PERFORMANCE OF CLASSICAL FACTORIZATION ALGORITHMS FOR STRUCTURED INTEGERS

ISABELLA LI

Abstract. Integer factorization is the foundation of some public-key cryptography, notably the Rivest-Shamir-Adleman (RSA) cryptosystem. In this paper, we will explore several factorization methods and two special families of integers, Mersenne numbers (2^p-1) and Fermat numbers ($2^{2^n}+1$), and discuss when each method is an efficient choice. We contrast trial division, Fermat's method, Pollard's $p-1$, and the Quadratic Sieve (QS). The measurements show clear patterns: trial division and Fermat's method are strong only when a small factor or a factor near $\sqrt{n}$ exists, respectively; Pollard's $p-1$ excels when a prime factor has a smooth $p-1$; and QS becomes increasingly desirable as n grows. We also include a short proof of correctness for QS in the "congruence of squares" step and charts indicating runtime of the factorization methods.

1. Introduction

From Fermat's difference of squares, invented in 1643, to Pollard's $p-1$ in 1974, progress in integer factorization accelerated. Continued fraction factorization [1] and Dixon's random squares method [2] led to the Quadratic Sieve [3] and, ultimately, the Number Field Sieve [4]. Lenstra's elliptic curve method [5] excels on large smooth cofactors, in other words, factors whose prime divisors are all below a chosen bound.

Early ideas for primality testing can be traced to Fermat's Little Theorem and Lucas's 19th-century criteria [6, 7]. These foundations led to fast randomized tests, in particular the Solovay–Strassen [8] and Miller–Rabin [9, 10] tests, and, eventually, the first unconditional polynomial-time algorithm, AKS [11]. Together with Diffie–Hellman

Date: November 7, 2025.
Key words and phrases. Integer factorization, Mersenne numbers, Fermat numbers, Pollard's $p-1$, Quadratic Sieve.

key exchange and Rivest-Shamir-Adleman (RSA) public-key encryption [12, 13], these results underpin much of modern cryptography [14, 15].

Modern protocols like RSA rely on the computational difficulty of factoring large composite integers to keep encrypted messages secure. While primality testing for Mersenne and Fermat numbers has specialized tests, for example, the Lucas–Lehmer test for Mersenne numbers and the Pépin test for Fermat numbers, factoring their nontrivial factors still benefits from general-purpose methods. A central goal of this paper is to highlight how the performance of these methods changes as the number of digits of the inputs grows, and to compare our results to discuss which algorithms remain practical at larger scales. We begin by summarizing the four methods that will be examined in this paper:

(1) **Trial division** (promising for numbers with small factors)
(2) **Fermat's factorization** (effective when factors are close to $\sqrt{n}$).
(3) **Pollard's** $p-1$ (efficient if some prime factor p has a smooth $p-1$).
(4) **Quadratic Sieve** (QS) (desirable for "medium-large" inputs)

For Mersenne numbers, if $q \mid 2^p - 1$ with p prime, then $q \equiv 1 \pmod{2p}$. For Fermat numbers, if $q \mid 2^{2^n} + 1$, then $q \equiv 1 \pmod{2^{n+1}}$. If these hold true, both conditions can be easily verified. Congruence conditions like this can help trial division and sieving by shrinking the set of admissible small primes.

We will implement these factorization methods with Python. Python is a convenient tool but introduces performance limits. For example, the `sqrt` function keeps sixteen digits, so it only works for numbers under such size. Since we use `time.process_time` as our timer, its granularity can blur small runtime differences.

For more than forty digits, the Quadratic Sieve becomes less efficient in our implementation due to the large relation matrix and memory-heavy linear algebra [3, 14, 16]. Elliptic curve methods are well suited to finding large prime factors, while the General Number Field Sieve is

preferred for general large composites [4, 16] and the Special Number Field Sieve for numbers of special form such as $a^n \pm b$ [17].

2. Methods

In this section, we describe the classical factorization methods that form the basis of our comparison. For each algorithm, we outline the main idea behind the method and give a short proof or justification of its correctness. These four algorithms represent a natural spectrum, from very elementary approaches to more advanced sieve-based techniques.

Trial Division

Test divisibility of n for prime numbers starting from 2, and return a divisor and iterate on the cofactor until the cofactor is prime.

Fermat's method

We begin with a classical observation.

Theorem 2.1. *Let n be odd and composite. Then, there exist integers $x \geq \lceil \sqrt{n} \rceil$ and $y \geq 0$ with $n = x^2 - y^2$, hence $n = (x-y)(x+y)$. To find such a pair, start from $x_0 = \lceil \sqrt{n} \rceil$ and increase x by 1 until $x^2 - n$ is a square. This finds (x, y) and thus a factorization of n.*

Proof. Write $n = ab$ with $a \geq b \geq 3$ both odd (since n is odd). Then

$$x := \frac{a+b}{2}, \qquad y := \frac{a-b}{2}$$

are integers with $x \geq \sqrt{ab} = \sqrt{n}$ and $n = (x-y)(x+y) = x^2 - y^2$. The algorithm examines $x = x_0, x_0 + 1, \ldots$ and halts at this x, where $x^2 - n = y^2$ is a perfect square. Returning $(x - y, x + y)$ yields a nontrivial factorization. □

Corollary 2.2. *If $n = ab$ with $a \geq b$ and $\delta := a - b$, then Fermat's search halts within at most $\delta/2$ increments of x; in particular it is fast when a and b are close.*

Pollard's $p-1$

We define concepts that are central to this method.

Definition 2.3 (B-smoothness)**.** *For $m \in \mathbb{N}$, m is B-smooth if all prime factors of m are $\leq B$.*

Theorem 2.4 (Pollard's $p-1$ criterion)**.** *Let $n > 1$ and suppose $p \mid n$ is a prime such that $p-1$ is B-smooth. Let $M = \operatorname{lcm}(1, 2, \ldots, B)$ and choose b with $\gcd(b, n) = 1$. Then, $b^M \equiv 1 \pmod{p}$. Consequently,*

$$d := \gcd(b^M - 1, n) \quad \textit{satisfies} \quad p \mid d.$$

If $b^M \not\equiv 1 \pmod{q}$ for at least one other prime $q \mid n$, then the following two statements are true: $1 < d < n$ and d is a nontrivial factor of n.

Proof. Because $p-1$ is B-smooth, we have $p-1 \mid M$. By Fermat's Little Theorem, $b^{p-1} \equiv 1 \pmod{p}$, hence $b^M \equiv 1 \pmod{p}$ and $p \mid (b^M - 1)$. Thus, $p \mid d$. If there is a prime $q \mid n$ with $b^M \not\equiv 1 \pmod{q}$, then $q \nmid (b^M - 1)$, so d is not divisible by q and cannot equal n; since $p \mid d$, we have $1 < d < n$. □

Corollary 2.5. *Pollard's $p-1$ succeeds (returns a nontrivial gcd) whenever n has a prime factor p with $p-1$ B-smooth and the chosen b makes $b^M \not\equiv 1 \pmod{q}$ for at least one other $q \mid n$. Increasing B increases the chance that some $p-1$ is B-smooth.*

Quadratic Sieve (QS)

Let $Q(t) = (\lfloor \sqrt{n} \rfloor + t)^2 - n$. We first introduce some concepts.

Definition 2.6 (Factor base, parity vector)**.** *Fix $B \in \mathbb{N}$. A* factor base *$\mathcal{P}$ is the set of primes $\leq B$ that divide some $Q(t)$. For $m \in \mathbb{Z}$, its* parity vector *is*

$$v(m) := (e_p \bmod 2)_{p \in \mathcal{P}} \in \mathbb{F}_2^{\mathcal{P}},$$

where e_p is the exponent of the prime p in the prime factorization of $|m|$.

Choose a factor base of small primes, collect values of $Q(t)$ that are B-smooth, and form the exponent matrix mod 2. Select a few rows, and for each column, sum up the terms in the selected rows in the matrix. If each of these sums is $0 \pmod 2$, we find a product $\prod Q(t_i)$ that is a perfect square. From this, we are able to derive $X^2 \equiv Y^2 \pmod{n}$ for some X, Y and obtain a factor through $\gcd(X - Y, n)$.

Theorem 2.7. *If $Q(t_i) = (r + t_i)^2 - n$ and the exponents of each prime in $\{Q(t_i)\}$ sum to an even number, then $\prod_i Q(t_i) = Y^2$ for some $Y \in \mathbb{N}$. Writing*

$$X := \prod_{i=1}^{k} (r + t_i), \quad Y := \sqrt{\prod_{i=1}^{k} Q(t_i)}.$$

we have $X^2 \equiv Y^2 \pmod{n}$. If $X \not\equiv \pm Y \pmod{n}$, which we hope is true, then $\gcd(X - Y, n)$ is a nontrivial factor of n.

Proof. Let $r = \lfloor \sqrt{n} \rfloor$ and $Q(t) = (r + t)^2 - n$. Suppose $t_1, \ldots, t_k$ are such that $\prod_{i=1}^{k} Q(t_i)$ is a perfect square. Set

$$X := \prod_{i=1}^{k} (r + t_i), \qquad Y := \sqrt{\prod_{i=1}^{k} Q(t_i)} \in \mathbb{N}.$$

Since $Q(t_i) \equiv (r + t_i)^2 \pmod{n}$, multiplying gives

$$\prod_{i=1}^{k} Q(t_i) \equiv \prod_{i=1}^{k} (r + t_i)^2 = X^2 \pmod{n},$$

hence $X^2 \equiv Y^2 \pmod{n}$. If $X \not\equiv \pm Y \pmod{n}$, then $d := \gcd(X - Y, n)$ satisfies $1 < d < n$ and is a nontrivial factor of n.

To find the square product, let $\mathcal{P}$ be the factor base and, for each $\mathcal{P}$-smooth $Q(t)$, let $v(t) \in \mathbb{F}_2^{\mathcal{P}}$ be the exponents of primes in $\mathcal{P}$ modulo 2. Form the matrix A whose rows are the vectors $v(t)$. Once the number of collected relations exceeds $|\mathcal{P}|$, linear algebra over $\mathbb{F}_2$ produces a nonzero selector vector $\mathbf{e}$ with $\mathbf{e}^\top A = 0$. Summing the corresponding rows gives $\sum_{t:\, \mathbf{e}_t = 1} v(t) \equiv 0$, so $\prod_{t:\, \mathbf{e}_t = 1} Q(t)$ is a perfect square. If the congruence satisfies $X \equiv \pm Y \pmod{n}$ one could choose a different factor base. □

3. Code setup

First, we run each method on two families: the Mersenne dataset, which are values of $2^p - 1$ for assorted p; the Fermat dataset, which are values of $2^{2^n} + 1$.

Then, we run the methods for random 20- and 30-digit numbers. 35-digit numbers take a lot of time for the trial division and require too

much memory for the Quadratic Sieve. In particular, when factoring 40-digit numbers, the console terminated the program with `zsh: killed` due to excessive memory usage.

For each input, we recorded its runtime in milliseconds (ms) of CPU time, using an Apple M3 chip. All data are rounded by three significant digits after the decimal place, and the code used can be found in this repository.

In some datasets, the sample sizes are relatively small because their respective methods terminate quickly or fail to produce usable outputs within time or memory constraints. For example, Fermat's method takes days or even months to run on Python when factors are not near $\sqrt{n}$.

4. Results

We first present the results from the Mersenne and Fermat datasets. Fermat numbers grow superexponentially, so only the first few cases run within reasonable time on a standard CPU.

TABLE 1. Timing summary for Mersenne numbers (ms).

Statistic	Trial Division	Fermat's method	Pollard's $p-1$	Quadratic Sieve
Median	0.171	0.009	0.137	29.91807461
Min	0.000	0.004	0.001	0.001192092896
Max	102 767.441	825.599	3478.245	221130.908012
Q1 (25%)	0.005	0.005	0.028	0.483751
Q3 (75%)	4.244	0.030	0.558	1507.411122
Sample size	95	74	101	106

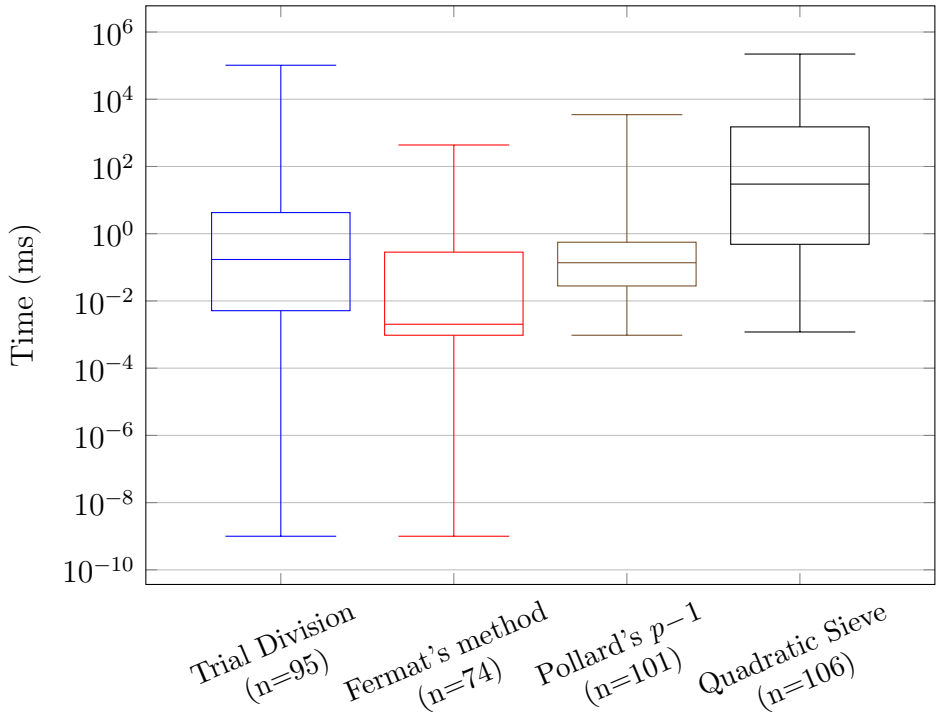

FIGURE 1. Box plots of runtime per method with sample sizes (log scale).

In the Mersenne dataset, Fermat's factorization method is generally inefficient, especially when the prime factors of the number we are factoring are far apart. Its success rate drops significantly for larger, unbalanced numbers, at 69.8% compared to other methods that are all above 89% for the Mersenne numbers $2^i - 1$, $2 \leq i \leq 107$. However, when factors are close, Fermat's method can be remarkably efficient.

TABLE 2. Success rates of factorization methods (sample size = 106).

Method	Success Rate (%)
Trial Division	89.6%
Fermat's method	69.8%
Pollard's $p-1$	95.3%
Quadratic Sieve	100.0%

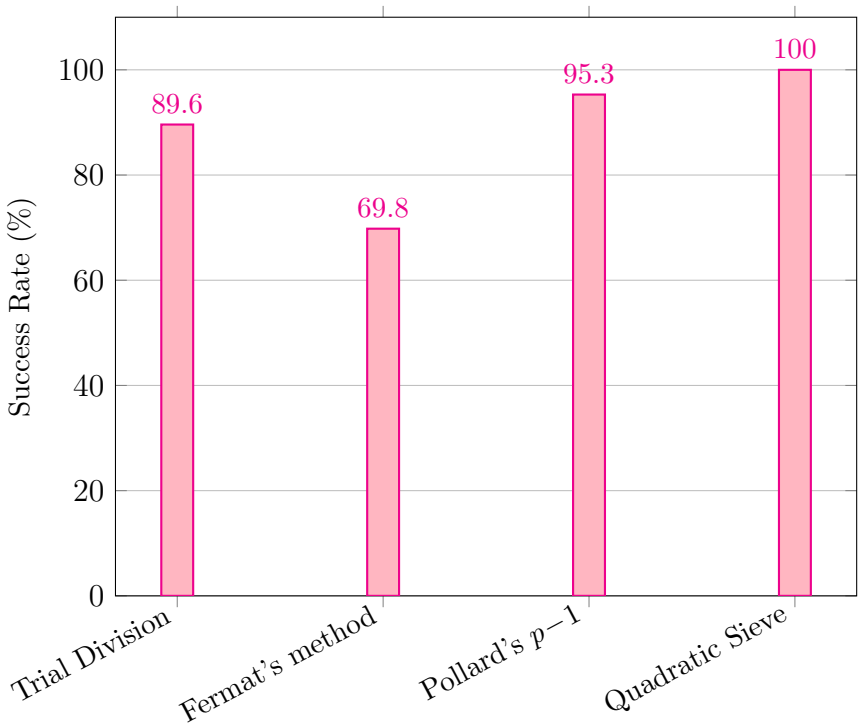

FIGURE 2. Success rates of factorization methods (sample size = 106).

TABLE 3. Timing summary for Fermat numbers (millisecond).

Statistic	Trial Division	Fermat's method	Pollard's $p-1$	Quadratic Sieve
Median	0.052	1.459	0.041	0.302
Min	0.002	0.005	0.001	0.031
Max	147.250	394.646	247 796.388	214.362
Q1 (25%)	0.005	0.010	0.014	0.280
Q3 (75%)	0.063	100.841	1.082	0.886
Sample size	5	4	6	5

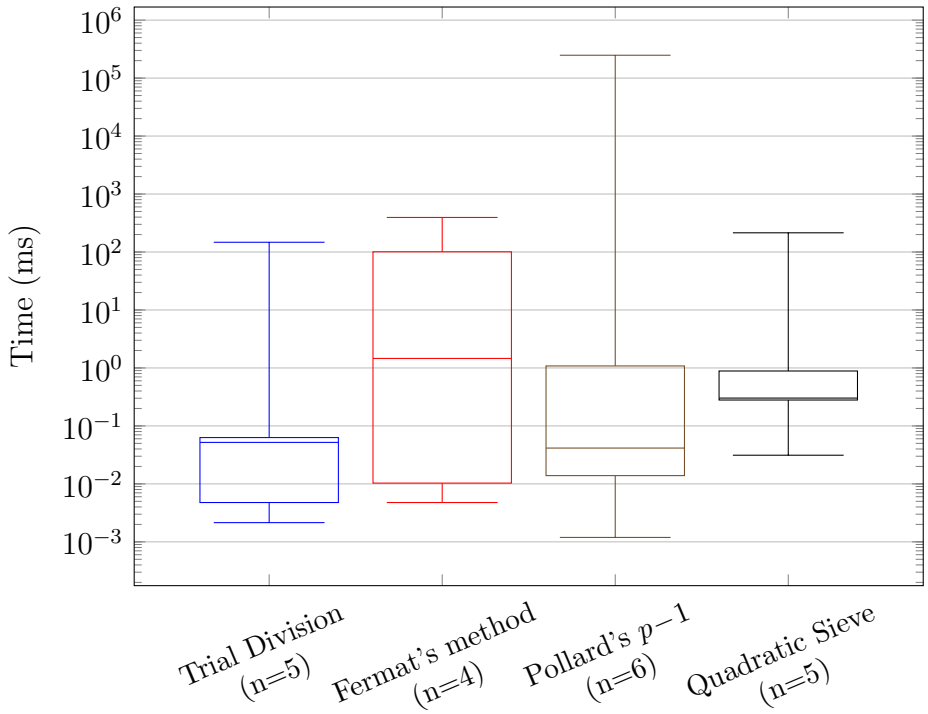

FIGURE 3. Box plots of runtime per method with sample sizes (log scale).

Next, we want to see what impact the size of a number has on the effectiveness of factorization methods, so we compare how they perform

in samples of 20 and 30. It is worth noting that Pollard's $p-1$ performed very well, so much so that the first-quartile results of the runtime on a random sample are unusually close to 0, which are reflected as 0 in the CPU recorded by Python.

TABLE 4. Timing summary (ms).

Statistic	Trial Division	Pollard's $p-1$	Quadratic Sieve
Median	109.510	0.001	301.880
Min	0.066	0.000	0.098
Max	392 481.974	2815.251	559.046
Q1 (25%)	15.080	0.000	256.980
Q3 (75%)	1725.358	0.005	347.848
Sample size	500	500	500

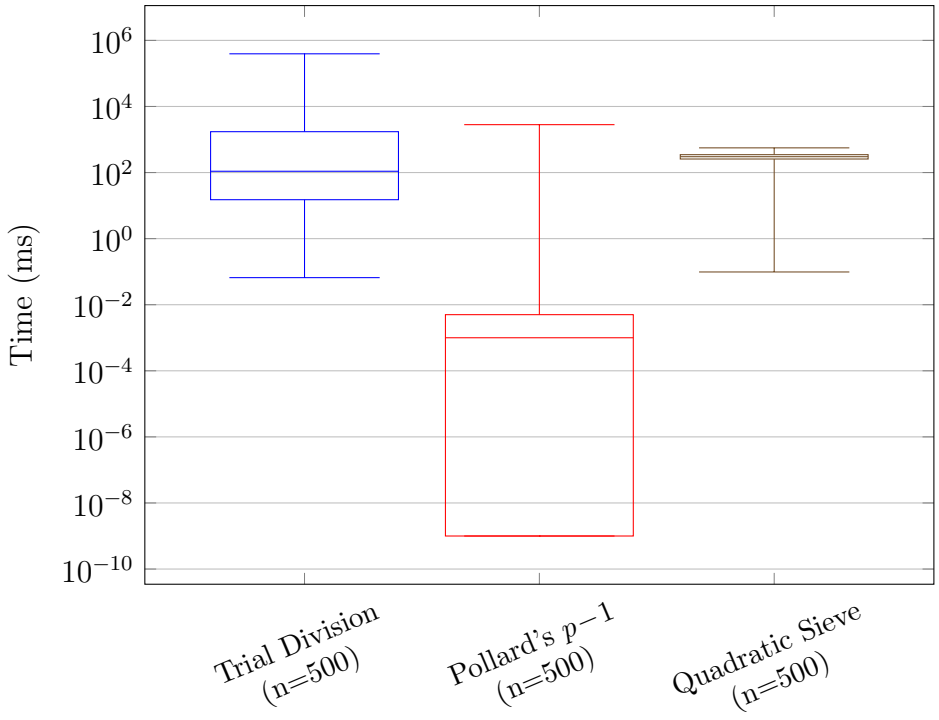

FIGURE 4. Box plots of runtime per method with sample sizes (log scale).

TABLE 5. Timing summary (ms).

Statistic	Pollard's $p-1$	Quadratic Sieve
Median	0.001	26 159.587
Min	0.000	0.156
Max	4294.464	38 616.475
Q1 (25%)	0.000	22 629.178
Q3 (75%)	0.008	29 775.944
Sample size	500	500

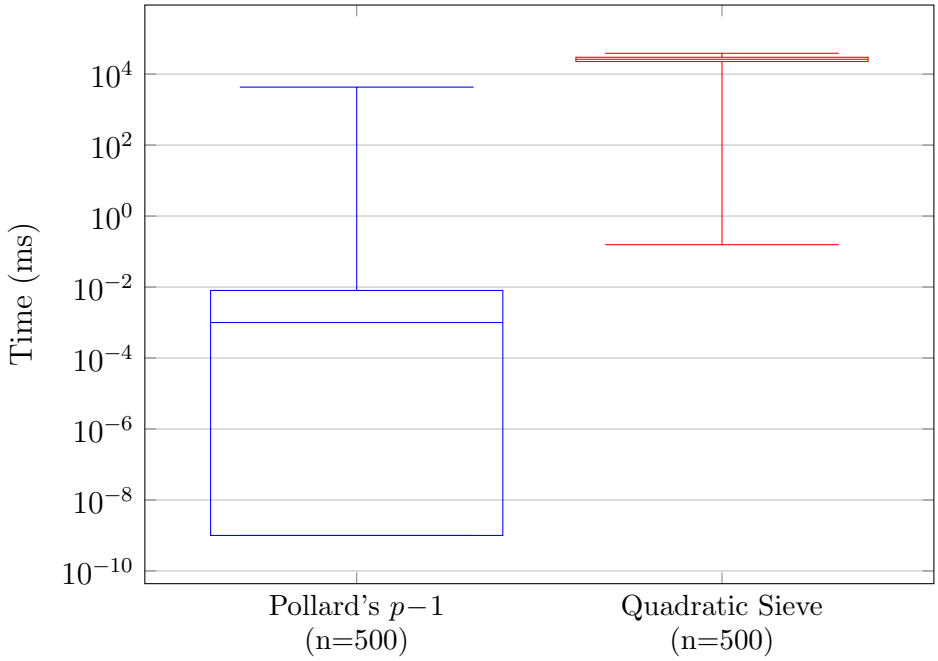

FIGURE 5. Box plots of runtime per method with sample sizes (log scale).

In summary, in the Mersenne dataset, the median runtimes are in the order of Fermat 0.0020, Pollard p–1 0.1369, Trial 0.1709, the Quadratic Sieve 29.9181. The Interquartile Ranges (IQRs) are tight for Fermat and broader for QS, consistent with the way that some cases can involve heavy linear algebra that slows down the process.

In the Fermat dataset, Pollard p–1 leads with median 0.0415, followed by QS 0.3021, Trial 0.0520, and Fermat 1.4591. The right tails are long for QS and Fermat's method.

On random 20–30 digit samples, the table shows Trial 109.51 and QS 301.88 vs. Pollard $p-1 \approx 0.001$. Pollard's $p-1$ performs especially well.

Although these experiments give meaningful comparisons, the feasibility of running them on the device used in the writing of this paper is limited. Python's limitations and the hardware used (a standard laptop CPU) cannot match today's best supercomputers, restricting our achievable sample sizes as input size grows.

5. Discussion

5.1. **Patterns**

For Mersenne inputs, Fermat's method is typically best in median time, with Pollard $p-1$ close behind, and QS is much slower from this perspective.

For Fermat inputs, Pollard $p-1$ is significantly faster than QS and trial division, while Fermat's method is comparatively slow.

For random composites of 20–30 digits, Pollard $p-1$ dominates at 20–30 digits, while trial division and QS are significantly slower.

Fermat's factorization is optimal for near-square integers, smooth $p-1$ cases (in other words, where $p-1$ can be expressed by a product of small prime numbers) favor Pollard, and, very large inputs that lack this property eventually require QS or the Number Field Sieve.

After comparing many factorization algorithms, it is important to keep in mind that not all tests are factorization algorithms. In certain cases, such as determining whether one could factor a number, one would use primality tests such as Fermat, Euler, Solovay–Strassen, Miller–Rabin, Lucas–Lehmer (for M_p), and Pépin (for F_n).

5.2. **More observations**

Runtimes are heavy-tailed, so medians and IQRs describe performance better than means.

In some cases, a runtime of 0 reflects a coarse timer resolution, not truly instantaneous runs. This makes the lower whiskers close to or even equal to 0.

6. Conclusions & future work

Takeaways

To find nontrivial factors of Mersenne numbers, a practical order of methods is: Fermat $\rightarrow$ Pollard $p-1$ $\rightarrow$ Quadratic Sieve. For Fermat numbers, a practical order is: Pollard p–1 $\rightarrow$ QS $\rightarrow$ Trial Division. For random 20–30 digit composites, start with Pollard $p-1$ and use QS only after easy smooth cases are exhausted.

As numbers increase, the Quadratic Sieve (and for even larger, the General Number Field Sieve) becomes essential. A natural next step is to (a) tune QS parameters, such as the size of the smooth factor base, and (b) add other factorization methods such as variants of Pollard's $p-1$ for comparison.

Future Work

It would be useful to further determine the ranges of n where methods like Pollard's p–1, the Elliptic Curve Method, the Quadratic Sieve, and the Number Field Sieve become most effective, and to mark their transition points.

Moreover, it would be valuable to investigate parameter tuning for each method, such as the smoothness bound B for Pollard's p–1, the size of the factor base in the Quadratic Sieve, or parameters in the Elliptic Curve Method, and study how these choices affect the performance of factorization methods.

We can also extend the experiments to larger datasets and alternative programming languages such as C to reduce limitations.

7. Acknowledgements

The author would like to thank Prof. Sandy Zabell for his lectures and guidance, inspiring me to write a paper on factorization methods, and for his invaluable suggestions and support throughout writing this paper.

References

[1] J. Brillhart and M. A. Morrison, "A method of factoring and the factorization of F_7," *Mathematics of Computation*, vol. 29, pp. 183–205, 1975. doi: 10.1090/S0025-5718-1981-0595059-1.

[2] J. D. Dixon, "Asymptotically fast factorization of integers," *Mathematics of Computation*, vol. 36, no. 153, pp. 255–260, 1981. doi: 10.1090/S0025-5718-1981-0595059-1.

[3] C. Pomerance, "The quadratic sieve factoring algorithm," in *Advances in Cryptology EUROCRYPT '84*, vol. 209 of *Lecture Notes in Computer Science*, pp. 169–182, Springer, 1985. doi: 10.1007/3-540-39757-4_17.

[4] J. P. Buhler, J. Hendrik W. Lenstra, and C. Pomerance, "Factoring integers with the number field sieve," in *The Development of the Number Field Sieve*, vol. 1554 of *Lecture Notes in Mathematics*, pp. 50–94, Springer, 2006. doi: 10.1007/BFb0091539.

[5] J. Hendrik W. Lenstra, "Factoring integers with elliptic curves," *Annals of Mathematics*, vol. 126, no. 3, pp. 649–673, 1987. doi: 10.2307/1971363.

[6] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Oxford University Press, 6 ed., 2008. doi: 10.1080/00107510903184414.

[7] Édouard Lucas, *Théorie des Nombres*. Paris: Gauthier-Villars, 1891.

[8] R. M. Solovay and V. Strassen, "A fast monte-carlo test for primality," *SIAM Journal on Computing*, vol. 6, no. 1, pp. 84–85, 1977. doi: 10.1137/0206006.

[9] G. L. Miller, "Riemann's hypothesis and tests for primality," *Journal of Computer and System Sciences*, vol. 13, no. 3, pp. 300–317, 1976. doi: 10.1016/S0022-0000(76)80043-8.

[10] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory*, vol. 12, no. 1, pp. 128–138, 1980. doi: 10.1016/0022-314X(80)90084-0.

[11] M. Agrawal, N. Kayal, and N. Saxena, "PRIMES is in P," *Annals of Mathematics*, vol. 160, no. 2, pp. 781–793, 2004. doi: 10.4007/annals.2004.160.781.

[12] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. doi: 10.1109/TIT.1976.1055638.

[13] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978. doi: 10.1145/359340.359342.

[14] C. Pomerance, "A tale of two sieves," *Notices of the American Mathematical Society*, vol. 43, no. 12, pp. 1473–1485, 1996.

[15] J. S. Kraft and L. C. Washington, *An Introduction to Number Theory with Cryptography*. CRC Press, 2018. doi: 10.1201/9781351664110.

[16] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*. Springer, 2 ed., 2005. doi: 10.1007/0-387-28979-8.

[17] A. K. Lenstra, H. W. L. Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pp. 564–572, ACM, 1990. doi: 10.1145/100216.100295.