How Hard Is Factoring? Classical and Quantum Approaches

A 25-minute tour of classical, modern, and quantum approaches

Riccardo Chimisso

Itinerary

- **Problem statement and why factoring matters** ($\sim$ 3 *minutes*)

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- **Classical algorithms: brute force $\rightarrow$ clever shortcuts** ($\sim$ 8 *minutes*)

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- Classical algorithms: brute force $\rightarrow$ clever shortcuts ($\sim$ 8 *minutes*)
- **Modern SotA: Number Field Sieve (NFS)** ($\sim$ 5 *minutes*)

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- Classical algorithms: brute force $\rightarrow$ clever shortcuts ($\sim$ 8 *minutes*)
- Modern SotA: Number Field Sieve (NFS) ($\sim$ 5 *minutes*)
- **Quantum mechanics: literally everything even remotely related**

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- Classical algorithms: brute force $\rightarrow$ clever shortcuts ($\sim$ 8 *minutes*)
- Modern SotA: Number Field Sieve (NFS) ($\sim$ 5 *minutes*)
- **Quantum mechanics: literally everything even remotely related** ($\sim$ 99999999999999999999

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- Classical algorithms: brute force $\rightarrow$ clever shortcuts ($\sim$ 8 *minutes*)
- Modern SotA: Number Field Sieve (NFS) ($\sim$ 5 *minutes*)
- ~~Quantum mechanics: literally everything even remotely related ($\sim$ 9999999999999999999999999~~
- **Quantum approach: Shor's algorithm and implications** ($\sim$ 4 *minutes*)

Itinerary

- Problem statement and why factoring matters ($\sim$ 3 *minutes*)
- Classical algorithms: brute force $\rightarrow$ clever shortcuts ($\sim$ 8 *minutes*)
- Modern SotA: Number Field Sieve (NFS) ($\sim$ 5 *minutes*)
- ~~Quantum mechanics: literally everything even remotely related ($\sim$ 9999999999999999999999999~~
- Quantum approach: Shor's algorithm and implications ($\sim$ 4 *minutes*)
- **Conclusion and takeaways** ($\sim$ 2 *minutes*)

Problem statement

The factoring problem

Input / Output

- Input: An integer $n \in \mathbb{N}_{>1}$
- Output: The unique prime factorization $n = \prod_{i=1}^{k} p_i^{e_i}$

Formal Goal

Determine the strictly increasing sequence of distinct primes $p_1 < p_2 < \cdots < p_k$ and their corresponding positive integer exponents $e_1, e_2, \ldots, e_k$ such that $n = \prod_{i=1}^{k} p_i^{e_i}$.

Why factoring matters

Key Case: Semiprimes

In cryptography (specifically RSA), we construct a modulus $n = p \cdot q$, where p and q are distinct, massive, secret prime numbers.

- **Public Key**
  Encrypt the message M by using the public modulus n and exponent e: $C \equiv M^e \pmod{n}$.
- **Private Key**
  Decrypt the ciphertext C using the private key d: $M \equiv C^d \pmod{n}$. For the exponents to mathematically cancel out and reveal M, d is defined such that $e \cdot d \equiv 1 \pmod{\varphi(n)}$.
- **Vulnerability**
  $\varphi(n) = (p-1)(q-1)$. If an attacker can factor n to learn p and q, they can compute $\varphi(n)$ and trivially derive the private key d.

Big picture

Practical security depends entirely on the computational cost of factoring n at a given bit size.

Complexity

- **Metric:** Input size is the bit-length $b \approx \log_2 n$.
- **Time Classes:**
  - **Polynomial (**Tractable**):** $O(b^k)$ for some constant k.
  - **Exponential (**Intractable**):** $2^{\Omega(b)}$.
  - **Sub-exponential (**Purgatory**):** $2^{o(b)} \wedge \omega(\text{poly}(b))$.
- **Factoring is hard:**
  - The best known classical algorithm is **sub-exponential**.
  - To date, no classical polynomial-time algorithm is known to exist.

Classical algorithms

Baseline: Trial division

Core idea

Find the smallest non-trivial divisor of n by testing 2, then odd integers $3, 5, 7, 9, \ldots, \lfloor \sqrt{n} \rfloor$. Whenever a divisor is found, divide it out completely and continue on the reduced cofactor.

1. Test divisibility by 2. While $2 \mid n$, output 2 and set $n \leftarrow n/2$.
2. For odd $d = 3, 5, 7, 9, \ldots$, while $d^2 \leq n$:
   - if $d \mid n$, output d and repeatedly set $n \leftarrow n/d$,
   - otherwise increment $d \leftarrow d + 2$.
3. If the remaining $n > 1$, output it as the final prime factor.

Performance profile

Worst case: $\Theta(\sqrt{n}) = \Theta(2^{\frac{b}{2}})$ trial divisions.

Fermat factorization

Core idea

For odd $n = pq$,

$$n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{q-p}{2}\right)^2.$$

So factoring n reduces to finding $x^2 - n = y^2$.

- Start $x \leftarrow \lceil \sqrt{n} \rceil$.
- If $x^2 - n$ is a square y^2, output $(x-y)(x+y)$.
- Otherwise increment x.

Performance profile

Effective when the two factors p and q are **close together**. In the worst case it can require about $\frac{q-p}{2}$ trials.

Pollard ρ

Core idea

Iterate $x_{i+1} = x_i^2 + c \pmod{n}$. If two values become equal modulo a hidden factor $p \mid n$ ($x_i \equiv x_j \pmod{p}$), then $p \mid (x_i - x_j)$.

1. Initialize x, y, c.
2. Iterate with cycle detection: $x \leftarrow f(x)\,, \quad y \leftarrow f(f(y)), \quad g = \gcd(|x - y|, n)$.
3. If $1 < g < n$, output g; otherwise restart with a new seed.

Performance profile

Randomized, tiny memory, expected $\widetilde{O}(\sqrt{p}) \approx \widetilde{O}(2^{\frac{b}{4}})$ for the smallest factor p.

Pollard $p-1$

Core idea

If $p \mid n$ and $p-1 \mid M$, then by Fermat $a^M \equiv 1 \pmod{p}$, so $p \mid (a^M - 1)$.

1. Choose a smoothness bound B and a base a.
2. Let M be a multiple of all prime powers $\leq B$.
3. Compute $g = \gcd(a^M - 1, n)$.
4. If $1 < g < n$, output g; otherwise change B or a.

Performance profile

One run costs roughly $O(B\,\mathrm{poly}(b))$. If $p-1$ is B-smooth, the method can be very fast; in the worst case B may need to reach $\sqrt{n} = 2^{\frac{b}{2}}$, matching trial division asymptotically.

Elliptic Curve Method (ECM)

Core idea

Pollard $p-1$ gets one hidden group per factor: its size is fixed at $p-1$. ECM keeps the same smoothness idea, but each new elliptic curve gives a *different* hidden group size modulo the same factor p.

- Pick a random elliptic curve E and a point P modulo n.
- Multiply P by a smooth scalar M.
- If the curve has smooth enough size modulo some factor p, a gcd reveals p.

Performance profile

Randomized; heuristic expected time

$$\exp\Big((\sqrt{2}+o(1))\sqrt{\ln p \ln\ln p}\Big)\cdot \mathrm{poly}(b)$$

So ECM is sub-exponential in the factor size.

Quadratic Sieve (QS)

Core idea

Fermat wants one value $x^2 - n$ that is already a square. QS is less demanding: it gathers many values of $Q(x) = x^2 - n$ that factor completely over small primes, then combines some of them so the total product becomes a square.

1. Study $Q(x)$ for x near $\sqrt{n}$, where the values are much smaller than n.
2. Keep the x-values for which $Q(x)$ is smooth over a small **factor base**.
3. Record only exponent parities (odd/even), and find a subset whose parities cancel mod 2.
4. The chosen relations give $X^2 \equiv Y^2 \pmod{n}$; try $\gcd(X - Y, n)$ and, if needed, $\gcd(X + Y, n)$.

Performance profile

Heuristic runtime $L_n\left[\frac{1}{2}, 1\right] = \exp\left((1 + o(1))\sqrt{\ln n \ln \ln n}\right)$: sub-exponential, and once the best practical general-purpose method before NFS.

Modern state of the art

From QS to GNFS

Insight

QS and GNFS have the same aim: build a non-trivial congruence of squares $X^2 \equiv Y^2 \pmod{n}$. GNFS wins on very large inputs because it has a more powerful way to collect the relations that make this possible.

- QS works with one family of values: $x^2 - n$.
- GNFS first chooses an integer m and a polynomial f with $f(m) \equiv 0 \pmod{n}$.
- It then sieves many pairs (a, b), looking for cases where both an **integer-side** value and an **algebraic-side** value are smooth.
- More machinery, but better asymptotics for large general integers such as RSA moduli.

Big picture

Same trick as QS, stronger relation-collection framework.

General Number Field Sieve (GNFS)

Why these relations?

Choose a polynomial f of degree d and an integer m with $f(m) \equiv 0 \pmod{n}$.
Let α denote the class of x in $\mathbb{Q}[x]/(f)$, so $f(\alpha) = 0$. Then GNFS evaluates the *same* linear form $a - bx$ in two compatible settings:

Rational side

$$a - bm$$

- ordinary integer arithmetic
- smoothness over rational primes

Algebraic side

$$a - b\alpha, \qquad N(a - b\alpha) = b^d f\left(\frac{a}{b}\right)$$

- arithmetic in a number field
- norm turns it into an integer test

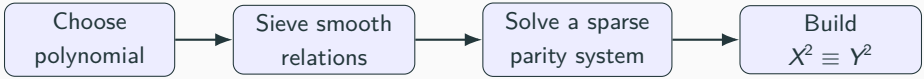
Why a pair (a, b)?

The pair represents the rational $\frac{a}{b}$. This is the natural coordinate system for the homogeneous form $b^d f\left(\frac{a}{b}\right)$, and it lets GNFS search many more candidates, while keeping them smaller.

General Number Field Sieve (GNFS)

Core idea

Sieve many pairs (a, b), keeping only those for which both $a - bm$ and $N(a - b\alpha)$ are smooth. A parity dependency among these relations eventually yields a factor of n.



- Sieving keeps only pairs (a, b) whose two sides are smooth.
- Linear algebra mod 2 finds a parity dependency.
- Square root + gcd to extract a non-trivial factor.

Performance profile

Heuristic runtime $L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right] = \exp\left(\left(\left(\frac{64}{9}\right)^{\frac{1}{3}} + o(1)\right)(\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}}\right)$, asymptotically better than QS. For large general integers, GNFS is the fastest known classical method.

Security implications (classical)

- For ordinary RSA moduli, the relevant general-purpose threat is **GNFS**.
- Practical risk depends on implementation quality, hardware, and how much parallel work can be thrown at sieving and linear algebra.
- The main defenses are:
  - sufficiently large moduli,
  - sound random prime generation,
  - avoiding primes that are too close or have obvious special structure.

Quantum algorithms

Quantum intuition for Shor

Three ideas only

- **Superposition:** a quantum register can encode many candidate exponents x at once.
- **Measurement:** in the end you only read one ordinary outcome, so "many values at once" is not enough by itself.
- **Interference:** a useful quantum algorithm arranges amplitudes so outcomes matching a hidden structure become more likely.

For Shor specifically

The hidden structure is a **period**. The quantum part is not faster brute force over divisors; it is a pattern-finding routine for that period.

Quantum reduction: factoring $\rightarrow$ order finding

From factoring to a cycle

Pick random a with $\gcd(a, n) = 1$, and study $f(x) = a^x \bmod n$.
Because modular powers eventually repeat, there is a smallest $r > 0$ such that $a^r \equiv 1 \pmod{n}$.
That r is the **order**, or period, and if it is even and $a^{\frac{r}{2}} \not\equiv \pm 1 \pmod{n}$, then

$$a^r - 1 = \left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right)$$

is divisible by n and one of

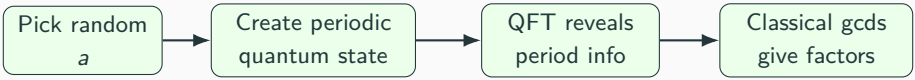
$$\gcd\left(a^{\frac{r}{2}} - 1, n\right), \qquad \gcd\left(a^{\frac{r}{2}} + 1, n\right)$$

often gives a non-trivial factor.

Shor's algorithm (high-level)

Where the quantum speedup lives

The hard part is finding the period of $f(x) = a^x \bmod n$. Shor replaces that step with a quantum period-finding subroutine.



- The repeated pattern in $a^x \bmod n$ is encoded in the quantum state.
- The QFT acts like a frequency detector: regular spacing leads to peaks tied to r.
- Measuring those peaks, then doing classical post-processing, gives the order r.

Practical constraint

In theory the runtime is polynomial in the bit-length b, but a cryptographically relevant attack still needs large fault-tolerant quantum hardware.

Conclusion

Looking back

- **Classical trip**: from simple ideas to special-purpose algorithms to the best currently known, GNFS.

Looking back

- Classical trip: from simple ideas to special-purpose algorithms to the best currently known, GNFS.
- **Quantum excursion**: Shor changes the asymptotic picture to polynomial time through specialized quantum order finding.

Looking back

- Classical trip: from simple ideas to special-purpose algorithms to the best currently known, GNFS.
- Quantum excursion: Shor changes the asymptotic picture to polynomial time through specialized quantum order finding.
- **Just a tour**: an overview of past and current important algorithms, often combined to exploit the strengths of each one.

Takeaways

- **Classical approach**: the best known general-purpose attack is sub-exponential (GNFS), and still extremely costly at modern RSA sizes.

Takeaways

- Classical approach: the best known general-purpose attack is sub-exponential (GNFS), and still extremely costly at modern RSA sizes.
- **Quantum approach**: Shor allows for theoretical polynomial time, but only once large fault-tolerant quantum computers exist (21 is *small*).

Takeaways

- Classical approach: the best known general-purpose attack is sub-exponential (GNFS), and still extremely costly at modern RSA sizes.
- Quantum approach: Shor allows for theoretical polynomial time, but only once large fault-tolerant quantum computers exist (21 is *small*).
- **Security lesson**: public-key systems built on factoring face a structural quantum risk; generic quantum attacks on symmetric cryptography top out at Grover's provably optimal quadratic speedup.

Thank you!

Appendix

Appendix: RSA Key Generation

1. **Choose primes:** Select two distinct, large prime numbers p and q.
2. **Compute modulus:** Calculate $n = p \cdot q$. This defines arithmetic modulo n and is shared publicly.
3. **Compute totient:** Calculate Euler's totient function, $\varphi(n) = (p-1)(q-1)$. This must be kept secret.
4. **Choose public exponent:** Select an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$ (they are coprime).
5. **Compute private exponent:** Determine d as the modular multiplicative inverse of e modulo $\varphi(n)$.

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

The resulting keys

Public Key: (n, e) **Private Key:** (d, p, q)

Appendix: RSA Encryption & Decryption

Assume Alice wants to send a secret plaintext message M to Bob. First, M must be padded and converted to an integer such that $0 \leq M < n$.

Encryption (Using Bob's Public Key)

Alice computes the ciphertext C using Bob's public modulus n and public exponent e:

$$C \equiv M^e \pmod{n}$$

Decryption (Using Bob's Private Key)

Bob recovers the original message M by raising the ciphertext C to his private exponent d:

$$M \equiv C^d \pmod{n}$$

Appendix: RSA Proof of Correctness

Goal: Prove that decrypting the ciphertext returns the original message: $C^d \equiv M \pmod{n}$.

- Substitute C: We want to show $(M^e)^d \equiv M \pmod{n}$, which means $M^{ed} \equiv M \pmod{n}$.
- By the definition of d, we know $ed \equiv 1 \pmod{\varphi(n)}$.
- Therefore, there exists some integer k such that $ed = 1 + k\varphi(n)$.
- Substituting this into our exponent:

$$M^{ed} = M^{1+k\varphi(n)} = M \cdot (M^{\varphi(n)})^k$$

- By **Euler's Theorem**, if $\gcd(M, n) = 1$, then $M^{\varphi(n)} \equiv 1 \pmod{n}$.

$$M \cdot (1)^k \equiv M \cdot 1 \equiv M \pmod{n}$$

Appendix: Wheel factorization

Idea

After removing small primes, many candidate divisors can be skipped automatically. Example: once 2, 3, and 5 have been handled, there is no point testing numbers divisible by any of them.

- The integers coprime to $30 = 2 \cdot 3 \cdot 5$ repeat modulo 30:

$$1, 7, 11, 13, 17, 19, 23, 29.$$

- So after 5, trial division only needs to test candidates in those residue classes.
- Equivalent step pattern: $+6, +4, +2, +4, +2, +4, +6, +2$, then repeat.

What it buys you

Wheel factorization is a constant-factor speedup: it reduces wasted divisibility tests, but it does *not* change the asymptotic complexity of trial division.

Appendix: Why Fermat starts at $\sqrt{n}$

Why start there?

In $n = x^2 - y^2 = (x-y)(x+y)$, a factor pair $p \le q$ corresponds to $x = \frac{p+q}{2}$ and $y = \frac{q-p}{2}$. Since $x^2 = n + y^2 \geq n \implies x \geq \sqrt{n}$, no solution can appear before $\sqrt{n}$.

Why is incrementing safe?

The correct value is $x = \frac{p+q}{2}$. Then $x^2 - n = \left(\frac{q-p}{2}\right)^2$, a perfect square. Starting at $\lceil \sqrt{n} \rceil$ and trying x, $x+1$, $x+2$, ... checks all possible integers in order, so Fermat must eventually hit that value.

Appendix: Smooth numbers

Definition

An integer is *B-smooth* if all of its prime factors are $\leq B$. A **factor base** is simply the list of small primes we allow.

- Example: $540 = 2^2 \cdot 3^3 \cdot 5$ is 5-smooth.
- Smooth numbers become rarer as numbers get larger.
- Many factorization algorithms try to make their target values small, or vary the hidden group size, so that smoothness becomes more likely.

Where this idea appears

Smoothness is the common thread behind Pollard $p-1$, ECM, QS, and GNFS.

Appendix: Why Pollard $p-1$ may fail

Pollard $p-1$

Fermat's little theorem says that for prime p and $\gcd(a,p)=1$, $a^{p-1}\equiv 1 \pmod{p}$.

In Pollard $p-1$, if $p-1$ is B-smooth, then $p-1 \mid M$, so $a^M \equiv 1 \pmod{p}$, but this is only a **sufficient** condition.

- If $p-1 \mid M$, then $a^M \equiv 1 \pmod{p}$, so a gcd can reveal p.
- Pollard $p-1$ may still succeed even when $p-1$ is not B-smooth (lucky choice of base a).
- However, if $p-1$ has a large prime factor, one chosen smooth exponent M may miss it.
- So the method is excellent on the *right* inputs, but unreliable as a general-purpose attack.

Appendix: From Pollard $p-1$ to ECM

Field notation

For a hidden prime factor $p \mid n$, arithmetic modulo p takes place in the finite field

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}.$$

- Pollard $p-1$ works in the multiplicative group $\mathbb{F}_p^\times$, whose size is fixed: $p-1$.
- ECM works with points on an elliptic curve $E(\mathbb{F}_p)$.
- The key advantage: the size $\#E(\mathbb{F}_p)$ changes when the curve changes.

ECM upgrade

Same smoothness idea, but many possible hidden group sizes for the same factor p.

Appendix: Visual intuition for ECM

Elliptic-curve background

Over $\mathbb{F}_p$, an elliptic curve is an equation of the form

$$y^2 = x^3 + Ax + B \pmod{p}$$

together with a rule for "adding" points.

- Over the real numbers, point addition has a nice geometric picture (chords and tangents).
- Over $\mathbb{F}_p$, the same algebraic formulas still define the group law.
- ECM exploits the size of that group, not the geometry itself.
- If $\#E(\mathbb{F}_p)$ is smooth, then $[M]P$ plays the same role as a^M in Pollard $p-1$.

Useful resource

For intuitive visual explanations, see curves.xargs.org and elliptic-curves.art.

Appendix: How ECM reveals a factor

Analogy with Pollard $p-1$

Pollard $p-1$ computes a^M; ECM computes $[M]P = \underbrace{P + P + \cdots + P}_{M \text{ times}}$.

- Choose a random elliptic curve E and a point P modulo n.
- Compute $Q = [M]P$.
- Modulo a hidden prime factor $p \mid n$, if the order of P divides M, then

$$Q \equiv O \pmod{p},$$

  where O is the **identity point** (the point at infinity).
- In projective coordinates, $O = (0 : 1 : 0)$, so Q_z becomes divisible by p.
- Then $g = \gcd(Q_z, n)$ can reveal the non-trivial factor p.

Factor extraction

If $1 < g < n$, then g is a non-trivial factor of n.

Appendix: Why ECM is better than Pollard $p-1$

Pollard $p-1$

Fast when one factor has $p-1$ smooth; otherwise it may fail, and in the worst case it is no better than trial division.

ECM

One hidden factor p, but many possible group sizes $\#E(\mathbb{F}_p)$ as the curve changes.

- One curve = one try.
- More curves = more chances to hit a smooth group size.
- Different curves can be run independently and in parallel.

Appendix: QS — what to keep?

Factorization of $Q(x)$ and x

For $Q(x) = x^2 - n$, we scan values of x near $\sqrt{n}$. We keep an x only when $Q(x)$ factors completely over the factor base.

- Example factor base: $\{2, 3, 5, 7\}$.
- Keep: $Q(x) = 2 \cdot 3^2 \cdot 7$.
- Discard: $Q(x) = 2 \cdot 11 \cdot 13$ (because 11 and 13 are outside the factor base).

Why keep x as well?

Later we will multiply the selected x-values together to build the X in $X^2 \equiv Y^2 \pmod{n}$.

Appendix: QS — what is linear algebra mod 2?

Parity vectors

For each kept factorization, record only whether each prime exponent is odd or even. With factor base $(2, 3, 5, 7)$:

$$2 \cdot 3^2 \cdot 7 \longmapsto (1, 0, 0, 1), \qquad 2^3 \cdot 5 \longmapsto (1, 0, 1, 0).$$

- "Mod 2" means $1 + 1 = 0$: odd + odd = even.
- So we add these vectors using XOR-like arithmetic.
- A subset whose sum is the zero vector means that every prime appears an even total number of times.

Why this matters

If all total exponents are even, then the product of those $Q(x)$ values is a perfect square.

Appendix: QS — from a dependency to a factor

After linear algebra

Suppose we found kept values $x_1, \ldots, x_k$ such that

$$Q(x_1) \cdots Q(x_k) = Y^2.$$

Since $Q(x_i) = x_i^2 - n \equiv x_i^2 \pmod{n}$, we get

$$Y^2 \equiv \left(x_1 \cdots x_k\right)^2 \pmod{n}.$$

- Set $X \equiv x_1 x_2 \cdots x_k \pmod{n}$, so $X^2 \equiv Y^2 \pmod{n}$.
- Try $\gcd(X - Y, n)$. If it is trivial, also try $\gcd(X + Y, n)$.

Why not just $X - Y$?

Because $n \mid (X - Y)(X + Y)$: either factor may contain the useful divisor.

Appendix: QS — toy example

Setup

Factor $n = 1649$. Start at $\lceil \sqrt{1649} \rceil = 41$, and use factor base $\{2, 5, 7\}$.

- $Q(41) = 41^2 - 1649 = 32 = 2^5$ keep: parity vector $(1, 0, 0)$
- $Q(42) = 42^2 - 1649 = 115 = 5 \cdot 23$ discard: 23 is outside the factor base
- $Q(43) = 43^2 - 1649 = 200 = 2^3 \cdot 5^2$ keep: parity vector $(1, 0, 0)$

Since the kept parity vectors are identical, they cancel mod 2. So

$$(41 \cdot 43)^2 \equiv 32 \cdot 200 = 2^8 \cdot 5^2 = 80^2 \pmod{1649}.$$

Reducing $41 \cdot 43 = 1763 \equiv 114 \pmod{1649}$, we get

$$114^2 \equiv 80^2 \pmod{1649}.$$

Therefore

$$\gcd(114 - 80, 1649) = 17, \qquad \gcd(114 + 80, 1649) = 97.$$

Appendix: GNFS — what is a relation?

Setup

Pick an integer m and a polynomial f with $f(m) \equiv 0 \pmod{n}$. Let α be a root of f.

- For each pair (a, b), GNFS forms two quantities:

$$\text{integer side: } a - bm, \qquad \text{algebraic side: } N(a - b\alpha).$$

- A **relation** is a pair (a, b) for which *both* sides factor completely over their chosen factor bases.
- After enough such rows are collected, linear algebra mod 2 finds a subset whose exponent parities are all even.

Endgame

That dependency is exactly what the square-root step needs to produce the congruence $X^2 \equiv Y^2 \pmod{n}$.

Appendix: GNFS — toy example

Setup

Factor $n = 91$. Choose $m = 10$ and $f(x) = x^2 + x - 19$, so $f(10) = 91 \equiv 0 \pmod{91}$.

Here the algebraic-side norm is (with α a root of f)

$$N(a - b\alpha) = a^2 + ab - 19b^2.$$

- For $(a, b) = (-30, 5)$: $a - bm = -80 = -2^4 \cdot 5$, and $N(a - b\alpha) = 275 = 5^2 \cdot 11$.
- For $(a, b) = (5, 1)$: $a - bm = -5$, and $N(a - b\alpha) = 11$.
- Their products, $(-80)(-5) = 20^2$ and $275 \cdot 11 = 55^2$, have even exponent parities.

Why this is useful

This is exactly the kind of dependency GNFS wants. In a real run, the square-root step turns many such dependencies into $X^2 \equiv Y^2 \pmod{n}$, and then a gcd can reveal a factor.

Appendix: Why GNFS beats QS on large inputs

Intuition

GNFS has a more flexible search space than QS, so it can find useful smooth relations faster when n is very large.

- QS looks at one family of values: $x^2 - n$.
- GNFS engineers two smoother worlds using a tailored polynomial.
- The price is more setup and more complicated bookkeeping.

Practical trade-off

For medium-sized inputs QS can be simpler; for very large general integers, GNFS wins.

Appendix: RSA primes — then and now

Historical advice

Earlier RSA guidance often recommended *strong primes*: choose p and q so that $p-1$, $p+1$, $q-1$, and $q+1$ each have a large prime factor.

- Motivation: make special-purpose attacks such as Pollard $p-1$ less effective.
- Limitation: this does not stop general attacks such as QS or GNFS.
- ECM also weakens the original motivation for obsessing over $p \pm 1$.

Modern takeaway

The main defenses are large enough modulus size, sound random prime generation, and avoiding obviously bad structure.

Appendix: Why the QFT helps

After modular exponentiation

Once the repeated structure of $f(x) = a^x \bmod n$ has been encoded, the relevant part of the quantum state behaves like

$$|x_0\rangle + |x_0 + r\rangle + |x_0 + 2r\rangle + \cdots$$

which is a regularly spaced pattern with spacing r.

A Fourier transform turns regular spacing in the input into peaks in frequency space.

Big picture

The QFT is not outputting all amplitudes. It reshapes the probabilities so the hidden period becomes visible in a small number of samples.

Appendix: Why order finding gives factors

Core algebra

If r is the order of a mod n, then $a^r \equiv 1 \pmod{n}$. When r is even,

$$a^r - 1 = \left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right).$$

- Because $n \mid a^r - 1$, the modulus n divides that product.
- If $a^{\frac{r}{2}} \not\equiv \pm 1 \pmod{n}$, then neither factor is trivial modulo n.
- So $\gcd(a^{\frac{r}{2}} - 1, n)$ or $\gcd(a^{\frac{r}{2}} + 1, n)$ often reveals a non-trivial factor.

Appendix: Shor — toy example

Factor $n = 15$ using $a = 2$

The powers of 2 modulo 15 are

$$2,\ 4,\ 8,\ 1,\ 2, \ldots$$

so the order is $r = 4$.

- r is even.
- $2^{\frac{r}{2}} = 2^2 = 4 \not\equiv \pm 1 \pmod{15}$.
- Therefore Shor's classical post-processing gives

$$\gcd(2^2 - 1, 15) = 3, \qquad \gcd(2^2 + 1, 15) = 5.$$

What the quantum part does

For large n, the hard step is finding the order r. Shor uses a quantum period-finding routine to get that r efficiently; the gcd step above is classical.

Appendix: Shor vs Grover — asymmetric vs symmetric

Public-key cryptography

Shor's algorithm attacks the algebraic structure behind RSA, Diffie–Hellman, and elliptic-curve cryptography, so it is a **game changer** for those systems.

Symmetric cryptography

The main generic quantum attack is **Grover's algorithm**, which gives only a quadratic speedup for brute-force search.

- A k-bit symmetric key falls from about 2^k work to about $2^{\frac{k}{2}}$ quantum work.
- For black-box unstructured search, that quadratic improvement is **provably optimal**: there is no generic quantum algorithm that beats Grover by more than a constant factor.
- So doubling key sizes is a standard rule of thumb for recovering the security margin.