

The Euclidean algorithm is not optimal among uniform processes

Jason Davies*

October 28, 2026

Abstract

Reusing a stored Euclidean prefix, we show that every pair $a \geq b$ has a derivation of $\gcd(a, b)$ from a, b with $O(\log a / \log \log a)$ applications of the remainder function alone; Euclid’s algorithm needs order $\log a$ in the worst case. These derivations bound the intrinsic complexity of $\gcd$ and coprimeness by the same order, refuting Moschovakis’s conjectured logarithmic lower bounds for uniform processes; by his best-uniform-process theorem, a single uniform process attains this bound. The Main Conjecture for algorithms remains open. We also prove an $\Omega(\log \log a)$ lower bound on this complexity for almost all coprime pairs and give, for each $\varepsilon > 0$, a nondeterministic register program computing $\gcd$ that, on every input $a \geq b \geq 1$, has a successful computation using at most $\varepsilon \log a + O(1)$ remainder calls. The upper bound also answers negatively an open problem of van den Dries and Moschovakis on the closure depth of the $\gcd$. The main bounds and the refutations are formalised in Lean 4; the best-uniform-process theorem is cited.

AI transparency statement. The main result, its formalisation in Lean and the manuscript were produced by artificial intelligence. The human author initiated the broader research and directed the AI systems, but did not write the text. Details are given at the end of the paper.

1 Introduction

Van den Dries and Moschovakis asked whether the Euclidean algorithm is optimal among its peers, the algorithms that compute the $\gcd$ from the remainder function [13, p. 390]. Moschovakis’s Main Conjecture (in his monograph [10], henceforth “the book”) asserts that for every algorithm α from rem and the zero test eq_0 computing $\gcd(x, y)$ there is an $r > 0$ such that α makes more than $r \log x$ remainder calls on infinitely many pairs $x > y \geq 1$ [10, p. 2]. Uniform processes are the book’s abstract model of computation from given primitives: every deterministic or nondeterministic recursive program induces one [10, §4A, Proposition 4A.1], but a uniform process need not be implemented by any finite program. Moschovakis stated versions of the conjecture for uniform processes in his 2012 notes and in the book; we refute both (Section 5). For algorithms the conjecture remains open.

Derivations. A *derivation* from a, b is a finite list that starts with a, b and in which each further entry is $\text{rem}(x, y)$ for earlier entries x, y with $y > 0$. Each further entry is one *instruction*, that is, one remainder call. Entries already in the list are *available* and may be reused; no constants and no other operations are supplied. Let $G(a, b)$ be the least number of instructions in a derivation containing $\gcd(a, b)$. Thus $G(a, 0) = G(a, 1) = 0$. Arithmetic used in the proofs, such as subtraction, sorting or comparison, only selects instructions; it is never itself an instruction. So G measures the length of a derivation, not the work of finding one. Logarithms are to base 2.

*jason@jasondavies.com

Theorem 1. *There is a constant C such that for all integers $a \geq 4$ and $0 \leq b \leq a$,*

$$G(a, b) \leq C \frac{\log a}{\log \log a}.$$

Idea of the proof. The key is to store one short Euclidean prefix and reuse it many times. For sufficiently large n , let $a > b$ be coprime with $a \leq N = 2^n$. Store a Euclidean prefix $r_0 = a, r_1 = b, \dots, r_h$, where $h = 3k$ and $k = 2^{\lfloor (\log n)/2 \rfloor}$ is about $\sqrt{n}$. If the prefix reaches a positive remainder at most $N^{3/4}$, take one further remainder and recurse on the resulting smaller pair. Otherwise, modulo every $m > N^{1/3}$, each residue class contains at most three prefix values, so the prefix occupies at least $k + 1$ residue classes (Lemma 3). A close pair of residues lets three remainder calls produce a positive value below m/k (Lemma 2). Reusing the same prefix, we repeatedly shrink the modulus from r_h to at most $N^{1/3}$ in $O(n/\log n)$ calls. Since m need not be coprime to a or b , two recursive gcd derivations on inputs at most $N^{1/3}$ finish: of $d = \gcd(m, \text{rem}(a, m))$, and then of $\gcd(d, \text{rem}(b, m))$. If $T(n)$ is the worst derivation cost for inputs at most 2^n , the resulting recurrence (5) gives $T(n) = O(n/\log n)$. Taking k about n^c with c close to 1 instead sharpens the upper coefficient to $3 + o(1)$ (Corollary 4): each contraction costs three calls and removes about $\log k \approx \log n$ bits, while the prefix stays negligible.

Consequences and further results. Section 5 shows that $G(a, b)$ equals the intrinsic remainder-call complexity of coprimeness in the book’s Euclidean structure on positive coprime inputs, and bounds those of gcd to within 2 (Proposition 7). Theorem 1 therefore refutes Conjecture 3.4 of [9] and version (4F-5) of the Main Conjecture, and shows that lower bounds from the homomorphism method cannot exceed $O(\log a / \log \log a)$ for these problems. The book’s best-uniform-process theorem then gives a single process with the same bound on every input. The upper bound also settles an open problem of van den Dries and Moschovakis on the closure depth of the gcd (Corollary 5).

We also prove that $G(a, b) \geq (1 - o(1)) \log \log N$ for all but $o(N^2)$ coprime pairs $2 \leq b < a \leq N$ (Proposition 11). Finally, fixing the prefix length gives, for each $\varepsilon > 0$, a nondeterministic register program with a successful computation costing at most $\varepsilon \log a + O(1)$ on each input $a \geq b \geq 1$ (Theorem 12). The number of registers grows as ε decreases.

Sections 2–4 prove Theorem 1 and its refinements, Section 5 treats certificates and uniform processes, Sections 6 and 7 give the lower bound and the register programs, Section 8 discusses open problems, and Section 9 and Appendix A describe the computational checks and the Lean formalisation.

Related work. The known intrinsic lower bounds give $\Omega(\log \log a)$ on special families of coprime pairs [13, 14], [10, Theorem 6C.5], also for richer primitives such as addition and integer quotient, and for depth. Pratt’s unpublished nondeterministic algorithm [10, Theorem 2E.2 and Problems x2E.17–x2E.19] matches this order on successive Fibonacci numbers [10, Corollary 6C.7]. For remainder-only derivations, our almost-all bound gives the same order on a set of full density among coprime pairs. Mansour, Schieber and Tiwari proved lower bounds for gcd and coprimeness in computation trees with richer arithmetic, including the floor operation [6, 7]; van den Dries and Moschovakis compare them with the intrinsic bounds [14, pp. 2:3–2:4].

In a different model, Bshouty obtained the optimal bound $\Theta(\log N / \log \log N)$ for gcd on inputs in $[0, N]$ using a random access machine with linear operations, shifts, comparisons and indirect addressing [1, Theorems i and ii, p. 3]. His construction tabulates blocks of the binary gcd algorithm [12], [5, §4.5.2], computing table entries on demand and counting their construction [1, §4,

Algorithm D and Lemma 5, pp. 12–14]. Here the saving comes from reusing a Euclidean prefix with remainder as the only instruction.

2 Normalisation

Write $g = \gcd(a, b) > 0$. For positive y ,

$$\text{rem}(gx, gy) = g \text{rem}(x, y).$$

Every generated value is divisible by g . Consequently scaling a derivation, or dividing all its values by g , preserves its length and the validity of every instruction. Hence

$$G(a, b) = G(a/g, b/g). \quad (1)$$

Define

$$T(n) = \max_{0 \leq b \leq a \leq 2^n, a > 0} G(a, b).$$

Euclid's remainders form a derivation containing $\gcd(a, b)$, so G and T are finite. By (1) it suffices to bound G on coprime pairs; the cases $a = b$, $b = 0$ and $b = 1$ need no instructions.

3 Three-call contractions

Lemma 2. *If $m > 0$ is available and a finite collection S of available values occupies $t \geq 2$ residue classes modulo m , at most three instructions generate w with*

$$0 < w < \frac{m}{t-1}.$$

Proof. Sort the distinct residues. Some adjacent pair $u < v$ has gap $\delta = v - u < m/(t-1)$, because the sum of the $t-1$ adjacent gaps is less than m . Each selected residue is $\text{rem}(x, m)$ for some $x \in S$, so it costs one instruction. If $u = 0$, use v . If $0 < u \leq \delta$, use u . Otherwise $0 < \delta < u$ and $v = u + \delta$, so $\text{rem}(v, u) = \delta$. This uses at most three instructions. $\square$

Lemma 3 (Euclidean-prefix lemma). *Let $r_0 = a > r_1 = b > \dots > r_h > 0$ be a Euclidean prefix of a coprime pair, with $r_{i-1} = q_i r_i + r_{i+1}$ and $q_i \geq 1$ for $1 \leq i < h$. For every integer $m > 2a/r_h$, each residue class modulo m contains r_i for at most three indices $0 \leq i \leq h$. The prefix therefore occupies t residue classes with $3t \geq h + 1$.*

Proof. Let $q_h = \lfloor r_{h-1}/r_h \rfloor \geq 1$ and $r_{h+1} = \text{rem}(r_{h-1}, r_h)$, so that the recurrence also holds for $i = h$. For $i < j \leq h$ set

$$M_{i,j} = \prod_{\ell=i+1}^j \begin{pmatrix} q_\ell & 1 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = M_{i,j} \begin{pmatrix} r_j \\ r_{j+1} \end{pmatrix}.$$

Right multiplication by $\begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$ sends (A, B, C, E) to $(qA+B, A, qC+E, C)$, starting from $(q, 1, 1, 0)$ for a single factor. By induction, the entries of $M_{i,j} = \begin{pmatrix} A & B \\ C & E \end{pmatrix}$ are nonnegative, $A \geq B \geq E$, $A \geq C \geq E$, $B, C \geq 1$, and $\det M_{i,j} = (-1)^{j-i}$; a product of at least two factors has $A \geq 2$.

Suppose $r_s \equiv r_u \equiv r_v \pmod{m}$ with $s+2 \leq u < v \leq h$; a class with four indices $i_1 < i_2 < i_3 < i_4$ contains such $s = i_1$, $u = i_3$ and $v = i_4$. Write $M_{s,u} = \begin{pmatrix} \sigma & \tau \\ * & * \end{pmatrix}$ and $M_{u,v} = \begin{pmatrix} A & B \\ C & E \end{pmatrix}$, so that $r_u = Ar_v + Br_{v+1}$ and $r_s = A'r_v + B'r_{v+1}$ with $A' = \sigma A + \tau C$ and $B' = \sigma B + \tau E$. Then m divides

$r_s - r_v = (A' - 1)r_v + B'r_{v+1}$ and $r_u - r_v = (A - 1)r_v + Br_{v+1}$. Eliminating r_{v+1} , and then r_v , shows that m divides Dr_v and Dr_{v+1} , where

$$D = (A' - 1)B - B'(A - 1) = -\tau \det M_{u,v} + (\sigma - 1)B + \tau E.$$

Consecutive remainders are coprime, so Bézout's identity gives $m \mid D$. If $\det M_{u,v} = -1$, then $D \geq \tau > 0$. If it is 1, then $E \geq 1$ as $AE = 1 + BC$, and $\sigma \geq 2$ because $M_{s,u}$ has $u - s \geq 2$ factors, so $D \geq B > 0$. In both cases $D \leq \tau + B' - B \leq \tau + A' - B < 2A'$, since $B' \leq A'$ (as $A \geq B$ and $C \geq E$) and $\tau \leq \sigma \leq \sigma A < A'$. Finally $A'r_v \leq r_s \leq a$ gives $2A' \leq 2a/r_h < m$, so $0 < D < m$, a contradiction. Hence each class contains at most three of $r_0, \dots, r_h$, and t classes hold at most $3t$ of them. $\square$

Since Lemma 3 holds for every $m > 2a/r_h$, one stored prefix serves every later modulus, although these moduli are not known in advance. If $h = 3k$ with $k \geq 1$, then $3t \geq 3k + 1$, so the prefix occupies at least $k + 1$ residue classes modulo every available $m > 2a/r_h$, and Lemma 2 applied to the prefix gives three instructions that generate

$$0 < w < \frac{m}{k}. \quad (2)$$

We call these three instructions a *contraction* of m , and w the new modulus.

For example, from 44, 17 one instruction stores the prefix 44, 17, 10. Modulo $m = 10 > 2 \cdot 44/10$ its residues are 4, 7, 0, and the close pair $4 < 7$ gives a contraction step as in Lemma 2 (this prefix is too short for (2), so we read off the residues directly):

$$\text{rem}(44, 10) = 4, \quad \text{rem}(17, 10) = 7, \quad \text{rem}(7, 4) = 3,$$

with new modulus $3 < 10/(3 - 1)$. One more instruction, $\text{rem}(10, 3) = 1$, finishes. This derivation takes five instructions; Euclid reaches 1 in four. The example illustrates contraction without a saving on this pair.

4 Construction and recurrence

Fix an integer $n > 16$ and put

$$N = 2^n, \quad p = \lfloor (\log n)/2 \rfloor \geq 2, \quad k = 2^p, \quad h = 3k, \quad J = \lceil n/p \rceil,$$

so that $k \leq \sqrt{n}$ and $k^J \geq N$. After normalisation, assume $a > b > 1$ are coprime and $a \leq N$. We use $N^{3/4}$ as the *early threshold* and $N^{1/3}$ as the *target modulus*. Any thresholds N^α, N^β with $\alpha < 1$, $\alpha + \beta > 1$ and $2\beta < 1$ would do: these make the early recursive problem smaller, let the prefix lemma apply above the target, and make the two final problems together smaller.

Phase (i): the prefix. Starting from $r_1 = b$, generate the Euclidean remainders $r_2, r_3, \dots$, one instruction each, until the most recent value is at most $N^{3/4}$ or r_h is available. This costs at most h instructions. Since the inputs are coprime, Euclid's algorithm reaches 1 before 0, so the stopping value is positive.

If the stopping value c is at most $N^{3/4}$, generate the next remainder $s < c$ with one more instruction. The pair (c, s) is coprime, with both entries at most $2^{\lceil 3n/4 \rceil}$. This gives the bound

$$h + 1 + T(\lceil 3n/4 \rceil). \quad (3)$$

Phase (ii): contractions. Otherwise $r_h > N^{3/4}$, so $2a/r_h < 2N^{1/4} < N^{1/3}$ (as $n > 12$), and Lemma 3 applies to every available $m > N^{1/3}$. Starting at $m = r_h$, repeatedly replace m by the value w of (2) until $0 < m \leq N^{1/3}$. After j contractions $m < N/k^j$; since $k^J \geq N$ and $m \geq 1$, fewer than J contractions occur. Since $p = \frac{1}{2} \log n + O(1)$, their cost is $3J = O(n/\log n)$. This is the source of the saving: the prefix costs only $O(\sqrt{n})$ instructions, and every contraction reuses it.

Phase (iii): two recursive derivations. The small m need not be coprime to either input. Generate $u = \text{rem}(a, m)$ and $v = \text{rem}(b, m)$, then append a derivation of $d = \text{gcd}(m, u)$ from m, u and one of $e = \text{gcd}(d, v)$ from d, v . Since $m \leq 2^{\lceil n/3 \rceil}$ and G is symmetric, each costs at most $T(\lceil n/3 \rceil)$. Finally

$$e = \text{gcd}(\text{gcd}(m, \text{rem}(a, m)), \text{rem}(b, m)) = \text{gcd}(m, \text{gcd}(a, b)) = 1.$$

The cost in this case is at most

$$h + 3J + 2 + 2T(\lceil n/3 \rceil). \quad (4)$$

The recurrence. The two case estimates (3) and (4) and normalisation imply

$$T(n) \leq \max\{T(\lceil 3n/4 \rceil) + O(\sqrt{n}), 2T(\lceil n/3 \rceil) + O(n/\log n)\}. \quad (5)$$

All implied constants are independent of the inputs.

Proof of Theorem 1. For $n \geq 2$ let $f(n) = n/\log n$. The ratios

$$\frac{f(\lceil 3n/4 \rceil)}{f(n)} \longrightarrow \frac{3}{4}, \quad \frac{2f(\lceil n/3 \rceil)}{f(n)} \longrightarrow \frac{2}{3}$$

show that, beyond a fixed threshold, $f(\lceil 3n/4 \rceil) \leq \frac{7}{8}f(n)$ and $2f(\lceil n/3 \rceil) \leq \frac{5}{6}f(n)$. Both overheads in (5) are at most $c_0 f(n)$ for a fixed c_0 . Choose the induction threshold above 16, where the recurrence holds, and large enough that both recursive indices are at least 2 and strictly less than n . Choose $C' \geq 8c_0$, increased to cover the finitely many values below the threshold. Strong induction then gives $T(n) \leq C' f(n)$: the two bounds are $(7C'/8 + c_0)f(n)$ and $(5C'/6 + c_0)f(n)$, both at most $C' f(n)$. Finally take $n = \lceil \log a \rceil$. For $a \geq 4$, $n \leq 2 \log a$ and $\log n \geq \log \log a \geq 1$, so $G(a, b) \leq T(n) \leq 2C' \log a / \log \log a$, and the theorem holds with $C = 2C'$. $\square$

4.1 The leading coefficient

The same construction with larger p gives a more precise bound.

Corollary 4. *For every $\varepsilon > 0$, there is an a_0 such that for every $a \geq a_0$ and $0 \leq b \leq a$,*

$$G(a, b) \leq (3 + \varepsilon) \frac{\log a}{\log \log a}.$$

Equivalently, the uniform upper bound has coefficient $3 + o(1)$.

Proof. Fix $0 < c < 1$ and $0 < \beta < 1/2$, and put $\alpha = 1 - \beta/2$. For large n put

$$p = \lfloor c \log n \rfloor, \quad k = 2^p, \quad h = 3k, \quad J = \lceil n/p \rceil.$$

Use N^α as the early threshold and N^β as the target modulus, where $N = 2^n$. In the late case $r_h > N^\alpha$, so $2a/r_h < 2N^{1-\alpha} = 2N^{\beta/2} < N^\beta$ for sufficiently large n , and the prefix lemma applies

to every modulus above N^β . By (2), each contraction reduces the modulus by a factor greater than k , and $k^J \geq N$, so J contractions suffice. Writing $f(n) = n/\log n$, we have

$$h = O(n^c) = o(f(n)), \quad 3J = \left(\frac{3}{c} + o(1)\right) f(n).$$

The two branches therefore give

$$T(n) \leq \max \left\{ T(\lceil \alpha n \rceil) + o(f(n)), \quad 2T(\lceil \beta n \rceil) + \left(\frac{3}{c} + o(1)\right) f(n) \right\}.$$

Theorem 1 implies that $\Lambda = \limsup_{n \rightarrow \infty} T(n)/f(n)$ is finite and nonnegative. For each fixed $q > 0$, the ratio $f(\lceil qn \rceil)/f(n)$ tends to q . Taking upper limits in the recurrence yields

$$\Lambda \leq \max\{\alpha\Lambda, 2\beta\Lambda + 3/c\}.$$

Since $\alpha < 1$ and $\Lambda \geq 0$, this implies $(1 - 2\beta)\Lambda \leq 3/c$. As T , and hence Λ , does not depend on c or β , this holds for all $0 < c < 1$ and $0 < \beta < 1/2$, so $\Lambda \leq 3$. Finally, $n = \lceil \log a \rceil$ satisfies $f(n) \sim \log a / \log \log a$, proving the assertion. All upper limits concern the worst-case function T , so the resulting threshold is independent of b . $\square$

We do not claim that 3 is optimal.

4.2 Closure depth

Van den Dries and Moschovakis [14] measure the parallel depth of the gcd. Put $S_0 = \{0, 1, a, b\}$, let S_{k+1} adjoin to S_k all values $x + y$, $x - y$, $\text{iq}(x, y)$ and $\text{rem}(x, y)$ with $x, y \in S_k$, and let $g(a, b)$ be the least k with $\text{gcd}(a, b) \in S_k$ [14, p. 2:6]. They ask whether there are positive integers $a_n \rightarrow \infty$ and b_n and a $C > 0$ with $g(a_n, b_n) > C \log a_n$ for all n [14, Open Problem, p. 2:11]. Here, following [14, p. 2:4], the stages live in $\mathbb{Z}$, $\text{iq}(x, y) = \lfloor x/y \rfloor$ and $\text{rem}(x, y) = x - y \text{iq}(x, y)$ for $y \neq 0$, and $\text{iq}(x, 0) = \text{rem}(x, 0) = x$; these total operations agree with the remainder used in derivations, whose divisors are always positive.

Corollary 5. *For integers $a \geq 4$ and $b \geq 1$,*

$$g(a, b) \leq 1 + G(a, \text{rem}(b, a)) = O\left(\frac{\log a}{\log \log a}\right).$$

Hence the Open Problem has a negative answer: no such sequences a_n, b_n exist.

Proof. Put $r = \text{rem}(b, a)$, so $r \in S_1$, $0 \leq r < a$ and $\text{gcd}(a, r) = \text{gcd}(a, b)$. In a derivation from a, r , the value of the k th instruction lies in S_{k+1} , by induction on k : its operands are a , r or earlier values, all in S_k , and every divisor in the derivation is positive. Theorem 1 bounds $G(a, r)$. $\square$

On a concurrent-read concurrent-write PRAM, counting bit operations, Kannan, Miller and Rudolph [4] and Chor and Goldreich [2] compute the gcd of n -bit integers in parallel time $O(n \log \log n / \log n)$ and $O(n / \log n)$ respectively. Van den Dries and Moschovakis posed their question later, for the closure under exact integer operations, where each operation on whole integers is one step; Corollary 5 gives depth of the same order there.

5 Certificates and uniform processes

Structures and certificates. Write tt, ff for the truth values and eq_c for the test $x = c$. We use the book's partial remainder operation, defined only when its second argument is nonzero, in the structures

$$\mathcal{N}_0 = (\mathbb{N}, \text{rem}, \text{eq}_0), \quad \mathcal{N}_\varepsilon = (\mathbb{N}, \text{rem}, \text{eq}_0, \text{eq}_1).$$

The latter is the book's Euclidean structure $\mathbf{N}_\varepsilon$ [10, (1C-10), p. 24]; our $\mathcal{N}_0$ should not be confused with the book's $\mathbf{N}_0$, which has no remainder operation [10, p. 171]. Coprimeness is the relation $x \perp\!\!\!\perp y \iff x, y \geq 1 \wedge \gcd(x, y) = 1$ [10, §1A].

A *certificate* for f at $\vec{x}$ in a structure $\mathcal{A}$ is a finite partial substructure $\mathcal{U}$ of $\mathcal{A}$, generated by $\vec{x}$ and containing $f(\vec{x})$ when f is a function, such that every homomorphism $\pi : \mathcal{U} \rightarrow \mathcal{A}$ satisfies $f(\pi\vec{x}) = \pi(f(\vec{x}))$; homomorphisms fix the truth values, so for a relation this says $f(\pi\vec{x}) = f(\vec{x})$. In other words, any assignment of values preserving the recorded operations and tests must preserve the answer. The *facts* of $\mathcal{U}$ are its recorded equations $\text{rem}(x, y) = z$ (*remainder facts*) and test values $\text{eq}_c(x) = \text{tt}$ or ff (*test facts*); together they form the *diagram* of $\mathcal{U}$. The remainder equations of a derivation form a finite partial substructure generated by the inputs; Proposition 7 adds the test facts that make it a certificate. The *intrinsic remainder-call complexity* $\text{calls}_{\text{rem}}(\mathcal{A}, f, \vec{x})$ is the least number of remainder facts in a certificate [10, §§1D, 4C–4E].

The following lemma lets us treat certificates as finite sets of facts and extract a derivation from any certificate; it is used in Proposition 7. It also justifies the fact-based Lean encoding, for which Lean proves the resulting equality of intrinsic complexities (Appendix A).

Lemma 6 (Fact representation). *Let F be a finite set of true remainder and test facts in which every number is generated from a, b by the remainder facts of F . Then F is the diagram of a finite partial substructure generated by a, b (of $\mathcal{N}_\varepsilon$, or of $\mathcal{N}_0$ if F has no eq_1 facts), and every such substructure arises in this way. A map $\pi : \mathbb{N} \rightarrow \mathbb{N}$ preserves the facts of F exactly when its restriction to the universe is a homomorphism, and every homomorphism extends to such a map. Every element of the universe is available after a derivation from a, b with at most as many instructions as F has remainder facts.*

Proof. Take the universe V to consist of a, b and the numbers occurring in F , and interpret each primitive only at the recorded arguments, with the recorded value. Truth of the facts makes these partial operations and tests restrictions of the primitives of $\mathbb{N}$, and the hypothesis says that V is generated by a, b . Conversely, the diagram of a finite generated partial substructure is such a set F . A homomorphism need only respect the recorded operations and tests on V , and values outside V are irrelevant; this gives the second assertion. For the last, start from a, b and repeatedly apply a remainder fact of F whose operands are available and whose value is new. While some element of V is unavailable such a fact exists: take the first unavailable value in a sequence of remainder facts of F that generates that element. Each fact is applied at most once. $\square$

Proposition 7. *For coprime $a, b \geq 1$, $\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \perp\!\!\!\perp, a, b) = G(a, b)$. For $(a, b) \neq (0, 0)$,*

$$G(a, b) \leq \text{calls}_{\text{rem}}(\mathcal{N}_0, \gcd, a, b) \leq G(a, b) + 2,$$

and the same bounds hold for $\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \gcd, a, b)$.

Proof. A derivation of 1, together with $\text{eq}_1(1) = \text{tt}$, $\text{eq}_0(a) = \text{ff}$ and $\text{eq}_0(b) = \text{ff}$, certifies $\perp\!\!\!\perp$: under any homomorphism the input images are positive, and every common divisor of them divides the image of every generated value, hence divides $\pi(1)$, which is 1 because the fact $\text{eq}_1(1) = \text{tt}$ is preserved. Conversely, if a certificate records no fact $\text{eq}_1(x) = \text{tt}$, then $x \mapsto 2x$ preserves its

remainder facts and all its other test facts, but not coprimeness. So every certificate contains 1, and Lemma 6 gives a derivation of 1 with at most as many instructions as the certificate has remainder facts. This proves the first assertion.

For gcd as a function, the certificate must reach its output g , giving a lower bound of $G(a, b)$. For an upper bound, append

$$\text{rem}(a, g) = 0, \quad \text{rem}(b, g) = 0, \quad \text{eq}_0(0) = \text{tt}.$$

Every homomorphism π has $\pi(0) = 0$ and $\pi(g) > 0$ (by definedness of either appended remainder). Thus $\pi(g)$ divides both input images. Conversely every common divisor of those images divides $\pi(g)$ by induction over the derivation. This proves the bounds for $\mathcal{N}_0$. In $\mathcal{N}_\varepsilon$ the lower-bound argument is unchanged, and every certificate in $\mathcal{N}_0$ is also one in $\mathcal{N}_\varepsilon$, so both bounds transfer to $\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \text{gcd}, a, b)$. $\square$

Remark 8. 1. At $(0, 0)$ the output is an input, and the substructure with universe $\{0\}$ and no facts is a certificate.

2. For gcd restricted to positive inputs, appending $\text{eq}_0(a) = \text{ff}$ and $\text{eq}_0(b) = \text{ff}$ keeps input images in the domain, at no remainder cost.
3. For a positive noncoprime input, adding $\text{eq}_1(g) = \text{ff}$ to the gcd certificate certifies noncoprimeness. So the sublogarithmic bound holds for both truth values of coprimeness, and since each overhead is a fixed additive constant, Corollary 4 holds for all three complexities of Corollary 9 below on positive inputs.

The conjectures refuted. Moschovakis's 2012 notes state:

Conjecture 3.4 [9, p. 19]. There is a rational number $r > 0$ such that for infinitely many $a \geq b \geq 1$, $\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \perp\!\!\!\perp, a, b) \geq r \log a$.

The book formulates one precise version of the Main Conjecture [10, p. 151]: there are a set $\mathcal{D} \subseteq \mathbb{N}^2$ and a $K > 0$ such that

$$(x, y) \in \mathcal{D} \implies \text{calls}_{\text{rem}}(\text{Eucl}, x, y) \leq L(x, y) \leq K \text{calls}_{\text{rem}}(\mathcal{A}, \text{gcd}, x, y), \quad (4F-5)$$

where $L(x, y) = 2 \log \max(x, y)$ for $(x, y) \neq (0, 0)$, $L(0, 0) = 0$, and $\text{calls}_{\text{rem}}(\text{Eucl}, x, y)$ counts the remainder calls of the Euclidean algorithm Eucl (the book writes ε) on (x, y) . The book states (4F-5) for the Euclidean algorithm in $\mathbf{N}_\varepsilon$, so its $\mathcal{A}$ is $\mathcal{N}_\varepsilon$; we treat both $\mathcal{A} = \mathcal{N}_\varepsilon$ and $\mathcal{A} = \mathcal{N}_0$. The book adds that $\mathcal{D}$ then satisfies its conditions (4F-3) and (4F-4); the latter makes the intrinsic complexity unbounded on $\mathcal{D}$, so $\mathcal{D}$ is infinite (for finite $\mathcal{D}$ the statement is trivial). We refute the weaker statement obtained by dropping the first inequality and requiring only that $\mathcal{D}$ be infinite.

Corollary 9. *The intrinsic complexities*

$$\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \perp\!\!\!\perp, a, b), \quad \text{calls}_{\text{rem}}(\mathcal{N}_0, \text{gcd}, a, b), \quad \text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \text{gcd}, a, b)$$

are $O(\log a / \log \log a)$, uniformly in $1 \leq b \leq a$. Consequently Conjecture 3.4 of [9] is false, and so is the version (4F-5) of the Main Conjecture with $\mathcal{A} = \mathcal{N}_0$ or $\mathcal{A} = \mathcal{N}_\varepsilon$: for every $r > 0$ only finitely many pairs $a \geq b \geq 1$ satisfy $\text{calls}_{\text{rem}}(\mathcal{N}_\varepsilon, \perp\!\!\!\perp, a, b) \geq r \log a$, and for every $K > 0$ only finitely many pairs $(x, y) \in \mathbb{N}^2$ satisfy $L(x, y) \leq K \text{calls}_{\text{rem}}(\mathcal{A}, \text{gcd}, x, y)$.

Proof. By Theorem 1, Proposition 7 and Remark 8, each complexity is at most $G(a, b) + 2$ on positive inputs, which gives the first assertion; the refutations follow since $\log a / \log \log a = o(\log a)$. For (4F-5), symmetry covers $x < y$, and pairs with a zero entry need no remainder facts. $\square$

A single uniform process. The book's best-uniform-process theorem (Theorem 4F.1) states that, whenever every input has a certificate, one uniform process attains the intrinsic complexity at every input [10, pp. 149–150].

Corollary 10. *For each of gcd and coprimeness, there is a single uniform process of $\mathcal{N}_\varepsilon$ computing it on $\mathbb{N}^2$ with $O(\log a / \log \log a)$ remainder calls, uniformly in $1 \leq b \leq a$. The gcd process can also be taken over $\mathcal{N}_0$.*

Proof. Corollary 9 bounds the intrinsic complexities, and by Theorem 4F.1 one process attains them at every input. Every input has a certificate (symmetry covers reversed inputs, and a zero-test fact suffices when either input is zero), so this process is defined on all of $\mathbb{N}^2$. $\square$

The corollary rules out the possibility raised after Corollary 6C.7 of the book that the Main Conjecture holds for all uniform processes of $\mathcal{N}_\varepsilon$ [10, p. 188].

The limit of the homomorphism method. The Homomorphism Test proves lower bounds on intrinsic complexity by showing that every generated diagram below the proposed cost admits a homomorphism that fails to preserve the answer [10, Lemma 4E.3]. Such bounds apply to every uniform process computing the function, hence to every recursive program [10, Proposition 4E.1, Corollary 4E.2]. By Corollary 9, no lower bound on the intrinsic complexities listed there can grow faster than $\log a / \log \log a$, so a logarithmic lower bound for algorithms must use properties of programs not shared by all uniform processes, such as finite program syntax or determinism.

6 An almost-all lower bound

The difficult pairs of [10, Theorem 6C.5] give $\Omega(\log \log a)$ on a sparse set. An elementary Diophantine argument gives the same order for almost all coprime pairs. Write each generated value as $ua + wb$. On the B -regular pairs defined below, small coefficients cannot give a small positive value, and each remainder call at most squares the coefficient bound, up to a factor $2B$. Reaching 1 therefore takes about $\log \log a$ calls, except on a sparse set of pairs.

Proposition 11. *For integers $d \geq 1$ and $N \geq 3$, at most $40(2 + \ln N)N^{2-2^{-d}}$ coprime pairs $2 \leq b < a \leq N$ satisfy $G(a, b) \leq d$. Hence, for each fixed $\varepsilon > 0$, all but $o(N^2)$ of these pairs satisfy $G(a, b) > (1 - \varepsilon) \log \log N$.*

Proof. For $B \geq 1$ call (a, b) B -regular if $|qb - pa| \geq a/(Bq)$ whenever $q \in \mathbb{N}$, $q \geq 1$, $p \in \mathbb{Z}$ and $qb \neq pa$. We first bound the cost on regular pairs, then count the irregular ones.

Every generated value has an integer representation $v = ua + wb$. On a regular pair, $v > 0$ implies $v \geq a/(B(|u| + |w|))$: for $w \neq 0$ use regularity with $q = |w|$, and for $w = 0$ note $v \geq a$. Let H_t bound the coefficient norm $|u| + |w|$ in one such representation of each value available after t instructions, starting with $H_0 = 1$. In an instruction $r = \text{rem}(x, y)$, write $x = sy + r$. Since generated values are at most a , the quotient satisfies $s \leq a/y \leq BH_t$. Subtracting s times the representation of y from that of x gives

$$H_{t+1} \leq H_t + BH_t^2 \leq 2BH_t^2, \quad H_t \leq (2B)^{2^t-1}.$$

Producing 1 within d instructions therefore forces

$$1 \geq \frac{a}{BH_d}, \quad a \leq \frac{(2B)^{2^d}}{2}.$$

To count irregular pairs below N , choose a witness $0 < |qb - pa| < a/(Bq)$. Dividing p, q by their gcd preserves this inequality, so assume $\gcd(p, q) = 1$. The nonzero integer $e = qb - pa$ then gives $q < a/B \leq N/B$ and $0 \leq p \leq q$ (as $pa = qb - e$ with $0 < b < a$ and $|e| < a$). For each q , there are at most $2q$ choices of p and $2N/(Bq)$ choices of the signed integer e . Given p, q, e , the equation $pa \equiv -e \pmod{q}$ determines one residue class for a , leaving at most $N/q + 1$ choices; b is then determined. With $Q = \lfloor N/B \rfloor$, the number of irregular pairs is at most

$$\begin{aligned} \sum_{q=1}^Q (2q) \frac{2N}{Bq} \left(\frac{N}{q} + 1 \right) &= \frac{4N^2}{B} \sum_{q=1}^Q \frac{1}{q} + \frac{4NQ}{B} \\ &\leq \frac{4N^2}{B} (2 + \ln N), \end{aligned}$$

using $Q \leq N$, $B \geq 1$, and the harmonic-sum bound; an empty sum is zero.

Now split into dyadic ranges $X/2 < a \leq X$, and take $B = X^{2^{-d}}/2$. If $B \geq 1$, a regular pair with $G \leq d$ would have $a \leq X/2$, so all such pairs in the range are irregular. Their number is at most $8(2 + \ln X)X^{2-2^{-d}}$. When $B < 1$, the same bound follows from the trivial bound X^2 , since $X^{2^{-d}} < 2$. Put $s = 2 - 2^{-d} \geq 3/2$. Summing over the nonempty ranges with $X = N/2^j \geq 3$, and bounding their logarithmic factors by $2 + \ln N$, gives

$$8(2 + \ln N)N^s \sum_{j \geq 0} 2^{-js} = \frac{8}{1 - 2^{-s}} (2 + \ln N)N^s \leq 40(2 + \ln N)N^s.$$

(The factor is below 13; we state 40, the constant proved in Lean, which bounds each pair separately; see Appendix A.) For the second assertion, take $d = \lfloor \log \ln N - 2 \log \ln \ln N \rfloor$ for sufficiently large N (with $\log$ to base 2 and $\ln$ natural). Then $N^{-2^{-d}} \leq \exp(-(\ln \ln N)^2)$, so the exceptional set has size $o(N^2)$, while $d = (1 - o(1)) \log \log N$, so eventually $d \geq (1 - \varepsilon) \log \log N$. $\square$

Since there are asymptotically $3N^2/\pi^2$ coprime pairs $2 \leq b < a \leq N$, the exceptional set has density zero among them. The saving 2^{-d} in the exponent of $N^{2-2^{-d}}$ shrinks as d grows, so this argument does not reach the upper-bound scale $\log N / \log \log N$.

7 Fixed programs with arbitrarily small coefficients

For a nondeterministic program P , let $c_P(a, b)$ be the minimum number of remainder calls among its convergent computations. We require at least one such computation on each input and require every convergent computation to return the gcd. These are the conventions of [10, (2E-2), (2E-5)].

A *register program* keeps its values in a fixed number of registers and executes assignments $x_i := E$, branches on a test E , and a halt returning E , where each E is a term in the registers; the nondeterministic version allows several commands at a line [10, §§2E-2F]. It is an iterator, so it uses no recursion stack, and in pointed structures register programs compute exactly the tail-recursive functions [10, Proposition 2F.1].

Theorem 12. *For each integer $p \geq 1$, there is a finite nondeterministic register program P_p over $\mathcal{N}_0$ with $3 \cdot 2^p + 6$ registers that computes the gcd and satisfies*

$$\limsup_{a \rightarrow \infty} \max_{1 \leq b \leq a} \frac{c_{P_p}(a, b)}{\log a} \leq \frac{3}{p}.$$

Consequently, for every $\varepsilon > 0$ there are a fixed register program P and a constant $B \geq 0$ such that

$$\forall a \geq b \geq 1, \quad c_P(a, b) \leq \varepsilon \log a + B.$$

Proof. Put $k = 2^p$, $h = 3k$ and $W = h+6$: two inputs, $h+1$ prefix values, a modulus and two scratch registers. Initialise the registers $x_0, \dots, x_{W-1}$ at $(a, b, a, b, a, \dots, a)$. If $b = 0$, return a ; otherwise enter the main line. There the program may perform any assignment $x_l := \text{rem}(x_i, x_j)$ with $i, j, l \geq 2$, or pass to verification with a candidate $x_i \neq 0$, $i \geq 2$. Verification sets $x_0 := \text{rem}(x_0, x_i)$ and $x_1 := \text{rem}(x_1, x_i)$, halts with x_i if both are 0, and loops otherwise; the main line never writes x_0, x_1 . Every register value is divisible by $\gcd(a, b)$, so a candidate that passes verification is the gcd, and Euclid's algorithm gives a convergent computation. The computations below move values between registers for readability; copying is never needed (Appendix A).

Multiplying every value by $g = \gcd(a, b)$ turns a computation on $(a/g, b/g)$ into one on (a, b) with the same remainder count, so it suffices to treat coprime pairs. Let $S(n)$ be the maximum, over coprime $1 \leq b \leq a \leq 2^n$ and over states with a, b in two work registers, of the least number of remainder calls with which main-line moves of P_p place 1 in a work register. Fix $0 < \beta < 1/2$ and put $\alpha = 1 - \beta/2$ and $N = 2^n$.

Compute the Euclidean remainders $r_0 = a, r_1 = b, r_2, \dots$ in working registers, stopping at r_h or at the first $r_j \leq N^\alpha$. If 1 has been reached, no further generation is needed. In the remaining early case, one further remainder gives the positive coprime pair r_j, r_{j+1} , bounded by N^α , at total cost at most h ; continue with that pair, in the same registers. This restart needs no stack, but the two recursive derivations of Phase (iii) in Section 4 would, so the final step below uses Euclid's algorithm instead. Otherwise $r_h > N^\alpha$. As in the late case of Corollary 4, now with p fixed, the prefix lemma applies to every modulus above N^β for large n , and by (2) at most $\lceil n/p \rceil$ contractions, using a modulus register and two scratch registers, produce a positive $m \leq N^\beta$. Then put $u = \text{rem}(a, m)$ and $v = \text{rem}(b, m)$, and run Euclid's algorithm in place on m, u and then on the result d and v . Each run takes at most $2 \log m + 3$ calls, and the result is $\gcd(d, v) = \gcd(m, a, b) = 1$. Thus, eventually in n ,

$$S(n) \leq \max \{S(\lceil \alpha n \rceil) + h, \quad h + 3\lceil n/p \rceil + 4\beta n + 7\}.$$

Let $\Lambda = \limsup_{n \rightarrow \infty} S(n)/n$, which is finite since Euclid gives $S(n) = O(n)$. Passing to upper limits, Λ satisfies $\Lambda \leq \max\{\alpha\Lambda, 3/p + 4\beta\}$, and since $\alpha < 1$, $\Lambda \leq 3/p + 4\beta$. The program does not depend on β , so $\Lambda \leq 3/p$. Take $n = \lceil \log a \rceil$ and add the two verification calls. Finally, choose $p > 3/\varepsilon$ and absorb the finitely many smaller inputs in B . $\square$

The program depends on ε but not on the inputs. The Main Conjecture lets its coefficient r depend on the program, so the family is consistent with it; no single sublogarithmic program is obtained.

Recursive programs. Theorem 12 also holds for the book's recursive programs: writing one function per line of P_p yields a finite nondeterministic tail-recursive program with the same remainder cost [10, §§2A, 2C, 2E]. Lean verifies this translation for every register program on its model of the book's recursive machine, a specialisation of the book's Table 1, and proves an input-independent bound on the stacks (Appendix A).

8 Discussion and open problems

The gap. The intrinsic complexity of coprimeness in $\mathcal{N}_\varepsilon$ is $\Omega(\log \log a)$ on almost all coprime pairs and $O(\log a / \log \log a)$ on all positive pairs. Closing this gap is open; unlike Bshouty's bound in his model, our upper bound is not known to be optimal.

One program. For this construction, an obstacle to one sublogarithmic program is reusable access to a prefix whose length grows with the input. A nondeterministic program can guess the close pair, but it must hold all $h + 1$ prefix values; regenerating them would cost about h calls per contraction. By Theorem 12, $W = 3 \cdot 2^p + 6$ registers give coefficient $3/p$, roughly $3/\log W$, so no lower bound $c \log a$ with $c > 0$ independent of the number of registers holds for nondeterministic register programs. A register program has only a fixed number of slots for prefix values. Does every nondeterministic register program computing gcd with W registers make at least $c \log a / \log W$ remainder calls on infinitely many inputs, for an absolute $c > 0$? Such a bound cannot rest on how quickly values shrink, since $\text{rem}(m + 1, m) = 1$. Pratt’s algorithm is itself a nondeterministic register program [10, Theorem 2E.2]; compare also the book’s question whether recursion reduces to tail recursion in the pointed Euclidean structure [10, Problem x2G.6].

Recursion permits unbounded saved state, but does not directly provide reusable access to an arbitrarily long prefix. Can one fixed recursive program retain and repeatedly access a growing prefix at the same remainder cost? Every certificate becomes a computation of some program [10, Proposition 2E.1], but not of one program for all certificates.

Stronger lower-bound methods. The limit established in Section 5 leaves room for arguments combining homomorphisms with additional restrictions on computations. In particular, the conjecture of a logarithmic lower bound for deterministic uniform processes remains open [9, Conjecture 3.9, p. 23], [10, §4H and p. 188].

9 Computational checks

The Python planner in the accompanying repository¹ chooses instructions offline with unrestricted arithmetic, using the parameters of Theorem 1 from Section 4; below a base size (256 bits by default) it uses Euclid’s algorithm. A separate replay script then checks operand availability, nonzero divisors, every remainder equation and the final gcd.

A benchmark script samples 25 coprime pairs at each size: a is uniform among integers of that bit length, and b is drawn uniformly from $\{2, \dots, a - 1\}$ until $\text{gcd}(a, b) = 1$. It also runs the successive Fibonacci pair (F_k, F_{k-1}) with F_k the first Fibonacci number of that bit length. The table gives remainder calls; the Euclidean count stops when 1 first appears.

Input bits	Random pairs: min / median / max		Fibonacci pair	
	Construction	Euclid	Construction	Euclid
512	193 / 210 / 228	266 / 295 / 337	53	735
2048	612 / 626 / 644	1136 / 1192 / 1260	106	2948
8192	1918 / 1957 / 1985	4664 / 4774 / 4890	212	11798

All 78 derivations and the example of Section 3 pass replay. On the Fibonacci pairs, Euclid’s worst case, the bits removed by successive contractions roughly double (32, 66, 131 and 248 at 512 bits), so few contractions are needed. These samples illustrate the construction; the asymptotic claim rests on the proof. With the parameters of Corollary 4 and c near 1, the prefix exceeds the whole Euclidean sequence at these sizes, so the coefficient 3 is asymptotic only. The accompanying repository also checks scaled and noncoprime inputs, rejects invalid traces, and tests the prefix lemma over a finite range. The repository README gives scripts, commands, test ranges, seeds and individual results. The Python checks are not formally verified; the unrestricted arithmetic used to plan instructions is outside the derivations.

¹<https://github.com/jasondavies/euclid-uniform-processes>; this paper describes release v1.0.

A Formal verification

The development uses Lean 4.35.0 [3] with mathlib [8] at its v4.35.0 tag; the pinned mathlib commit is 56a005855ea17e0b0958f73ec801ae140b0a1679. The proved library contains no proof placeholders or added axioms; the axiom audit reports only `propext`, `Classical.choice`, and `Quot.sound`. Table 1 gives the main declarations; the repository README gives the complete map and reproduction commands. The general theory of uniform processes, including Theorem 4F.1 used in Corollary 10, and the historical lower bounds are cited, not formalised. The certificate bounds and conjecture refutations are formalised independently of that theory, with certificates represented by their facts. The consequence of Lemma 6 that we use, equality of the book-style and fact-based complexities, is formalised too: Lean states the book’s definitions for $\mathcal{N}_\varepsilon$ and $\mathcal{N}_0$ directly (finite partial substructures with partial operations, generation, homomorphisms on the universe, certification and the norm `callsrem`) and proves that the resulting intrinsic complexities equal the fact-based ones used in the refutations. Corollary 5 is formalised with the source’s definitions of `iq`, `rem` and the closure stages, including their conventions at zero divisors.

The development is also packaged for the comparator of the Palomar registry: `Challenge.lean` imports only mathlib and states the main theorems with every definition visible and deliberate `sorry` placeholders, and `Solution.lean` imports the proved library, independently of these placeholders. The comparator checks that the proved statements are exactly those of `Challenge.lean` and replays the proofs through Lean’s kernel and the independent kernels NanoDa and con-ron; we ran it on release v1.0 of the repository. `comparator.json` lists all twenty-two compared theorems, including that G and the closure depth are attained (`generates_G`, `ClosureDepth.depth_mem`), so the bounds on them are not about empty infima; Table 1 marks those included here with [†]. Version 2 of the Palomar registry entry PALOMAR-2026-10-23-000012 [11] records this verification for release v1.0 (commit 2548681), covering all twenty-two compared theorems.

Arithmetic bounds. Costs are lengths of explicit remainder derivations. The finite theorem `sizeCost_step` proves the recurrence from natural-number size parameters; it does not assume a recurrence for the cost function. For the initial Big-O proof, Lean uses the parameters of Section 4, computing p as $\lfloor \log n \rfloor / 2$, which equals $\lfloor (\log n) / 2 \rfloor$, and checks the integer rounding estimates before passing to real logarithms. The refinement uses the tunable parameters of Corollary 4 and checks the rounding limits preserving coefficient 3. For Proposition 11, Lean takes $B = a^{2^{-d}} / 2$ separately for each pair instead of splitting into dyadic ranges. It proves the stated constant 40 and the exceptional-set limit for every fixed $0 < \varepsilon \leq 1$ (larger ε are trivial), with threshold $(1 - \varepsilon) \log \log N$.

Programs and machine computations. For Theorem 12, Lean defines the book’s register programs over $\mathcal{N}_0$ (`Register.Program`): lines numbered from 0, each a halt or a list of alternative assignments and branchings, with the book’s prime terms x_i , `rem`(x_i, x_j), `tt`, `ff` and `eq0`(x_i), and the book’s iterator semantics, in which `rem`($x, 0$) is undefined. The program P_p is written in this syntax (`TailSource.program`), with no stack, and each remainder evaluation is charged once, including the two verification calls. Every convergent execution returns the `gcd`, and one exists on every input. The cost proofs use a generator that may also relabel all registers at once. `TailSource.run_trace_relabel` replays such runs by single-register commands at the same cost and with no extra registers: it tracks the relabelling in the proof, and writes each new remainder to a register no other value occupies, which exists by pigeonhole whenever two names share a register.

The tail-recursive form (`TailRecursive`) compiles the register program into a language of non-deterministic first-order programs (`Recursive`): terms with variables, remainder, a zero-test con-

ditional and calls of internal symbols, each symbol with a set of alternative bodies, evaluated call by value. The compiler (`Register.compile`) follows the book’s reduction of iteration to tail recursion [10, §2C]: each line becomes a function of the W registers, an assignment a tail call with one argument replaced, a branching on $\text{eq}_0(x_i)$ a zero-test conditional between two tail calls and one on `tt` or `ff` a single tail call, a halt its term, and alternative commands alternative bodies. For every register program, `Register.eval_iff_halts` matches evaluations exactly, in value and remainder cost, with convergent computations; every call is a tail call, whose arguments, remainder operands and conditional guards contain no internal calls (`Register.compile_tailRecursive`); and the machine’s control and value stacks never hold more than $2W+4$ entries in all, on any input and however long the computation runs (`Register.stacks_bounded`). For Theorem 12, $W = 3 \cdot 2^p + 6$, so the stacks never exceed $6 \cdot 2^p + 16$ entries (`TailRecursive.stacks_bounded`); each closure environment has at most W entries. For every program in the language, `Recursive.Machine.eval_iff_steps` proves exact-cost equivalence with a specialisation of the book’s two-stack machine: arguments are evaluated right to left, zero tests and branches are separate transitions, and only remainder transitions are charged [10, Table 1, p. 75]. Closures represent substitution in machine states, not extra program data.

Table 1: Main results and Lean declarations, all in the namespace `ShortGCD`. Names marked $\dagger$ are compared against `Challenge.lean`.

Paper	Lean declaration(s)
Theorem 1	<code>uniform_real_bound</code> , <code>generates_G$\dagger$</code>
Normalisation (1)	<code>G_normalize</code>
Lemma 2	<code>small_remainder</code>
Lemma 3	<code>prefix_class_card_le_three</code> , <code>residue_card_three</code> ; for (2), <code>prefix_reservoir</code>
Lemma 6 (consequence used)	<code>Book.callsCoprimeeness_eq$\dagger$</code> , <code>Book.callsRemGcd_eq$\dagger$</code> , <code>Book.callsGcd_eq$\dagger$</code>
Corollary 4	<code>uniform_real_bound_three$\dagger$</code> , <code>uniform_intrinsic_bound_three$\dagger$</code>
Proposition 7	<code>callsGcd_le</code> , <code>callsGcd_zero_left</code> , <code>callsRemGcd_le_callsGcd</code> , <code>exists_certifiesGcd$\dagger$</code> , <code>G_le_callsRemGcd$\dagger$</code> , <code>G_le_callsGcd$\dagger$</code> , <code>callsCoprimeeness_eq_G$\dagger$</code>
Remark 8	<code>callsCoprimeeness_le</code> , <code>exists_certifiesCoprimeeness$\dagger$</code>
Corollary 9	<code>not_conjecture_3_4$\dagger$</code> , <code>not_main_conjecture_4F5$\dagger$</code> , <code>not_main_conjecture_4F5_Neps$\dagger$</code> , and the literal readings <code>not_conjecture_3_4_literal$\dagger$</code> , <code>not_main_conjecture_4F5_literal$\dagger$</code>
Corollary 10	Cited Theorem 4F.1; not formalised here
Corollary 5	<code>ClosureDepth.depth_le</code> , <code>ClosureDepth.depth_mem$\dagger$</code> , <code>ClosureDepth.depth_bound</code> , <code>ClosureDepth.no_logarithmic_sequences$\dagger$</code>
Proposition 11	<code>AlmostAll.card_D_le</code> , <code>G_eq_D</code> , <code>tendsto_card_callsCoprimeeness_loglog$\dagger$</code>
Theorem 12	<code>TailProgram.Fixed.limsup_profile_le</code> , <code>TailProgram.Fixed.eventually_accepts_bound</code> , <code>TailSource.executes_iff</code> , <code>TailSource.total_registers</code> , <code>TailSource.executes_correct$\dagger$</code> , <code>TailSource.executes_total$\dagger$</code> , <code>TailSource.arbitrarily_small_coefficient$\dagger$</code>
Recursive programs (Section 7)	<code>TailRecursive.program_tailRecursive</code> , <code>TailRecursive.stacks_bounded</code> , <code>TailRecursive.executes_iff_steps</code> , <code>TailRecursive.arbitrarily_small_coefficient</code>

Acknowledgements

I thank Shubhendu Trivedi for helpful comments.

Declaration of AI-generated research and authorship

This manuscript is the product of an AI-driven research process. During a broader investigation initiated by the human author, an AI system, GPT-6-Astra, identified the principal result. The author then directed AI systems to formalise the results in Lean and to prepare this manuscript. The AI systems developed the results and their presentation, including the mathematical exposition, the formalisation, and the organisation and writing of the text; the human author wrote no part of the manuscript. The systems were GPT-6-Astra (OpenAI), used through Codex, and Claude Opus 5.5 (Anthropic), used through Claude Code. Both contributed to the Lean formalisation and to the review and revision of the manuscript.

The Lean development gives a machine-checkable version of the main claims. Lean’s kernel checks the proofs, including the mathlib results they use, relative to the axioms listed in Appendix A. That includes the equality of the book-style and fact-based complexities, the consequence of Lemma 6 that we use; the book’s general theory of uniform processes, used through the cited Theorem 4F.1, is not formalised.

The human author has checked the arguments and takes full responsibility for the content of this paper, including the correctness of its mathematics, and for the accuracy of this disclosure.

References

- [1] Nader H. Bshouty. *Euclidean GCD algorithm is not optimal*. Research Report 1989-376-38, Department of Computer Science, University of Calgary, December 1989. [doi:10.11575/PRISM/30475](https://doi.org/10.11575/PRISM/30475).
- [2] Benny Chor and Oded Goldreich. An improved parallel algorithm for integer GCD. *Algorithmica* 5(1–4):1–10, 1990. [doi:10.1007/BF01840374](https://doi.org/10.1007/BF01840374).
- [3] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, Lecture Notes in Computer Science, vol. 12699, pp. 625–635. Springer, 2021. [doi:10.1007/978-3-030-79876-5_37](https://doi.org/10.1007/978-3-030-79876-5_37).
- [4] Ravindran Kannan, Gary Miller and Larry Rudolph. Sublinear parallel algorithm for computing the greatest common divisor of two integers. *SIAM Journal on Computing* 16(1):7–16, 1987. [doi:10.1137/0216002](https://doi.org/10.1137/0216002).
- [5] Donald E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, Reading, MA, 1997.
- [6] Yishay Mansour, Baruch Schieber and Prasoon Tiwari. A lower bound for integer greatest common divisor computations. *Journal of the ACM* 38(2):453–471, 1991. [doi:10.1145/103516.103522](https://doi.org/10.1145/103516.103522).
- [7] Yishay Mansour, Baruch Schieber and Prasoon Tiwari. Lower bounds for computations with the floor operation. *SIAM Journal on Computing* 20(2):315–327, 1991. [doi:10.1137/0220020](https://doi.org/10.1137/0220020).

- [8] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pp. 367–381. ACM, 2020. doi:10.1145/3372885.3373824.
- [9] Yiannis N. Moschovakis. *Absolute lower bounds for uniform processes*. Unpublished lecture notes (first draft), January 25, 2012. <https://www.math.ucla.edu/~ynm/nadalb.pdf>.
- [10] Yiannis N. Moschovakis. *Abstract Recursion and Intrinsic Complexity*. Lecture Notes in Logic, vol. 48. Cambridge University Press and Association for Symbolic Logic, 2019. doi:10.1017/9781108234238.
- [11] Palomar registry. Entry PALOMAR-2026-10-23-000012, version 2, for commit 2548681 of <https://github.com/jasondavies/euclid-uniform-processes>. <https://palomar-registry.org/entry?id=PALOMAR-2026-10-23-000012&version=2>.
- [12] J. Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics* 1(3):397–405, 1967. doi:10.1016/0021-9991(67)90047-2.
- [13] Lou van den Dries and Yiannis N. Moschovakis. Is the Euclidean algorithm optimal among its peers? *Bulletin of Symbolic Logic* 10(3):390–418, 2004. doi:10.2178/bsl/1102022663.
- [14] Lou van den Dries and Yiannis N. Moschovakis. Arithmetic complexity. *ACM Transactions on Computational Logic* 10(1), Article 2, 49 pp., 2009. doi:10.1145/1459010.1459012. Preprint: <https://www.math.ucla.edu/~ynm/papers/ac.pdf>.