hexagon:2611.00115v2

CLASS NUMBERS AND LATTICE-POINT COUNTS IN TIME $2^{O(n/\sqrt{\log n})}$, ASSUMING THE NEW #SAT ALGORITHMS

VAIBHAV TULSYAN

Abstract. Basic counting problems of computational number theory have proven deterministic bounds that are fixed powers of the input. The class number $h(D)$ of an imaginary quadratic discriminant takes $|D|^{1/2+o(1)}$ steps unconditionally and $|D|^{1/5+o(1)}$ under the generalised Riemann hypothesis (GRH); exact divisor sums $\sum_{m\le x}\tau(m)$ take $x^{1/3+o(1)}$. However, none of these bounds is of the form $X^{o(1)}$, even conditionally. We present deterministic algorithms in time $2^{O(n/\sqrt{\log n})}$, where n is the input bit length, for the number of reduced forms of discriminant $-N$ (for fundamental discriminants, $h(-N)$) and for $\sum_{m\le x}\tau(m)$, $\sum_{m\le x}\sigma(m)$ and the Gauss circle count. Our approach writes each answer as a count of solutions of equations with one or two products and evaluates it with Anthropic's counting engine through one network library with $O(n)$ shared bits. The same library gives the least square root modulo any m and the least quadratic non-residue modulo a prime. These are not new bounds (OpenAI's results on the quasi-Riemann hypothesis and on deterministic polynomial factorisation over $\mathbb{F}_p$ give polynomial time modulo a prime), but they are machine-checked. We prove the seven resulting theorems in Lean 4 (for example `hForm_TM_rate`; axioms `propext`, `Classical.choice`, `Quot.sound`): a finite-alphabet Mathlib Turing machine outputs the answer within $\lfloor C\cdot 2^{Cn/\sqrt{\log_2(n+2)}}\rfloor$ steps on every input. Lean proves the count of Gauss's reduced forms; its equality with the ideal class number is classical and not formalised. The constants are astronomical, the machines use exponential space, and every result rests on Anthropic's Lean development of its claimed refutation of the Exponential Time Hypothesis, which is under review. Version 2 corrects the claims of version 1 about non-residues and square roots.

1. Introduction

Computing the class number $h(D)$ of an imaginary quadratic discriminant $D=-N$ is one of the oldest problems of computational number theory. Gauss's reduction theory gives the simplest algorithm: $h(D)$ is the number of reduced forms $ax^2+bxy+cy^2$ of discriminant D, and every such form has $a\le\sqrt{N/3}$, so listing them takes $N^{1/2+o(1)}$ steps. To our knowledge this is still the best proven deterministic bound without hypotheses [18, §5]. Under GRH, Shanks's baby-step giant-step method, analysed by Lenstra [17] and Schoof [27, §3], takes $N^{1/5+o(1)}$, and the subexponential algorithm of Hafner and McCurley [13] is randomised and also rests on GRH [18, §5]. Exact divisor sums $\sum_{m\le x}\tau(m)$, a lattice-point count under a hyperbola, take $x^{1/3+o(1)}$ steps [31, 32].

In October 2026 OpenAI [24] and Anthropic [2] (below [O] and [A]) each claimed a deterministic algorithm that counts the satisfying assignments of a k-CNF in subexponential time, refuting the Exponential Time Hypothesis; both claims come with Lean 4 developments and are under review. Rudich and Rousseau [26] used the engines to factor in $N^{o(1)}$, Shunia [30] proved the rate $2^{O(n/\sqrt{\log n})}$ on paper, and our factoring note [35] proved it in Lean on [A]'s engine. OpenAI's catalogue also settles two neighbouring problems. Its proof of the quasi-Riemann hypothesis [22] makes the least quadratic non-residue modulo a prime polylogarithmic, so it is found in deterministic polynomial time, and its deterministic factorisation of polynomials over $\mathbb{F}_p$ [23] takes square roots modulo a prime in polynomial time (Section 10). However, the bounds above for class numbers and divisor sums are all fixed powers of the input, even under GRH for $h(D)$. We are unaware of any derivation of better bounds from the new counting engines or from these results; our searches are described in Section 10.

Date: November 2026 (version 2, 5 November 2026).

2020 *Mathematics Subject Classification.* 11Y16, 11Y40, 11A15, 68Q25, 03B35.

Key words and phrases. Class number, binary quadratic forms, divisor summatory function, Gauss circle problem, exact counting, square roots modulo m, quadratic non-residue, Exponential Time Hypothesis, Lean 4, formal verification.

1.1. **Counting problems with one or two products.** We consider problems whose answer is, or is found from, the number of integer solutions of a fixed system of polynomial equations and inequalities in $O(1)$ unknowns of $O(n)$ bits, in which each equation has at most two products. The class number is such a count: $h(-N)$ counts the triples (a, b, c) with $b^2 + N = 4ac$ and $|b| \le a \le c$, up to a tie-break.

Example 1.1 (Running example)**.** Take $N = 23$. The reduced forms of discriminant -23 are $(a, b, c) = (1, 1, 6)$, $(2, 1, 3)$ and $(2, -1, 3)$, so $h(-23) = 3$. Our machine asks two counts. Query 1 counts the triples of naturals with $b^2 + 23 = 4ac$ and $b \le a \le c$; they are $(1, 1, 6)$ and $(2, 1, 3)$. Query 2 counts the triples with $b^2 + 23 = 4ac$ and $0 < b < a < c$; it is $(2, 1, 3)$ alone, which stands for the form $(2, -1, 3)$. The answer is $2 + 1 = 3$.

1.2. **The naive encodings fail; the principle.** At first one might reduce each count to #SAT and apply the engines' headline theorems. However, Tseitin's transformation of a multiplication circuit has $\Omega(n \log n)$ clauses for every known circuit, and [A]'s clause bound then gives $2^{O(n\sqrt{\log n})}$, worse than the classical bounds. Our factoring note [35] showed how to avoid this for one equation $pq = N$: [A]'s network model charges $2^{e \cdot U}$ for U bits shared between nodes, with $e = O(1/\sqrt{\log n})$, and a two-dimensional Schönhage–Strassen transform multiplies with $O(n)$ shared bits. Writing a new network for each problem would cost thousands of Lean lines each. Instead, we rely on a different principle: *a fixed system of equations with few products has a network assembled from three kinds of block (products, carry chains and comparison chains), each cutting $O(n)$ bits, so one library proved once serves every problem.* The per-problem work is a data value, the description, together with a counting identity in pure Mathlib.

1.3. **What we prove.** In this paper we build that library, which we call ArithNet, write six descriptions, and prove in Lean that one Turing machine per problem computes the answer. All six theorems have the same form.

Theorem 1.2 (Section 8; machine-checked)**.** *For each of the following functions F there are a Turing machine M in Mathlib's `TM2` model, with finite alphabets on all stacks, and a constant $C \in \mathbb{N}$, such that on every input word w of length n (as specified) M halts within $\lfloor C \cdot 2^{Cn/\sqrt{\log_2(n+2)}} \rfloor$ steps with output `encodeNat` F:*

(i) `hForm_TM_rate`*: $w =$* `encodeNat` *N, $F = h_{\mathrm{form}}(-N)$, the number of reduced forms of discriminant $-N$ (Theorem 5.1);*

(ii) `hFormPrim_TM_rate`*: the same machine, for $-N$ a fundamental discriminant, outputs the number of primitive reduced forms, the form class number $h(-N)$;*

(iii) `sqrtMod_TM_rate`*: $w =$* `encodePair` *$m\ a$, F the least $y < m$ with $y^2 \equiv a \pmod m$, and m if there is none;*

(iv) `lqnr_TM_rate`*: $w =$* `encodeNat` *p with p prime, F the least quadratic non-residue modulo p (p if there is none, i.e. $p = 2$); on every other p the machine halts within the same bound with some $y \le p$;*

(v) `divisorSum_TM_rate`*,* `sigmaSum_TM_rate`*,* `circle_TM_rate`*: $w =$* `encodeNat` *x, $F = \sum_{m \le x} \tau(m)$, $\sum_{m \le x} \sigma(m)$, and $\#\{(u, v) \in \mathbb{Z}^2 : u^2 + v^2 \le x\}$.*

Since $n = \log_2 N + O(1)$, the bound in (i) is $N^{O(1/\sqrt{\log \log N})} = |D|^{o(1)}$, against $|D|^{1/2+o(1)}$ unconditionally and $|D|^{1/5+o(1)}$ under GRH. In (v) it is $x^{o(1)}$, against $x^{1/3+o(1)}$. To our knowledge, no deterministic algorithm with a bound of the form $X^{o(1)}$ was known for (i), (ii) or (v), even under GRH.

Parts (iii) and (iv) are not new bounds. Modulo a prime p, the least non-residue and the square roots of a are found in deterministic polynomial time, by OpenAI's proof of the quasi-Riemann hypothesis [22] and by OpenAI's polynomial factorisation over $\mathbb{F}_p$ [23]. For composite m, the least root can also be found in time $2^{O(n/\sqrt{\log n})}$ by factoring m first; we derive this on paper in Section 10. We keep (iii) and (iv) because they come from the same library at no extra cost and are machine-checked. They are an independent route, not an improvement.

1.4. **Contributions.** Our contributions are:

- We introduce ArithNet, a library that turns a *description* (variables, products of a variable by a variable or a constant, linear equations with coefficient mass at most 32, comparisons, bit patterns) into a parsimonious network, and prove in Lean that its value is the number of solutions, that its shared bits total $O(n)$ and that each of its g nodes holds $O(n/g + n^{3/4})$ bits (Section 3).

- We present generic counted prefix-search and one/two-query machines that turn any polynomial-time description reader into a Turing machine running in $2^{O(n/\sqrt{\log n})}$ steps (Section 4).
- We give the six descriptions and their counting identities, including the two-query split of reduced forms and an exact formula for non-residues with a prefix from a count of squares (Section 5).
- We prove the six theorems of Theorem 1.2 in Lean, and state precisely what the class-number theorem counts and what remains classical (Sections 6, 8 and 9).

The ingredients (reduction theory, prefix search, the transform network) are classical or from our factoring note; what we add is the library, its proofs, and the descriptions.

1.5. **Status of each claim.** We separate four kinds of claim. *Machine-checked in Lean*: the seven theorems of Theorem 1.2, with the axioms `propext`, `Classical.choice` and `Quot.sound` only, and every lemma whose Lean name we give. *Proved on paper only*: the equality of the form class number with the ideal class number of $\mathbb{Q}(\sqrt{-N})$ (classical; Dirichlet and Dedekind, see [10, Thm. 7.7]); the decomposition of h_{form} for non-fundamental discriminants in Theorem 6.1; the discussion of real quadratic fields; the comparisons with earlier bounds; and the derivations from [22, 23] and from factoring in Section 10. *Assumed*: nothing beyond the three axioms; [A]'s results enter as Lean theorems of its development, which the kernel checks with ours. *Checked numerically*: the running examples and the decomposition of Theorem 6.1 for $N < 400$, by a short exact computation that is not part of the proof; some of the small values, such as the split $2 + 1$ of Theorem 1.1 and $\mathrm{lqnr}(7) = 3$ of Theorem 5.3, are also checked in Lean by `decide`.

Outline. Section 2 recalls [A]'s counting theorem. Section 3 builds ArithNet and Section 4 the machines. Section 5 gives the descriptions, and Section 6 says what the class-number theorem counts. Sections 7 to 9 cover the exponent, the statements and the Lean development, and Sections 10 and 11 related work and limitations.

Changes in v2. Version 1 (hexagon:2611.00115v1) said that Burgess's bound is the best unconditional bound for the least quadratic non-residue, that polynomial time for it needs GRH, and that no deterministic $X^{o(1)}$ algorithm was known for square roots modulo m. These statements were out of date. Two entries of OpenAI's catalogue, which predate version 1, supersede them: entry 003, *The quasi-Riemann hypothesis* [22] (preprint dated 30 September 2026), and entry 142, *Deterministic polynomial factorization over prime fields* [23] (preprint dated 4 October 2026). The corrections below credit them. In version 2 we

- retitle the note, which now leads with class numbers and lattice-point counts;
- correct every statement about non-residues and square roots in the abstract, the introduction, the related work and the limitations, and present (iii) and (iv) of Theorem 1.2 as a machine-checked route, not as the best bounds;
- add a derivation on paper of the least square root modulo composite m from deterministic factoring (Section 10);
- cite Frei, Sofos and Widmer [12] and Vega [37], and report the repeated novelty search.

No Lean theorem, statement or proof changed. The Lean development is the one of version 1.

2. The engine counts accepted labellings

In this section we recall the one theorem of [A] that we compose with, in the form used by our factoring note [35, §2]. We use [A]'s Lean development at commit `16a5ce4` (`anthropics/formal-math`, folder `eth`, Lean v4.33.1, Mathlib `0df444a`).

A *network on* $[g]$ *with parameters* P, B, τ has, for every node a, settings of at most P bits and a weight W_a, and for every pair of nodes an interaction that factors through $c_{a,b}$ shared bits, with $\sum_b c_{a,b} \le P$; its value is $Z = \sum_z \prod_a W_a(z_a) \prod_{\{a,b\}} \Psi_{a,b}$ and its total is $U = \sum_{a \ne b} c_{a,b}$. We use only *label systems*: items (wires) placed on pairs of nodes, each carrying a W-bit label seen by both nodes, with W_a a 0/1 table. The value is then the number of labellings accepted by every table.

Imported Theorem 2.1 ([A], Lemma 8(i) and Theorem A(b); see [35, Imported Thm. 2.2])**.** *Let $K = K_\ell \ge 2$ be a level of* [A]*'s schedule, $g = 3K^8$, and $e_\ell \le (3x_0 + 6)/\sqrt{\log_2 K}$ for an absolute x_0. A network on $[g]$ with*

parameters P, B, τ and total U can be evaluated within $\big(K^g + 2^{e_\ell K^2 L}(K(L+B'))^{C_0 K^2} 2^{2gP/K}\tau\big)(g(P+B))^{C_1}$ operations, where $L = \lceil (U + gP/K)/K^2 \rceil$, $B' = \lceil 2gP/K \rceil + g^2 B$ and C_0, C_1 are absolute.

In Lean this is `ExplicitCost.groupingExplicit` together with the clique machine of the counted recursion (`CountedRoute.onTheCountedRecursion`); [A]'s host lemma `CountKSat.exists_tmW` turns the step-by-step machine into a Mathlib Turing machine with polynomial overhead. So a network family with $U = O(n)$ and $gP = O(n)$ is evaluated in $2^{O(n/\sqrt{\log n})}$ steps at the level $K^{40} \lesssim n$ of our factoring note.

3. ArithNet: one network library for few-product systems

In this section we define descriptions and the count they denote, build the network of a description, and state its value and parameter theorems. The definitions are pure Mathlib (module `ArithDesc`); the network is in the bridge modules `ArithParams` and `ArithNet*`.

3.1. **Descriptions.** A description is plain data:

```
inductive Operand | var (v : ℕ) | const (limbs : List ℕ)
structure ProdD where u : ℕ; w : Operand           -- the product x_u * w
inductive Src | prod (p : ℕ) | lin (v : ℕ) | const (limbs : List ℕ)
structure Term where coeff : ℕ; src : Src
structure ChainD where lhs : List Term; rhs : List Term
structure CmpD where v : ℕ; rhs : Operand; rel : CmpRel   -- lt | le | eq | ge | gt
structure Desc where nV : ℕ; pat : List (List (Option Bool));
  prods : List ProdD; chains : List ChainD; cmps : List CmpD
```

Constants are lists of b-bit limbs, with value $\sum_i l_i 2^{bi}$. A chain asserts $\sum_{\text{lhs}} \text{coeff} \cdot \text{value} = \sum_{\text{rhs}} \text{coeff} \cdot \text{value}$, where the value of a term is that of a product, a variable or a constant. A pattern fixes some bits of a variable (`none` is a free bit). `Desc.Sat` b D x holds when every pattern matches, every comparison holds and every chain balances, and the count of D at size n is

$$\text{count}_n(D) = \#\big\{x \in [0, 2^{n_B})^{\text{nV}} : \texttt{Desc.Sat}\ b\ D\ x\big\},$$

with b the limb width and $n_B = \ell b > n$ the box width of [35, §4]. A description is well formed (`Desc.WFB`) when its indices are in range, its constants have at most ℓ limbs below 2^b, its patterns have length at most n_B, each side of each chain has coefficient mass at most 32, and no product occurs in two chains.

3.2. **Wires, vertices and checks.** Every wire carries a label of $W' = J + 8$ bits, where $J = 2^{\lfloor \nu/2 \rfloor + 7}$, $\nu = \lfloor \log_2(n+2) \rfloor$, is the transform modulus exponent of [35].

- *Variables.* Each variable has ℓ limb wires, one copy per chain in which it occurs linearly, and one copy per product side in which it is an operand.
- *Products.* Each product $x_u \cdot w$ gets its own two-dimensional Schönhage–Strassen pipeline over $\mathbb{Z}/(2^J + 1)$: row and column transforms of both operands, pointwise products, inverse columns and inverse rows, ending in the 2ℓ coefficients of the product (the stages of [35, §4]). A constant operand enters through limb wires whose values the cells fix.
- *Chains: signed carries.* Chain c has a carry vertex at each position k, which reads the coefficient digits $d_k^{\text{L}}, d_k^{\text{R}}$ of its two sides (sums of coefficient times product coefficient, limb or constant limb) and the carry wires, and checks the single equation

$$R_{k+1} \cdot 2^b + d_k^{\text{R}} + \text{OFF} = d_k^{\text{L}} + R_k + \text{OFF} \cdot 2^b, \qquad R_0 = R_K = \text{OFF} = 2^{J+6}. \tag{3.1}$$

  The offset makes the signed carry $R_k - \text{OFF}$ a natural number. By `carryChk_iff`, valid carries exist if and only if $\sum_k d_k^{\text{L}} 2^{bk} = \sum_k d_k^{\text{R}} 2^{bk}$, and then they are unique (`carryChk_unique`) and lie in $(0, 2^{J+7})$ (`carryChk_range`). The mass bound 32 keeps every digit below $\text{OFF} \cdot (2^b - 1)$ (`side_le_OFF`).
- *Comparisons: state chains.* A comparison $x_v \bowtie r$ has state wires $s_j \in \{0, 1, 2\}$ (less, equal, greater) passed from limb $\ell - 1$ down to limb 0; the cell at limb i updates the state lexicographically, and cell 0 tests the relation (`cmpChain_iff`, `cmpChain_unique`).
- *Cells.* There is one cell per limb position i; it checks the range and pattern of limb i of every variable, that all copies agree, and the comparison transitions at i.

Every wire has exactly two ends. Index bookkeeping goes through a generic mixed-radix block layout (module `Blocks`) instead of hand-written case splits.

3.3. **Value and parameters.** For a description with nV variables, nP products, nC chains and nQ comparisons let $c_A(D) = 64(\mathrm{nV} + \mathrm{nP} + \mathrm{nQ} + 1)(\mathrm{nC} + 1)$ and $d(n) = 2^{\lfloor 3\nu/4 \rfloor + 11} = O(n^{3/4})$.

Theorem 3.1 (Parsimony; `arithNet_card`, `arithNet_value`)**.** *Let $g \geq 2$ and let D be well formed at size n. The labellings accepted by every check are in bijection with the solutions counted by $\mathrm{count}_n(D)$, so the value of the network is $\mathrm{count}_n(D)$.*

Sketch. Forward (`aRun_all_iff`): a solution x labels every wire with its canonical value $\mathrm{run}(x)$; the transform checks pass by construction, the pipelines compute the products (`pipeline_eq_mul` of [35]), so the carry checks pass if and only if every chain balances, and the cells pass if and only if the patterns match and the comparisons hold. *Backward* (`aEq_run`): an accepted labelling equals run of the values read off the limb wires, stage by stage, with the carries and states forced by uniqueness. □

Proposition 3.2 (`arithSys_U_le`, `arithSys_is`)**.** *For $g \geq 2$ the network has total $U \leq 2^{11} c_A(D)(n+2)$, and it is a network with $P = 2^{12} c_A(D)\big((n+2)/g + d(n)\big)$ and $B = 1$.*

The network is in [A]'s `Net` format through our label systems [34], and its tables are answered by polynomial-time procedures (`tabAF_polyT`, `arithNet_isNet`); no table is written out. For the descriptions of Section 5, $c_A(D) \leq 1152$.

4. Counted searches and queries

In this section we describe how counts become answers. The interface is a *counting package* `CntPackage` c_0: for every input word w and query word π a network whose value is $Q(w,\pi) \in \mathbb{N}$, with $U \leq c_0(|w|+2)$, $P \leq c_0((|w|+2)/g + d(|w|))$, polynomial reader and procedures, and a number of rounds $R(w) \leq |w|$. `ArithSpec.pkg_exists` builds a package with $Q(w,\pi) = \mathrm{count}_{n(w)}(D(w,\pi))$ from any polynomial-time map $(w,\pi) \mapsto D(w,\pi)$ to well-formed descriptions with $c_A \leq c_D$, with $c_0 = 2^{12} c_D$.
Prefix search. A `SearchSpec` has a test for small inputs, an answer for them, a step and a finish, all polynomial-time. The machine `GR.searchM` starts from the empty word, and for $R(w)$ rounds replaces π by $\mathrm{step}(w,\pi,Q(w,\pi0))$; then it outputs $\mathrm{fin}(w,\pi)$. With the step "append 0 if the count is positive, else 1", the word found is the least element of the counted set (`searchWith_eq_minOr`: if the oracle is positive on $\pi0$ exactly when some element of a set $S \subseteq [0,2^n)$ has prefix $\pi0$, the result has value $\min S$, or $2^n - 1$ if $S = \emptyset$).

Theorem 4.1 (`GR.searchM_TM`, `GR.queryM_TM`)**.** *For every counting package and every search specification (resp. query specification with two query words $q_1(w), q_2(w)$ and a finish $\mathrm{fin}(w, Q(w,q_1), Q(w,q_2))$) whose parts are polynomial-time and whose outputs have length $O(|w|)$, there are a finite-alphabet Turing machine and C that output the specified answer on every w within $\lfloor C \cdot 2^{C|w|/\sqrt{\log_2(|w|+2)}} \rfloor$ steps.*

Both are proved through one generic tail, `GR.TM_of_runs`, on top of the round bound `FR.round_bound` and the level selection of the factoring note.

5. Six descriptions

In this section we give each description and the pure counting identity that connects its count to the answer. All identities are in pure Mathlib (modules `ClassForms`, `QRCount`, `Lattice`, `ArithInstances`).

5.1. **Reduced forms with two queries.** We follow Cox's convention [10, §2]: a positive definite form (a,b,c) of discriminant $D = b^2 - 4ac < 0$ is *reduced* if $|b| \leq a \leq c$, and $b \geq 0$ whenever $|b| = a$ or $a = c$.

Definition 5.1 (`hForm`)**.** For $D \in \mathbb{Z}$ and $N = |D|$, $h_{\mathrm{form}}(D)$ is the number of $(b,a,c) \in [-N,N] \times [0,N] \times [0,N]$ with $b^2 - 4ac = D$, $|b| \leq a \leq c$, and $(|b| = a \vee a = c) \Rightarrow b \geq 0$.

For $D < 0$ the box imposes nothing (`mem_redForms_iff`): a reduced form has $c \leq N$. The description of query j has variables a, b, c, products $b \cdot b$ and $a \cdot c$, the chain $1 \cdot (b \cdot b) + 1 \cdot N = 4 \cdot (a \cdot c)$ with N a constant, and the comparisons

$$\text{query 1: } b \leq a,\ a \leq c; \qquad \text{query 2: } b > 0,\ b < a,\ a < c.$$

Here b ranges over naturals. Query 1 counts the reduced forms with $b \geq 0$; in them the tie-break is automatic. A reduced form with $b < 0$ has $|b| < a < c$, since equality would force $b \geq 0$, so $b \mapsto -b$ maps these forms bijectively onto the solutions of query 2.

Lemma 5.2 (`hForm_split`, `hDesc_count`). *For $0 < N < B$, $h_{\mathrm{form}}(-N) = h_{Q1}(B,N) + h_{Q2}(B,N)$, where $h_{Qj}(B,N)$ is the number of solutions of query j in $[0,B)^3$. The two descriptions count exactly these.*

The machine (`specE`) answers 1 on $N = 0$ (where $h_{\mathrm{form}}(0) = 1$ counts the degenerate triple $(0,0,0)$), and otherwise asks the two queries and adds. In Theorem 1.1 the counts are 2 and 1.

5.2. **Square roots modulo any m.** The input is `encodePair` m a, two little-endian words padded to a common length n. The reader takes m and $a \bmod m$; inputs with $m < 2$ are answered directly (0). The description has variables y, k, x, products $y \cdot y$ and $k \cdot m$, the chain $y \cdot y = x + k \cdot m$, the comparisons $y < m$, $x < m$, $x = a \bmod m$, and the pattern "y has the prefix $\pi 0$". Since $y < m$ and $x < m$ determine k and x, the count is the number of roots $y < m$ of a with that prefix (`S3A_eq`). The search keeps 0 when the count is positive; at the end, $y = \mathrm{val}(\pi)$ is output if it is a root and m otherwise, which covers the case of no roots, where the search returns $2^n - 1 \geq m$. For $m = 7$, $a = 2$ (roots $3, 4$; $n = 3$) the counts of 0, 00, 010 are $1, 0, 0$, and the result is $011 = 3$.

5.3. **The least non-residue from counts of squares.** For a prime p we need, for each prefix, whether some non-residue $x < p$ has it. The same description with the pattern on x instead of y, and no condition $x = a$, counts the pairs (y, x) with $y^2 \equiv x$ and x in the prefix block. For an odd prime, 0 has one square root and every nonzero square has two, so this count is $S(\pi) = 2\,Q(\pi) + z_0(\pi)$, where $Q(\pi)$ counts the nonzero squares in the block and $z_0(\pi) = 1$ if the block contains 0 (`S3B_eq_Qc`). The block has $r(\pi)$ elements below p, so the number of non-residues in it is

$$\mathrm{NR}(\pi) = r(\pi) - z_0(\pi) - \big(S(\pi) - z_0(\pi)\big)/2 \qquad (\texttt{NR_eq}), \tag{5.1}$$

which the step computes from the count in polynomial time; it keeps 0 when $\mathrm{NR}(\pi 0) > 0$.

Example 5.3 ($p = 7$)**.** The squares modulo 7 are $0, 1, 2, 4$, so the least non-residue is $3 = 011_2$. The search runs as follows.

query $\pi 0$	0	00	010
block	$\{0,1,2,3\}$	$\{0,1\}$	$\{2\}$
$S(\pi 0)$	5	3	2
$r,\ z_0$	$4,\ 1$	$2,\ 1$	$1,\ 0$
$\mathrm{NR}(\pi 0)$	1	0	0
new π	0	01	011

For $p \leq 2$ the machine answers p (0 and 1 are squares modulo 2). For p composite the formula may be wrong, but the finish outputs some $y \leq p$ (`specB_answer_le`). The pure characterisation is `lqnr_spec`: for a prime $p > 2$, $\mathrm{lqnr}(p) < p$, it is not a square in `ZMod` p, and every smaller residue is a square.

5.4. **Divisor sums and the circle.** These need one query each. The divisor sum is $\sum_{m \leq x} \tau(m) = \#\{(a,b,s) : ab + s = x,\ a, b > 0\}$ (`divQ_eq`); $\sum_{m \leq x} \sigma(m) = \#\{(a,b,c,s) : ab + s = x,\ b > 0,\ 1 \leq c \leq a\}$ (`sigQ_eq`); and the circle count is $4\,\#\{(a,b,s) : a^2 + b^2 + s = x,\ a > 0\} + 1$, by rotating the four quadrants (`circ_eq`). Each is one product (two for the circle) and one chain with the constant x.

Example 5.4. For $x = 10$: $\sum_{m \leq 10} \tau(m) = 27$, $\sum_{m \leq 10} \sigma(m) = 87$, and the circle count is $4 \cdot 9 + 1 = 37$.

6. What the class-number theorem counts

In this section we say exactly what Lean proves about class numbers and what is classical.
Forms, not ideals. Lean proves that the machine outputs $h_{\mathrm{form}}(-N)$, the number of Gauss reduced forms of discriminant $-N$, and, for a fundamental discriminant, the number of *primitive* reduced forms (`hForm_eq_card_primitive`, from `primitive_of_fund`: no prime divides all of a, b, c when $b^2 - 4ac$ is fundamental). That number is the form class number $h(D)$, the order of Gauss's form class group. That $h(D)$ equals the ideal class number of the imaginary quadratic field $\mathbb{Q}(\sqrt{D})$ for fundamental D, and of the order of discriminant D in general, is the classical correspondence of Dirichlet and Dedekind [10, Thm. 7.7], [5]. It is not in Mathlib at the commit we use: Mathlib has `NumberField.classNumber`, but no binary quadratic forms and no forms–ideals correspondence. So the link to `NumberField.classNumber` is proved on paper only. We define fundamental discriminants locally (`IsFundDisc`: $D \equiv 1 \pmod 4$ squarefree, or $D = 4m$ with $m \equiv 2, 3 \pmod 4$ squarefree), since Mathlib at that commit has no such definition.

Remark 6.1 (Non-fundamental discriminants). If $N \equiv 1, 2 \pmod 4$ there are no forms, and $h_{\mathrm{form}}(-N) = 0$. Otherwise $h_{\mathrm{form}}(-N)$ also counts imprimitive forms: every form of discriminant D is f times a primitive form of discriminant D/f^2, and scaling preserves reduction, so

$$h_{\mathrm{form}}(D) = \sum_{f^2 \mid D,\ D/f^2 \equiv 0,1 \ (4)} h(D/f^2), \qquad h(D) = \sum_{f^2 \mid D,\ D/f^2 \equiv 0,1 \ (4)} \mu(f)\, h_{\mathrm{form}}(D/f^2).$$

For example $h_{\mathrm{form}}(-12) = 2$ counts $(1, 0, 3)$ and $(2, 2, 2) = 2 \cdot (1, 1, 1)$, while $h(-12) = 1$. The inversion recovers $h(D)$ from at most $\tau(N)$ runs of the machine once the square divisors of N are known, and these can be found by factoring N, also in time $2^{O(n/\sqrt{\log n})}$ [35]. This is on paper only (we checked the decomposition numerically for $N < 400$); the Lean theorem for non-fundamental D is about h_{form}.

Real quadratic fields are not covered. For $D > 0$ the reduced indefinite forms fall into cycles under the reduction operator, and each class corresponds to a whole cycle, whose length is tied to the regulator. Counting reduced forms therefore gives neither h nor hR: one must count cycles, or weight each form by its distance to its successor, which yields hR [27, §4–5]. The indicator "this form is the first of its cycle" is a condition over the whole cycle, not a local check on $O(n)$ bits, and separating h from the regulator R is a baby-step giant-step computation, not a count. We know of no description with few products for $h(D)$, $D > 0$.

7. The exponent

In this section we check that the descriptions give the rate. Every description of Section 5 has at most four variables, two products, one chain and three comparisons, so $c_A(D) \le 1152 = c_D$ for every input. By Theorem 3.2, $U \le 2^{11} \cdot 1152\,(n+2)$ and $gP \le 2^{12} \cdot 1152\,((n+2) + g\,d(n))$. At the level K with $K^{40} \le K_0^{40}(n+2)$ of [35, §5] we have $g\,d(n) = O(n)$, so $U, gP = O(n)$, and Theorem 2.1 gives one count in $2^{O(n/\sqrt{\log n})}$ steps. The searches ask at most n counts and the class-number and lattice machines one or two; the overhead of the host lemma is polynomial. In Lean this is `FR.round_bound` with $c_0 = 2^{12} \cdot 1152$, inside Theorem 4.1. For (A) the network lives at $n = |w|/2$ while the bounds are stated in $|w|$; this uses that d is monotone (`dg_mono`).

8. What a reader must trust

A reader who wants to trust Theorem 1.2 must check the following definitions and nothing else of ours. They are in the pure modules `ClassForms`, `QRCount` and `Encode`; we render Unicode symbols as mathematics (`x.1` and `x.2` are the projections of a pair).

```
def hForm (D : ℤ) : ℕ :=
  ((Finset.Icc (-(D.natAbs : ℤ)) D.natAbs ×ˢ Finset.Icc 0 (D.natAbs : ℤ) ×ˢ
      Finset.Icc 0 (D.natAbs : ℤ)).filter
    fun t : ℤ × ℤ × ℤ => t.1 * t.1 - 4 * t.2.1 * t.2.2 = D ∧ |t.1| ≤ t.2.1 ∧
      t.2.1 ≤ t.2.2 ∧
      ((|t.1| = t.2.1 ∨ t.2.1 = t.2.2) → 0 ≤ t.1)).card
def redForms (D : ℤ) : Finset (ℤ × ℤ × ℤ) := -- the same finset as in hForm
def IsFundDisc (D : ℤ) : Prop :=
  (D % 4 = 1 ∧ Squarefree D) ∨
    (D % 4 = 0 ∧ (D / 4 % 4 = 2 ∨ D / 4 % 4 = 3) ∧ Squarefree (D / 4))
def primRedForms (D : ℤ) : Finset (ℤ × ℤ × ℤ) :=
  (redForms D).filter fun t => Int.gcd t.2.1 (Int.gcd t.1 t.2.2) = 1
def minRoot (m a : ℕ) : ℕ :=
  Nat.find (⟨m, Or.inl rfl⟩ : ∃ y, y = m ∨ (y < m ∧ y * y % m = a % m))
def lqnr (p : ℕ) : ℕ :=
  Nat.find (⟨p, Or.inl rfl⟩ : ∃ x, x = p ∨ (x < p ∧ ∀ y < p, y * y % p ≠ x))
def padR (n : ℕ) (l : List Bool) : List Bool := l ++ List.replicate (n - l.length) false
def encodePair (m a : ℕ) : List Bool :=
  padR (max (encodeNat m).length (encodeNat a).length) (encodeNat m) ++
    padR (max (encodeNat m).length (encodeNat a).length) (encodeNat a)
```

A triple is `(b, a, c)`, so `t.1` is the middle coefficient. The final statements, verbatim from `Final` up to line breaks, except that we abbreviate the common bound as $\mathrm{T}(x)$, for the input word x, and write `Outputs` for `GraphProblems.Outputs`. In the source, $\mathrm{T}(x)$ is

```
⌊(C : ℝ) * 2 ^ ((C : ℝ) * x.length /
    Real.sqrt (Real.logb 2 (x.length + 2)))⌋₊
```

that is, $\lfloor C \cdot 2^{C|x|/\sqrt{\log_2(|x|+2)}} \rfloor$. The statements are:

```
theorem hForm_TM_rate :
  ∃ (M : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (M.tm.Γ i)) ∧
  ∀ N : ℕ, Outputs M (encodeNat N) (encodeNat (hForm (-(N : ℤ)))) T(encodeNat N)
theorem hFormPrim_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧
  ∀ N : ℕ, IsFundDisc (-(N : ℤ)) → 0 < N →
    Outputs M (encodeNat N) (encodeNat (primRedForms (-(N : ℤ))).card) T(encodeNat N)
theorem sqrtMod_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧
  ∀ m a : ℕ, Outputs M (encodePair m a) (encodeNat (minRoot m a)) T(encodePair m a)
theorem lqnr_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧
  (∀ p : ℕ, p.Prime →
     Outputs M (encodeNat p) (encodeNat (lqnr p)) T(encodeNat p)) ∧
  (∀ p : ℕ, ∃ y ≤ p, Outputs M (encodeNat p) (encodeNat y) T(encodeNat p))
theorem divisorSum_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧ ∀ x : ℕ,
  Outputs M (encodeNat x) (encodeNat (∑ m ∈ Finset.Icc 1 x, (Nat.divisors m).card))
    T(encodeNat x)
theorem sigmaSum_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧ ∀ x : ℕ,
  Outputs M (encodeNat x) (encodeNat (∑ m ∈ Finset.Icc 1 x, ∑ d ∈ Nat.divisors m, d))
    T(encodeNat x)
theorem circle_TM_rate : ∃ M C, (∀ i, Finite (M.tm.Γ i)) ∧ ∀ x : ℕ,
  Outputs M (encodeNat x)
    (encodeNat (((Finset.Icc (-(x : ℤ)) x ×ˢ Finset.Icc (-(x : ℤ)) x).filter
      fun q : ℤ × ℤ => q.1 ^ 2 + q.2 ^ 2 ≤ x).card)) T(encodeNat x)
```

In the source, each statement spells out the bound and writes `Computability.encodeNat`; the types of `M` and `C` are as in the first. `Outputs` is the one-line definition of [A]'s statement file, which we also trust:

```
def Outputs (M : TM2ComputableAux Bool Bool) (x y : List Bool) (T : ℕ) : Prop :=
  Nonempty (TM2OutputsInTime M.tm (x.map M.inputAlphabet.invFun)
    (some (y.map M.outputAlphabet.invFun)) T)
```

Behind these are Mathlib's `TM2OutputsInTime`, `Computability.encodeNat` (little-endian binary, the empty word for 0), `Nat.find`, `Nat.divisors`, `Finset.Icc`, `Squarefree`, `Int.gcd`, `Real.logb` and `Real.sqrt`, from Mathlib `0df444a`. Integer division and remainder are Lean's (`Int.emod` is non-negative), which in `IsFundDisc` gives the usual definition for $D < 0$. For a quick check of the definitions, Lean evaluates $h_{\mathrm{form}}(-3) = h_{\mathrm{form}}(-4) = h_{\mathrm{form}}(-7) = h_{\mathrm{form}}(-8) = 1$, $h_{\mathrm{form}}(-15) = h_{\mathrm{form}}(-20) = 2$ and $\mathrm{lqnr}(7, 11, 13, 17, 41) = 3, 2, 2, 3, 3$ by `decide`.

9. The Lean development

In this section we list the modules and state the trusted base. The development is in the public repository [36]: directories `lean/EthFollowup/ModP` (pure Mathlib; the files compile with Lean v4.33.1 and v4.34.1) and `lean-bridge-anthropic/bridge/EthFollowup/ModP` (on top of [A]'s development). Table 1 lists the modules.

Trusted base and axioms. The theorems depend only on `propext`, `Classical.choice` and `Quot.sound`. A fresh `leanchecker` replay of the dependency closure of `EthFollowup.ModP.Final`, the module holding the seven final theorems, has passed. What must be trusted is the Lean kernel, the Mathlib definitions named in Section 8, our definitions quoted there, and [A]'s definition of `Outputs`. [A]'s development, Mathlib's proofs and our other modules are checked by the kernel; they are part of the checked closure, not of the trusted base.

One design point. Everything polynomial-time must be polynomial on *every* context, not only on those the reader produces. Our first context stored the number of variables and vertices in binary and looped over them, which is exponential on adversarial contexts; the context now stores them in unary (`ArithProcs`).

10. Related work

We organise related work by idea: class numbers, lattice-point counts, non-residues and square roots, the counting engines applied to number theory, and formal proofs.

Class numbers. Counting reduced forms is Gauss's method and is used to tabulate class numbers [27, §2], [5]; the best proven deterministic unconditional bound we know is $|D|^{1/2+o(1)}$ [18, §5]. Under GRH, Shanks's method gives $|D|^{1/5+o(1)}$ deterministically [17, 27], [9, §5.4], and Hafner and McCurley's Las Vegas algorithm runs in expected time $L(|D|)^{\sqrt{2}+o(1)}$, $L(x) = \exp\sqrt{\log x \log\log x}$ [13]; its analysis assumes GRH [18, §5]. Mosunov and Jacobson [21] tabulate all class groups with $|D| < 2^{40}$ unconditionally, which is a different,

module	content	lines
`SignedCarry`	signed carry chains: existence iff sum, uniqueness, range	417
`Compare`	comparison state chains	213
`MinSearch`	prefix search finds the minimum	164
`QRCount`	`minRoot`, `lqnr`, root and square counts, (5.1)	509
`Encode`	`encodePair` and its halves	146
`ClassForms`	`hForm`, the two-query split, primitivity	277
`Lattice`	divisor sums and the circle as solution counts	293
`ArithDesc`	descriptions and their count	192
`ArithInstances`	the six descriptions and their counts	427
`Blocks`	mixed-radix block layouts	248
`ArithBounds`, `ArithParams`	digit bounds; wires, vertices, checks, canonical run	957
`ArithNetDefs`, `ArithNetParsimony`, `ArithNetCount`, `ArithNet`	the label system, parsimony, U and P	947
`ArithEnc`, `ArithReaderPoly`, `ArithProcs`, `ArithPoly`	encodings, procedures, polynomial time	1647
`CntPackage`, `GRate`	counting packages; search and query machines	878
`ArithPackage`, `ArithPkgs`	packages from descriptions; the six packages	589
`SpecSq`, `SpecE`, `SpecLattice`	the search and query specifications	616
`Final`	the seven final theorems	112
total		8632

TABLE 1. The modules, with line counts (`wc -l`). The first nine are pure Mathlib. The sanity module `GRateMinFac` (77 lines, re-deriving the factoring theorem through `GR.searchM_TM`) is not imported by `Final`. The development reuses, unchanged, the factoring modules of [35] (`Params`, `FermatBridge`, `MulProcs`, `MulPackage`, `Level40`, `Rate`, `Search`) and the bridge modules of [34].

amortised problem. OpenAI's zero-free half-plane [22] covers the Dirichlet L-functions $L(s,\chi_D)$ and, among Hecke L-functions, those over $\mathbb{Q}(\sqrt{-3})$. Frei, Sofos and Widmer [12] derive consequences of the quasi-Riemann hypothesis for class groups and small generators of number fields, some of them made unconditional by [22]; they give no class-number algorithm. Some GRH-based analyses of class-number algorithms may become unconditional in this way, with larger exponents; we have not checked which, and we know of none that would give $|D|^{o(1)}$. Vega [37] shows that $h(-d)$ is computable in deterministic polynomial time with an oracle for a language in Σ_2^{P}. This is a statement of a different kind: it gives polynomial time if P = NP, but no time bound otherwise. In contrast, our algorithm is deterministic and unconditional except for [A]'s development, and it is $|D|^{o(1)}$; but it computes only the number, not the group.

Lattice-point counts. Sladkey [31] computes $\sum_{m\le x}\tau(m)$ in $O(x^{1/3})$ steps, and Tao, Croot and Helfgott [32, §2.1] sketch an $x^{1/3+o(1)}$ algorithm; Li [19] reports $x^{9/28+o(1)}$ expected time. For the circle count we found no dedicated reference; we expect the same convex-hull techniques to give $x^{1/3+o(1)}$. In contrast, we obtain $x^{o(1)}$, at the price of galactic constants.

Non-residues and square roots. Burgess's bound $p^{1/(4\sqrt{e})+\varepsilon}$ [6], from his character-sum method [7], was for decades the best unconditional bound for the least non-residue. Under GRH it is $O(\log^2 p)$ by Ankeny [1], with explicit constants by Bach [3] and Lamzouri, Li and Soundararajan [15]. OpenAI [22] prove that every Dirichlet L-function is zero-free in $\Re s > 7/8$, and state the consequence for non-residues as their Corollary 1.2: the least quadratic non-residue $n(p)$ satisfies $n(p) \le C(\log p)^A$ for absolute constants $A, C > 0$ (for example $A = 32$), via the weak-GRH bound of Bhargava, Ivanyos, Mittal and Saxena. This proves Vinogradov's least non-residue conjecture, and trying $2, 3, \dots$ with Euler's criterion finds $n(p)$ in deterministic polynomial time.

Given a non-residue, square roots modulo p take polynomial time by Tonelli [33] and Shanks [29], or Cipolla [8]; Schoof [28] computes $\sqrt{a} \bmod p$ in polynomial time for fixed a. Finding a non-residue and taking square roots modulo p are deterministic polynomial-time equivalent (folklore; [4]). The same Corollary 1.2 of [22] states the algorithm: find a non-residue, then run Tonelli–Shanks, all in time polynomial in $\log p$.

OpenAI's second result [23] gives them directly: it factors every polynomial over $\mathbb{F}_p$ deterministically in time polynomial in $(\deg f + 1)\log p$, so it finds the roots of $y^2 - a$.

For composite m, taking square roots is randomised-equivalent to factoring m [25]. Deterministic factoring takes $m^{1/5+o(1)}$ steps classically [11], and $2^{O(n/\sqrt{\log n})}$ steps assuming the new engines [26, 30, 35]. The least square root modulo m can also be derived from the factorisation, as follows; this is on paper only. Write $m = \prod p^e$ and $a \bmod p^e = p^k a'$ with $p \nmid a'$. The roots of $y^2 \equiv a \pmod{p^e}$ form at most two residue classes modulo a power of p for odd p, and at most four for $p = 2$. If $k = 0$, they are the Hensel lifts of the roots modulo p, which [22] or [23] gives (for $p = 2$, of the roots modulo $2^{\min(e,3)}$). If $0 < k < e$, there are roots only when k is even, and they are $p^{k/2}$ times the roots of $y'^2 \equiv a' \pmod{p^{e-k}}$, so they are classes modulo $p^{e-k/2}$. If $p^e \mid a$, they are the multiples of $p^{\lceil e/2 \rceil}$. The Chinese remainder theorem combines these into at most $2^{\omega(m)+1}$ classes modulo one divisor d of m, and the least root is the least of their representatives in $[0, d)$. Since $\omega(m) = O(n/\log n)$, this takes $2^{O(n/\log n)}$ polynomial-time steps after factoring, so the total is $2^{O(n/\sqrt{\log n})}$, under the same assumptions as the factoring algorithm. We have not formalised this route. In contrast, (iii) and (iv) compute the least root and the least non-residue directly by counting, without factoring and without a zero-free region, and they are machine-checked; but modulo a prime they are slower than the polynomial-time algorithms above.

Counting engines applied to number theory. Rudich and Rousseau [26] introduced prefix search by divisor counting and proved factoring in $N^{o(1)}$ in Lean; Shunia [30] proved the rate $2^{O(n/\sqrt{\log n})}$ on paper, and [14] and our factoring note [35] proved it in Lean. Closest to our work is the factoring note, whose multiplication network and level selection we reuse. In contrast, we replace its single network by a library for any few-product system and apply it to problems other than factoring. For version 1 (4 November 2026) we searched the web for each problem together with "ETH", "#SAT" and "2026"; searched GitHub repositories for the problem names; checked the Hexagon records and repositories of the October and November 2026 follow-ups known to us ([26, 30, 14] and the fork `orda-aska/integer-factoring`), which treat factoring only; and read our survey of targets [36]. That search missed [22, 23]. For version 2 (5 November 2026) we read OpenAI's catalogue (`CONTENTS.md` at commit `4eaee87`); grepped the titles and abstracts of all 716 Hexagon works through its public query interface, and the full text of the 181 works since 26 October in cs.*, math.NT, math.CO and quant-ph; and searched arXiv for the problem names, the quasi-Riemann hypothesis and polynomial factorisation. This found [22, 23, 12, 37], and no derivation of (i), (ii) or (v) from either ETH refutation. A negative search result is weak evidence.

Formal proofs. We use Lean 4 [16] and Mathlib [20]. Mathlib defines the class number of a number field (`NumberField.classNumber`) and proves its finiteness, but has no theory of binary quadratic forms. In contrast, our statements are about forms, and the bridge to ideals is the classical theorem cited in Section 6.

11. Limitations

- *The results stand or fall with* [A]*'s development.* Every theorem rests on [A]'s Lean development of its claimed refutation of the Exponential Time Hypothesis, and the labs' claims are under review. Lean checks that development as part of our proof, and our statements use only Mathlib, our quoted definitions and [A]'s `Outputs`. We do need the development, built from source at the pinned commit, to be a sound Lean development, and the toolchain to be sound.
- *Parts (iii) and (iv) are not new bounds.* Modulo a prime, the least non-residue and square roots take deterministic polynomial time by [22, 23]. For composite m, the least root follows on paper from deterministic factoring in time $2^{O(n/\sqrt{\log n})}$ as well (Section 10). What (iii) and (iv) add is a machine-checked proof that does not use those results. All our results, (i) and (v) included, come from the same engine.
- *The class number is the form class number.* Its equality with the ideal class number is classical but not formalised; for non-fundamental D the theorem is about h_{form} (Theorem 6.1); the class group and real quadratic fields are not covered.
- *The algorithms are galactic.* [A]'s levels start at $K_0 = 2^{x_0}$ with $x_0 \geq 18$, the level needs $K^{40} \lesssim n$, and the constants $c_0 = 2^{12} \cdot 1152$ and C are astronomical. The machines are of no practical use, and they beat the classical bounds only for astronomically large inputs.
- *The machines use exponential space.*

- *No human has checked the work.* The Lean code and this text were written with AI assistance. No human expert has yet checked that Section 8 lists everything a reader must trust.

Acknowledgements

This work was done with AI assistance (Claude Code, Anthropic): the constructions, the Lean development, the literature and citation checks and the text were produced in sessions with Claude Code. The Lean theorems are checked by the Lean kernel; everything else awaits human expert review. Responsibility for any errors rests with the author. This note continues the project of [34, 35].

References

[1] N. C. Ankeny, *The least quadratic non residue*, Ann. of Math. (2) **55** (1952), 65–72.
[2] Claude (Anthropic), *A $2^{o(n)}$ time algorithm for 3-SAT*, preprint dated 26 October 2026. Paper and Lean formalisation in the repository `https://github.com/anthropics/formal-math`, folder `eth`; we use commit `16a5ce41304fcdaea4e0be7c132e1c361dc26da5`.
[3] E. Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), no. 191, 355–380.
[4] E. Bach and J. Shallit, *Algorithmic Number Theory, Vol. 1: Efficient Algorithms*, MIT Press, Cambridge, MA, 1996.
[5] D. A. Buell, *Binary Quadratic Forms: Classical Theory and Modern Computations*, Springer-Verlag, New York, 1989.
[6] D. A. Burgess, *The distribution of quadratic residues and non-residues*, Mathematika **4** (1957), 106–112.
[7] D. A. Burgess, *On character sums and primitive roots*, Proc. London Math. Soc. (3) **12** (1962), 179–192.
[8] M. Cipolla, *Un metodo per la risoluzione della congruenza di secondo grado*, Rend. Accad. Sci. Fis. Mat. Napoli (3) **9** (1903), 154–163.
[9] H. Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics, vol. 138, Springer-Verlag, Berlin, 1993.
[10] D. A. Cox, *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication*, 2nd ed., John Wiley & Sons, Hoboken, NJ, 2013.
[11] D. Harvey, *An exponent one-fifth algorithm for deterministic integer factorisation*, Math. Comp. **90** (2021), no. 332, 2937–2950.
[12] C. Frei, E. Sofos, and M. Widmer, *Some applications of the quasi-Riemann hypothesis to class groups and small generators of number fields*, arXiv:2610.18491, October 2026.
[13] J. L. Hafner and K. S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, J. Amer. Math. Soc. **2** (1989), no. 4, 837–850.
[14] 1337isnot1337 (GitHub user) and contributors, *integer-factoring: deterministic integer factoring bounds*, public repository `https://github.com/1337isnot1337/integer-factoring`; Lean formalisation in `formal-appendix/` (pull request #20, merged 30 October 2026).
[15] Y. Lamzouri, X. Li, and K. Soundararajan, *Conditional bounds for the least quadratic non-residue and related problems*, Math. Comp. **84** (2015), no. 295, 2391–2412; corrigendum, Math. Comp. **86** (2017), 2551–2554.
[16] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, Lecture Notes in Comput. Sci., vol. 12699, Springer, 2021, pp. 625–635.
[17] H. W. Lenstra, Jr., *On the calculation of regulators and class numbers of quadratic fields*, Journées Arithmétiques 1980 (J. V. Armitage, ed.), London Math. Soc. Lecture Note Ser., vol. 56, Cambridge Univ. Press, 1982, pp. 123–150.
[18] H. W. Lenstra, Jr., *Algorithms in algebraic number theory*, Bull. Amer. Math. Soc. (N.S.) **26** (1992), no. 2, 211–244.
[19] D. Li, *Computing the divisor summatory function in $N^{9/28+o(1)}$*, arXiv:2609.29949, September 2026.
[20] The mathlib Community, *The Lean mathematical library*, Proc. 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), 2020, pp. 367–381.
[21] A. S. Mosunov and M. J. Jacobson, Jr., *Unconditional class group tabulation of imaginary quadratic fields to $|\Delta| < 2^{40}$*, Math. Comp. **85** (2016), no. 300, 1983–2009.
[22] OpenAI, *The Quasi-Riemann Hypothesis: A Zero-Free Half-Plane $\Re s > 7/8$*, preprint dated 30 September 2026; entry 003, "The quasi-Riemann hypothesis", of the catalogue `CONTENTS.md` in the repository `https://github.com/openai/math`.
[23] OpenAI, *Deterministic Polynomial Factorization over Prime Fields*, preprint dated 4 October 2026; entry 142, "Deterministic polynomial factorization over prime fields", of the catalogue `CONTENTS.md` in the repository `https://github.com/openai/math`.
[24] OpenAI, *The Exponential Time Hypothesis is false*, preprint dated 17 October 2026, revised 25 October 2026. Paper and Lean formalisation in the repository `https://github.com/openai/math`.
[25] M. O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.
[26] I. Rudich and L.-M. Rousseau, *Deterministic factoring in time $N^{o(1)}$*, Hexagon preprint hexagon:2610.00600v1, published 29 October 2026, `https://hexagonmath.org/2610.00600v1`.
[27] R. J. Schoof, *Quadratic fields and factorization*, Computational Methods in Number Theory, Part II (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Mathematical Centre Tracts, vol. 155, Mathematisch Centrum, Amsterdam, 1982, pp. 235–286.
[28] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Math. Comp. **44** (1985), no. 170, 483–494.

[29] D. Shanks, *Five number-theoretic algorithms*, Proc. Second Manitoba Conference on Numerical Mathematics (Winnipeg, 1972), Congressus Numerantium VII, Utilitas Math., 1973, pp. 51–70.
[30] J. Shunia, *An improved subexponential bound for deterministic integer factorization*, Hexagon preprint hexagon:2610.00711v1, published 2 November 2026, https://hexagonmath.org/2610.00711v1.
[31] R. Sladkey, *A successive approximation algorithm for computing the divisor summatory function*, arXiv:1206.3369, June 2012.
[32] T. Tao, E. Croot III, and H. Helfgott, *Deterministic methods to find primes*, Math. Comp. **81** (2012), no. 278, 1233–1246.
[33] A. Tonelli, *Bemerkung über die Auflösung quadratischer Congruenzen*, Nachr. Königl. Ges. Wiss. Göttingen (1891), 344–346.
[34] V. Tulsyan, *Counting Hamiltonian cycles in sparse graphs in subexponential time, assuming the new #SAT algorithms*, Hexagon preprint hexagon:2610.00672v1, October 2026; source and Lean development at https://github.com/xennygrimmato/eth-hamiltonicity.
[35] V. Tulsyan, *An independent machine-checked proof of deterministic factoring in time* $2^{O(n/\sqrt{\log n})}$*, via the new #SAT algorithms*, preprint, November 2026; source in the directory `factoring-note/` of [36].
[36] V. Tulsyan, *eth-hamiltonicity: Lean 4 formalisation and research notes*, public repository https://github.com/xennygrimmato/eth-hamiltonicity, 2026; Lean sources as of commit `b9ea3f1`.
[37] F. Vega, *Class Numbers of Imaginary Quadratic Orders in the Polynomial Hierarchy*, Hexagon preprint hexagon:2611.00086v1, published 4 November 2026, https://hexagonmath.org/2611.00086v1.