On the Time Complexity of the Verification of the Factorization of $2^{67} - 1$

Isaac Grosof
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
igrosof@cmu.edu

Isaac Grosof
Computing Hardware

April 1, 2019

Abstract

In 1903, Frank Nelson Cole [4] demonstrated that the 67th Mersenne number, 147573952589676412927, is equal to $193707721 \times 761838257287$, disproving Marin Mersenne's claim of primality from 1644 [9]. Cole demonstrated this equality by silently multiplying the two factors on a blackboard, receiving a standing ovation [3, 5]. Modern sources include the additional embellishment that this demonstration took one hour. We find this claim suspicious, as we grow bored with hour-long lectures in the best of circumstances.

To resolve this discrepancy, we investigate the time complexity of the multiplication of middling-large numbers. We use similar computational hardware to the original demonstration, namely the second author. We investigate a variety of multiplication algorithms, including algorithms that Cole might have used and more recent algorithms. We find that Cole could have performed the demonstration in about 10 minutes.

1 Introduction

In 1644 [9], Marin Mersenne claimed without proof that $2^{67}-1$ is prime. In 1903 [4], Frank Nelson Cole disproved with claim by exhibiting the two prime factors of $2^{67} - 1$: 193707721 and 761838257287. An account of this demonstration is given by N. T. Gridgeman [5]:

> At a mathematical meeting in New York in 1903, F. N. Cole walked on to the platform and, without saying a single word, wrote two large numbers on the blackboard. He multiplied them out in longhand, and equated the result to $2^{67} - 1$. (Subsequently, in private, Cole said that those **few minutes** at the blackboard had cost him three years of Sundays.)

Wikipedia [1], whose accuracy on historical figures has been found lacking [6], gives the following account of the same event, citing N. T. Gridgeman's account above as its only relevant source:

> On October 31, 1903, Cole famously made a presentation to a meeting of the American Mathematical Society where he identified the factors of the Mersenne number $2^{67} - 1$, or M_{67}. . . . During Cole's so called "lecture", he approached the chalkboard and in complete silence proceeded to calculate the value of M_{67}, with the result being $147,573,952,589,676,412,927$. Cole then moved to the other side of the board and wrote $193,707,721 \times 761,838,257,287$, and worked through the tedious calculations by hand. Upon completing the multiplication and demonstrating that the result equaled M67, Cole returned to his seat, not having uttered a word during the **hour-long** presentation. His audience greeted the presentation with a standing ovation. Cole later admitted that finding the factors had taken "three years of Sundays."

Note the key discrepancy: a “few minutes" at the blackboard became an “hour-long" lecture. Given that both accounts agree that the silent demonstration received a standing ovation, the former is far more plausible.

Another modern account [2] also describes the demonstration as taking “nearly an hour", while citing a source [3] that makes no such claim.

To resolve this discrepancy in the literature, we turn to simulation. We investigate the time complexity of the multiplication $193707721 \times 761838257287$ under various algorithms. We use the second author as computing hardware, which is of approximately the same computational performance as that used in the original demonstration.

We investigate four multiplication algorithms, each performed by hand on paper in base 10:

- Lattice multiplication
- Double-halve multiplication
- Quarter-Square multiplication
- Karatsuba multiplication

In addition to recording the runtime of each algorithm, we also investigate the number of digits written during the computation, which we observe to be highly correlated with runtime. We also comment on the fault-tolerance of each algorithm, which we find to be a major source of variance in runtime.

We find that the fastest algorithm, lattice multiplication, consistently takes 10 to 12 minutes. We would be far more likely to applaud 10 minutes of silent multiplication than an hour thereof, so we judge the older accounts [3, 5] to be the true ones.

2 Lattice Multiplication

In the lattice multiplication algorithm, the two multiplicands are written at the top and right of a grid. The product of each pair of digits is written in the cell at the intersection of the row and column, with the high digit to the upper left and the low digit to the lower right. Diagonals are summed, then those sums are summed to give the final result.

Figure 1 shows the computation. Note that partway through the computation, the computing hardware stopped writing zeros, because it couldn’t be bothered.

This method involved writing down 249 digits, which took 11 minutes, resulting in a computation frequency of 0.38 Hz. For multiplication of an m digit number by an n digit number, approximately

$$2mn + 4(m + n)$$

digits need to be written down.

On fault-tolerance, this algorithm performs well: the computing hardware only made 3 errors, each of which affected one or two output bits. The computing hardware found each error found relatively easily, only costing a minute or two in total.

In the end, this algorithm took 11 minutes to compute, which was a bit boring but not too bad. We could give a standing ovation to this.

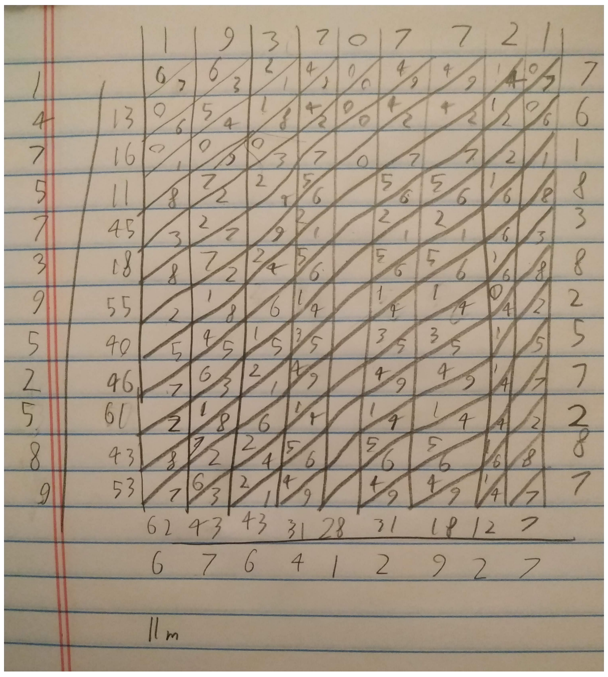


Figure 1: Lattice multiplication computation

2.1 Long Multiplication

Long multiplication is just a worse version of lattice multiplication. It’s like lattice multiplication, but with lots of extra additions interspersed among the multiplications. It’s called "long" for a reason. It’s really a shame that long multiplication is the standard method taught to most people. This is nothing short of a major failing of the education system.

3 Double-Halve Multiplication

In double-halve multiplication, the smaller multiplicand is repeatedly halved while the larger is repeatedly doubled. Each pair where the halving of the smaller multiplicand produces an even number is crossed out. The remaining doublings of the larger multiplicand are summed, giving the product.

This algorithm essentially consists of a binary decomposition of the smaller multiplicand, which is used to perform a shift-and-add multiplication on the larger multiplicand.

Figure 2 shows the computation. Note that only the first digit and the last two digits of the output are correct, because the computing hardware made a mistake partway through, and definitely isn't being paid enough to do this again.

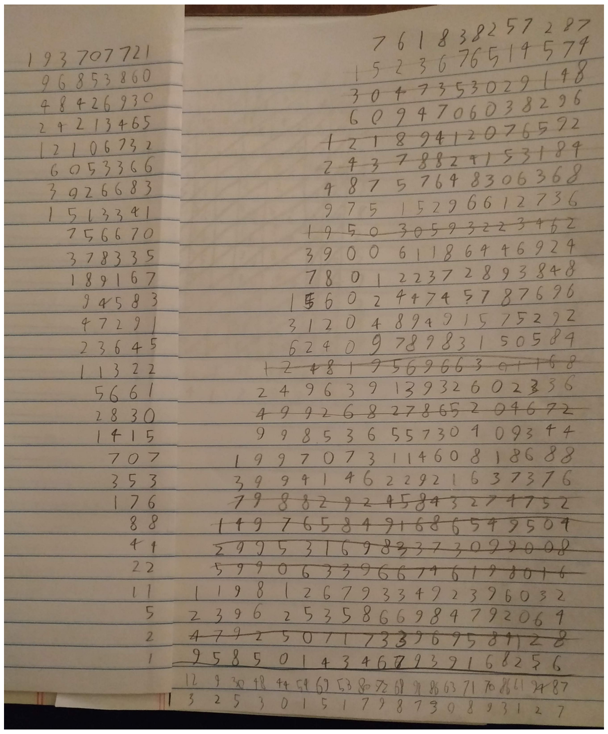
This method required writing down 651 numbers, which took 21 minutes, resulting in a computation frequency of 0.52 Hz. For a multiplication of a m digit number by an n digit number, where $m \leq n$, approximately

$$(4 + m \log_2 10)(m + n)$$

digits need to be written down.

Based on the number of digits written as a proxy for runtime, we should expect double-halve multiplication to have better performance than lattice multiplication when $m \leq 0.423n$. Or it would if it wasn't for all of the errors.

On fault-tolerance, this algorithm is terrible. A single error partway through is almost impossible to find and rectify, as demonstrated in Figure 2, beats us where.



193707721	761838257287
96853860	~~1523676514574~~
48426930	~~3047353029148~~
24213465	6094706038296
12106732	~~12189412076592~~
6053366	~~24378824153184~~
3026683	48757648306368
1513341	97515296612736
756670	~~195030593223462~~
378335	390061186446924
189167	780122372893848
94583	1560244745787696
47291	3120489491575292
23645	6240978983150584
11322	~~12481956966301168~~
5661	24963913932602336
2830	~~49926827865204672~~
1415	99853655730409344
707	199707311460818688
353	399414622921637376
176	~~798829245843274752~~
88	~~1497658491686549504~~
44	~~2995316983373099008~~
22	~~5990633966746198016~~
11	119812679334792396032
5	23962535866984792064
2	~~47925071733969587128~~
1	95850143467939168256
	12 9 30 48 44 54 69 53 80 72 68 98 86 63 71 70 86 61 94 87
	1 3 2 5 3 0 1 5 1 7 9 8 7 3 0 8 9 3 1 2 7

Figure 2: Double-halve multiplication computation

In the end, this algorithm took 21 minutes to utterly fail to verify the factorization. Definitely no standing ovation.

4 Quarter-Square Multiplication

Quarter-square multiplication is based on the following identity:

$$xy = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4}$$

As a result, given a pre-computed table of $\frac{n^2}{4}$, one can efficiently perform multiplications. Of course, this is less "computation by writing on a blackboard" and more "computation by looking in a book", but the comparison is still interesting.

Given a quarter-square book with entries up to at least $193,707,721 + 761,838,257,287$, the demonstration could be performed with two lookups and a couple of additions and subtractions. However, that quarter-square book would have approximately 10^{12} words. At no more than 1000 words to the page and 10000 pages to the meter (see Figure 3), the book would stretch approximately 100 kilometers. As a result, the multiplication would have taken at least a couple of hours, given transportation methods available in 1903.

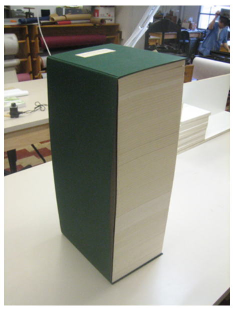


Figure 3: A 10,119 page book [10]. We're going to need a quarter-square table 100,000 times longer.

The largest quarter-square book ever published had entries up to $200,000$ [7], and was in print in 1903. Using this book to multiply the two numbers of interest would require breaking the multiplication into at least 6 sub-multiplications, requiring 12 lookups. Assuming a lookup takes in the range of 30 seconds to a minute, this method is competitive with the other multiplication algorithms in this paper.

With all that said, if someone silently looked up 12 numbers in a book and added them up, we would not give a standing ovation. We want to see the chalk fly!

5 Karatsuba Multiplication

Karatsuba multiplication was invented in 1960 [8], well after Prof. Cole's demonstration. Nonetheless, we may investigate whether it could have been used to improve the performance of the performance.

Karatsuba multiplication makes use of the following insight:

Let $x = x_1 b + x_0$ and let $y = y_1 b + y_0$. Then

$$xy = x_1 y_1 b^2 + (x_1 y_0 + y_1 x_0) b + x_0 y_0.$$

However, we can compute the middle term using the outer two terms and one additional multiplication:

$$x_1 y_0 + y_1 x_0 = x_1 y_1 + x_0 y_0 - (x_1 - x_0)(y_1 - y_0).$$

For the recursive calls, Karatsuba multiplication may be applied again, or another multiplication method may be used for the smaller multiplications.

Figure 4 shows the computation. Karatsuba multiplication was used with $b = 10^6$, and lattice multiplication was used for the recursive calls. Note that the computation hardware had a lot more fun calculating this way, it should do this more often.

This method required writing down 460 digits, which took 17 minutes, resulting in a computation speed of 0.45 Hz. For a multiplication of two n digit numbers, approximately

$$17n + \frac{3}{2}n^2$$

digits need to be written down.

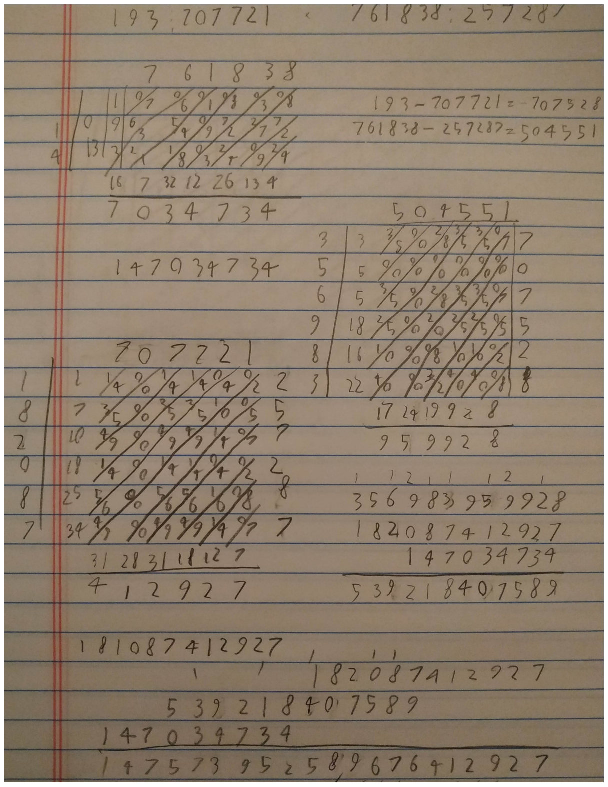


Figure 4: Karatsuba multiplication computation

Based on digits written as a proxy for runtime, we should expect Karatsuba multiplication to be an improvement over lattice multiplication when $n > 18$.

On fault tolerance, mistakes were reasonably locally recoverable, thanks to the use of lattice multiplication for the recursive calls. That being said, the use of many different types of operations did increase the error rate.

This algorithm took 17 minutes, but we got to do some fun and interesting computations along the way. Ovation awarded, but we'd probably stay seated.

6 Conclusion

Prof. Cole probably took about 10 minutes to show that

$$193,707,721 \times 761,838,257,287 = 147,573,952,589,676,412,927.$$

We don't know where you got an hour from, Wikipedia, but you're (probably) wrong. Also, he either used or should have used lattice multiplication to do it.

Other multiplication methods have their merits: Quarter-square multiplication might be faster, given a large book's worth of precomputation, while Karatsuba multiplication is a lot of fun. Double-halve multiplication doesn't have merits. Please don't make us do it again.

References

[1] Anonymous. Frank nelson cole. `https://en.wikipedia.org/wiki/Frank_Nelson_Cole`. Accessed: 2019-03-13.

[2] Anonymous. Lecture sans paroles: the factors of m67. `https://thatsmaths.com/2016/06/30/lecture-sans-paroles-the-factors-of-m67/`. Accessed: 2019-03-13.

[3] Eric T. Bell. *Mathematics, Queen and Servant of Science.* 1952.

[4] Frank N Cole. On the factoring of large numbers. *Bulletin of the American Mathematical Society*, 10(3):134–137, 1903.

[5] N. T. Gridgeman. The search for perfect numbers. *New Scientist*, 18(334):86–88, 1963.

[6] Lucy Holman Rector. Comparison of wikipedia and other encyclopedias for accuracy, breadth, and depth in historical articles. *Reference services review*, 36(1):7–22, 2008.

[7] Neville Holmes. Multiplying with quarter squares. *The Mathematical Gazette*, 87(509):296–299, 2003.

[8] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.

[9] Marin Mersenne. *Cogitata Physica-Mathematica*. 1644.

[10] Bill Voss. 10,000 page book bound at conservation lab. `https://blog.lib.uiowa.edu/preservation/2011/01/07/10000-page-book-bound-at-conservation-lab/`. Accessed: 2019-03-13.