Synchronization Phenomena in Classical Factorization Algorithms on Structured Integers

Isabella Li

Lexington High School, 251 Waltham St, Lexington, MA 02421, USA

isabellali434@gmail.com

August 2026

Preprint DOI: 10.5281/zenodo.21924106

Abstract

We compared the time it takes for five classical factorization methods, trial division, Fermat's method, Pollard's rho, Pollard's $p-1$, and the Quadratic Sieve, to factor Mersenne numbers, Fermat numbers, and random 20- and 30-digit composites. The places where these methods fall short were informative: on the Mersenne family, the two Pollard methods fail on disjoint sets, each for a provable structural reason. Rho fails on exactly the exponents divisible by 20, whose values are divisible by 25, a modulus whose two factors of 5 the iteration $x \mapsto x^2 + 1$ provably cannot pull apart, while $p-1$ fails on exactly three prime exponents, driven by the congruence $q \equiv 1 \pmod{2p}$. We prove both mechanisms and then test them predictively: they reproduce the observed failure set exactly, reproduce the measured fraction of failing bases exactly on two inputs, classify the blind spots of the maps $x^2 + c$, and correctly predict the outcomes on repunits. We argue that the observed failures are driven by synchronization, and the classical remedies, restarting rho with a new polynomial and retrying $p-1$ with a new base, are precisely desynchronization strategies. We further compare success rates and runtime distributions for all five methods and describe when each method is the right choice.

Keywords: integer factorization, Mersenne numbers, Fermat numbers, Pollard's rho, Pollard's $p-1$, Quadratic Sieve

1 Introduction

From Fermat's difference of squares, invented in 1643, to Pollard's $p-1$ and rho methods in 1974 and 1975 [1, 2], progress in integer factorization accelerated. Continued fraction factorization [3] and Dixon's random squares method [4] led to the Quadratic Sieve [5] and, ultimately, the Number Field Sieve [6]. Lenstra's elliptic curve method [7] excels at extracting comparatively small prime factors from large inputs.

Early ideas for primality testing can be traced to Fermat's Little Theorem and Lucas's 19th-century criteria [8, 9]. These foundations led to fast randomized tests, in particular the Solovay–Strassen [10] and Miller–Rabin [11, 12] tests, and, eventually, the first unconditional polynomial-time algorithm, AKS [13]. Together with Diffie–Hellman key exchange and Rivest–Shamir–Adleman (RSA) public-key encryption [14, 15], these results are the cornerstone of much of modern cryptography [16, 17].

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 five methods that will be examined in this paper:

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

For Mersenne numbers, if $q \mid 2^p-1$ with p an odd prime, then $q \equiv 1 \pmod{2p}$. For Fermat numbers, if $q \mid 2^{2^n}+1$, then $q \equiv 1 \pmod{2^{n+1}}$. Both congruences are cheap to test, which makes them useful filters for candidate divisors. Congruence conditions like this can help trial division and sieving by shrinking the set of admissible small primes. We will explain how this congruence is the mechanism behind the failure of $p-1$ on certain Mersenne inputs (Section 5).

We will implement these factorization methods with Python. Python is a convenient tool but introduces performance limits. For example, floating-point square roots carry only about

sixteen digits, so we use exact integer square roots (`math.isqrt`) in trial division and the Quadratic Sieve, and high-precision decimal square roots in Fermat's method. 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 [5, 16, 18]. Elliptic curve methods and the General Number Field Sieve are thus preferred for general large composites [6, 18] and the Special Number Field Sieve for numbers of special form such as $a^n \pm b$ [19].

2 Methods

In this section, we describe the classical factorization methods that we will compare. For each algorithm, we outline the main idea behind the method and give its justification. These five algorithms range from very elementary approaches to more advanced sieve-based techniques.

Trial Division

Test divisibility of n by successive primes starting from 2; each time a divisor is found, record it and continue with the cofactor until the cofactor is prime. Unlike the other four methods, our trial division runs to a complete factorization rather than stopping at the first split; we return to this asymmetry in Section 7.3.

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 the first x for which $x^2 - n$ is a perfect square; this happens no later than the x above. At the halting x, writing $y^2 = x^2 - n$, returning $(x - y, x + y)$ yields a factorization, and it is nontrivial: $x - y = 1$ would force $x = (n + 1)/2$, while the halting x satisfies $x \leq \frac{a+b}{2} < \frac{n+1}{2}$, since $(a - 1)(b - 1) > 0$. □

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.*

Proof. The number of increments is at most

$$\frac{a+b}{2} - \lceil \sqrt{n} \rceil \;\leq\; \frac{a+b}{2} - \sqrt{ab} \;=\; \frac{(\sqrt{a}-\sqrt{b})^2}{2} \;\leq\; \frac{(\sqrt{a}-\sqrt{b})(\sqrt{a}+\sqrt{b})}{2} \;=\; \frac{a-b}{2}. \quad \square$$

Pollard's rho

Pollard's rho method [2] iterates $f(x) = x^2 + 1 \pmod{n}$, keeping a slow copy x_k and a fast copy $y_k = x_{2k}$ (Floyd cycle detection), and computes $\gcd(|x_k - y_k|, n)$ at each step. The idea is probabilistic. If p is an unknown prime factor of n, the sequence read modulo p behaves like a random sequence in $\mathbb{Z}/p\mathbb{Z}$, so by the birthday paradox two iterates collide modulo p within $O(\sqrt{p})$ steps in expectation. A collision modulo p that is not a collision modulo n makes the gcd a nontrivial factor. The trajectory looks like a tail leading into a cycle, the shape of the letter ρ, which is where the name comes from.

Remark 2.3. The $O(\sqrt{p})$ bound is heuristic: it assumes $x \mapsto x^2 + 1$ behaves like a random map modulo p, which is unproven; the strongest rigorous results, due to Bach [20], average over both the seed and the additive constant and therefore say nothing about the fixed map $x^2 + 1$. Our data contains a concrete failure of the analogous randomness assumption at the prime-power modulus 25 (Proposition 5.1); the heuristic for prime moduli is untouched.

Our implementation performs up to 8 random restarts of 200,000 iterations each, which reliably finds prime factors of up to roughly ten digits; a full failure costs about 1.6 million iterations. A recursive wrapper applies a Miller–Rabin test [11, 12] and factors cofactors completely, retrying failed rho calls without bound; the failure protocol for runs that cannot make progress is described in Section 3.

Pollard's $p-1$

We define concepts that are central to this method, introduced by Pollard [1].

Definition 2.4 (B-smoothness, B-powersmoothness)**.** For $m \in \mathbb{N}$, m is *B-smooth* if all prime factors of m are $\leq B$, and *B-powersmooth* if every prime power dividing m is at most B. Powersmoothness is strictly stronger: $1024 = 2^{10}$ is 2-smooth, but it is not 100-powersmooth, and indeed 1024 does not divide $\operatorname{lcm}(1, \ldots, 100)$.

Theorem 2.5 (Pollard's $p-1$ criterion)**.** *Let $n > 1$ and suppose $p \mid n$ is a prime such that $p-1$ is B-powersmooth. 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. For each prime q, the q-part of $M = \operatorname{lcm}(1, \ldots, B)$ is the largest power of q not exceeding B. Since $p-1$ is B-powersmooth, every prime power dividing $p-1$ is at most B and therefore divides M; as these prime powers are pairwise coprime, $p-1 \mid M$. 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 2.6. *Pollard's* $p-1$ *succeeds (returns a nontrivial gcd) whenever* n *has a prime factor* p *with* $p-1$ *B-powersmooth and the chosen* b *makes* $b^M \not\equiv 1 \pmod{q}$ *for at least one other prime* $q \mid n$*. Increasing* B *increases the chance that some* $p-1$ *is* B*-powersmooth; the exponent* M *has about* $1.44B$ *bits, so a full pass costs roughly* $2B$ *modular multiplications (one squaring per bit plus one multiplication per set bit).*

Our implementation uses $B = 10^5$ and bases $2, 3, 5$, builds the exponent from all prime powers up to B, checks the gcd every 200 primes so that easy factors exit early, and on an overshoot (gcd equal to n) backtracks from the last checkpoint one multiplication at a time. Backtracking separates factors unless every factor surfaces at the same single multiplication; Section 5 shows this synchronization genuinely occurs on Mersenne inputs.

Quadratic Sieve (QS)

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

Definition 2.7 (Factor base, parity vector)**.** Fix $B \in \mathbb{N}$. A *factor base* $\mathcal{P}$ is the set of primes $\leq B$ that divide some $Q(t)$ in the sieving range. 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 2.8. *Let $r = \lfloor \sqrt{n} \rfloor$ and let $t_1, \ldots, t_k \geq 1$, so that each $Q(t_i) = (r + t_i)^2 - n$ is positive. If 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 $X \not\equiv \pm Y \pmod{n}$, 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 $X^2 \equiv Y^2 \pmod{n}$. Now suppose $X \not\equiv \pm Y \pmod{n}$ and let $d := \gcd(X - Y, n)$. If $d = n$, then $X \equiv Y \pmod{n}$; if $d = 1$, then from $n \mid (X - Y)(X + Y)$ we get $n \mid X + Y$, that is, $X \equiv -Y \pmod{n}$. Both are excluded, so $1 < d < n$ and d is a nontrivial factor of n. □

To find the square product in practice, 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 resulting congruence has $X \equiv \pm Y \pmod{n}$, one selects a different dependency from the null space, or collects more relations; when n has at least two distinct prime factors, at most about half of the possible congruences fail in this way.

Our implementation chooses B adaptively from the input size, roughly $B \approx L(n)^{\sqrt{2}/4}$ with $L(n) = \exp(\sqrt{\ln n \ln \ln n})$ and a floor of 50 (about 100 at 20 digits and 400 at 30 digits), sieves a single interval of length about B^3 starting at $\lfloor \sqrt{n} \rfloor$ (only positive $Q(t)$ can produce relations, so the theorem's hypothesis is met), uses the Tonelli–Shanks algorithm to locate the residue classes each factor-base prime can divide, and performs dense Gaussian elimination over $\mathbb{F}_2$. The implementation proceeds once the number of smooth relations reaches $|\mathcal{P}|$;

the dependency guarantee above requires $|\mathcal{P}| + 1$, a harmless off-by-one that can cost an occasional failure.

3 Code setup

First, we run each method on two families: the Mersenne dataset, the values $2^i - 1$ for $2 \le i \le 107$ (using the general definition, so both prime and composite exponents appear; 106 inputs, of which 11 are prime); and the Fermat dataset, the values $F_n = 2^{2^n} + 1$ for $0 \le n \le 7$ (note $F_0, \ldots, F_4$ are prime); Table 4 records the outcome of every Fermat-number run.

Then, we run the methods for random 20- and 30-digit numbers: each dataset contains 500 integers drawn uniformly at random from $[10^{19}, 10^{20})$ and $[10^{29}, 10^{30})$ respectively, with no screening applied, so a handful of prime inputs occur; the exact lists are archived in the repository. Fermat's method is absent from these comparisons because random inputs typically have factors far from $\sqrt{n}$, its worst case, and trial division was likewise dropped at 30 digits. 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, under Python 3.12 on an Apple M3 chip; the timer's granularity is about one microsecond on this machine, and runs faster than one tick are recorded as 0. All timings are reported to three decimal places, and the code used can be found in this repository.

Sample sizes below the dataset size mean that some runs did not complete. Failure is algorithmic for $p-1$ (all three bases yield a trivial gcd) and for QS (the sieved interval produces too few smooth relations, or no dependency splits n). Rho's recursive wrapper retries failed calls without bound, so a rho run on an input it cannot split does not halt on its own; such runs, and trial division and Fermat's method runs with no prospect of finishing (for Fermat's method, the projected search time reaches days or more when the factors are far from $\sqrt{n}$), were terminated by hand and recorded as failures, without a uniform time budget. Section 5 proves that the five rho terminations were forced: no restart budget could have succeeded there. Prime inputs are detected by the Miller–Rabin check in the rho, $p-1$, and QS wrappers and count as successes for those methods; trial division and Fermat's method have no primality check and must process prime inputs in full.

Remark 3.1 (Provenance of the Pollard timings). In an earlier version of this work, the rho implementation above was mislabeled Pollard's $p-1$; we had conflated the two methods, and all previously reported "$p-1$" timings were in fact rho timings. This revision separates the methods: the rho columns below reproduce the original measurements under their correct

name, and the $p-1$ columns report new measurements of the implementation described in Section 2, collected on the same hardware. The Fermat-number table was re-measured for this version with the archived implementations, one timed run per input; the other tables report the original measurements. The repository documents the history.

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 rho	Pollard's $p-1$	Quadratic Sieve
Median	0.171	0.009	0.137	0.036	29.918
Min	0.000	0.004	0.001	0.001	0.001
Max	102 767.441	825.599	3478.245	0.681	221 130.908
Q1 (25%)	0.005	0.005	0.028	0.032	0.484
Q3 (75%)	4.244	0.030	0.558	0.042	1507.411
Sample size	95	74	101	103	106

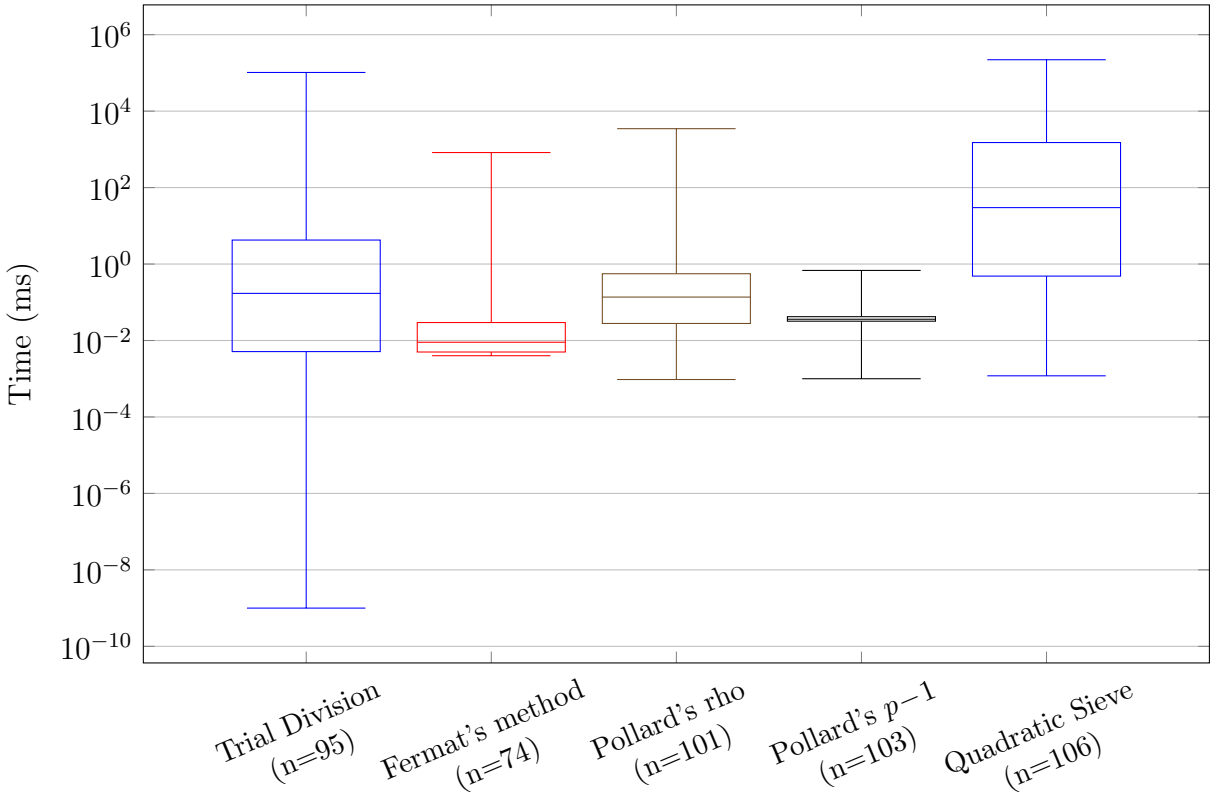

Figure 1: Box plots of runtimes on the Mersenne dataset (log scale); zero timings are plotted at 10^{-9} ms to admit the log axis.

In the Mersenne dataset (Table 1, Figure 1), Fermat's factorization method is generally inefficient, especially when the prime factors of the number we are factoring are far apart. Its overall success rate, 69.8%, is driven down by failures on larger, unbalanced inputs, while the other methods are all above 89% on the Mersenne numbers 2^i-1, $2 \leq i \leq 107$ (Table 2). With $n = 106$, the Wilson intervals of rho and p–1 overlap substantially, so their ranking should not be read as strict. The failure sets themselves are informative. Trial division's eleven failures split into seven inputs that are out of its reach outright, $i \in \{83, 89, 97, 98, 101, 103, 107\}$, each needing from about half an hour to more than a decade of CPU in our implementation, and four manual terminations of runs in the roughly 13 to 70 second range. Fermat's method completes every even exponent instantly, since $2^{2m}-1 = (2^m-1)(2^m+1)$ is a difference of squares split at the very first step, together with 21 small odd exponents; its 32 failures are the remaining odd exponents. Both characterizations were reconstructed by rerunning the archived implementations under fixed budgets, since the original terminations were manual. However, when factors are close, Fermat's method can be remarkably efficient.

Table 2: Success rates of factorization methods (sample size = 106), with 95% Wilson score intervals.

Method	Success Rate (%)	95% CI (%)
Trial Division	89.6%	82.4–94.1
Fermat's method	69.8%	60.5–77.7
Pollard's rho	95.3%	89.4–98.0
Pollard's $p-1$	97.2%	92.0–99.0
Quadratic Sieve	100.0%	96.5–100.0

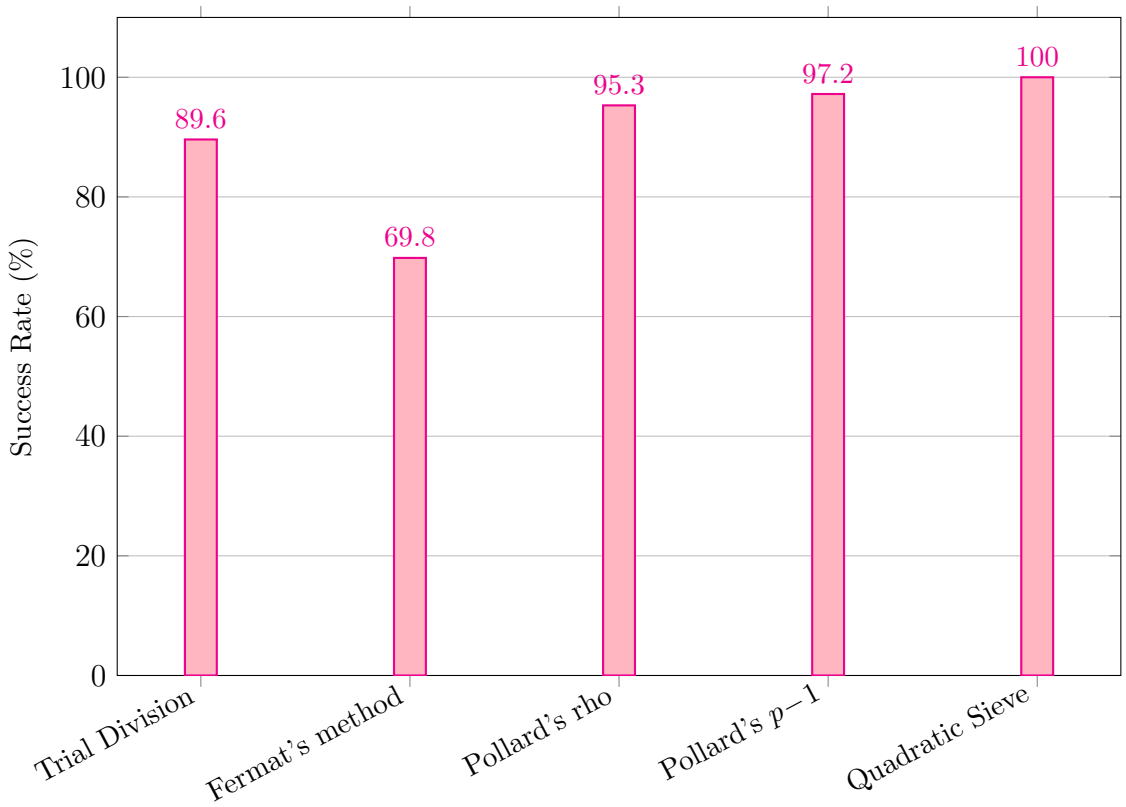

Figure 2: Success rates of factorization methods on the Mersenne dataset (sample size = 106).

Table 3: Timing summary for Fermat numbers (ms).

Statistic	Trial Division	Fermat's method	Pollard's rho	Pollard's $p-1$	Quadratic Sieve
Median	0.003	0.121	0.013	0.007	0.002
Min	0.000	0.006	0.001	0.000	0.001
Max	144.365	6374.951	1.370	0.287	0.012
Q1 (25%)	0.001	0.016	0.003	0.001	0.001
Q3 (75%)	0.028	45.247	0.040	0.135	0.007
Sample size	7	6	7	7	5

Table 4: Outcome of every Fermat-number run (ms; re-measured with the archived implementations). A time marks a completed run; on the prime inputs $F_0, \ldots, F_4$, trial division and Fermat's method process the input in full, the other three methods recognize primality, and Fermat's method halts on a prime p at the trivial pair $(1, p)$. An f marks a run that ended without a factorization: $p-1$ on F_7 exhausts all three bases (no factor's order is 10^5-powersmooth), QS on F_5 and F_6 finds too few smooth relations in its sieve interval, and Fermat's method on F_7 stops at a decimal-precision error. A dash marks a run that is not feasible: projected years of CPU for trial division on F_7 and Fermat's method on F_6, an expected 47 CPU-days under the restart budget for rho on F_7, and memory exhaustion for QS on F_7.

Input	Trial Division	Fermat's method	Pollard's rho	Pollard's $p-1$	QS
F_0	0.003	0.026	0.004	0.001	0.002
F_1	0.001	0.006	0.001	0.001	0.001
F_2	0.000	0.013	0.001	0.000	0.001
F_3	0.001	0.216	0.019	0.007	0.007
F_4	0.004	60.257	0.013	0.012	0.012
F_5	0.052	6374.951	0.060	0.258	f
F_6	144.365	–	1.370	0.287	f
F_7	–	f	–	f	–

Next, we want to see what impact the size of a number has on the effectiveness of factorization methods, so we compare their performance on the random 20- and 30-digit datasets (Tables 5 and 6, Figures 4 and 5). Both Pollard methods performed so well that many individual runs fall below the timer's resolution and are recorded as 0, which is why the first quartiles are 0.

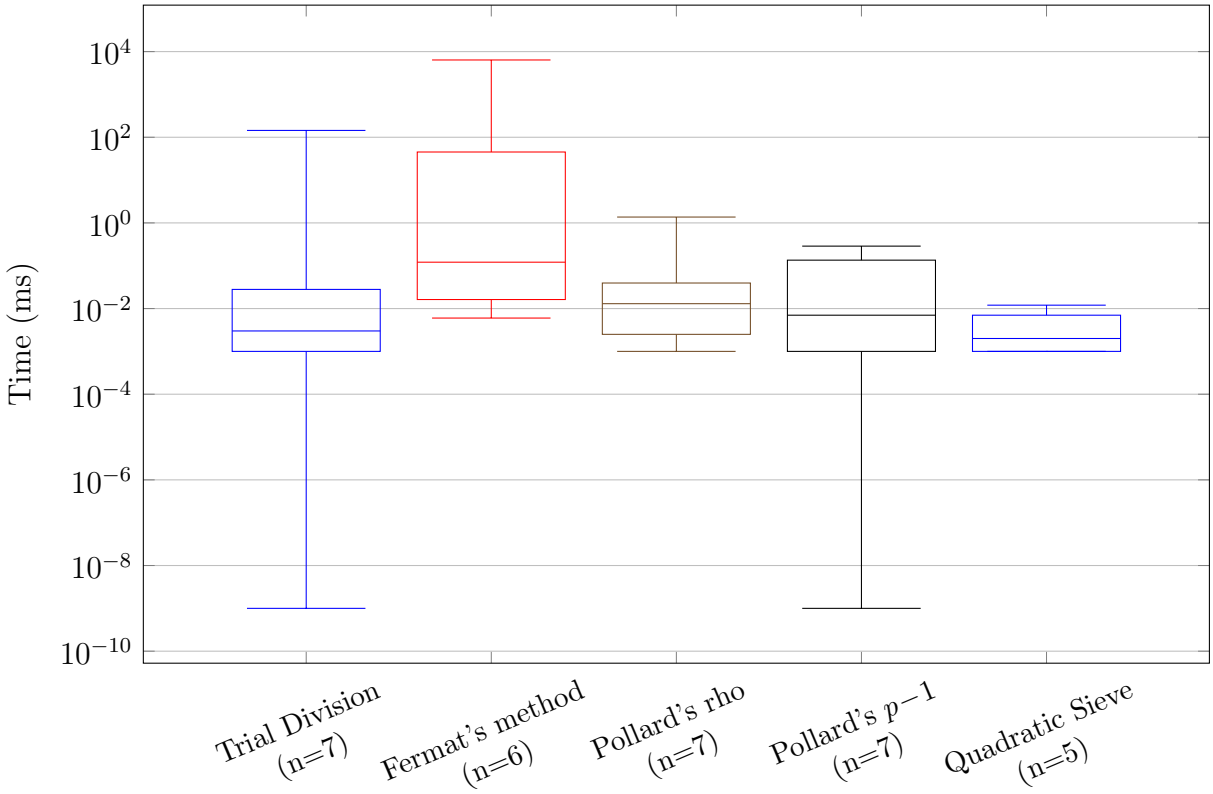

Figure 3: Box plots of runtimes on the Fermat-number dataset (log scale); zero timings are plotted at 10^{-9} ms to admit the log axis.

Table 5: Timing summary for the random 20-digit dataset (ms).

Statistic	Trial Division	Pollard's rho	Pollard's $p-1$	Quadratic Sieve
Median	109.510	0.001	0.001	301.880
Min	0.066	0.000	0.000	0.098
Max	392 481.974	2815.251	11.470	559.046
Q1 (25%)	15.080	0.000	0.000	256.980
Q3 (75%)	1725.358	0.005	0.265	347.848
Sample size	500	500	498	500

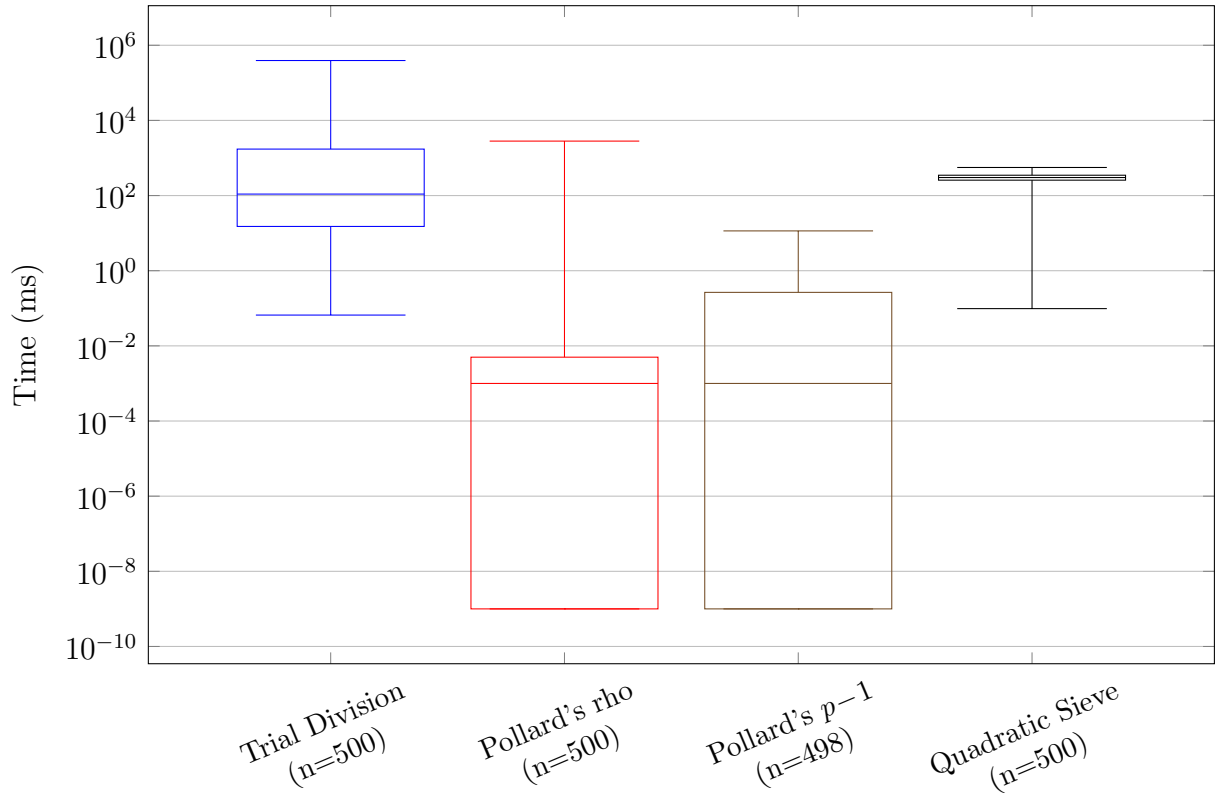

Figure 4: Box plots of runtimes on the random 20-digit dataset (log scale); zero timings are plotted at 10^{-9} ms to admit the log axis.

Table 6: Timing summary for the random 30-digit dataset (ms).

Statistic	Pollard's rho	Pollard's $p-1$	Quadratic Sieve
Median	0.001	0.001	26 159.587
Min	0.000	0.000	0.156
Max	4294.464	8.964	38 616.475
Q1 (25%)	0.000	0.000	22 629.178
Q3 (75%)	0.008	0.343	29 775.944
Sample size	500	493	500

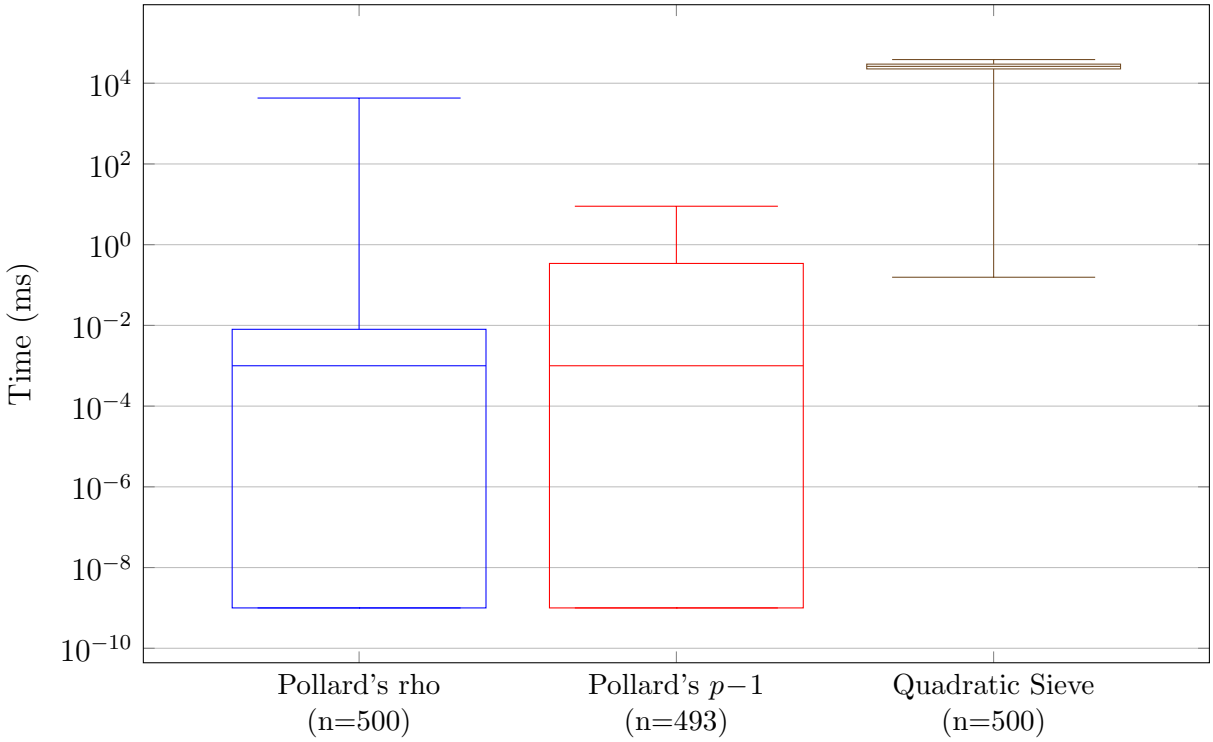

Figure 5: Box plots of runtimes on the random 30-digit dataset (log scale); zero timings are plotted at 10^{-9} ms to admit the log axis.

In summary, in the Mersenne dataset, the median runtimes are in the order of Fermat's method 0.009, Pollard's p–1 0.036, Pollard's rho 0.137, trial division 0.171, and the Quadratic Sieve 29.918; these medians cover completed runs only, over different completion sets (Table 2). The Interquartile Ranges (IQRs) are tight for Fermat's method 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 (Tables 3 and 4), the medians mostly measure how quickly each method disposes of the five prime inputs; the informative composite cases are $F_5 = 641 \times 6700417$ and $F_6 = 274177 \times 67280421310721$. Trial division, rho, and $p-1$ split both in at most a few hundred milliseconds; Fermat's method needs about 6.4 seconds on F_5, whose factors are far apart, and cannot reach F_6; and QS fails on both composites for want of smooth relations in its single sieve interval. The right tails are long for trial division and Fermat's method.

On the random 20-digit dataset, Table 5 shows trial division at 109.510 and QS at 301.880 against both Pollard methods at ≈ 0.001, the timer's resolution floor; at 30 digits (Table 6), the QS median grows to about 26.2 seconds while both Pollard methods stay at the floor. A random integer usually has a small prime factor, which is exactly the structure both Pollard methods exploit, for different reasons. Pollard's p–1 failed on 2 of the 500 twenty-digit inputs and 7 of the 500 thirty-digit inputs; in every case the cause is ordinary non-smoothness (no prime factor q of the input has $q-1$ being 10^5-powersmooth), not synchronization.

The measurements can also be compared, roughly, against the standard cost estimates. The Quadratic Sieve's heuristic cost grows like $L(n)^{1+o(1)}$ with $L(n) = \exp(\sqrt{\ln n \ln\ln n})$; between our 20- and 30-digit datasets this predicts a slowdown of about $L(10^{30})/L(10^{20}) \approx 46$, and the measured median slowdown is $26159.586/301.880 \approx 87$, the right order of magnitude, with the gap attributable to the $o(1)$ term and Python's memory-bound linear algebra. For rho, the birthday heuristic predicts on the order of $\sqrt{p}$ iterations to find a factor p: at $p = 7432339208719$ that is about 2.7 million iterations, consistent with the repeated $8 \times 200{,}000$-iteration calls that $2^{101}-1$ cost us, and it explains why the same budget finds factors of up to roughly ten digits ($\sqrt{p} \le 10^5$) reliably within a single call.

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 The disjoint failure sets

The failures in the Mersenne data turned out to be as interesting as the successes. In our runs, Pollard's rho fails on exactly the five inputs 2^i-1 with $i \in \{20, 40, 60, 80, 100\}$, and Pollard's $p-1$ fails on exactly the three inputs with $i \in \{11, 29, 101\}$. The two sets are disjoint, and each method factors the other's failures, though not at the same cost: $p-1$ splits all five of rho's stalls in tens of microseconds, and rho factors $2^{11}-1$ and $2^{29}-1$ in microseconds, while $2^{101}-1$ costs rho repeated restarts and anywhere from seconds to minutes of CPU time, since its smaller prime factor 7432339208719 has thirteen digits. Both phenomena have proofs, of

different strengths: the rho failures are forced for every seed and every restart (Section 5.1), while the successes elsewhere are empirical outcomes of the randomized search.

5.1 Why rho fails on the multiples of 20

The order of 2 modulo 25 is 20, so $25 \mid 2^i - 1$ exactly when $20 \mid i$. A single factor of 5 appears whenever $4 \mid i$ (the order of 2 mod 5 is 4) and is harmless; the obstruction is the prime square. One of our five inputs carries more: $5^3 \mid 2^{100} - 1$, while the other four carry exactly 5^2. This changes nothing, because the iteration does split 125: an exhaustive check shows the nontrivial gcds reachable modulo 125 are 25 and 125, so rho peels off a single 5 and is left with the same unsplittable 25.

Proposition 5.1. *The rho iteration with $f(x) = x^2 + 1$ cannot split 25: for every starting value $x_0 \in \mathbb{Z}/25\mathbb{Z}$ and every pair of iterates x, y produced by Floyd cycle detection,* $\gcd(|x - y|, 25) \in \{1, 25\}$.

Proof. Direct computation over all 25 starting values shows every orbit of f on $\mathbb{Z}/25\mathbb{Z}$ enters the 3-cycle $1 \mapsto 2 \mapsto 5 \mapsto 1$ within at most four steps. The cycle elements $\{1, 2, 5\}$ are pairwise distinct modulo 5. For $k \geq 4$, both Floyd iterates x_k and x_{2k} lie on the cycle, and there any congruence $x \equiv y \pmod 5$ forces $x = y$ in $\mathbb{Z}/25\mathbb{Z}$, so $|x - y|$ is divisible by 25 or coprime to 5: the gcd is never exactly 5. This leaves only the finitely many pairs with $1 \leq k \leq 3$; enumerating them over all 25 starting values shows the gcd never equals 5 there either. □

Consequently, on the five inputs divisible by 25, rho strips the other prime factors (composite exponents provide many, since $2^d - 1 \mid 2^i - 1$ for every $d \mid i$) and then gets stuck forever on the unsplittable 25; restarts cannot help because all 25 seeds funnel into the same cycle. Pollard's $p-1$ handles these inputs instantly, since $5 - 1 = 4$ is 4-powersmooth. The failure is specific to the map: for instance, $x^2 + 2$ splits 25 without difficulty, though Section 6 shows it acquires a blind spot of its own at 15.

5.2 Why $p-1$ fails on 11, 29, 101

Every prime factor q of $2^p - 1$ with p an odd prime satisfies $q \equiv 1 \pmod{2p}$, so p divides $q - 1$ for *every* factor simultaneously; the same congruence is what the GIMPS project exploits to accelerate its factor searches [21]. When, for a given base b, every factor's multiplicative order retains the prime p, and p is in each case the largest prime power dividing the order, all factors of n surface at the same single multiplication in the staged exponentiation, the one at which the exponent first absorbs p: the gcd jumps from 1 directly to n, and no backtracking granularity can separate them. We call this *order synchronization*. A base can also fail outright, with the gcd stuck at 1, when no factor is ever exposed because some

prime power of its order exceeds the smoothness bound; both failure routes occur on the three inputs below.

Example 5.2. $2^{11}-1 = 2047 = 23 \times 89$, with $23-1 = 2 \cdot 11$ and $89-1 = 2^3 \cdot 11$. For each of the bases $b \in \{2,3,5\}$, the orders of b modulo both factors retain the factor 11 (they are $11, 11$; $11, 88$; and $22, 44$), with 11 the largest prime power dividing each, so both factors surface exactly when the exponent absorbs 11, and the gcd returns 2047. The same synchronization, for all three bases, occurs for $i = 29$. For $i = 101$, base 2 synchronizes in the same way (the orders of 2 modulo both factors equal 101), while bases 3 and 5 fail by non-exposure: their orders modulo the two factors are divisible by the primes 278557 and 295985357 respectively, both exceeding $B = 10^5$, so neither factor ever surfaces and the gcd stays at 1.

Synchronization is a tendency rather than a law: most prime-exponent Mersenne inputs escape because some base has a desynchronized order (for $2^{23}-1$, base 3 exposes the factor 47 alone), or because the two-power parts of the $q-1$ differ. Three of the 17 composite prime-exponent inputs resisted all three bases. Interestingly, rho's failures all have composite exponents, while $p-1$'s failures all have prime exponents. The first method is defeated by extra algebraic structure, and the second by too much uniformity, which the congruence $q \equiv 1 \pmod{2p}$ forces.

6 Predictions and validation

The mechanisms of Section 5 would be more convincing if they could predict measurements we had not yet made. This section reports four such tests. In every case the prediction was computed first, from the mechanism alone, and the experiment was run afterward.

6.1 Predicting the $p-1$ failure set

Our implementation builds the exponent one prime multiplication at a time, so for a base b, each prime factor q of n has a well-defined *exposure step*: the multiplication at which the accumulated exponent first becomes divisible by the multiplicative order of b modulo q. Since smaller primes are processed first, the exposure step is determined by the prime power of the largest prime dividing the order; we call that pair, the largest prime together with its exponent in the order, the *exposure key*. The factor q is never exposed if some prime power of its order exceeds B. A base therefore fails exactly when no factor of n is exposed at all, or when every factor shares the same first exposure key.

Using only the factorizations of the inputs and the multiplicative orders of $2, 3, 5$, we computed a predicted verdict for each of the 17 composite prime-exponent Mersenne numbers

in our range, before consulting the timing data. Fourteen inputs were conclusively predicted (for the exponents 47, 53, and 59, an order resisted factoring within our time limits), and all fourteen predictions were correct; the three undecided inputs all factored successfully in the runs. In particular, the predicted failure set is exactly $\{11, 29, 101\}$, matching the observed one.

6.2 Predicting base-failure densities

The same model predicts, for a fixed input, the fraction of *all* bases that fail: one computes the exposure keys over every residue and counts collisions. One implementation detail matters here. The first gcd check occurs after the first squaring rather than at the base itself, so a factor whose order is 1 and a factor whose order is 2 are first detected at the same check and must therefore share an exposure key. With that convention, for $2047 = 23 \times 89$ the model predicts 1602 failing bases out of the 1934 valid ones (the residues $2 \le b \le 2045$ with $\gcd(b, 2047) = 1$), a fraction of 0.8283, and measurement returns the same 1602: model and algorithm agree base by base, with no exceptions. Giving the order 1 a class of its own instead predicts 1600, or 0.8273, and mispredicts exactly two bases. The simple group-theoretic estimate $(10/11)^2 \approx 0.826$ explains the bulk of the value, since an order retains the factor 11 for 10 of every 11 bases, independently modulo each factor. For $2^{29}{-}1 = 233 \times 1103 \times 2089$ the model predicts 1352 failures over the 1493 sampled bases $2 \le b \le 1501$ with $\gcd(b, 2^{29}{-}1) = 1$, a fraction of 0.9056, and measurement returns the same 1352, again base by base.

6.3 Classifying rho's blind spots

Exhaustive computation over all primes $p \le 97$ shows that the only prime squares p^2 that the iteration $x^2 + 1$ cannot split are 4 (irrelevant, since even inputs are handled separately) and 25: within this range, 25 is the *unique* odd prime-square blind spot. Varying the map, the values $c \in \{1, \ldots, 24\}$ for which $x^2 + c$ cannot split 25 are exactly $1, 6, 11, 16, 21$, and 19: the first five are the residues $c \equiv 1 \pmod 5$, whose mod-5 dynamics coincide with those of $c = 1$, while $c = 19$ is a single anomaly whose lifted cycle structure happens to collapse as well.

Switching maps relocates blind spots rather than removing them. The iteration $x^2 + 2$ splits 25 easily, but it cannot split 15. For $k \ge 1$, Floyd's iterates satisfy $x_k \equiv x_{2k}$ modulo a divisor d exactly when k is at least the tail length of the orbit modulo d and is a multiple of its cycle length (if $x_k = x_{2k}$, then x_k is a periodic point whose period divides k, which forces both conditions; the converse is immediate), so the steps at which the two iterates agree modulo d depend only on that pair of lengths. Modulo 3 and modulo 5, the orbits of $x^2 + 2$ both fall into cycles of length two with tails no longer than two, so both primes yield the

identical list of agreement steps: the iterates are either equal modulo both primes or differ modulo both, and the partial collision that produces a factor never occurs. The obstruction is the coincidence of the cycle lengths combined with the short tails, not the relative phase of the two cycles; phase cannot matter, since it does not enter the agreement condition. The qualitative criterion, equal cycle and tail data across the factors forcing a trivial gcd, is in print in Galbraith [22, Section 14.9]. Consequently, factoring our five rho-failing Mersenne inputs with fixed $c = 2$ leaves all five stuck, though not all at the same place: four stall at the cofactor 15, while $2^{60}-1$, the only one of the five divisible by 9 (the order of 2 modulo 9 is 6, and $6 \mid 60$), stalls at $45 = 9 \cdot 5$. The extra factor of 3 survives for the same reason 25 survives $x^2 + 1$: every orbit of $x^2 + 2$ on $\mathbb{Z}/9\mathbb{Z}$ collapses into the 2-cycle $\{2, 6\}$, whose elements are distinct modulo 3, so the gcd is never exactly 3; an exhaustive check modulo 45 confirms the only nontrivial gcd the iteration can produce there is 45 itself. Choosing a fresh random c at every restart, which is essentially what practical implementations do when a polynomial fails, factored all five completely. The classical remedy of varying the polynomial is exactly a desynchronization strategy. Matching the polynomial to the input's structure is classical in the positive direction as well: Brent and Pollard factored the eighth Fermat number with $f(x) = x^{2^{10}} + 1$, chosen because the factors of F_8 satisfy $p \equiv 1 \pmod{2^{10}}$ [23].

6.4 An unseen family: repunits

Repunits $R_p = (10^p - 1)/9$ carry the same congruence structure as Mersenne numbers in base ten: for prime p, every prime factor q of R_p (other than p itself) satisfies $q \equiv 1 \pmod{p}$: the order of 10 modulo q divides the prime p, and it equals 1 only for $q = 3$, which divides R_p only when $p = 3$. We computed exposure-key predictions for all ten composite repunits with $p \leq 41$, again before running any timings: the model predicted no synchronization failures, and indeed all ten repunits factored successfully under $p-1$. Ten out-of-sample predictions, ten confirmations, including the less glamorous but equally falsifiable prediction of an *absence* of failures.

Taken together, these tests support a single reading of our data: the observed failures are driven by synchronization. Rho fails when the steps at which its iterates agree coincide across the prime powers dividing the input, which happens when the component cycle lengths match; $p-1$ fails when exposure steps coincide (on $2^{101}-1$, base 2 synchronizes while the other two bases are never exposed at all). The standard folklore remedies, restarting rho with a new polynomial and retrying $p-1$ with a new base, are precisely anti-synchronization measures.

6.5 What is classical here, and what is not

It is worth stating plainly which parts of the above are standard and which are ours, since the ingredients are elementary and several of them are in print. Classical, and not claimed as new: that rho fails when a collision modulo one prime factor is simultaneously a collision modulo the whole input, which is the standard cautionary remark about the method; the qualitative criterion behind it, that equal cycle and tail data across the factors force a trivial gcd [22, Section 14.9]; even the specific example, that x^2+1 started at $x_0 = 1$ returns 25 on the input 25 while $c = 2$ splits it, is algorithmic folklore [24]; that every prime factor q of 2^p-1 with p an odd prime satisfies $q \equiv 1 \pmod{2p}$; that $p-1$ returns gcd $= n$ when every factor is exposed at the same stage, whose qualitative form (all factors sharing the same largest prime power in their orders) is likewise folklore; and the elementary counting used throughout. The cycle and tail structure of the special maps x^2 and x^2-2 over prime fields was analyzed in detail by Vasiga and Shallit [25], and the general theory of how cycles of polynomial maps lift from $\mathbb{Z}/p\mathbb{Z}$ to $\mathbb{Z}/p^k\mathbb{Z}$, which is the dynamical machinery underneath our blind-spot computations, has recently been worked out in full generality by Nara [26]. What we claim is the resolution and the validation, not the ingredients. Specifically: (i) we exhibit two failure sets that are exact, disjoint, and arithmetically described, rather than anecdotal, over a fixed family of 106 inputs; (ii) we sharpen the folklore example at 25 to an impossibility statement, for every seed and every Floyd pair, which is stronger than the usual "restart and try again" advice and explains why restarting cannot help; (iii) we turn the $p-1$ mechanism into a per-base predictive model, the exposure key, calibrated to our implementation's checkpoint boundary, and test it before measuring rather than describing failures after the fact; and (iv) we classify, exhaustively within stated ranges, the blind spots at 25 of the maps x^2+c for $c \in \{1, \ldots, 24\}$ and the odd prime-square blind spots of x^2+1 for $p \leq 97$, which shows that changing the polynomial relocates the obstruction instead of removing it. To our knowledge this combination, an exact failure set together with a proof, a per-base model with exact base-by-base agreement, and an out-of-sample confirmation on repunits, has not appeared together, though the dynamical ingredients are available in [26].

6.6 Reproducibility

The public repository at `https://github.com/mintylemon66/factorization` contains our implementations of all five methods, the random 20- and 30-digit input lists, the Mersenne generator, the timing loops, and, in its August 2026 revision, the analysis scripts behind Sections 5 and 6: the exposure-key model and its base-by-base density counts, the exhaustive x^2+c and prime-square sweeps, the fixed-c stall check, and the repunit predictions. The repository also documents the labeling error described in the remark of Section 3 and the

revision that corrected it. We encourage readers to rerun the experiments: the exhaustive checks in this section, over all 25 seeds modulo 25, over all c in the stated ranges, and over the odd primes $p \leq 97$, complete in seconds on an ordinary laptop. The timing tables themselves depend on hardware and, for trial division and Fermat's method, on the manual termination protocol of Section 3.

7 Discussion

7.1 Patterns

For Mersenne number inputs, Fermat's method is best in median time, with the two Pollard methods close behind, and QS is much slower in this number class. Note that the time for Fermat's method is calculated over its completed runs, which are biased toward its easy case.

For Fermat numbers, the Pollard methods are significantly faster than QS and trial division, while Fermat's method is slower in comparison.

For random composites of 20–30 digits, the Pollard methods are significantly faster than the other methods.

Fermat's factorization is optimal for near-square integers; Pollard's rho is optimal for integers whose smallest prime factor is small; Pollard's $p-1$ is optimal for powersmooth $p-1$ cases; and very large inputs that lack the above structures require QS to finish in reasonable time.

Note that not all tests are factorization algorithms. When 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).

7.2 More observations

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

In some cases, a runtime of 0 reflects a coarse timer resolution, not truly instantaneous runs. Runs recorded as 0 are plotted at the 10^{-9} ms floor in the box plots and flagged in the captions.

7.3 Limitations

There are several caveats. First, medians are computed over completed runs only, so they exhibit survivorship bias; this is why success rates are reported beside the Mersenne timings. Each input was timed once; the large per-dataset samples and the reported quartiles mitigate,

but do not replace, repeated measurement. Trial division runs to a complete factorization while the other four methods stop at the first nontrivial split, so its columns are upper bounds relative to the others. Prime inputs are handled asymmetrically: the rho wrapper, the $p-1$ wrapper, and QS apply a Miller–Rabin test and return immediately on primes, while trial division and Fermat's method have no such check. Finally, our implementations are baselines: we compare our five implementations against one another, not each algorithm at its tuned best; a multiple-polynomial QS with sparse linear algebra would substantially outperform ours.

8 Conclusions & future work

Takeaways

To find nontrivial factors of Mersenne numbers, a practical order of methods is: Fermat $\rightarrow$ the Pollard methods $\rightarrow$ Quadratic Sieve. For Fermat numbers, a practical order is: the Pollard methods $\rightarrow$ Trial Division $\rightarrow$ QS. For random 20–30 digit composites, start with the Pollard methods and use QS only after the easy factors are exhausted. The disjoint failure sets of Section 5 suggest running rho and $p-1$ together: on our data, each one covers exactly the structured inputs that defeat the other. The bigger lesson for us was that a method's name matters less than its mechanism.

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 the Elliptic Curve Method and a second stage for 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.

Acknowledgments

We thank Prof. Sandy Zabell, whose mentorship through the Pioneer Research Institute inspired this paper, for his lectures, guidance, and invaluable suggestions and support throughout.

References

[1] J. M. Pollard, "Theorems on factorization and primality testing," *Proceedings of the Cambridge Philosophical Society*, vol. 76, no. 3, pp. 521–528, 1974. doi: 10.1017/S0305004100049252.

[2] J. M. Pollard, "A Monte Carlo method for factorization," *BIT*, vol. 15, no. 3, pp. 331–334, 1975. doi: 10.1007/BF01933667.

[3] M. A. Morrison and J. Brillhart, "A method of factoring and the factorization of F_7," *Mathematics of Computation*, vol. 29, no. 129, pp. 183–205, 1975. doi: 10.1090/S0025-5718-1975-0371800-5.

[4] 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.

[5] 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.

[6] J. P. Buhler, H. W. Lenstra, Jr., 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, 1993. doi: 10.1007/BFb0091539.

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

[8] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Oxford University Press, 6 ed., 2008. doi: 10.1093/oso/9780199219858.001.0001.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

[19] A. K. Lenstra, H. W. Lenstra, 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.

[20] E. Bach, "Toward a theory of Pollard's rho method," *Information and Computation*, vol. 90, no. 2, pp. 139–155, 1991. doi: 10.1016/0890-5401(91)90001-I.

[21] GIMPS, "Mathematics and research strategy." `https://www.mersenne.org/various/math.php`. Great Internet Mersenne Prime Search. Accessed August 2026.

[22] S. D. Galbraith, *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012. doi: 10.1017/CBO9781139012843.

[23] R. P. Brent and J. M. Pollard, "Factorization of the eighth Fermat number," *Mathematics of Computation*, vol. 36, no. 154, pp. 627–630, 1981. doi: 10.1090/S0025-5718-1981-0606520-5.

[24] cp-algorithms contributors, "Integer factorization." `https://cp-algorithms.com/algebra/factorization.html`. Accessed August 2026.

[25] T. Vasiga and J. Shallit, "On the iteration of certain quadratic maps over GF(p)," *Discrete Mathematics*, vol. 277, no. 1–3, pp. 219–240, 2004. doi: 10.1016/S0012-365X(03)00158-4.

[26] T. Nara, "Lifting of cycles in functional graphs," 2025. arXiv:2509.16234.