
Factoring large integers using Quadratic Sieve

CHRISTOPHER TELJSTEDT

chte@kth.se

HENRIK VIKSTEN

hviksten@kth.se

November 11, 2013

Abstract

This report will give you an insight to factorization of integers using quadratic sieve. It covers the basics of the factoring problem and mentions other commonly used algorithms, but the main focus is on the quadratic sieve which is the best algorithm for this problem up to 110 bit integers.

This report was written during the course DD2440 Advanced Algorithms at KTH 2013 and recieved the maximum amount of points in the scoring system KATTIS.

CONTENTS

I	Introduction	3
I.1	Purpose	3
I.2	Problem	3
I.3	Scope	3
I.4	Statement of Collaboration	3
II	Preliminaries	3
III	Background	4
III.1	Trial Division	4
III.2	Pollard's ρ	4
III.2.1	Basic idea	4
IV	Quadratic Sieve	5
IV.1	Brief explanation	5
IV.2	Deciding on factor base	6
IV.3	Sieving step	6
V	Implementation	7
VI	Results	7
VII	Discussion	7
VIII	Conclusion	7

I. INTRODUCTION

This report will cover the work of a integer factorization program written in the course DD2440 advanced algorithms. It was decided to focus primarily on quadratic sieve which should give a good understand of the problem in general.

I.1 Purpose

The goal of the program is to be able to factorize integers in an efficient way and also pass KATTIS test cases with as high score as possible, this is done by improving the program step-by-step and gradually increasing the performance and intelligence in the task at hand. In the report the methods used will be analyzed described. How they work independently and their correlation in our implementation.

I.2 Problem

An unknown set of 100 integers in varying size are used as input to the algorithm from KATTIS. The output is either the factors of the input if it is solved or the string fail otherwise. If number $n_i = p_1^{k_1} \dots p_m^{k_m}$, then the program should print the following on stdout.

$$\begin{array}{l} \left. \begin{array}{c} p_1 \\ p_1 \\ \vdots \\ p_1 \end{array} \right\} k_1 \text{ times} \\ \vdots \\ \left. \begin{array}{c} p_m \\ p_m \\ \vdots \\ p_m \end{array} \right\} k_m \text{ times} \end{array}$$

One of the problems is dealing with large integers within the timeframe but also finding every single non-trivial factor. The restrictions in kattis are 15 seconds and 64MB of memory.

I.3 Scope

There is no intention of reinventing any methods that already exists, the idea is to create a solver that can factorize unknown integers and in the end get a high score on KATTIS.

I.4 Statement of Collaboration

Some text.

II. PRELIMINARIES

Notation. By ' \log_b ' we denote the base b logarithm and the natural logarithm denotes by $\ln = \log_e$ with $e \approx 2.71828$. The largest integer $\leq x$ is denoted by ' $\lfloor x \rfloor$ '. The number of primes $\leq x$ is denoted by ' $\pi(x)$ ', and due the *Prime number theorem*[1] we know that $\pi(x) \approx x / \ln(x)$.

Smoothness. A positive integer is *B-smooth* if all its prime factors are lesser than a boundary B . Furthermore an integer, say x , is said to be *smooth with respect to S*, if x can be completely factored using integers by some set S alone.

Modular arithmetic. Throughout this paper ' $x \equiv y \pmod{n}$ ' means that $(x - y)$ is a multiple of n whereas x, y and $n \in \mathbb{N}_{\neq 0}$.

Similarly, ' $x \not\equiv y \pmod{n}$ ' mean that $(x - y)$ is not a multiple of n . Euclid's algorithm for finding the *greatest common divisor* of two non-negative integers, say x and z , is denoted by $\gcd(x, z)$.

III. BACKGROUND

The Integer Factorization problem is defined as follows; given a composite integer n , find any non-trivial factor e of n , such that $e|n$. At first it might seem like a trivial task, but for large integers this has proven to be a very difficult task.

III.1 Trial Division

The most straight-forward approach to factoring composite numbers is *trial division*, which essentially (just like its name suggests) just check for every prime number $p \leq \sqrt{n}$ if $p|n$.

Improvement is to only do tests when p is a prime number, i.e. keep a list of pre-computed prime numbers. Although trial division is easy to implement and guaranteed to grant an answer it is not efficient with large integers. There is improvements that can be applied such as storing a pre-computed tables of primes to bring up the speed. However, this instead requires alot of memory during run-time as well as storage, when dealing with larger integers. A

III.2 Pollard's ρ

In 1975 John M. Pollard proposed a new and very efficient Monte Carlo algorithm for factoring integers, now known as Pollard's ρ (rho) method. It was a breakthrough and proved to be alot more faster than its predecessor trial division for finding small non-trivial factors of a large integer n .

III.2.1 Basic idea

Pollard's ρ basic concept is that a sequence of pseudo-random integers constructed as

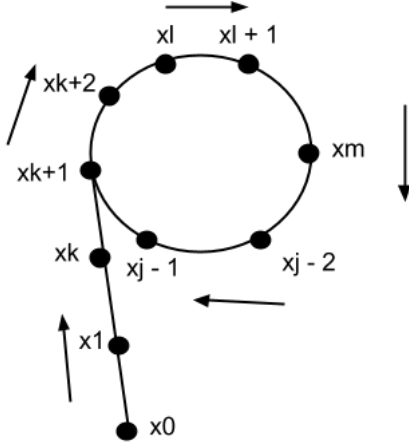
$$x_0 = \text{rand}(0, n - 1) \quad (1)$$

$$x_i = f(x_{i-1}) \pmod{n}, \text{ for } i = 1, 2, \dots \quad (2)$$

where f is polynomial which in most practical implementations has the form $f(x) = x^2 + \alpha, \alpha \neq 0, -2$. The key observation that if we consider a non-trivial divisor of n , say d , that is small compared to n . Then there exists smaller congruence groups modulo d compared to n . Because of this there is a probability that there exists two x_i and x_j such that $x_i \equiv x_j \pmod{d}$, while $x_i \not\equiv x_j \pmod{n}$. Thus it follows from that $\text{gcd}(x_i - x_j, n)$ is a non-trivial factor of n .

The idea of pollard rho algorithm is to iterate a formula until it falls into a cycle. We want to find a x and y whereas the x makes twice as many iterations as the y using a function \pmod{n} as a generator of a pseudo-random sequence. The $\text{gcd}(x - y, n)$ is taken each step, and it reaches n we have not found an answer and the algorithm terminates with a failure.

We can write $n = p \cdot q$, when x , which iterates twice as fast as y , catches up with y which will happen eventually, at this point factor p will be found. The time it will take cannot be proven matematically and can only be proven by heuristics. If the sequence behaves randomly it would take approximately p steps to find p , which is not very efficient. [2]



The figure above illustrates the Pollard's Rho cycle. The mapping of x_{i+1} is instead replaced with a function $x^2 + 1$ so that we get $x_i^2 + 1 \pmod N$ the factor p will be found after $O(\sqrt{p}) \in O(N^{1/4})$ steps. [2]

Pollard's rho algorithm can be improved further by implementing Brent's cycle finding method. [5]

IV. QUADRATIC SIEVE

IV.1 Brief explanation

The quadratic sieve algorithm is today the algorithm of choice when factoring very large composite numbers with no small factors. The general idea behind quadratic factoring is based on Fermat's observation that a composite number n can be factored if one can find two integers $x, y \in \mathbb{Z}$, such that $x^2 \equiv y^2 \pmod n$ and $x \not\equiv \pm y \pmod n$. This would imply that,

$$n \mid x^2 - y^2 = (x - y) \cdot (x + y) \quad (3)$$

but n neither divides $(x - y)$ nor $(x + y)$. Furthermore it can be rewritten as $(x - y) \cdot (x + y) = k \cdot p \cdot q$ for some integer k , thus becoming two possible cases.

- either p divides $(x - y)$ and q divides $(x + y)$, or vice versa.
- or both p and q divides $(x - y)$ and neither of them divides $(x + y)$, or vice versa.

Hence, the greatest common divisor of $(x - y, n)$ and $(x + y, n)$ would in first case yield p or q and a non-trivial factor of n is found. In the second case we get n or 1 and trivial solution is found.

Carl Pomerance suggested a method to find such squares[3]. The first step in doing so is to define the polynomial

$$q(r) = (r + \lfloor \sqrt{n} \rfloor)^2 - n \approx r^2 - n \quad (4)$$

Now, consider we set of primes $P = \{p_1, p_2, \dots, p_k\}$ lesser than a bound B , i.e $k < \pi(B)$. We then want to construct a subset of integers r_1, r_2, \dots, r_k such that $\forall i : q(r_i) = r_i^2 - n \pmod n$ is smooth in respect to P , more specifically $\forall i :$

$$q(r_i) = r_i^2 \equiv p_1^{e_{i1}} \cdot \dots \cdot p_k^{e_{ik}} \pmod n \quad (5)$$

where e_{ij} is the exponent of p_j of factorization of $q(r_i)$. If the exponents for all primes sums to an even number we arrive at following relation:

$$\prod_{i=1}^n q(r_i) = \prod_{i=1}^n r_i^2 \equiv (p_1^{e_1} \cdot \dots \cdot p_k^{e_k})^2 \pmod n \quad (6)$$

and the integers we x, y we sought to find are simply:

$$x = \prod_{i=1}^n q(r_i) \quad (7)$$

$$y = \prod_{i=1}^{\pi(B)} p_1^{e_i} \quad (8)$$

Lets consider a factor base P with the primes p_1, p_2, \dots, p_k that is $k \geq B$ and co-prime to n . We want to search for small r_i so that $q(r_i)$ is smooth in respect to P . If we find such r 's we say that $q(r_i) = (r + \lfloor \sqrt{n} \rfloor)^2 - n$ is B-smooth and we can factor completely over the factor base.

A prime factorization $p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_{\pi(B)}}$ of a B-smooth number can then be expressed be entirely expressed by the factor base and exponent vector $(e_1, e_2, \dots, e_{\pi(B)})$.

Since a prime p only divides $q(r_i)$ if and only if it divides $q(r_i + kp)$ for any integer k , we can found these values efficiently using a sieve. For this reason its called the *sievingstep* and because only primes $(\frac{n}{p_i}) = 1$ can divide $q(r_i)$, explains the definition of the factor base P . Naivly one could simply randomly select r in a range of interest and verify that $q(r)$ is divisible by all the primes in the factor base. Instead in quadratic sieve algorithm we can first solve the quadratic congruence $r^2 \equiv n \pmod{p}$ and then cleverly only divide by prime numbers corresponding to the interval r and $r + kp$ for some any integer k . Thus, finding these r 's becomes easier than just a simple trail division as we apriori know at that a B-smooth $q(r_i)$ is divisible by a subset of primes in P . This is the reason why trail division of B-smooth numbers preferable over is a plain naive trail division.

Then the product of subsequence of x i.e. $q(x_1) \cdot q(x_2) \cdot \dots \cdot q(x_k)$ produced a square *iff* the exponent vectors has only even entries. That is that it's the *null*-vector in mod 2 space. So with the collection of smooth numbers we want to form their exponent vectors, reduce them modulo 2. For that vector space (finite field of two elements) the sequence of vectors

thus becomes linearly dependent and can be solved with Gaussian Elimination to find the non-empty subsequences. The exponent vectors is represented as the rows of a matrix. Hence, the column sums must be even.

IV.2 Deciding on factor base

To factorize during the sieving step we want all the $q(r)$'s to be smooth in respect to the factor base, i.e. $q(r)$ must be divisible by only primes in the factor base. If a prime p divides $q(r_i)$ earlier mentioned polynomial implies following

$$(r_i + \lfloor \sqrt{n} \rfloor)^2 \equiv n \pmod{p_i} \quad (9)$$

Hence, n is a quadratic residue modulo p_i and we only need to consider those primes. Thus the set of primes p_i for which the Legendre symbol $(\frac{n}{p_i})$ is 1 will form the factor base.

IV.3 Sieving step

Only a small fraction of number will be completely factorized by the primes of the factor base. Therefore, a crucial step is to sieve as many numbers as possible. From following observation we can decide on intervals of primes in P that to divide prime numbers of $q(r)$.

$$q(r) = \tilde{r}^2 - n, \tilde{r} = (\lfloor \sqrt{n} \rfloor + r) \quad (10)$$

$$q(r + kp) = (\tilde{r} + kp)^2 - n \quad (11)$$

$$q(r + kp) = \tilde{r}^2 + 2\tilde{r}kp + (kp)^2 - n \quad (12)$$

$$q(r + kp) = q(r) + 2\tilde{r}kp + (kp)^2 \quad (13)$$

$$\equiv q(r) \pmod{p} \quad (14)$$

So the solution $q(r) \equiv 0p$ for r_i yields a sequence $q(r_i)$ that is divisible by p , thus

sieving will find many factorizable numbers in the defined factor base.

Gauss-eliminering När tillräckligt många användbara x_i och $f(x_i)$ har påträffats återstår det bara att pröva olika linjärbaserade kombinationer av dessa som ger jämna kvadrater för produkten av $f(x_i), i \in [1, k]$. För att vara säkra på att ett linjärt beroende existerar behöver vi samla in ett mer r än antalet primtal i faktorbasen.

Att hitta sådana linjärbaserade kombinationer kan göras genom att bilda matrisen A med radvektorer så som exponenterna för primtalen i faktorbasen för vart och ett av de utvalda x_i :na, räknat modulo 2. Uppgiften blir sedan att finna nollrummet för A^T , vilket görs enklast med Gauss-eliminering.

Härifrån återstår endast att pröva de jämn-kvadrat-bildande linjärbaserade kombinationer av x_i och y_i som nollrummet medger tills en icke-trivial lösning hittas med hjälp av \gcd .

V. IMPLEMENTATION

For our implementation we used several papers as reference. One few that really come handy lecture notes by Carl Pomerance [4]

VI. RESULTS

These tests were solely based on the score and running time on the KATTIS problem `oldkattis:factoring`.

VII. DISCUSSION

We chose to work with quadratic sieve since it's one of the most efficient integer factorization algorithms out there. It is the

fastest algorithm for factorization up to approximately 110 bit integers, there was no doubt it would be a challenge, but since we had knowledge that Pollard's Rho combined with Brent's cycle finding granted approximately 75 points in KATTIS and that the code was not a great challenge to write we wanted to give ourselves a challenge and write a state of the art algorithm. We gave ourselves the time to investigate the more complex alternative and ended up succeeding but still have the basic knowledge about Pollard's Rho since that was our lifeline if our implementation would not have worked.

The result was as expected from a correctly implemented quadratic sieve method, however, on the first submit (that was working) we got a score of 85 points which was due to partly a lack of general optimization but the main reason was making sure the smoothness bounds worked correctly. There are a lot of mathematical parts to implementing this algorithm, although not very complex math in theory getting all the parts of the algorithm to work well was a tough job. The understanding of the concept came as we went along and step-by-step we improved our code and realized the earlier errors we made and in the end realized that the concept is not hard to grasp if you divide the problem in to subproblems and follow pre-existing formulas to execute it.

VIII. CONCLUSION

In hindsight there is no regret of choosing the more complex solution to the problem, it did not only give us more understanding of how to solve the problem we were given in an intelligent way but we also managed

to successfully implement the quadratic sieve algorithm to achieve maximum score in KATTIS. Of course the solution can be optimized further but considering the time constraints we are satisfied with our work.

REFERENCES

- [1] G.H. Hardy, E.M. Wright, D.R. Heath-Brown, and J. Silverman. *An Introduction to the Theory of Numbers*. Oxford mathematics. OUP Oxford, 2008.
- [2] Johan Hastad. Notes for the course advanced algorithms, 2000.
- [3] Carl Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemars-son, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer Berlin Heidelberg, 1985.
- [4] Carl Pomerance. Smooth numbers and the quadratic sieve. In in *[Buhler and Steenhagen 2007]*. *Citations in this document*: §5. University Press, 2005.
- [5] Weissteinm Eric W. Brent’s factorization method.