A Thorough Analysis of Quadratic Sieve Factoring Algorithm and Its Comparison to Pollard-rho Factoring Algorithm

Zongxia Li
Computer Science and Mathematics
University of Maryland
zli12321@umd.edu

William Gasarch
Computer Science Affiliate in Mathematics
University of Maryland
gasarch@umd.edu

1 Abstract

One of the most significant challenges on cryptography today is the problem of factoring large integers since there are no algorithms can factor in polynomial time and factoring large numbers more than some limits(200 digits) remain difficult. The security of the current crypto systems depend on the hardness of factoring large public keys. In this work, we want to use the implement two existing factoring algorithms - pollard-rho and quadratic sieve - and compare their performance. In addition, we want to analyze how close is the theoretical time complexity of both algorithms compare to their actual time complexity and how bit length of numbers can affect quadratic sieve's performance. Finally, we verify whether quadratic sieve would do better than pollard-rho for factoring numbers smaller than 80 bits.

2 Introduction

The idea of public key cryptography was first introduced in 1975 by Martin Hellman, Ralph Merkle, and Whitfield Diffie at Stanford University (Diffie, 1988). Before the era of public key crypto system, it two people want to exchange secret information without anybody else knowing, they have to agree in advance on a secret key that is known only by them but not anyone else. After the invention of public key systems, two people can exchange secret information without ever meeting each other. The idea is that the secret message can only be decrypted in a reasonable amount of time using secret keys possessed by two people who are exchanging information. Then Ron Rivest, Adi Shamor, and Leonard Adleman introduced the RSA public key cryptosystem that is considered to be more secure than previous cryptosystems. This cryptosystem is implemented based on two ideas: public-key encryption and digital signatures (Milanov, 2009). In the public-key encryption part, the sender generates a random prime p, a base number g, and a random number a. Then the sender sends need to send (p, g, g^a((mod p)) to the receiver. After the receiver receives the number, the receiver then generates a random number b, and send g^b (mod p) back to the sender. The sender and the receiver can then compute the shared secret key easily based on the information they have (Diffie and Hellman, 1976). However, for a third party to know the secret, they have to compute g^{ab} with four values - p, g, g^a((mod p), and g^b (mod p). This is the place where factoring algorithm comes in. If the public key is a products consisting of two large primes, where each prime is roughly 512 bits, or 1024 bits, or 2048 bits, then the third party would have to find the factors for the public key to find out the secret key. There are already many existing factoring algorithms. However, none of them can factor large products in a reasonable time- polynomial time. Nowadays, it is feasible to factor 155- decimal digit numbers but it is still considered hard to factor numbers more than 150 decimal digits (Duta et al., 2016). Although we cannot examine the runtime of those algorithms on large products- more than 512 bits, we can analyze some of those algorithms with relative small scale numbers- numbers fewer than 100 bits. Pollard-rho and quadratic-sieve are two of popular factoring algorithms we analyze in this work.

We implement the pollard-rho algorithm based on the birthday paradox and probability theory (Gasarch, fall 2020). Then we implement the quadratic sieve algorithm and optimize some important steps using fast Gaussian elimination. These two algorithms serve to factor products of two primes. Runtime between the two algorithms will be compared and analyzed, then runtime of quadratic sieve will be analyzed when the factors of the products have different bit lengths.

The contribution of this work has is summarized as follows:

- Our experiment shows that that pollard-rho performs better than quadratic sieve most of the time for numbers under 80 bits.
- We do an extensive analysis on the runtime of quadratic sieve under different circumstances- the bit difference of the two factors are large and the bit difference of the two factors are small. Our result shows that for products of the same bit length, the average runtime of quadratic sieve tends to be shorter when the bit difference of two factors are smaller. We verified that quadratic sieve does better when the bit difference of the two factors are small.

3 Related Work

In this section, we briefly review the work related to pollard rho and quadratic sieve. In 1983, Joseph L. Gerver implemented quadratic sieve algorithm(QS), continued algorithm(CF) of Brillhart and Morrison and continued algorithm with early abort modification(CFEA) (Gerver, Jul 1983). In his work, he factored a 47-digit number that has never been factored before into three primes using quadratic sieve, and that number is 176749718190056652686682009038227579300761 1. He also compared the runtime of QS, CF, and CFEA. It turns out that QS starts to do better than CF when the product exceeds 40 bits, and QS does better than CFEA when the product exceeds 60 bits. Although QS could beat CF and CFEA easily, when the product bit length exceeds 60 bits, pollard-rho is considered to do better than QS theoretically for products under 100 bits.

Peter Montgomery, an American mathematician who wokred at the System Development Corporation and Microsoft Research then did a modification to quadratic sieve and named it **Multiple Polynomial Quadratic Sieve**. Robert D. Silverman later implemented this modification of quadratic sieve and factored 45 digit numbers in 0.25 hours and 82 digit numbers in 1265 hours (Silverman, 1987). Before Silverman's time, there are only two implementations of quadratic sieve algorithm. The second implementation, done from the 'Cunningha, Project', used a Cray XMP supercomputer to factor the number $(10^{71}$ - 1)/9 (about 70 digits) in 9.5 hours (Silverman, 1987). By the year 1994, quadratic sieve was able to a 129- digit RSA number (Pomerance, 1967). As better supercomputers and parallel machines are made, quadratic sieve is able to factor numbers with more digits.

Pollard-Rho factoring algorithm. Aminudin et al. analyzed the runtime of pollard-rho and their experiment results shows that their algorithm was able to factor a 44 bit number- 11752700814259- at 7.394 seconds and a 66 bit number- 49808531654765413631- at 28 seconds. They concluded that pollard-rho was significantly faster than Fermat's factorization (Aminudin and Cahyono, April 2021). As a comparison, our pollard-rho algorithm is able to factor 11752700814259 in 0.0007 seconds and 49808531654765413631 in 0.06 seconds.

4 Methodology

We first show the theories behind pollard rho and our implementation of it. Then we show the mathematical steps required to implement quadratic sieve and what techniques we incorporate into optimizing the the quadratic sieve algorithm.

4.1 Pollard Rho Factorization

The pollard-rho factoring algorithm is based on a probabilistic method to factor composite numbers N by finding the greatest common divisor(gcd) between the difference of two random numbers x and y (x and y are between 1 and N - 1) generated by an arbitrary function and N iteratively. The hope is that we could somehow find a pair of random numbers x and y such that the gcd(x - y, N) is the factor of N. The method is published by J.M. Pollard in 1975 and is based on the Birthday Paradox problem (Barnes, Dec 7 2004). The Birthday Paradox states that if we are to choose m samples from N items with replacement, with m to be large enough, some items will be chosen twice. The pollard-rho algorithm uses the following sequences to generate random x and y values for finding the gcd of x-y and N:

$$x_0 \leftarrow \text{random positive integer (mod N)}$$

$$\text{c} \leftarrow \text{random positive integer (mod N)}$$

$$x_i \leftarrow f_c(x) = x_{i-1} * x_{i-1} + c \text{ (mod N)}$$

$$y_0 \leftarrow f_c(x) \text{ (mod N)}$$

$$y_i \leftarrow f_c(f_c(y_{i-1})) \text{ (mod N)}$$

Since the sequence can be at most N-1, when the sequence gets larger, the sequence will finally become periodic. We find the nontrivial factor of the composite if gcd(x-y, N) $\neq$ 1 and gcd(x-y, N) $\neq$ N. According to the Birthday Paradox, the time to find a nontrivial factor is proportional to the size of N[1]. The expected number of steps to find the factor is approximately $N^{\frac{1}{4}}$. The runtime is considered extremely good with one flaw: the algorithm never stops if it cannot find a nontrivial factor.

The implementation of our algorithm is shown below:

Algorithm 1 Pollard-Rho Algorithm

$c \leftarrow rand(1, N-1)$
$f_c(x) \leftarrow x * x + c$
$x \leftarrow rand(1, N-1)$
$y \leftarrow f_c(x)$ (mod N)
while True **do**
  $x \leftarrow f_c(x)$
  $y \leftarrow f_c(f_c(y))$
  $d \leftarrow gcd(x-y, N)$
  if $d \neq 1$ and $d \neq$ N **then**
    return d
  end if
end while

Besides the pseudocode of the shown above, we also added two extra tricks into the algorithm. We first check whether N is a prime or not then we run the algorithm to prevent infinite loop. Before we run the major loop in the algorithm above, we check whether N is divisible by first ten primes to speed up the algorithm.

4.2 Quadratic Sieve Algorithm

The quadratic sieve algorithm is a more complicated factoring algorithm which contains several parts.

4.3 Safe Prime Generators

We want to generate random primes of certain lengths but sometimes it is expensive to check whether a large number is a prime, or even impossible. Thus, we want to generate numbers that are considered to be primes with high probabilities, but can still be composites with a really low probability.

[1] https://www.cs.umd.edu/users/gasarch/COURSES/456/F20/lecfactoring/bday.pdf

5 Experiments

In this section, we introduce experimental setup and present the results of our experiment. Concrete and specific examples will be presented to give a better understanding of our results.

5.1 Experimental Setup

Dataset We want to test the performance of the two algorithms on composites with different bit lengths. We used safe prime method to generate safe primes of different bit lengths- 40 bits, 50 bits, and 60 bits. We also want to compare the performance of both algorithms for composites of certain bit lengths but with prime factors of different bit lengths. An example for generating 40 bit composites is a combination of 5 bit and 35 bit primes multiplied together to get a 40 bit product. The list of combinations of primes we generated are shown in table 1. In addition, we want to examine the two algorithms' performance on composites of random bit lengths (The bit lengths of the composites are not neccessarily multiples of 5). Thus, we generated 4462 pairs of primes with random bit lengths but the products of those pairs of primes do not exceed 70 bits. The experiment is run in a personal laptop macbook pro with a ram memory of 16 GB and 2GHz Quad-Core Intel Core i5. no parallel machine nor GPU is used in the whole experiment.

Composite Size	Prime 1 Size	Prime 2 Size
40	5	35
40	10	30
40	15	25
40	20	20
50	5	45
50	10	40
50	15	35
50	20	30
50	25	25
60	5	55
60	10	50
60	15	45
60	20	40
60	25	35
60	30	30

5.2 Bit Length of Products and Runtime of the Two Algorithms Comparison on Random Bit Composites

In this section, we limit the runtime of finding suitable B and M parameters for quadratic sieve

algorithm to 3 minutes for the sake of limited time for this project. We want to compare the runtime of the two algorithms on products in different bit lengths. We generated 4462 products of random bit lengths and tested the runtime of both programs. Figure 1 visualizes the runtime of quadratic sieve algorithm on products of random bit lengths. Figure 2 shows the runtime of pollard rho algorithms for the same set of products. Among 4462 products, only 4268 products are being successfully factored in the time limit. From the two graphs, we see that the maximum runtime of quadratic sieve is about 1.2 seconds. On the contrary, the highest runtime for pollard rho on the products is about 0.175 seconds. Thus, we see that pollard rho completely beats quadratic sieve on random bit length products no greater than 70 bits. In addition, it seems to take quadratic sieve the longest time to factor products of around 65 bit length. The runtime of quadratic sieve seems to increase as the bit length of the products increase, which looks reasonable. However, there are some numbers between bit length of 50 bits and 60 bits that took significantly longer runtime than other numbers smaller than 60 bits. Here, we consider the time to factor a number smaller than 60 bits more than 0.6 seconds to be long. Table 2 shows the numbers smaller than 60 bits but took more than 0.6 seconds to factor.

For Pollard rho, it is not surprising that the runtime of of the program gradually increases as the bit length of the products increases. As the number gets larger, the number of cycles to hit the correct candidate factors becomes bigger. There is a significant jump of the runtime of Pollard rho from 60 bit to 70 bits.

5.3 Analysis of Quadratic Sieve's Parameters

B value is a list of primes smaller than B. We want to use the list of primes to decompose numbers into factors only from that list of primes. Then we use the derived B prime factors for each number to form a matrix. The larger the B value is, the more number of primes we will have, which means we will form a bigger matrix in our program to solve. Theoretically, the runtime of the program is positively related to the B value. Figure 3 shows the logistic regression line of the size of the B value and the runtime of quadratic sieving algorithm. The B value is indeed positively correlated to the runtime of the quadratic sieve.

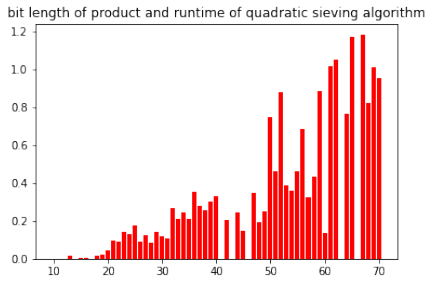

Figure 1: The histogram shows the bit length of composites and the runtime corresponding to that bit length for quadratic sieving algorithm

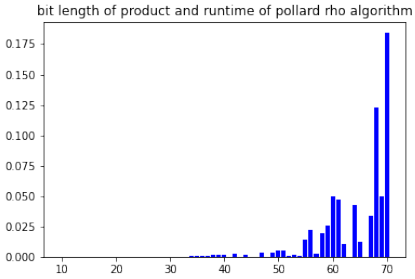

Figure 2: Bit length of composites and the runtime corresponding to that bit length for Pollard ρ

5.4 M Values and Runtime of Quadratic Sieve

Now we want to examine how the M values relate to the runtime of quadratic sieve algorithm. Starting from the square root of the number we want to factor, M value is how much farther we want to go from that number. For example, if the number we want to factor is 337102662581 and square root of 337102662581 is 580605. If we pick an M value of 100, we will do B factoring on numbers 580605 to 580705. The large the M value is, the more numbers we will have to do B factoring. This also means we will have more rows in the matrix we have to deal with, Thus, the M value should be positively correlated to the runtime of the program. Figure 4 shows the size of the M value and the runtime of the program. To factor the same composite using quadratic sieving algorithm, if we find an optimal M value, and we use an M value that is larger than the optimal M value, then the runtime of the algorithm will increase.

5.5 Comparison of Runtime on Products of fixed length

In the following section, we will compare the runtime between quadratic sieve and pollard

Table2. Bit difference and average runtime of quadratic sieve

Product	Facto 1	Factor 2	factor 1 length	factor 2 length
27522003582288223	2963	9288560102021	12	44
1751357076372217	4441	394360971937	13	39
349739602979093	291148573	1201241	29	21
185456974731188183	18796061	9866800003	25	34

rho for 40-bit products, 50-bit products, and 60- bit products. We randomly generated 1276 products of 40 bits and we factored them using both algorithms. It turns out that pollard rho does better than quadratic sieve for all 1276 numbers. Then we generated 2749 products of 50 bits and factored them using both algorithms. There are only two numbers quadratic sieve does better than pollard rho. These two numbers are 332984151088201 and 455170965415169. Both numbers are products of two 25-bit primes. We then generated 3561 products of 60 bits and there are 5 products quadratic sieve does better than pollard rho. These five products are 433201593543320029, 35565484892954009, 466227204332054881, 427100149055786449, and 485273147388660049. These numbers are all products of two 30-bit primes. Although the number of products quadratic sieve does better than pollard rho increases as the bit length of products increases, this does not mean there will be more numbers quadratic sieve performs better than pollard rho because the numbers these two algorithms factor are increasing from 1276 to 2749 to 3561. In all of the cases, quadratic sieve only does better than pollard rho when the two factors have the same bit length. We will look into whether quadratic sieve will perform better when the bit length of two factors are close together in later sections.

6 Checking the Failure Rate of Quadratic Sieve for 40-bit, 50-bit, and 60-bit Products Within Three Minutes

Due to the time limits of this project, it is impossible to allow the program to run infinite amount of time to factor some numbers that are hard to factor. Thus, if it takes the program more than three minutes to find the suitable B and M values and factor a product, we will cut off the program and factor the next product.

6.1 40-bit Products Failure Analysis

For 40-bit products, there are 14 numbers quadratic sieve cannot factor. The numbers the program cannot factor is shown is table 3 in the appendix. From table 3, we see that all the products quadratic sieve cannot factor are composed of a 10-bit prime factor and a 30-bit prime factor. We will examine more thoroughly on 50-bit and 60-bit products and see whether the same pattern appears.

6.2 50-bit Products Failure Analysis

For 50-bit products, there are a total of 206 products quadratic sieve cannot factor within three minutes. The following table 4 shows the bit length of two primes and how many of them cannot be factored on time. From table 4, we see that the most number of products that cannot be factored are combinations of 10-bit primes and 40-bit primes. We will question whether the products that contain 10-bit primes are harder to factor.

6.3 60-bit Products Failure Analysis

The number of 60-bit products that cannot be factored and combination of bit length of primes is shown in table 5 in the appendix. However, for 60-bit products, the most of products that cannot be factored is not combination of a 10-bit prime and a 50-bit prime. It is rather a combination of a 5-bit prime and a 55-bit prime. One possible reason for why the most number of products cannot be factored is the combination of bit lengths between the two prime factors. When the difference of bit lengths of two primes are farther apart, then it takes more time for quadratic sieve to factor the numbers.

7 Percentage of Products that Cannot Be Factored For 40-bit, 50-bit and 60-bit Products

Because we have different number of samples for 40-bit, 50-bit and 60-bit products, we cannot just

look at how many products quadratic sieve cannot factor for these bit length products. We must look at the percentage of products quadratic sieve cannot factor given the bit length of the product. It turns out that for 40-bit products, about 1.097 percent of the numbers cannot be successfully factored within three minutes. For 50-bit products, about 9.657 percent of the numbers cannot be factored within three minutes. For 60-bit products, the failure rate is up to 65.403 percent. Thus, the longer the bit length of the product is, the harder it is for quadratic sieve to find out the suitable B and M values and factored the products on time. We cannot say quadratic sieve performs worse when the product gets larger. When the size of a product gets larger, the algorithm may spend most of its time finding the correct B and M values instead of doing the actual process of factoring the number.

8 How Well Does Quadratic Sieve Perform When the Bit Difference of Primes Changes

Table 6 shows how quadratic sieve performs when the bit difference of the two prime factors vary. Especially, they show the percentage of numbers that can be successfully factored within three minutes for different combinations of bit length primes.

From the table 6, we see that except for 40-bit products, the success rate of factoring in three minutes seem to be the lowest when the bit difference of two primes are farthest apart and as the bit difference gets closer, the success rate slightly increases for 50-bit and 60-bit products.

9 Comparing Average Runtime for Quadratic Sieve when the Difference of Bit Lengths for the Two Primes are Close or Far

From the last section, we concluded that the success rate within three minutes for quadratic sieve to factor a number is the lowest when the bit difference of the two primes are farthest apart. Thus, we also want to make a hypothesis that as the bit difference between the two products are farthest, the average runtime of quadratic sieve should take the longest among other bit differences. Table 7 in the shows the average runtime of for the algorithm for different bit length products and different bit difference

Indeed, from table 7, we sees that for 40-bit, 50-bit and 60-bit products, the average runtime in seconds to factor a product is the highest when the bit difference are the greatest. Thus, the runtime of quadratic sieve does not only depends on the bit length of the product, but also on the bit difference of the two prime factors (Çetin K.Koç and N.Arachchige, 1991).

10 Limitations

Computer Resources We have limited computing power for running the two algorithms. The only computer we use is a mac book pro with 16GB memory. Quadratic sieve algorithm would speed up if it runs in a parallel machine, since it can Gaussian elimination simultaneously on multiple rows in a matrix. However, for a normal personal computer, quadratic sieve algorithm starts to take a long time to find the correct B and M values. With the time limit, the success rate of factoring a number for quadratic sieve becomes much lower when the bit length of the product exceeds 70 bits. Thus, we did not run quadratic sieve and pollard rho on numbers greater than 70 bits. In theory, however, quadratic sieve should does better than pollard rho when the product is over 100 bits. We ran an experiment on both algorithms for an 80-bit number, the runtime of pollard rho is about 22.95 seconds, and the runtime of quadratic sieve is about 44.25 seconds. Afterwards, we ran both algorithms on a 100 bit number and pollard rho did not successfully factor the number in ten minutes. Quadratic sieve factored the number in 587.66 seconds. Although there is only one example for 100-bit numbers, we at least find an example that satisfies the theory.

Time Limits Although the time limit of three minutes for quadratic sieve and pollard rho greatly saves our time on factoring thousands of examples, we forgot to record the time for finding the correct B and M values for quadratic sieve. This is important because for most numbers, quadratic sieve cannot factor them for not finding the correct B and M values on time, not for other reasons. If we have a better algorithm for finding the correct B and M values for quadratic sieve, then the success rate of factoring numbers of quadratic sieve would be much higher.

Limited Sample Sizes Because we can only do analysis for 40-bit, 50-bit and 60-bit products, the comparisons we can do for pollard rho and quadratic sieve are very limited. We cannot see a significant performance of quadratic sieve over

pollard rho for 100-bit length numbers since for each pair of B and M values, it would take 10 minutes for quadratic sieve to factor and maybe more than 10 minutes for pollard rho to factor. The time cost of doing large bit length numbers is very high. Also, the failure rate of factoring 80-bit numbers for quadratic sieve is high for a three minute timing- about 90 percent failure rate. The limited sample size really limits the type comparison we want to do between two algorithms.

11 Summary and Outlook

11.1 Time Span to run the algorithms

Increase the timespan for factoring each product. It takes 587.6639738 seconds- almost 10 minute for quadratic sieve to factor the product. Also, we have to count for the time to find out the B and M values, which means 20 minutes to factor each product. The time to run the algorithms might take a month.

11.2 Factor more specific bit length primes

Find how much time it takes to factor a 40 bit number when the two products are 5 bit and 35 bit, 6 bit and 34 bit, 7 bit and 33 bit, etc... Do the same for 45 bit, 50 bit, 55 bit 60 bit products.

11.3 Finding a better algorithm for finding B and M values

Instead of increment B and M by 10 each alternately starting B = 10 and M = 10, for larger numbers, we start B and M with 100 and 1000 and increment the two values by 100 or other intervals alternately.

References

Aminudin and Eko Budi Cahyono. April 2021. A practical analysis of the fermat factorization and pollard rho method for factoring integers.

Connelly Barnes. Dec 7 2004. Integer factorization algorithms.

Whitfield Diffie. 1988. The first ten years of public key cryptography.

Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography.

Cristina-Loredana Duta, Laura Gheorghe, and Nicolae Tapus. 2016. Framework for evaluation and comparison of integer factorization algorithms.

William Gasarch. fall 2020. Pollard's ρ algorithm for factoring(1975).

Joseph L. Gerver. Jul 1983. Factoring large numbers with a quadratic sieve.

Çetin K.Koç and Sarath N.Arachchige. 1991. A fast algorithm for gaussian elimination over gf(2) and its implementation on the gapp.

Evgeny Milanov. 2009. The rsa algorithm.

Carl Pomerance. 1967. A tale of two sieves.

Robert D. Silverman. 1987. The multiple polynomial quadratic sieve.

Product	Bit length of factor 1	Bit length of factor 2
369062373143	10	30
534106988197	10	30
508085497589	10	30
369901521617	10	30
369901521617	10	30
458343342553	10	30
341238699311	10	30
421135689817	10	30
563755932497	10	30
482560209653	10	30
477265583519	10	30
456886122281	10	30
442321850393	10	30
386706654493	10	30

Table 3. 40-bit products that cannot be factored by quadratic sieve within three minutes

Bit length of factor 1	Bit length of factor 2	Number of them can't be factored
5	45	38
10	40	51
15	35	38
20	30	37
25	25	42

Table 4. 50-bit products that cannot be factored by quadratic sieve within three minutes

Bit length of factor 1	Bit length of factor 2	Number of them can't be factored
5	55	417
10	50	383
15	45	367
20	40	383
25	35	373
30	30	406

Table 5. 60-bit products that cannot be factored by quadratic sieve within three minutes

Bit difference	Bit length of factor 1	Bit length of factor 2	Success rate in three minutes
30	5	35	1.0000
20	10	30	0.9646
10	15	25	1.0
0	20	20	1.0
40	5	45	0.9023
30	10	40	0.9136
20	15	35	0.9354
10	20	30	0.9370
0	25	25	0.9294
50	5	55	0.2992
40	10	50	0.3541
30	15	45	0.3822
20	20	40	0.3541
10	25	35	0.3721
0	30	30	0.3142

Table 6. Performance of quadratic sieve on different combination of primes of different bit lengths

Bit length of the product	Bit difference	average runtime in seconds
40	30	0.123
40	20	0.122
40	10	0.096
40	0	0.098
50	40	0.245
50	30	0.154
50	20	0.146
50	10	0.143
50	0	0.140
60	50	0.242
60	40	0.233
60	30	0.240
60	20	0.236
60	10	0.242
60	0	0.224

Table 7. Bit difference and average runtime of quadratic sieve

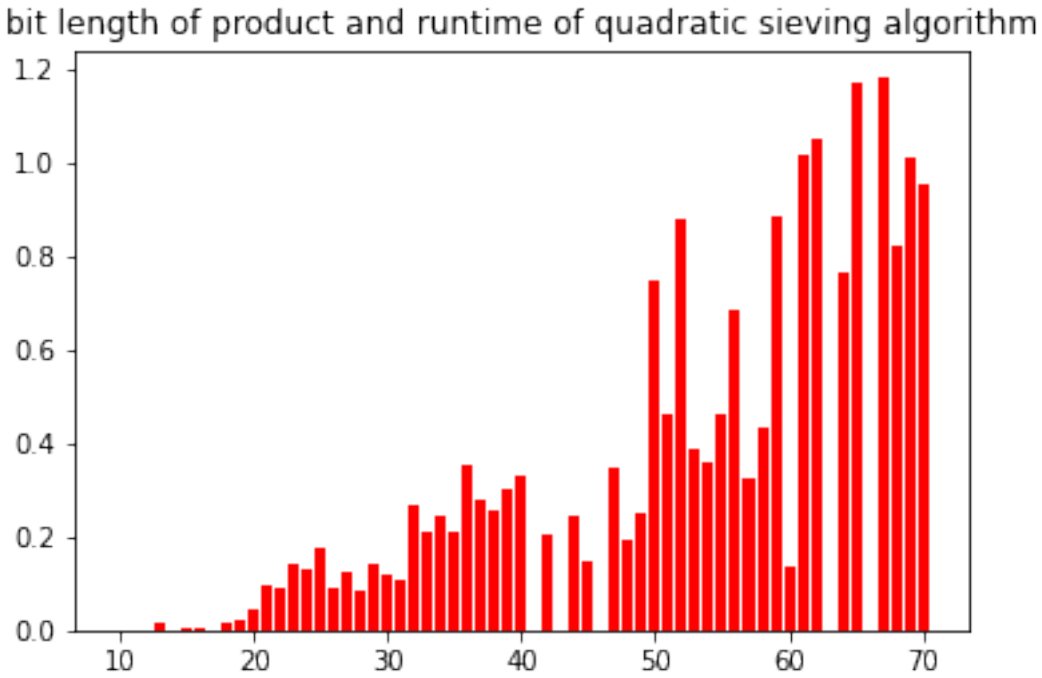

Figure 3: runtime vs. bit length for quadratic sieve

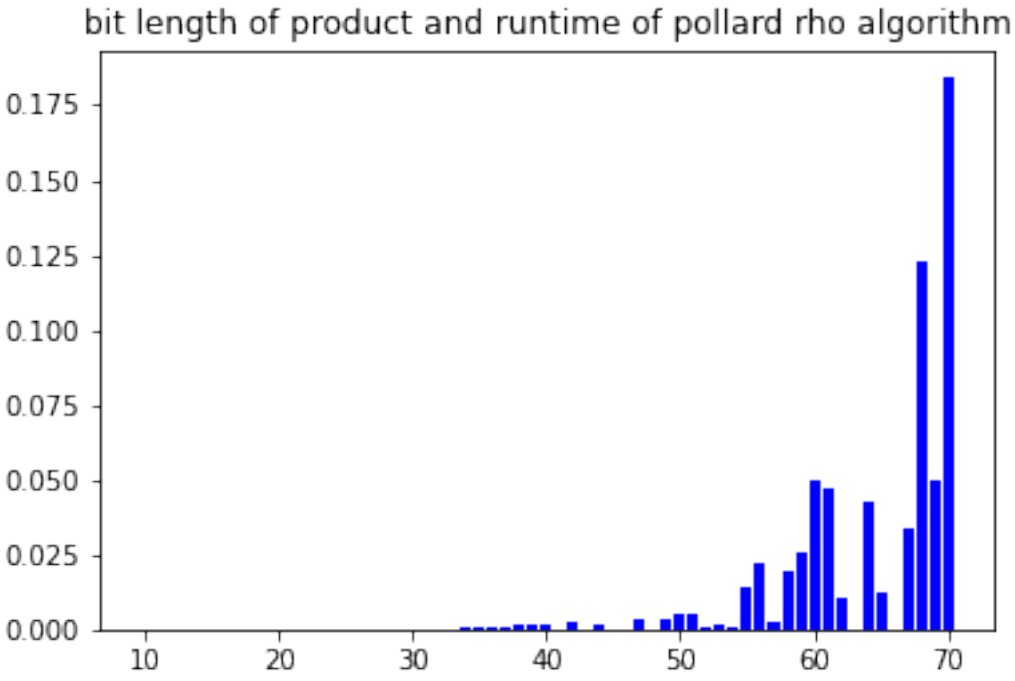

Figure 4: runtime vs. bit length for pollard rho