A finite range where the SubexponentialSAT bound's exponential factor sits above 2^n with $A = 1$, and the receipt of a comparator rerun

Luiz Antonio Busnello (Independent Researcher, ORCID 0009-0007-5911-8141)

Version 1.0.2 · 30 October 2026 · doi:10.5281/zenodo.23820111

Abstract

Two checkable things about the SubexponentialSAT challenge of `openai/math`. The first is the receipt of a rerun of leanprover's comparator on the published solution, carried out under a real sandbox on 2026-10-28, with the provenance needed to repeat it and the raw log of all three runs, the two earlier failures included. Those two were failures of the preparation script rather than verdicts on the proof. Three limits of the run are stated here rather than left to be discovered: it used Lean's default kernel, so it is a rerun and not a check by a second kernel implementation; the recorded commit identifies the comparator source rather than the binary executed; and a comparator result says nothing about whether a formal statement means what the informal claim says.

The second is a recomputation of the stated time bound. Dividing runtimeScale(n) by n leaves $\ln(L)^2/\sqrt{L}$ with $L = \ln(n + 16)$, which tends to zero, so the exponent is $o(n)$. What the asymptotics do not show is that this ratio is not monotone: it starts below 1, rises to a maximum of $16/e^2$ at $L = e^4$, and falls back. Stipulating $A = 1$, it crosses 1 twice, and both crossings are closed-form in the Lambert W function through $L = \exp(-4W(-1/4))$ on the two real branches: ascending at $n = 49.175096$ and descending near $10^{2390.21}$.

What this does not establish matters as much as what it does. An upper bound above 2^n says nothing about running time, because the algorithm may run far below its own bound. The exponent is $o(n)$ for any fixed A, so a finite range cannot bear on an asymptotic statement, and nothing here refutes anything. The constant A is existential in the statement and is not determined by it. Two further conditions are left open and are claimed nowhere: that the formal statement is a faithful rendering of k-SAT, and that the unary encoding keeps the input size polynomial.

1. The comparator accepts the solution

On 2026-10-28 the comparator from `leanprover/comparator`, run under a real sandbox, reported that the Lean kernel accepts the solution for the challenge's own declaration, `OAI.Algorithms.SubexponentialSAT.SqrtLogSpace.exact_co`
Of the solution's declarations that print their own axiom dependencies, 19 in total, every one declares exactly `[propext, Classical.choice, Quot.sound]`, the three the challenge permits. That is a statement about those 19 outputs and not about the whole development: whether the final declaration is among them needs the log, which I did not check and which is included with this note.

Three limits of that run, stated so nobody has to find them:

1. It used Lean's default kernel, so it did not provide an independent check from a second kernel implementation. The challenge sets `enable_nanoda` to false and the log reads `Running Lean default kernel`.
2. The binary is the comparator from `leanprover/comparator` at the commit recorded below, and that commit identifies the source rather than the binary I executed.
3. A comparator result says nothing about whether the formal statement means what the informal claim says.

2. Reading the formal statement by hand

What follows is four checks rather than a verdict. Three found no mismatch; the fourth I could not complete, and I say where it stops.

Quantifier order: one fixed deterministic machine and fixed constants come before the universal quantifier over n and over formulas.

Locality of the transition function: it reads only the tape symbol designated by the current state.

Strength and reach: within the formal definitions, the statement counts satisfying assignments exactly, which is at least as strong as deciding satisfiability, and it covers every $k \geq 2$, so $k = 3$ is included.

Input encoding (not completed): the encoding is unary. I did not establish what that costs in encoding length. Unary encoding of an integer is in general exponential in its binary length, and whether it stays polynomial here depends on the bounds on the encoded integers and on the formula size, which I did not check.

3. The bound, and the ratio

The stated bound is

$$T \leq C \cdot (\text{bits} + 1)^d \cdot 2^{A \cdot \text{runtimeScale}(n)}, \qquad \text{runtimeScale}(n) = \frac{n \ln(\ln(n+16))^2}{\sqrt{\ln(n+16)}}$$

with C, A and d existential. Everything below concerns the exponential factor alone, taking $A = 1$. The polynomial factor and the actual value of A are not determined by the statement, and nothing here claims either.

Dividing runtimeScale(n) by n leaves

$$\text{ratio}(n) = \frac{\ln(L)^2}{\sqrt{L}}, \qquad L = \ln(n+16)$$

which tends to zero. For any fixed A, that makes the exponent $o(n)$ in the number of variables. Whether an $o(n)$ exponent contradicts the ETH needs the two conditions named in the abstract, neither established nor claimed here.

4. The ratio is not monotone, and both crossings are closed-form

What the asymptotics do not show is that ratio(n) starts below 1, rises to a maximum, and falls back. At $A = 1$ it crosses 1 twice. Writing $u = \ln L$, the equation ratio $= 1$ becomes

$$u^2 = e^{u/2} \quad \Longrightarrow \quad \left(-\tfrac{u}{4}\right) e^{-u/4} = -\tfrac{1}{4} \quad \Longrightarrow \quad L = \exp\left(-4\, W(-1/4)\right)$$

on the two real branches of the Lambert W function:

crossing	branch	L	n
ascending	principal, W_0	4.177077	49.175096
descending	W_{-1}	5503.6647	$\sim 10^{2390.2112}$

At those exact roots the ratio is exactly 1; the printed values are rounded and do not satisfy the equation exactly. Between the crossings the ratio rises to a maximum of $16/e^2 = 2.165$ at $L = e^4$, that is $n \sim 10^{23.7117}$.

The maximum needs no numerical search. Differentiating gives

$$\frac{d}{dL}\left[\frac{\ln(L)^2}{\sqrt{L}}\right] = \frac{\ln(L)\,(4 - \ln(L))}{2\,L^{3/2}}$$

which is positive for $1 < L < e^4$ and negative beyond, so the only interior maximum is at $L = e^4$, where the ratio is $4^2/e^2$.

The ascending crossing is written as 49.175096 rather than as a power of ten for readability at that scale. The +16 inside the logarithms still matters there: the same equation without it has its root at $n = 65.175096$, which is 16 higher, and the original ratio evaluated at that point is about 1.0458 rather than 1.

So, for $A = 1$ and for the exponential factor alone, there is a finite range of n in which that factor sits above 2^n. Since n counts variables and is therefore an integer, the range is $n = 50$ up to 10^{2390}: the crossing falls at 49.175096, so at $n = 49$ the ratio is still 0.999421 and at $n = 50$ it is 1.002702.

5. Two things that do not follow

It does not show the algorithm is slower than exhaustive search. An upper bound above 2^n says nothing about the running time, which may sit far below its own bound. The comparison here is between bounds, not between running times.

It does not change the asymptotics. The exponent is $o(n)$ for any fixed A, so a finite range cannot bear on an asymptotic statement.

The value of A decides how much of this is visible at all. With A below about $e^2/16 \approx 0.46$, the exponential factor stays under 2^n for every $n \geq 1$. With A above 1 the range is wider. Without A, the effective range is undetermined. The bound on n is not pedantry: at $n = 0$ the exponent is $A \cdot \text{runtimeScale}(0) = 0$, so both sides equal 1 and neither is below the other.

6. Provenance, and how to repeat it

```
examined clone  openai/math at 7ef5c0cc5ca4b0e072b7a050fca09ab24bc6fa3a
challenge json  sha256 debbfdc9c01104d2369be80da5180978275106df8b685f693fba7ce7da7c5b19
challenge lean  sha256 4183b2057cff350d66848215ff7167196dfc20e1891b598fdddea59077a76685
toolchain       leanprover/lean4:v4.34.1, declared in that repository's lean-toolchain
comparator      leanprover/comparator @ 865edc14486c3388e9225daa971da8e157488c3e
lean4export     5a9a355dc6c5fd82b69058d433e2ae2e8dfbf8fe
sandbox         landrun 0.1.18
cost            3 h 47 min for the verdict step, 12,014 build jobs
```

The two challenge hashes can be reproduced from the two files at that commit; I did that on 2026-10-29 and both match byte for byte.

The two parts differ in what checking them costs. The arithmetic above reproduces in seconds with nothing but a Python interpreter, and the script that does it runs eight self-tests before printing any number, plus a `--plant` mode that corrupts the transcribed formula and requires the self-tests to reject it. The comparator run took 3 h 47 min over 12,014 build jobs and needs a Linux host with Lean, elan and landrun installed, so for most readers the receipt and the raw log are the practical object rather than a rerun of this rerun.

Everything cited here, including the receipt, the raw log of all three runs and the script, is in the deposit: `https://doi.org/10.5281/zenodo.23820111`, with the working repository at `https://github.com/totobusnello/eth-sube`
MIT licence.