Analysis of the One Line Factoring Algorithm

Tejas Gopalakrishna and Yichi Zhang

Abstract

For integers that fit within 42 bits, a competitive factoring algorithm is the so-called One Line Factoring Algorithm proposed by William B. Hart. We analyze this algorithm in special cases, in particular, for semiprimes $N = pq$, and look for optimizations. We first observe the cases in which the larger or smaller prime is returned. We then show that when p and q are sufficiently close, we always finish on the first iteration. An upper bound can be found for the first iteration that successfully factors an odd semiprime. Using this upper bound, we demonstrate some simplifications to the algorithm for odd semiprimes in particular. One of our observations is that we only need to iterate numbers $\{0,1,3,5,7\}$ modulo 8, as the other iterators are very rarely the first that successfully factor the semiprime. Finally, we inspect the performance of the optimized algorithm.

1 Introduction

It is well known that the optimal algorithm for factoring large numbers greater than 10^{100} is the general number field sieve, and for numbers less than 10^{100}, the quadratic sieve. However, for numbers that fit within 42 bits, there are many other algorithms that are competitive. One of them is Fermat's method which searches for squares a and b such that $N = a^2 - b^2 = (a+b)(a-b)$, hence finding factors. Some others are Lehman's algorithm [1] and Shanks' Square Forms Factoring algorithm [2]. More recently, Hart [3] revealed a competitive algorithm for integers of this size:

OLF(N):

For $k := 1, \ldots, N$:

$s := \lceil \sqrt{k \cdot N} \rceil$

$m := s^2 \bmod N$

If m is square:

$t := \sqrt{m}$

Return GCD$(N, s-t)$

This algorithm gets its name from the fact that it can be implemented in one line of Pari/GP or Sage. It is a variant of Lehman's algorithm, but only requires one variable to be iterated instead of two.

The goal of this paper is to analyze runtime improvements upon changing the set of k that the algorithm iterates over, and explore ways to optimize the algorithm, in particular, for odd semiprimes $N = p \cdot q$.

In Section 2, we observe some cases when the smaller or larger prime is returned by the algorithm, we look at the set of semiprimes that are factored in the first iteration, and conjecture a bound on the number of iterations required to factor odd semiprimes.

In Section 3, we use this conjecture to reduce the modulus operation in the algorithm to a subtraction operation, and we propose an improvement to the algorithm.

In Section 4, we observe the practical runtime of the algorithm on semiprimes, and compare our proposed algorithm to demonstrate the speedup.

2 Observations

2.1 Notation

We use the nonstandard notation $a \bmod N$ to represent a unique number $x \in \{0, 1, \ldots, N-1\}$ such that $a \equiv x \pmod{N}$.

We also refer to the algorithm described by Hart as OLF(N). We refer to our modified version of the algorithm (for semiprimes) as OLF_MOD(N).

2.2 Result of factoring semiprimes

Consider the graph in Figure 1. We let the y-axis coordinate represent the index of prime p (with $y = \pi(p)$), and the x-axis coordinate represent the index of prime q (with $x = \pi(q)$), where π is the prime-counting function. Each point in this graph represents a semiprime $N = pq$. A point is colored green if the smaller factor is returned by the algorithm, and colored blue if the larger factor is returned.

Note that the diagonal "band" from the bottom left to top right is solid green. This suggests that where p, q are close, the algorithm returns the lower number. In particular, we claim:

Proposition 1. *When p and q are odd primes, and $q - p < 2\sqrt{2p} + 2$, the algorithm always returns the smaller prime.*

Proof. Without loss of generality, let $q > p > 2$. In the iteration $k = 1$, s in the algorithm is $\frac{p+q}{2}$, and m in the algorithm is $\frac{(q-p)^2}{4}$, as in the proof of Proposition 2. Here, m is a perfect square.

t in the algorithm is $\sqrt{m} = \frac{q-p}{2}$. The factor is $\text{GCD}(N, s-t) = \text{GCD}(pq, \frac{p+q}{2} - \frac{q-p}{2}) = \text{GCD}(pq, p) = p$. Therefore, the smaller factor, p, is returned. □

2.3 Numbers factored in the first iteration

In Figure 2, we visualize the relative number of iterations required to factor a semiprime. Each point represents a semiprime, as in Figure 1. A semiprime point is colored more black if it is factored in fewer iterations, and more white if it needs more iterations.

Let $2 < p < q$, where p, q are primes. We consider semiprime $N = pq$. From the algorithm, if $m = \lceil \sqrt{iN} \rceil^2 \bmod N$ is a square, then the number can be factored on the i-th iteration. We look the case $i = 1$, the numbers that may be factored within one iteration.

At iteration $i = 1$, we check if $(\lceil \sqrt{N} \rceil^2 \bmod N) = (\lceil \sqrt{pq} \rceil^2 \bmod N)$ is a square.

Lemma 1. $\lceil \sqrt{pq} \rceil^2 = \frac{(p+q)^2}{4}$ *when* $q - p < 2\sqrt{2p} + 2$.

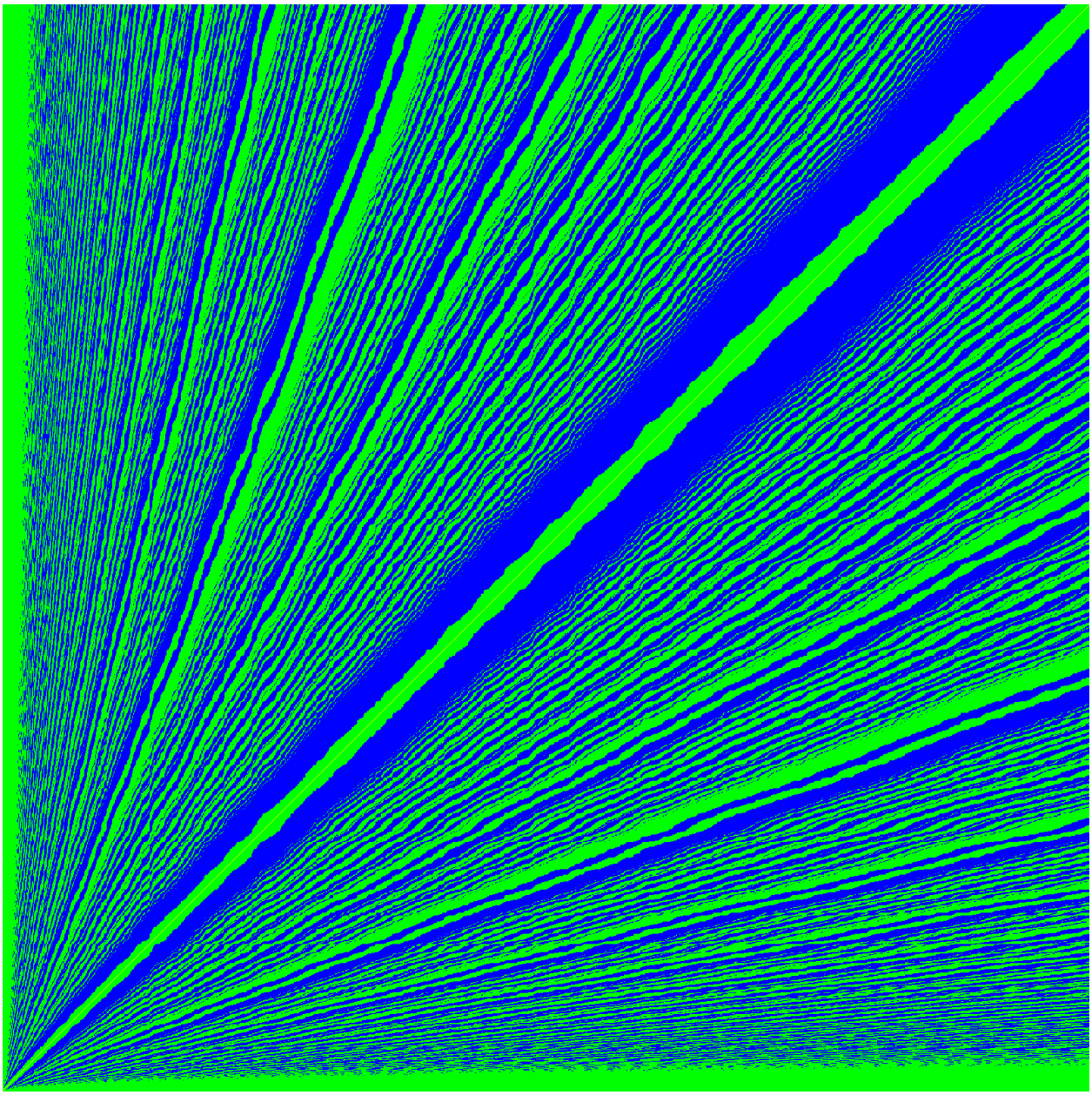

Figure 1: Number returned by OLF(pq): The point is green when factoring the corresponding semiprime returns the smaller prime, and the blue when factoring it returns the larger prime.

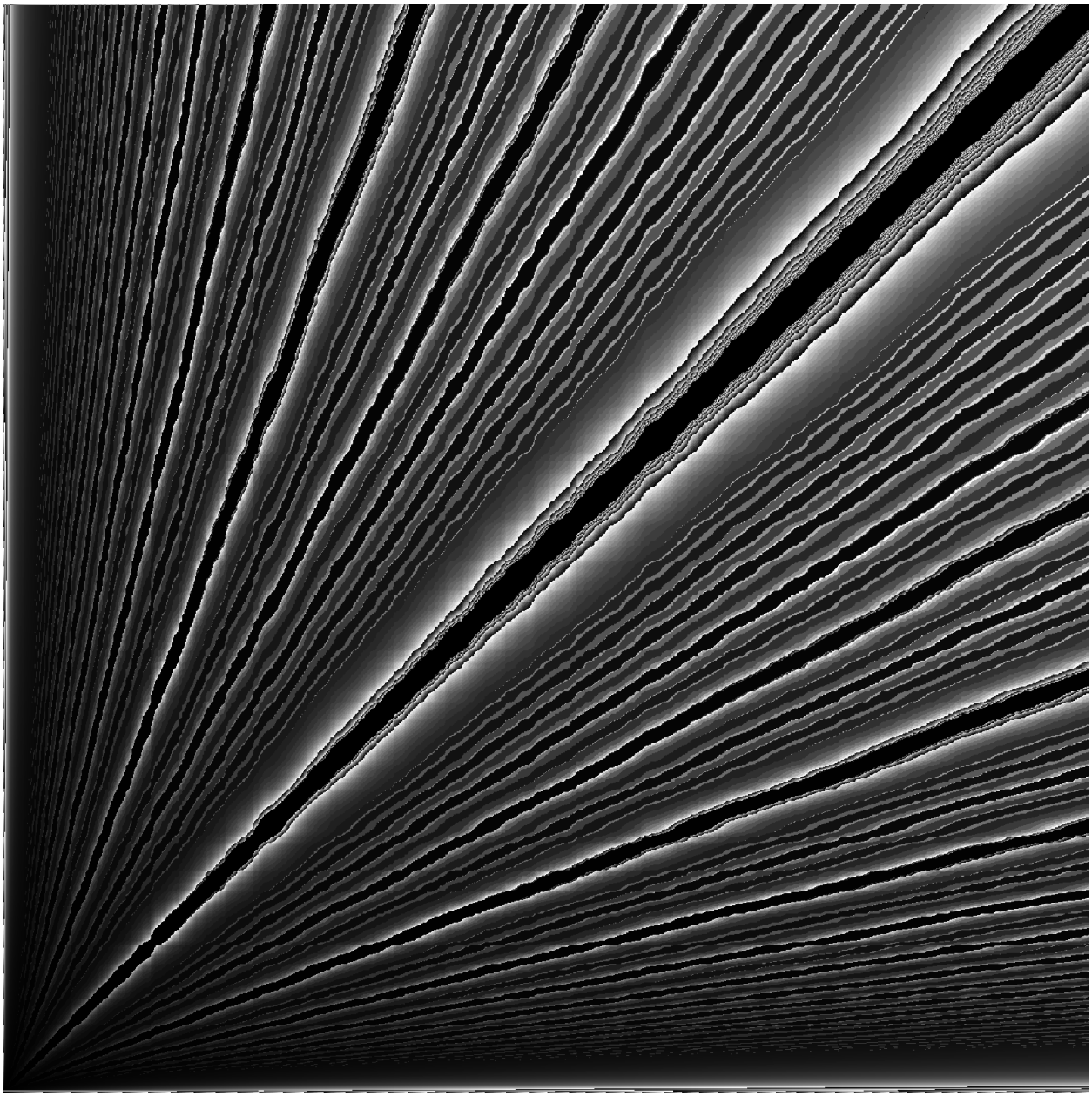

Figure 2: The number of iterations required by OLF(pq) to complete. The point is colored more black when fewer iterations are required, and more white when more iterations are required.

Proof. The expression $\lceil\sqrt{pq}\rceil^2$ can be interpreted as the next perfect square after pq.

Let $a = q - p$. Then $N = pq = p(p+a)$. Note that $(p+\frac{a}{2})^2 = p^2 + pa + \frac{a^2}{4}$ and $(p+\frac{a}{2}-1)^2 = p^2 + \frac{a^2}{4} + 1 + pa - 2p - a$, and that $\frac{a}{2}$ must be an integer (since a is even, the difference between odd primes).

Clearly, $pq = p(p+a) = p^2 + pa \leq p^2 + pa + \frac{a^2}{4} = (p+\frac{a}{2})^2$, since $\frac{a^2}{4} \geq 0$ ($a = q - p$, and $q > p$). So, $N = pq \leq (p+\frac{a}{2})^2$.

$(p+\frac{a}{2}-1)^2 < pq$ follows from $q - p < 2\sqrt{2p}+2$. Since $(p+\frac{a}{2}-1)^2 < pq \leq (p+\frac{a}{2})^2$, and since $\lceil\sqrt{pq}\rceil^2$ is the next perfect square after pq, $\lceil\sqrt{pq}\rceil^2 = (p+\frac{a}{2})^2$. Since $a = q - p$, this can be further simplified to $(p+\frac{q-p}{2})^2 = (\frac{p+q}{2})^2 = \frac{(p+q)^2}{4}$. □

Lemma 2. *$pq < \frac{(p+q)^2}{4} < 2pq$ when $q - p < 2\sqrt{2p}+2$.*

Proof. Let $a = q - p$, so $pq = p^2 + pa$. Then, $\frac{p^2+q^2}{2} = \frac{p^2+(p+a)^2}{2} = \frac{2p^2+2pa+a^2}{2} = p^2 + pa + \frac{a^2}{2} > p^2 + pa = pq$ (since $\frac{a^2}{2} > 0$).

From here we show the lower bound: $\frac{(p+q)^2}{4} = \frac{p^2+2pq+q^2}{4} = \frac{p^2+q^2}{4} + \frac{pq}{2} = \frac{1}{2}(\frac{p^2+q^2}{2} + pq) > \frac{1}{2}(2pq) = pq$.

The upper bound is also easy to derive. It can be shown from $q - p < 2\sqrt{2p}+2$ that $p^2 + q^2 < 8p + 2pq + 2$.

Since $q > p > 2$, it is always true that $1 < p(2q-4)$, so $2 + 8p < 4pq$. This yields $p^2 + q^2 < 8p + 2pq + 2 < 6pq$. Since $(p+q)^2 = p^2 + q^2 + 2pq < 8pq$, the upper bound is $\frac{(p+q)^2}{4} < 2pq$. □

Proposition 2. *$N = pq$ can be factored in the first iteration $i = 1$ when $q - p < 2\sqrt{2p}+2$.*

Proof. We must check if $\lceil\sqrt{N}\rceil^2 \bmod N = \lceil\sqrt{pq}\rceil^2 \bmod pq$ is a square. By Lemma 1, this is the same as $\frac{(p+q)^2}{4} \bmod pq$. By Lemma 2, $0 < \frac{(p+q)^2}{4} - pq < pq$, so $\frac{(p+q)^2}{4} \bmod pq = \frac{(p+q)^2}{4} - pq$. From this, it is clear that $\lceil\sqrt{N}\rceil^2 \bmod N = \frac{(p+q)^2}{4} - pq = \frac{(q-p)^2}{4}$. Since p and q are odd primes, their difference is even, so $(q-p)^2$ contains a factor of 4. Therefore, $\frac{(q-p)^2}{4}$ is a perfect square, and $\lceil\sqrt{N}\rceil^2 \bmod N$ is a perfect square, so such numbers can be factored on the first iteration. □

In Figure 3, we see the region $q - p < 2\sqrt{2p}+2$ that may be factored in one iteration highlighted in red. Note that this is the same as the diagonal black "band" region in Figure 2.

An interesting point we observed is that each of the black "bands" (that start at the bottom left of the figure) consist of points that may be factored in the same iteration. In other words, we observed that in the ranges we manually checked, given a point in a band, all other points in the same band can be factored in the same iteration.

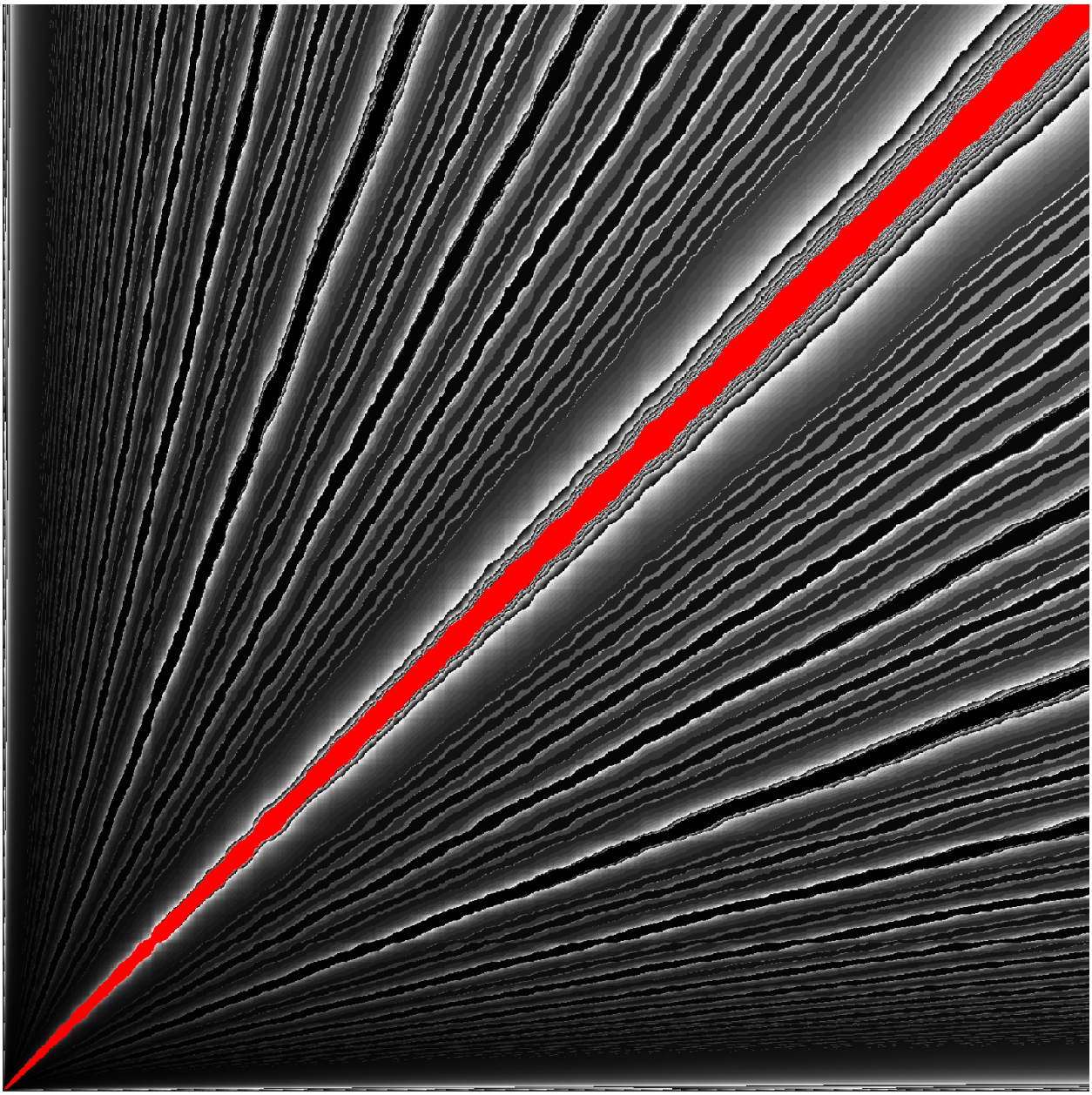

Figure 3: The number of iteration required by OLF(pq) to complete. The points are colored as in Figure 2. The region $q - p < 2\sqrt{2p} + 2$, where one iteration is sufficient, is highlighted in red.

2.4 Bound on the number of iterations

In Figure 4, we plot semiprimes $N = pq$ against their first successful iteration k. We note that the approximate line of points formed outside the region $k < \frac{(N-2)^2}{4N}$ consist of only even semiprimes, which leads to the following conjecture.

Conjecture 1. *For all odd semiprimes N, the first successful iteration k satisfies* $k < \frac{(N-2)^2}{4N}$.

This conjecture has been verified experimentally for the first 10^{15} semiprimes.

Another interesting observation is that similar to the line of even semiprimes, semiprimes divisible by 3 form an approximate line. The same can be observed for semiprimes divisible by other primes, although it is not as clear in the figure.

3 Optimizations

In this section, we look at methods to reduce the runtime of the algorithm for odd semiprimes. This optimized algorithm follows from the conjecture in the previous section.

3.1 The modulus operation

Using Conjecture 1, we can eliminate the modulus operation from the algorithm.

Proposition 3. *If* $k < \frac{(N-2)^2}{4N}$*, where* $N = pq$ *is an odd semiprime, then* $\lceil\sqrt{kN}\rceil^2 \bmod N = \lceil\sqrt{kN}\rceil^2 - kN$.

Proof. Since $k < \frac{(N-2)^2}{4N}$, we see that $4kN < (N-2)^2$, and $2\sqrt{kN} < N-2$. This can be rewritten as $2(\sqrt{kN}+1) < N$.

Clearly, $\lceil\sqrt{kN}\rceil \leq \sqrt{kN}+1$. It follows that $2\lceil\sqrt{kN}\rceil \leq 2(\sqrt{kN}+1) < N$.

In the algorithm, $s = \lceil\sqrt{kN}\rceil$. This means the next square after kN is s^2. More formally, $(s-1)^2 < kN \leq s^2$. It follows that $s^2 - (s-1)^2 > s^2 - kN \geq s^2 - s^2$. This, of course, is $2s-1 > s^2 - kN \geq 0$.

As we found, $2\lceil\sqrt{kN}\rceil < N$, so, $2s < N$. Hence, $N > 2s > 2s-1 > s^2 - kN \geq 0$.

Since $0 \leq \lceil\sqrt{kN}\rceil^2 - kN < N$, $\lceil\sqrt{kN}\rceil^2 \bmod N = \lceil\sqrt{kN}\rceil^2 - kN$.

□

3.2 Iterations 2 modulo 4

We may now optimize the algorithm by removing particular iterations.

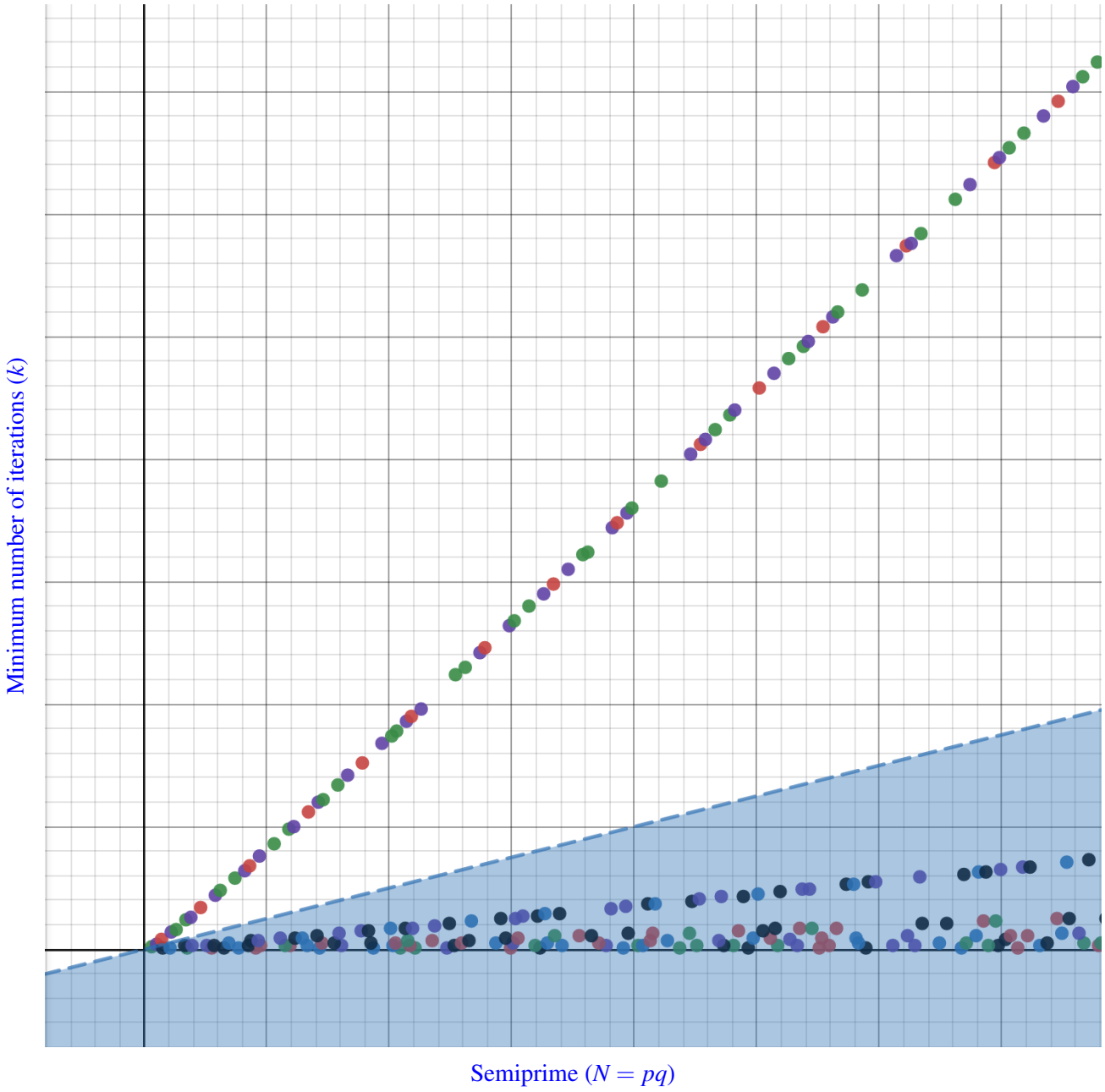

Figure 4: In this graph the points represent semiprimes factored by $OLF(N)$. The graphed region is an approximation of $k < \frac{(N-2)^2}{4N}$.

Proposition 4. *Let $N = pq$ be an odd semiprime. Then, iterations i with only one factor of 2 never result in a factor.*

Proof. For an iteration to result in a factor, m in the algorithm must be a perfect square. Let that square be c^2. Then, since $m = s^2 \bmod N = s^2 - iN$ by Proposition 3, $m = s^2 - iN = c^2$. It follows that $(s+c)(s-c) = iN$.

Clearly, since both s and c are integers, $(s+c)$ and $(s-c)$ share the same parity. If i is even, then there must be two factors of two in i, since $(s+c)$ and $(s-c)$ would both be even (recall that N is odd). Therefore, it is impossible for i to have only one factor of 2. □

3.3 Proposed algorithm

We have shown that if the conjecture in the previous section is true, we do not have to check iterations that are equivalent to 2 mod 4. In addition, we also conjecture that for larger odd semiprimes, iterations equivalent to 4 mod 8 are never the first iterations that result in m being a square. This conjecture has been tested for the first 10^{15} semiprimes.

From these conjectures, we note that the iterations $\{2,4,6\}$ mod 8 are not essential, as they never result in m being a square. Our proposed optimization of the algorithm (for odd semiprimes) follows.

```
OLF_MOD(N):
    For k := 1,...,N:
        If k mod 8 ∈ {0,1,3,5,7}:
            s := ⌈√(kN)⌉
            m := s^2 − kN
            If m is square:
              t := √m
              Return GCD(N, s − t)
```

4 Performance and Comparison

In this section we summarize our results in comparing the performance of the algorithms.

Our implementations were written in the Pascal language using the GNU Multiple Precision Arithmetic Library [4] and compiled with the Free Pascal Compiler version 3.0.4 [5].

Figure 5 compares the runtime, given various sizes of factors in the input semiprime. We note, as in the initial paper by Hart, that semiprimes whose factors are close together are factored quickly.

In Figure 6, we show the average runtimes of the algorithms. Since both algorithms finish quickly when the factors are close, we show an example when the factors are 20 bits apart. Our algorithm has a runtime that is approximately 37.5% faster than the original algorithm, due to the iterators $\{2,4,6\}$ mod 8 being skipped.

	22	**25**	**28**	**31**	**34**	**37**	**40**	**43**
22	<0.05ms	<0.05ms	<0.05ms	0.3ms	9.1ms	15.0ms	23.5ms	46.9ms
25	<0.05ms	<0.05ms	<0.05ms	<0.05ms	0.4ms	53.1ms	26.1ms	12.8ms
28	<0.05ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	0.3ms	1.2ms	9.8ms
31	0.3ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	0.4ms	1.2ms
34	9.1ms	0.4ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	0.4ms
37	15.0ms	53.1ms	0.3ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms
40	23.5ms	26.1ms	1.2ms	0.4ms	<0.05ms	<0.05ms	<0.05ms	<0.05ms
43	46.9ms	12.8ms	9.8ms	1.2ms	0.4ms	<0.05ms	<0.05ms	<0.05ms

Figure 5: Average performance on different semiprime sizes. The top and side headers of the table specify the number of binary bits of p and q, respectively.

Bits of N	**OLF(N)**	**OLF_MOD(N)**
$5+25=30$	1ms	0.9ms
$10+30=40$	1.1ms	0.9ms
$15+35=50$	5.6ms	3.9ms
$20+40=60$	12.8ms	8.1ms
$25+45=70$	13ms	8.2ms
$30+50=80$	325ms	209.8ms
$35+55=90$	337ms	210.4ms
$40+60=100$	315ms	199.2ms

Figure 6: Comparison of the algorithms when $N = pq$, and p and q are 20 bits apart.

5 Summary

We have analyzed particular cases of the One Line Factoring Algorithm, showing areas for which the algorithm is optimal, and we have conjectured a bound for the number of iterations. Using this bound, we have proposed an easily implemented optimization for odd semiprimes that results in a speedup of rougly 37.5%.

6 Acknowledgements

This project was part of the PRIMES program. We are very grateful to the program for the support.

We would like to thank Dr. Tanya Khovanova for her useful suggestions, and Yongyi Chen, for the comments.

We also thank Dr. Stefan Wehmeir from MathWorks for suggesting the project, and for his feedback.

References

[1] R. S. Lehman, Factoring Large Integers, *Math. Comp.* **28** (1974), 637-646

[2] D. Shanks, On Gauss and Composition II, *Number Theory and Applications (Banff, AB, 1988)*, NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., 265 (ed. R. A. Mollin) (Kluwer, Dordrecht, 1989), pp. 179-204.

[3] W. B. Hart, A One Line Factoring Algorithm, *J. Aust. Math. Soc.* **92** (2012), 61-69

[4] Torbjörn Granlund and the GMP development team, GNU MP: The GNU Multiple Precision Arithmetic Library, `https://gmplib.org`

[5] Florian Klaempfl et al., The Free Pascal Compiler, `https://www.freepascal.org`