In order to avoid to perform Erathostenes Sieve to generate prime numbers in runtime the program uses a precompiled prime number list. The prime number list contains the first 10000 primes, as the judge KATTIS only supports uploads of up to 128 kB source code.

3.2 Perfect Powers

Since QS does not detect numbers that are perfect powers, i.e numbers $n = n^k, k \geq 1$, it is neccessary to add a procedure that detects these numbers and solve them through other means. If N is a perfect power p^k it is easy to realize that $k \leq \log_2 n$. The perfect power test performs i:th root computations on n from $i = 2$ to $i = \log_2 n$. If the computation was exact we detect a perfect power.

3.3 QS optimizations

Recall the polynomial $Q(a) = a^2 - n$. Observe that

$$Q(a + kp) = (a + kp)^2 - n =$$

$$a^2 + 2akp + (kp)^2 - n =$$

$$Q(a) + 2akp + (kp)^2 \equiv Q(a) \pmod p$$

If we can find x such that $Q(x) \equiv 0 \pmod p$ we will thus find an index for which $Q(x)$ is divisible by p and also all other indices for which $Q(x)$ is divible by p. Find $Q(x) \equiv 0 \pmod p \implies a^2 \equiv N \pmod p$. The modular quadratic

equation can be solved efficiently by the Shanks-Tonelli algorithm which we will not delve any further into. By this solution and the realisation that every p:th $Q(x)$ is divisible by p we can effiently visit only those $Q(x)$ that p divides.

3.3.1 Speeding with logarithms

Observe that $\frac{e^a}{e^b} = e^{a-b}$. Suppose that we represent all $Q(x)$ as $\ln Q(x)$. We can now subtract $\ln p$ at each $Q(x)$ visited instead of performing a more expensive division. The major gain of using logarithmed $Q(x)$ is that we can approximate the logarithm of $Q(x)$ over large interval to avoid to evaluate $Q(x)$ often. Since $\ln 1 = 0$, we should ideally check for $\ln Q(x) = 0$, but since $\ln Q(x)$ is approximated we need a tolerance. Checking if $\ln Q(x) < \log p_{max}$, where p_{max} is the largest prime in $\mathbb{F}$ ensures that $Q(x)$ could not be factored with factors larger than p_{max}. The optimization comes at the expense of having to refactor the $Q(x)$ over $\mathbb{F}$, but it is not an expensive fee, since $|F|$ often is small. Another disadvantage is the fact that we miss some smooth numbers, but it is over shadowed by the sheer increase in performance.

4 PERFORMANCE

The running times and overall performance for the different algorithm implementation and input varies. For very small inputs, a naive solution performs well, and for somewhat smooth inputs Pollard Rho performs well. QS performs well on most inputs. In this chapter, the aim is to give an overview on how the different implementations perform.

4.1 Number Suites

The input data used to analyze the algorithms are suites of numbers with specific attributes and of varying size. The size of the numbers in the suites match the possible sizes for KATTIS-cases. Each suite contains 10 numbers and the different test suites are listed below:

- 100-Smooth Numbers with $\approx$ 20 digits (smooth10)
- 100-Smooth Numbers with $\approx$ 50 digits (smooth50)
- Perfect Squares with $\approx$ 10 digits (psquare10)
- Perfect Squares with $\approx$ 20 digits (psquare20)
- Perfect Squares with $\approx$ 30 digits (psquare30)
- Semiprimes with $\approx$ 10 digits (semi10)
- Semiprimes with $\approx$ 20 digits (semi20)
- Semiprimes with $\approx$ 30 digits (semi30)

Smooth numbers are interesting because they are quite easily factorized; many factors can be taken care of by Pollard Rho or Naive.

Perfect Squares are numbers that the QS algorithms do not detect, which is why they are interesting.

Semiprimes are hard to factorize, especially when the two factors involved are both very big (but not the same, as in perfect squares).

4.2 Results

Below, the results of the major algorithms are shown. For each algorithm and test suite, an avarage runnig time is listed, and the last column contains the KATTIS results. Each number had a time limit of 1s; the factorization is considered a fail if this limit is exceeded. The algorithms (and algorithm variants) included in the test are:

- Pollard Rho, Brent's Method (PR-B)
- QS
- QS + Pollard (QS+PR)

Below is the results. Each cell has a success rate, and the average running time in seconds:

algorithm	PR-B	QS	QS+PR
suite			
smooth10	0.006	0.037	0.005
smooth50	0.006	0.032	0.007
psquare10	0.006	0.027	0.006
psquare20	1.109	0.027	0.008
psquare30	-	0.032	0.008
semi10	0.06	0.026	0.008
semi20	0.493	0.240	0.370
semi30	-	2.703	2.914
KATTIS	67, 14.47	100, 8.06	100, 8.18

As expected, all three algorithms perform well on the smooth numbers, where the factors are not very large. PR-B handles small factors well, but can not solve any numbers in the 30-digit suites, simply because it reaches the time limit before such big factors are investigated. The interesting bit is the difference between QS and QS+PR. Both algorithms persform well overall, but PR-B has taken care of smaller inputs which is apparent from smaller inputs. However, looking at the larger inputs, mainly KATTIS test cases and semi30, QS is slightly faster. This is because there is a trade of in the QS+PR algorithm, where a certain amount of time is spent trying to see whether PR-B can factorize it quickly; the input might be smooth(or small). Then, when it seems like Pollard Rho (PR) might not be the best choice, it switches to QS which solves the case.

The final KATTIS submission ID is 458789 for the 100p full QS solution, 458793 for the 100p QS+Pollard Solution and 458797 for the 67p Pollard solution.

To see whether the QS can perform good for even larger testcases, higher and higher semi-prime number were generetaded with Wolfram Alpha [9]. The largest prime that were solved in reasonable time was the number 311939497648045357687131564489914540384581 13 = 59604644783353249 * 523347633027360537213687137. It contained 44 digits and the algorithm ran for 22.187s.

5 CONCLUSION

While inspecting and implementing several different algorithms for integer factorization, the implementation of Quadratic Sieves did manage to factorize numbers up to 44 digits long, and in reasonable time. The KATTIS results were satisfactory, and a mix between Quadratic Sieves and

Pollard's Rho Heuristic procuded the best result for general purpose integer factorization.

References

[1] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*. Wiley, ISBN: 0471625469. [Online]. Available: `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0471625469`.

[2] (Nov. 2013). The kattis system., [Online]. Available: `https://kth.kattis.scrool.se`.

[3] (Nov. 2013). Aks primality test, [Online]. Available: `http://mathworld.wolfram.com/AKSPrimalityTest.html`.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd. The MIT Press, 2009, ISBN: 0262033844, 9780262033848.

[5] (Nov. 2013). Pollard's rho algorithm, [Online]. Available: `http://en.wikipedia.org/wiki/Pollard\%27s_rho\_algorithm`.

[6] H. Riesel, *Prime numbers and computer methods for factorization*. Cambridge, MA, USA: Birkhauser Boston Inc., 1985, ISBN: 0-8176-3291-3.

[7] R. Brent. (Nov. 2013). Brent's factorization method., [Online]. Available: `http://mathworld.wolfram.com/BrentsFactorizationMethod.html`.

[8] Wikipedia. (Nov. 2013). Quadratic sieve, [Online]. Available: `http://en.wikipedia.org/wiki/Quadratic_sieve`.

[9] (Nov. 2013). Wolfram alhpa, [Online]. Available: `http://www.wolframalpha.com/`.