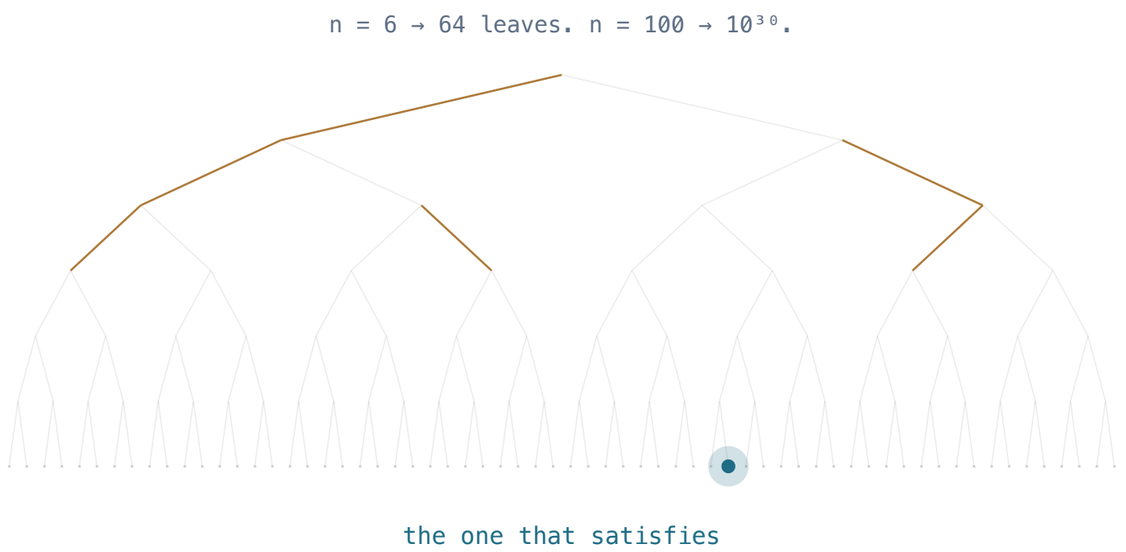
INTRODUCTION TO COMPUTER SCIENCE · WEEK 11 · STATION IV

Cost

Suppose a question is decidable. You still have to pay for the answer — in time, space, and energy.

DECIDABLE ≠ DOABLE

A question with a guaranteed answer you will never receive.



2ⁿ branches · one satisfying leaf

Given a logical formula over 100 yes/no variables: is there an assignment making it true? Perfectly decidable — try all 2^{100}.

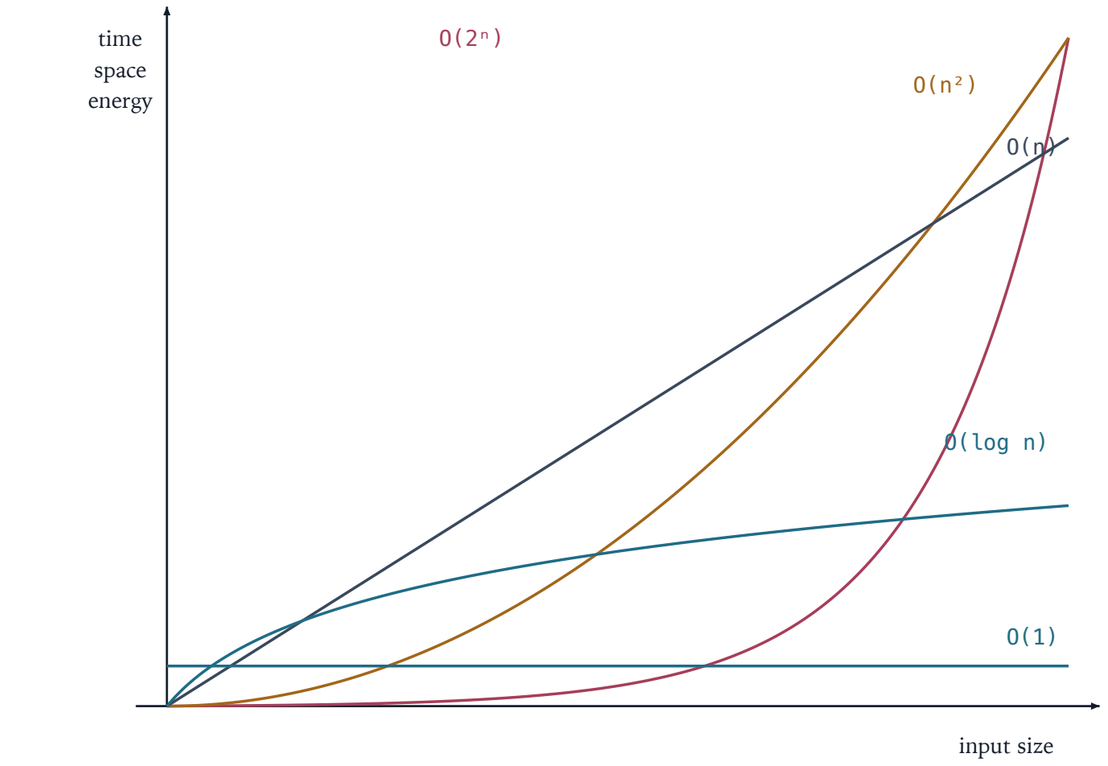
$$2^{100} \approx 1.3 \times 10^{30}$$

ASSIGNMENTS · 10^{13} YEARS AT A BILLION CHECKS PER SECOND

Chess has about 10^{44} legal positions; Go, 10^{170}. Nobody solves these. We navigate them — with heuristics, structure, and luck.

THREE CURRENCIES

Time, space, energy — and the only distinction that matters: polynomial or exponential.



cost against input size

Time is instructions, as the emulator counts them. Space is memory. Energy is what the wall delivers, and it has a law of physics attached — last slides.

A method that takes n^2 steps on an input of size n is polynomial; n can be a million before it hurts. A method that takes 2^n is exponential; n cannot be a hundred.

The questions some method answers in polynomial time form the class P. Sorting, routing, checking that a text is a C* program, running mipster for k steps: nobody argues about whether these are affordable.

INTRACTABILITY

Hard to find. Easy to check.

That asymmetry has a name: NP — problems whose solutions are quick to verify even if finding them seems to need exponential search. The satisfying assignment is the certificate; plug it in and check.

Cook and Levin, 1971–73: thousands of such problems are the same problem in disguise. Crack one efficiently and you crack scheduling, routing, folding, packing, proving. Whether that is possible — P = NP? — is the most consequential open question in the exact sciences. Most researchers bet no.

THE ASYMMETRY, EVERYWHERE

Writing a proof versus reading it. Designing a protein versus assaying it. Writing the program versus running the test. Doing the homework versus grading it.

REMEMBER THIS ONE

Generation is expensive; verification is cheap. Every healthy division of labour — and every safe way to use an AI — is built on that gap.

IN THE TERMINAL

A SAT solver in 400 lines.

```
$ cat examples/sat/rivest.cnf
p cnf 4 7
2 3 -4 0
1 3 4 0
-1 2 4 0
-1 -2 3 0
-2 -3 4 0
-1 -3 -4 0
1 -2 -4 0
$ make sat
./babysat: 7 clauses with 4 declared variables loaded from
./babysat: examples/sat/rivest.cnf is satisfiable with -1 -
```

A formula in conjunctive normal form: variables 1 to 4, a minus sign for *not*, each line an *or* of its literals, the whole thing an *and* of lines. Is there a way to set the four variables that makes every line true?

`babysat` tries all $2^4 = 16$ assignments and finds one: x1 false, x2 false, x3 true, x4 true. Written in C*, on top of selfie, so mipster can run the solver too.

It is not meant to be fast. It is the executable definition of what a SAT solver is. Add variables and each one doubles the work; at a hundred it is the machine that answers after the Sun.

WHY SAT IS HARD, AND WHY SOLVERS WORK ANYWAY

The exponential is a worst case. Real formulas have structure.

Nobody has a polynomial algorithm for SAT, and if the standard conjecture holds nobody will. Random formulas near 4.26 clauses per variable are hard for every solver ever built.

And yet modern solvers handle formulas with millions of variables from chip design, scheduling and software checking in seconds. The formulas that come from real problems are not random.

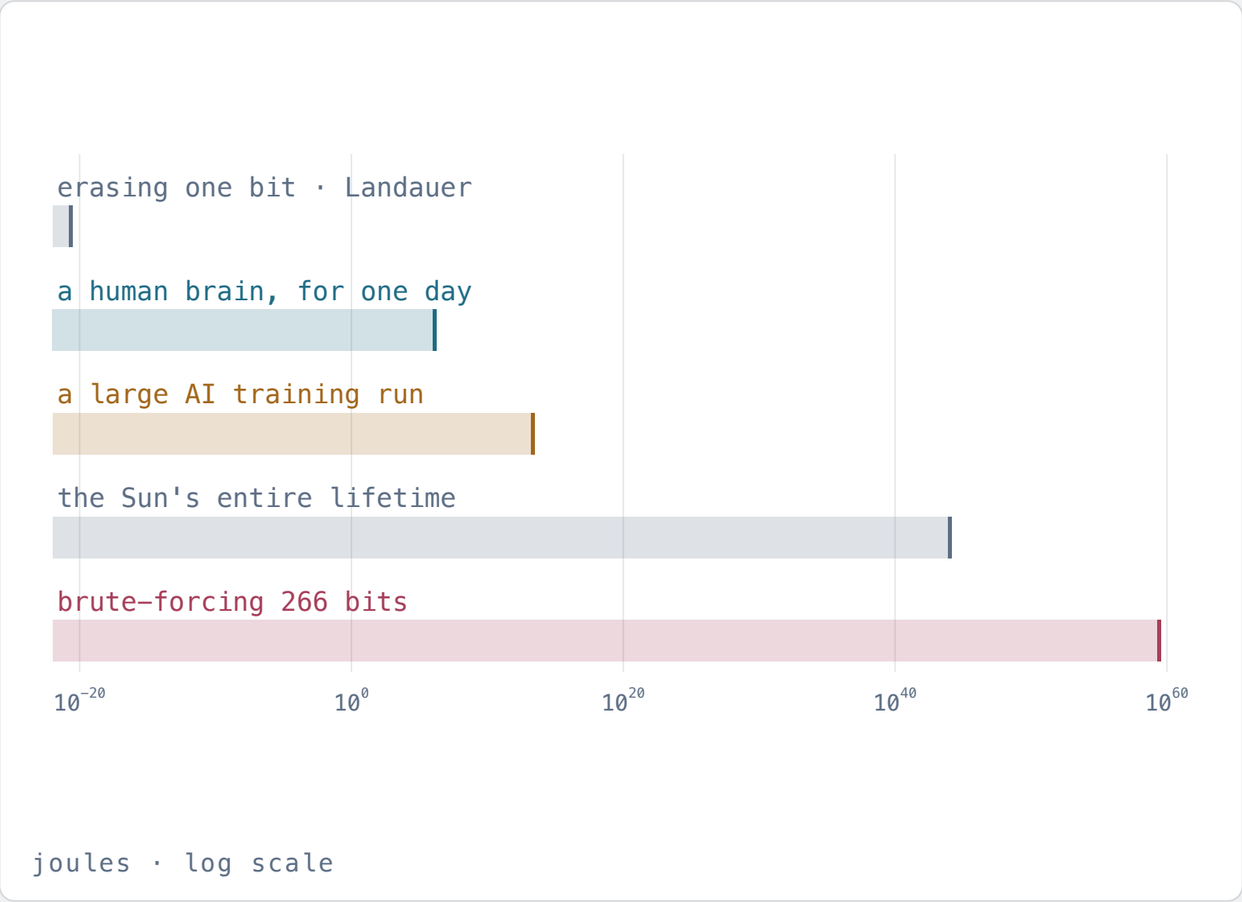
WHAT A MODERN SOLVER DOES

Propagate what is forced. Decide only when nothing is. When a decision fails, work out *why* and write it down as a new clause — a theorem about the formula — so the same failure never recurs. Restart, wiser.

On Rivest's formula that learns, after two failures, that x1 must be false, and after one more that x2 must be false — which is the comment in the file. Four assignments examined instead of sixteen.

PHYSICS

Computation is not abstract. Every step that forgets costs energy.



Landauer, 1961: erasing one bit at room temperature costs at least kT ln 2 ≈ 3×10^{-21} joules — because erasing is irreversible: two states become one, and the lost distinction leaves as heat.

A search moves on, and moving on is erasing. Just *counting* through week 2's 266-bit space flips the lowest bit 2^{266} times: about 10^{59} joules, 10^{15} times everything the Sun will ever radiate. You cannot even count the states.

Meanwhile a human brain runs on 20 watts. Time, space, energy: one budget. Any claim about intelligence that ignores it is a claim about magic.

REFRAME · 4 OF 5

Hardness is load-bearing.

WHAT IT FORBIDS

Brute force as a strategy. You cannot search your way to correctness, to a cure, or to a business plan.

WHAT IT OPENS

Every private message, every signature, every payment, every password you rely on today exists because some problems are hard. Difficulty is the raw material of security — and the reason abstraction, theory, and taste are worth more than compute.

A world where P = NP would be spectacular and also strange: mathematics automated, cryptography dead, and, since anything findable would be found, nothing left to discover.

BEFORE NEXT WEEK

Recommended exercises.

1. Read the Cost chapter up to *Models of Machines*.
2. Check by hand that $-1\ -2\ 3\ 4$ satisfies every clause of `rivest.cnf`. Add the eighth clause `1 2 -3 0` back and show that nothing satisfies all eight.
3. Write a CNF file of your own with five variables and run `babysat` on it. Then one with twenty-five, and time it.
4. How many years to try 2^{60} assignments at a billion per second? At a trillion?
5. Compute the minimum energy to erase 8 GB once, by Landauer, and compare it with a phone battery, about 4×10^4 joules.

NEXT WEEK

Formal methods: machine code in, formula out, solver next. A tool built on selfie turns a program into a formula, and the solver finds the input that makes it divide by zero — within a bound.