hexagon:2610.00711v1

An Improved Subexponential Bound for Deterministic Integer Factorization

Joseph M. Shunia*

October 30, 2026

Abstract

We prove that the complete prime factorization of an n-bit integer can be computed deterministically in time $2^{O(n/\sqrt{\log n})}$ and polynomial space. This establishes the bound conjectured by Rudich and Rousseau, sharpening their qualitative $N^{o(1)}$ result. The proof combines OpenAI's quantitative Boolean tensor-contraction theorem with the linear-size multiplication encoding of Akshima et al. The encoding permits exact counting of divisors with prescribed binary prefixes without loss in the exponent. The least prime factor is then recovered by successive prefix queries, and iteration yields the complete factorization within the same asymptotic bound.

2020 Mathematics Subject Classification. Primary 11Y05; Secondary 11Y16, 68Q25.
Key words and phrases. Integer factorization, deterministic algorithms, subexponential time, exact counting, Boolean tensor networks.

1 Introduction

Rudich and Rousseau [3] obtained a deterministic algorithm for integer factorization with running time $N^{o(1)}$. They further conjectured that the running time can be bounded by

$$\exp\Big(O\Big(\frac{\log N}{\sqrt{\log\log N}}\Big)\Big). \tag{1}$$

We prove this refinement by combining the quantitative contraction theorem of OpenAI [2] with the multiplication encoding of Akshima et al. [1].

For $N \geq 2$, write $n = \lfloor \log_2 N \rfloor + 1$ for its binary length. Running times are measured in bit operations, and space in bits. All logarithms are to base two; their base does not affect the asymptotic bounds.

Theorem 1. *There is a uniform deterministic algorithm which computes the least prime factor and the complete prime factorization of every n-bit integer $N \geq 2$ in time*

$$2^{O(n/\sqrt{\log n})}$$

and space $n^{O(1)}$.

Since $n = \Theta(\log N)$, this is the bound in (1), and hence proves Conjecture 1 of [3]. The contribution is the quantitative estimate. The search reduction is the divisor-prefix procedure of Rudich and Rousseau: count divisors with a prescribed initial binary word, then recover the least prime factor one bit at a time. To retain the desired running time, however, the counting instances must have size linear in n.

*Independent researcher, Ann Arbor, Michigan, United States. Email: jshunia@gmail.com.

The multiplication network of Akshima et al. has $O(n)$ edges and maximum degree $O(n^{3/4})$. Fixing its product word to N and restricting the leading bits of one factor preserves both bounds. OpenAI's contraction theorem, which allows degree $O(s^{9/10})$ on a network with s edges, therefore evaluates each count in $2^{O(n/\sqrt{\log n})}$ time. Polynomially many such counts suffice for complete factorization. The inequalities between these size and degree bounds are what allow the conjectured rate to pass through the reduction unchanged.

We recall the two results in the form needed here, then give the counting reduction and the factorization algorithm. We also record the integer-rounded version of the bound used in the formulation of the conjecture.

2 Boolean tensor networks

A *Boolean tensor network* consists of a finite undirected multigraph $G = (V, E)$ without self-loops and, at each vertex v, a predicate $P_v : \{0,1\}^{\deg(v)} \to \{0,1\}$. The edges incident to each vertex are ordered. Its contraction is

$$\mathrm{Count}(G) = \sum_{x \in \{0,1\}^E} \prod_{v \in V} P_v(x|_{E(v)}),$$

where $x|_{E(v)}$ denotes the ordered tuple of incident edge bits. Thus the contraction counts the assignments satisfying all vertex predicates. The predicates are specified by algorithms for evaluating them, rather than by their full truth tables.

The following form of OpenAI's contraction theorem is recorded in [1, Theorem 2].

Theorem 2 (OpenAI [2, Theorem D.1])**.** *Let G be a Boolean tensor network with $s \geq 2$ edges and maximum degree $O(s^{9/10})$. Suppose that the number of vertices and the total description length are polynomial in s, and that the vertex predicates are uniformly evaluable in polynomial time. Then* $\mathrm{Count}(G)$ *can be computed deterministically in time*

$$2^{O(s/\sqrt{\log s})}$$

and polynomial space. The constants are uniform for fixed bounds on the degree, description length, and predicate evaluation time.

For completeness, we explain the specialization from the original statement; see also [1, Remark 1]. That statement requires a bipartite graph. Subdividing every edge by a vertex carrying the binary equality predicate produces such a graph, doubles the edge count, and preserves the contraction. The maximum degree becomes at most the larger of 2 and the original degree. The original statement also computes over a finite coefficient ring. Taking $\mathbb{Z}/2^{s+1}\mathbb{Z}$ recovers the exact integer count, since

$$0 \leq \mathrm{Count}(G) \leq 2^s < 2^{s+1}.$$

The required precision is linear in the network size, including after subdivision. These changes therefore preserve the asserted running time and space.

To encode an arithmetic relation, one designates distinct *interface edges* for its argument bits. An encoding is *parsimonious* if each valid assignment to the interface extends to exactly one satisfying assignment on the remaining edges, and an invalid assignment has no such extension.

Theorem 3 (Akshima et al. [1, Theorem 3])**.** *For every $d \geq 1$, the relation $ab = z$, with $0 \leq a, b < 2^d$ and $0 \leq z < 2^{2d}$, has a parsimonious Boolean tensor-network encoding with fewer than $1100(d+1)$ edges and maximum degree less than $136(d+1)^{3/4}$. The network is uniformly constructible in polynomial time, and its vertex predicates are uniformly evaluable in polynomial time.*

The construction in [1] uses a two-stage Fourier-transform decomposition to obtain the linear edge bound. Canonical representations of the intermediate words ensure parsimony. We shall use both properties: the size bound controls the running time, while uniqueness of the auxiliary assignments identifies the contraction with the desired divisor count.

3 Divisor counts and factorization

Represent every integer in $[0, 2^n)$ by a word of length n, allowing leading zeros. For a binary word u of length at most n, let

$$C_N(u) = \#\{d : d \mid N,\ d \geq 2, \text{the } n\text{-bit word for } d \text{ begins with } u\}. \tag{2}$$

In particular, the empty word is a prefix of every divisor.

Lemma 4. *The count $C_N(u)$ can be computed deterministically in time $2^{O(n/\sqrt{\log n})}$ and polynomial space, uniformly in N and u.*

Proof. Apply Theorem 3 with $d = n$. Fix the product word z to the $2n$-bit representation of N, and fix the first $|u|$ bits of a to u. Each fixed-bit condition can be conjoined to the predicate at one endpoint of its interface edge. This leaves the graph unchanged and preserves uniqueness of the auxiliary assignments. The modified predicates remain uniformly evaluable in polynomial time.

The satisfying assignments now correspond bijectively to the positive divisors a of N whose n-bit representations begin with u. Indeed, each such divisor determines its unique cofactor $b = N/a$, and both are at most $N < 2^n$. Parsimony gives exactly one extension to the remaining edges. Conversely, a satisfying assignment has $ab = N > 0$ and therefore supplies such a divisor. To obtain $C_N(u)$, subtract 1 from the contraction if u is a prefix of the n-bit representation of 1, and subtract 0 otherwise.

The network has $s = \Theta(n)$ edges: its $4n$ interface edges give the lower bound, and Theorem 3 gives the upper bound. Its maximum degree is $O(n^{3/4}) = O(s^{3/4})$, which satisfies the hypothesis of Theorem 2. Its construction and description have polynomial size and cost. The contraction, and hence the required count, is therefore computed in time

$$2^{O(s/\sqrt{\log s})} = 2^{O(n/\sqrt{\log n})}$$

and polynomial space. The count is at most N, so its binary representation requires only $O(n)$ bits. All bounds are uniform over the prefixes of length at most n. □

Proof of Theorem 1. Start with the empty word u. The invariant $C_N(u) > 0$ holds initially because N itself is counted. At each step, compute $C_N(u0)$, where juxtaposition denotes concatenation. Replace u by $u0$ if the count is positive, and by $u1$ otherwise. Continue until u has length n.

For $|u| < n$, the divisors counted by $C_N(u)$ partition according to their next bit, so

$$C_N(u) = C_N(u0) + C_N(u1).$$

The invariant is therefore preserved. Moreover, at every step the smaller next bit is chosen whenever it admits a completion. The final word is consequently the lexicographically least word representing an eligible divisor. Since fixed-length binary words have the same lexicographic and numerical orders, this is the smallest divisor of N greater than 1. It is prime, for otherwise one of its proper prime divisors would be a smaller eligible divisor of N.

This procedure uses n calls to Lemma 4. Having found the least prime factor, divide out all its powers and repeat on the remaining cofactor, stopping at 1. There are at most n iterations,

each on an integer of at most n bits, and hence at most n^2 counting calls. The remaining arithmetic takes polynomial time. Thus the total running time is

$$n^{O(1)}2^{O(n/\sqrt{\log n})} = 2^{O(n/\sqrt{\log n})}.$$

The calls are made sequentially and their workspaces reused. The current cofactor, prefix, count, and accumulated factorization require only polynomial space. Every step, including construction of the counting networks, is uniform and deterministic. □

Corollary 5. *Conjecture 1 of Rudich and Rousseau [3] holds. More precisely, there is a positive integer C and a deterministic machine with finite alphabets which computes the least prime factor of every n-bit integer $N \geq 2$ within*

$$C\,2^{\left\lfloor \frac{Cn}{\lfloor\sqrt{\lfloor \log_2 n \rfloor}\rfloor + 1} \right\rfloor}$$

steps.

Proof. For $n \geq 2$, the denominator in the exponent is $\Theta(\sqrt{\log n})$. Enlarging C absorbs the constants in Theorem 1, the integer rounding, and the finitely many small input lengths. The algorithm admits a finite-alphabet machine implementation; a polynomial simulation overhead changes only the constant in the exponent. If the inputs 0 and 1 are included, their prescribed outputs are handled separately. □

The linear edge bound is essential to this deduction. Substituting only an $O(n \log n)$ size bound into Theorem 2 would yield

$$2^{O\left(n \log n/\sqrt{\log(n \log n)}\right)} = 2^{O(n\sqrt{\log n})},$$

which does not give subexponential time in n. The multiplication encoding of Theorem 3 removes this loss without exceeding the allowed degree. It is this combination that establishes the conjectured rate. The estimate concerns asymptotic complexity; determining useful concrete running times would require an explicit analysis of the contraction algorithm's constants.

Acknowledgment

OpenAI's ChatGPT assisted the author in developing, drafting, and checking the argument. The author supplied the source manuscripts and directed the investigation.

References

[1] Akshima, R. Bacho, Y.-H. Huang, E. Kiltz, A. Nishaburi, S. Nooripoor, N. Tittelbach, and E. Wiedijk, *A Subexponential Time Algorithm for Elliptic Curve Discrete Logarithm*, preprint, version 0.2, October 29, 2026. https://github.com/bochum-crypto/subexponential-ecdl.

[2] OpenAI, *The Exponential Time Hypothesis is False*, preprint, October 17, 2026; revised October 25, 2026. https://github.com/openai/math/blob/main/preprints/The-Exponential-Time-Hypothesis-is-False-October-25-2026/paper.pdf.

[3] I. Rudich and L.-M. Rousseau, *Deterministic Factoring in Time $N^{o(1)}$*, Hexagon preprint 2610.00600, version 1, October 2026. https://hexagonmath.org/2610.00600v1.