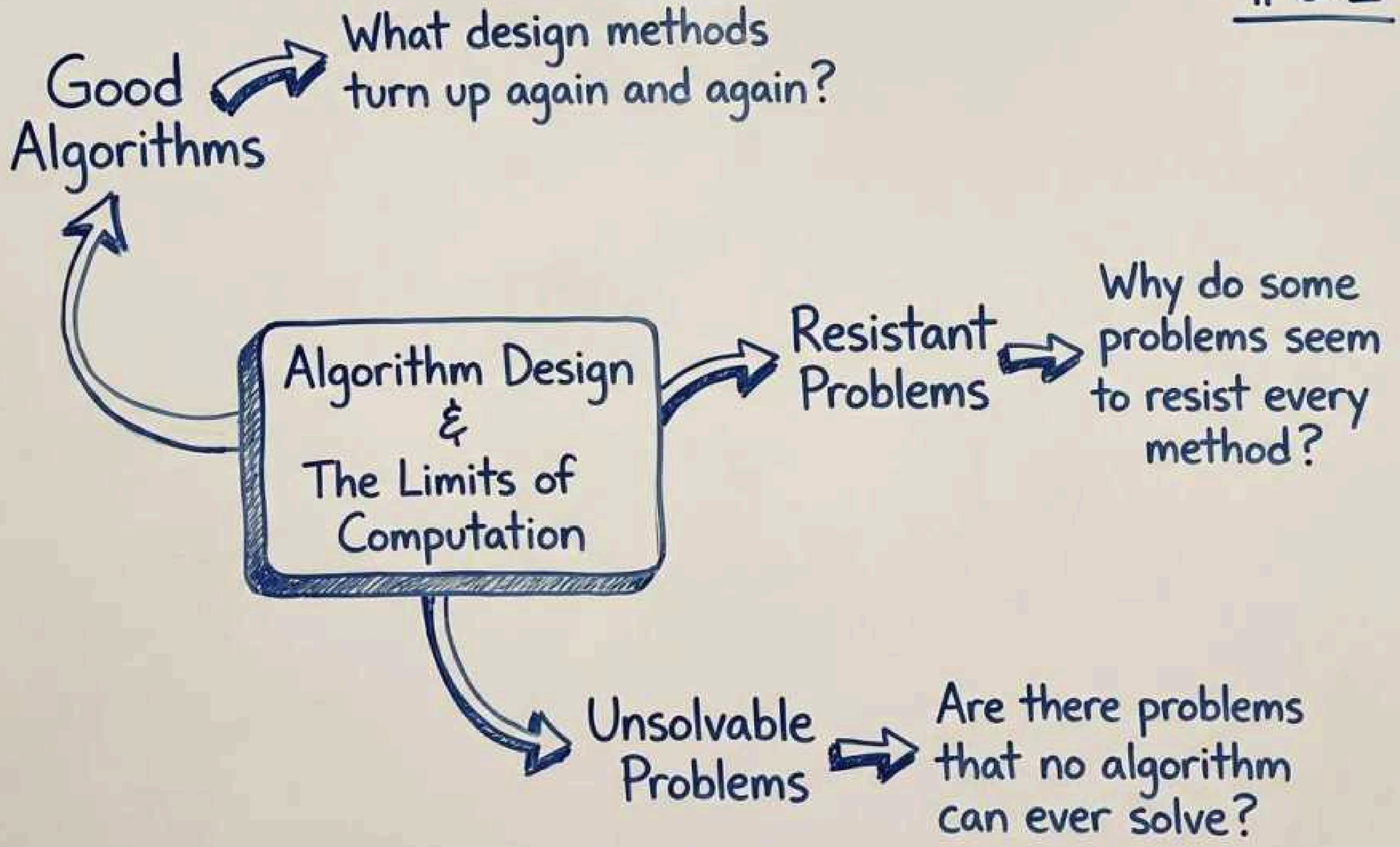
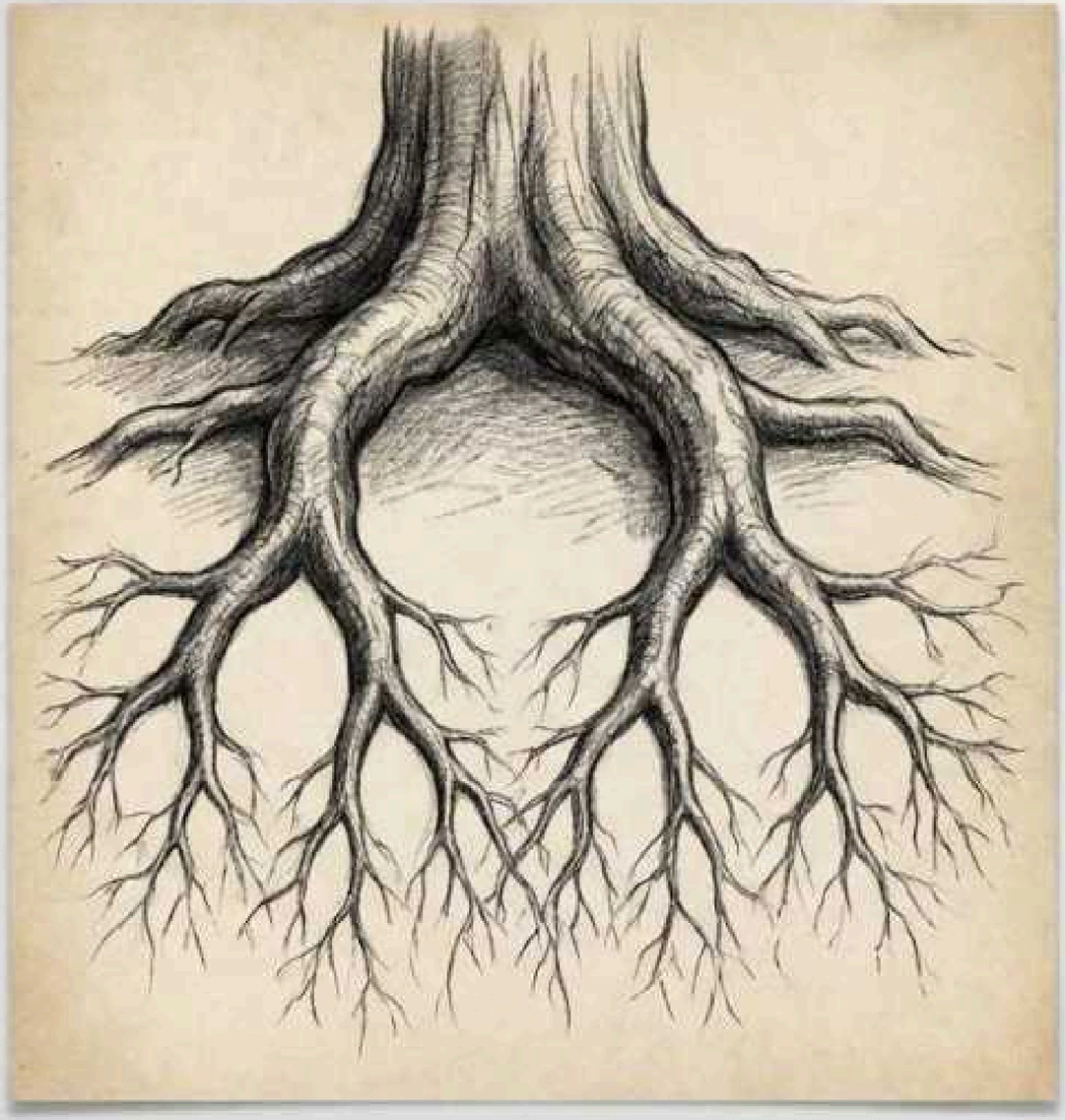
¶6.1

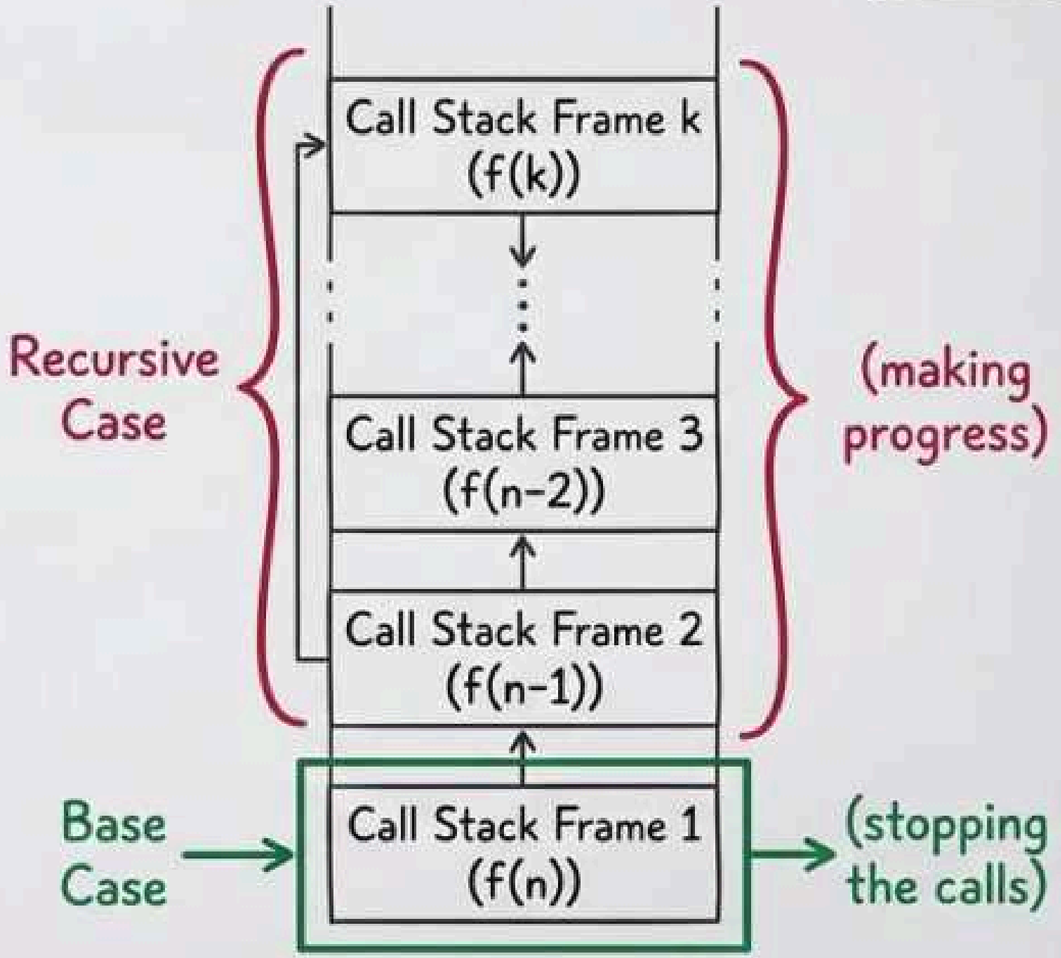


Gemini Notebook



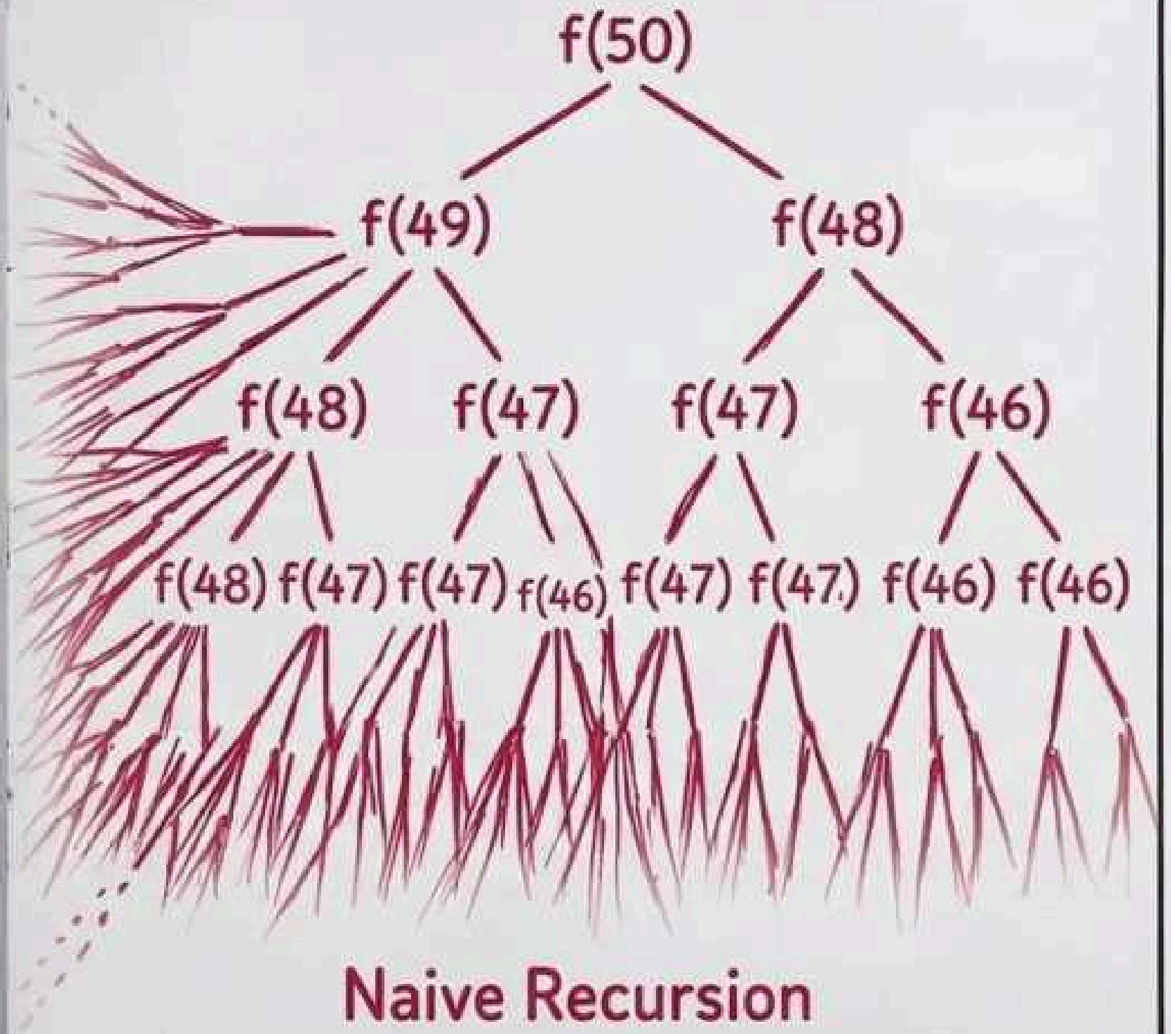
Many structures naturally mirror recursive shapes.

¶ 6.2



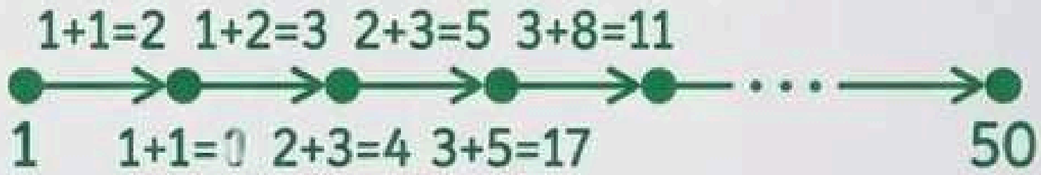
⚠ Deep recursion = Stack Overflow.
Loops may be required.

¶ 6.3



Naive Recursion

Disastrous: Solves identical smaller problems repeatedly. Tens of billions of calls. Exponential Time.



Obvious Loop

50 simple additions.
Linear Time.

Dynamic Programming

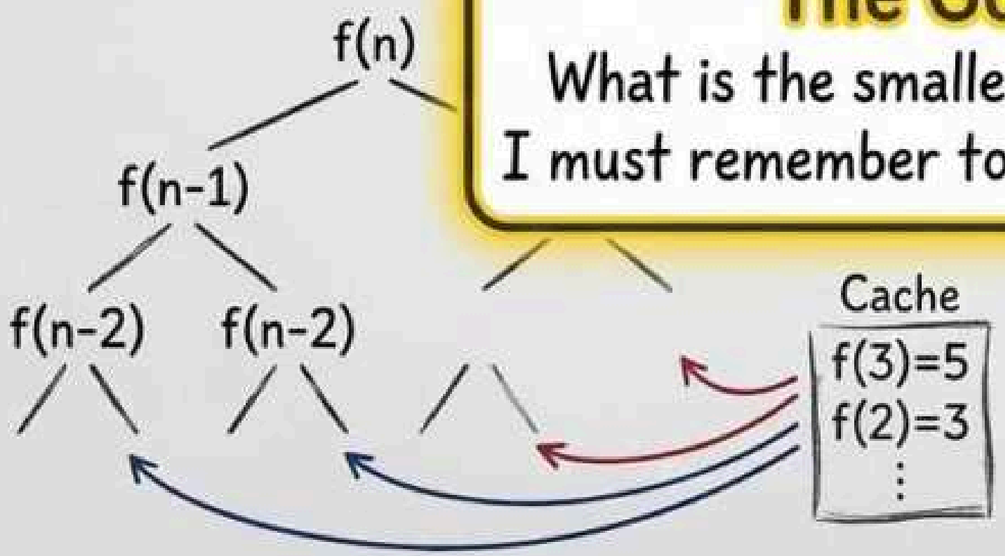
¶ 6.4

(Richard Bellman, 1950s – 'Chosen because it sounded impressive')

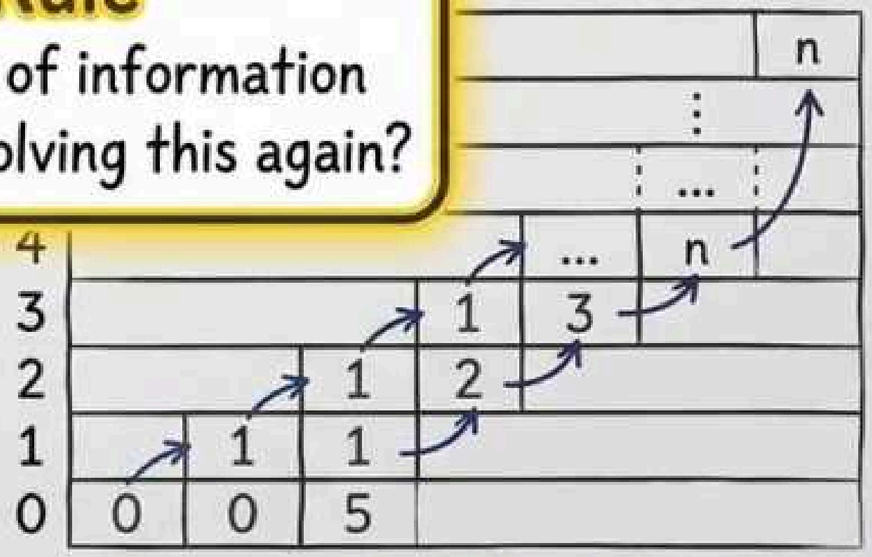
Core Requirements:
1) Overlapping Subproblems
2) Optimal Substructure (Optimal solutions built from optimal smaller parts).

The Golden Rule

What is the smallest piece of information
I must remember to avoid solving this again?



Top-Down (Memoization)



Bottom-Up (Tabulation)

The Knapsack Problem



Item	Weight (w_i)	Value (v_i)
1	2kg	$10
2	3kg	$15
3	5kg	$25
...	...	...

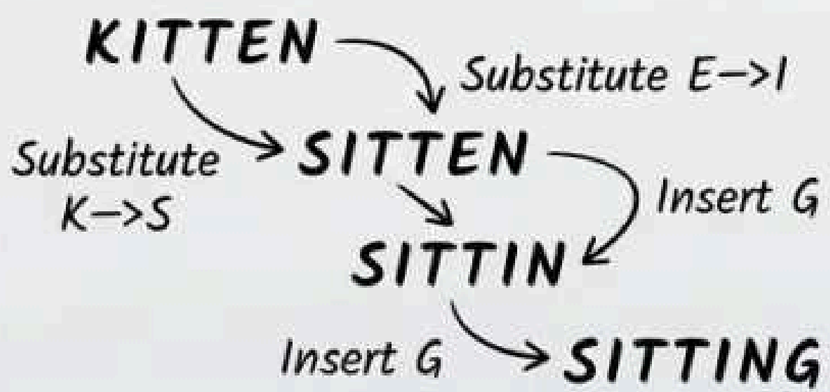
↳ Time proportional to items × weight limit.

Longest Common Subsequence

		G	E	N	E	S	I	S
	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1
E	0	1	2	2	2	2	2	2
N	0	1	2	3	3	3	3	3
O	0	1	2	3	4	4	4	4
M	0	1	2	3	4	5	5	5
E	0	1	2	3	4	5	6	6

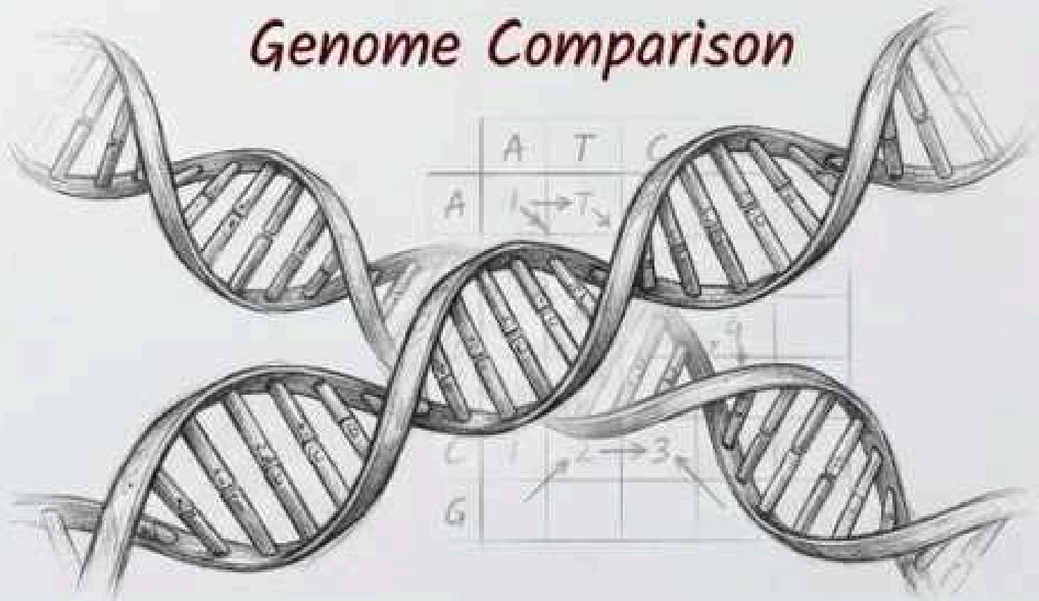
O(mn) — Powers file comparison tools.

Edit Distance



Levenshtein, 1965.

Genome Comparison



Gemini Notebook

Greedy Algorithms

¶ 6.6

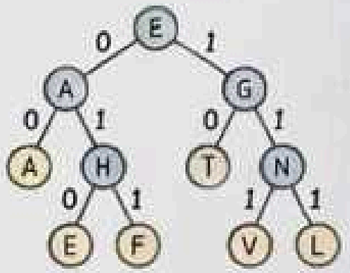
(The path of least resistance – always take the immediate best option)



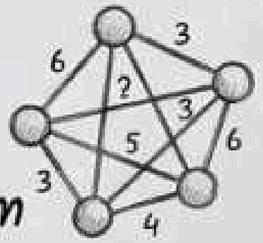
Make the locally best choice at every step. Never reconsider.

Classic Examples

- **Activity Scheduling**
  Earliest finishing time = most activities.
- **Huffman Coding**
  (David Huffman, 1952)
  Optimal binary compression tree



- **Graph Algorithms**
  Dijkstra, Kruskal, Prim



Gemini Notebook

Target: Make 6.
Available Coins: 1, 3, 4.

Takes 4
↓
Needs 2
↓
Takes 1, then 1

Total: 3 coins
(4+1+1)



Chooses 3
↓
Chooses 3 again

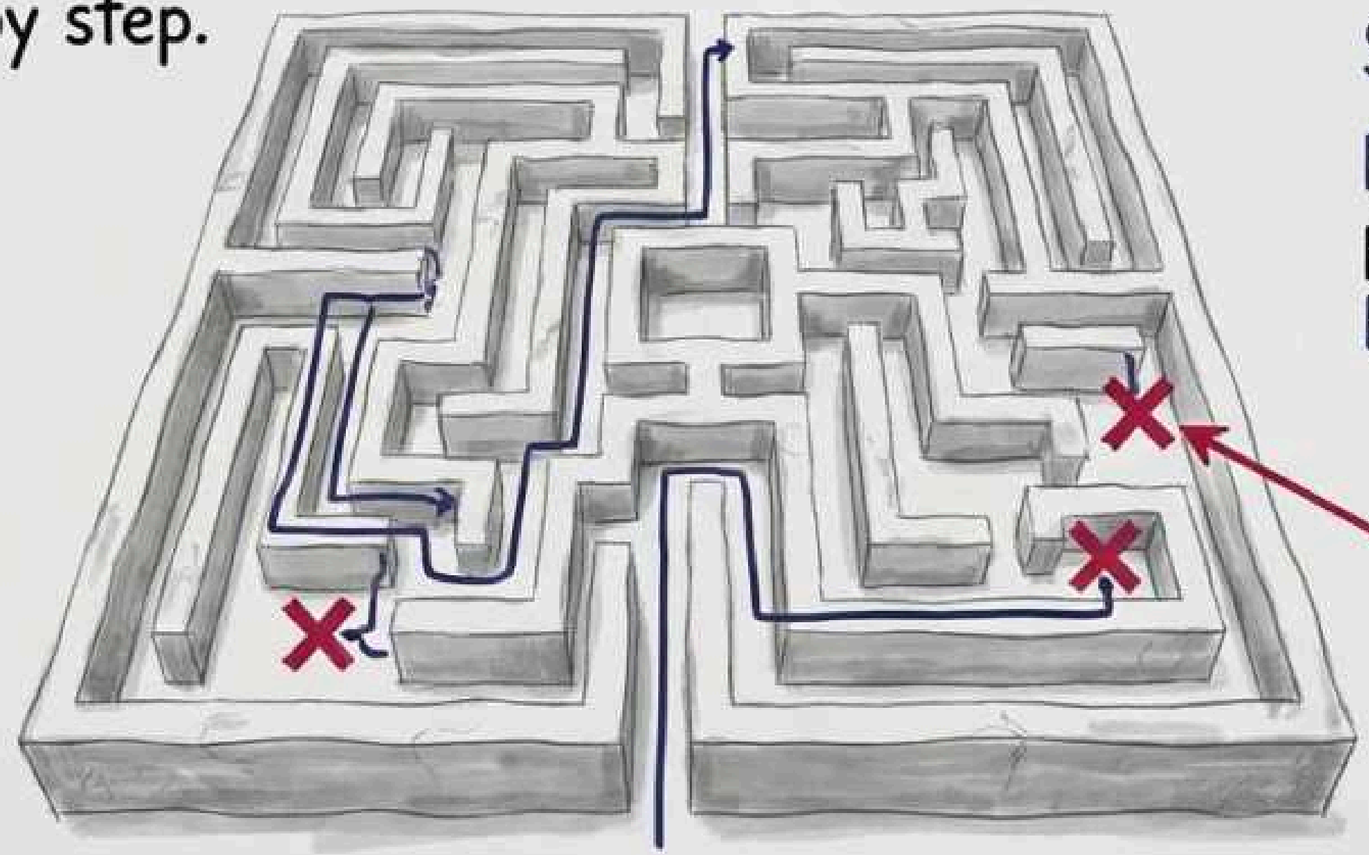
Total: 2 coins
(3+3)



Greedy methods can fail. They require formal proofs. (Standard currency denominations happen to work, which is why cashiers use them).

Gemini Notebook

Builds a solution step by step.

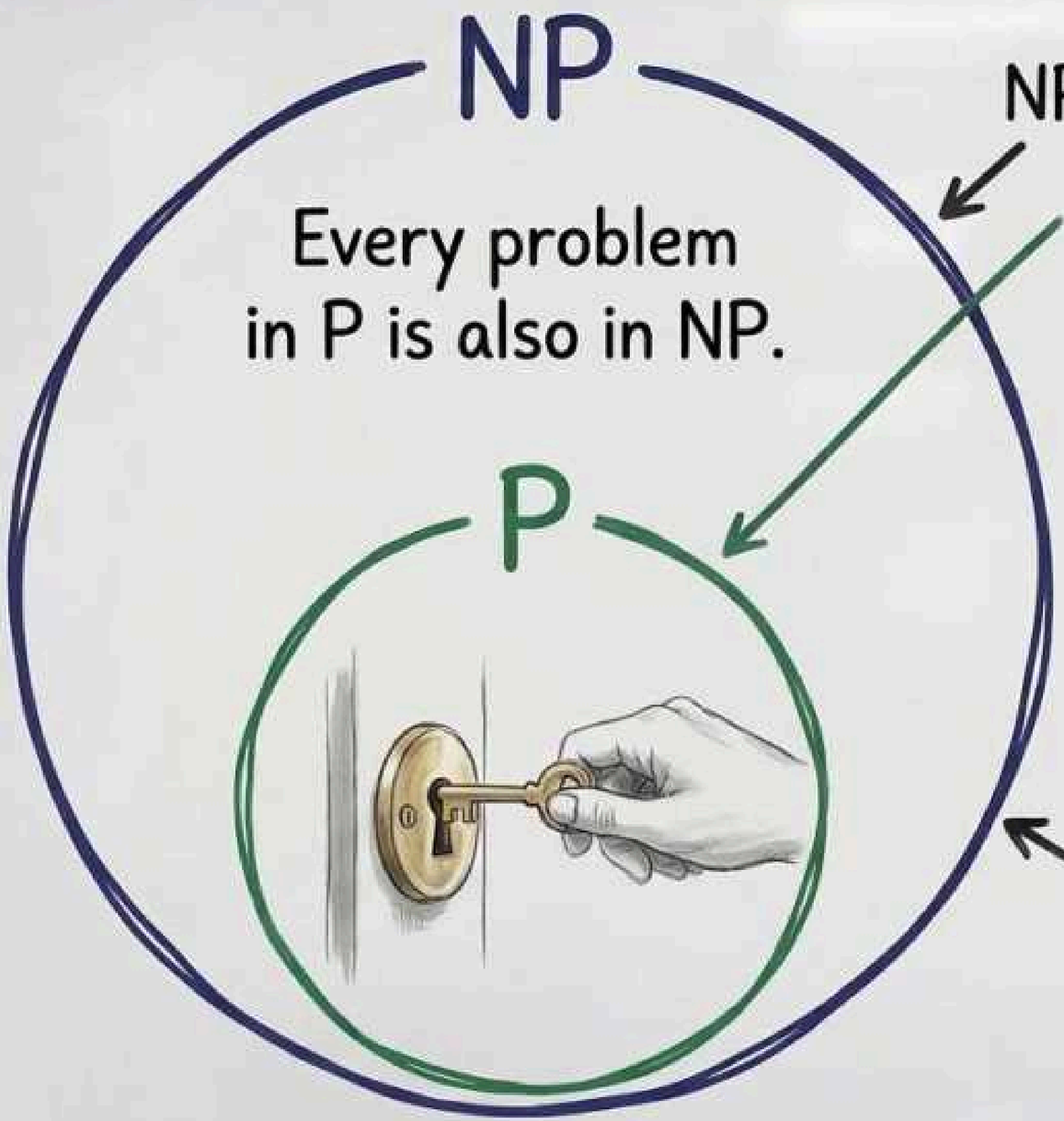


Classic applications: Sudoku, the 8 Queens problem, parsing text patterns, and Branch & Bound for optimization.

Pruning: Abandoning a partial solution as soon as it cannot succeed.

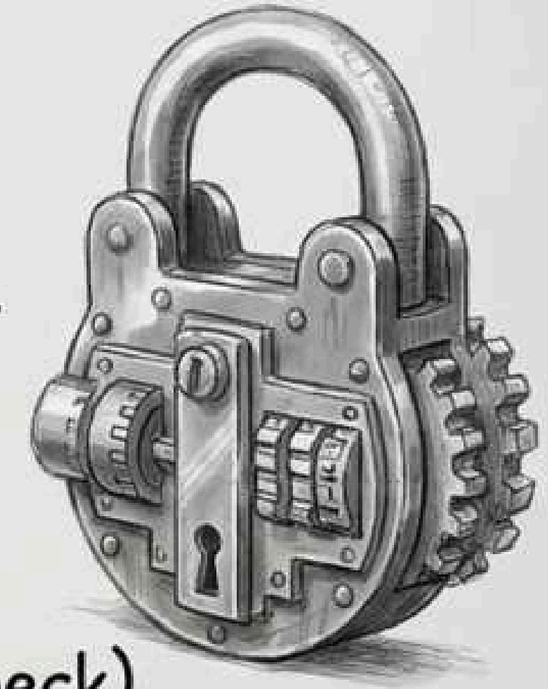
Worst case is exponential, but good pruning makes it practical.

Gemini Notebook



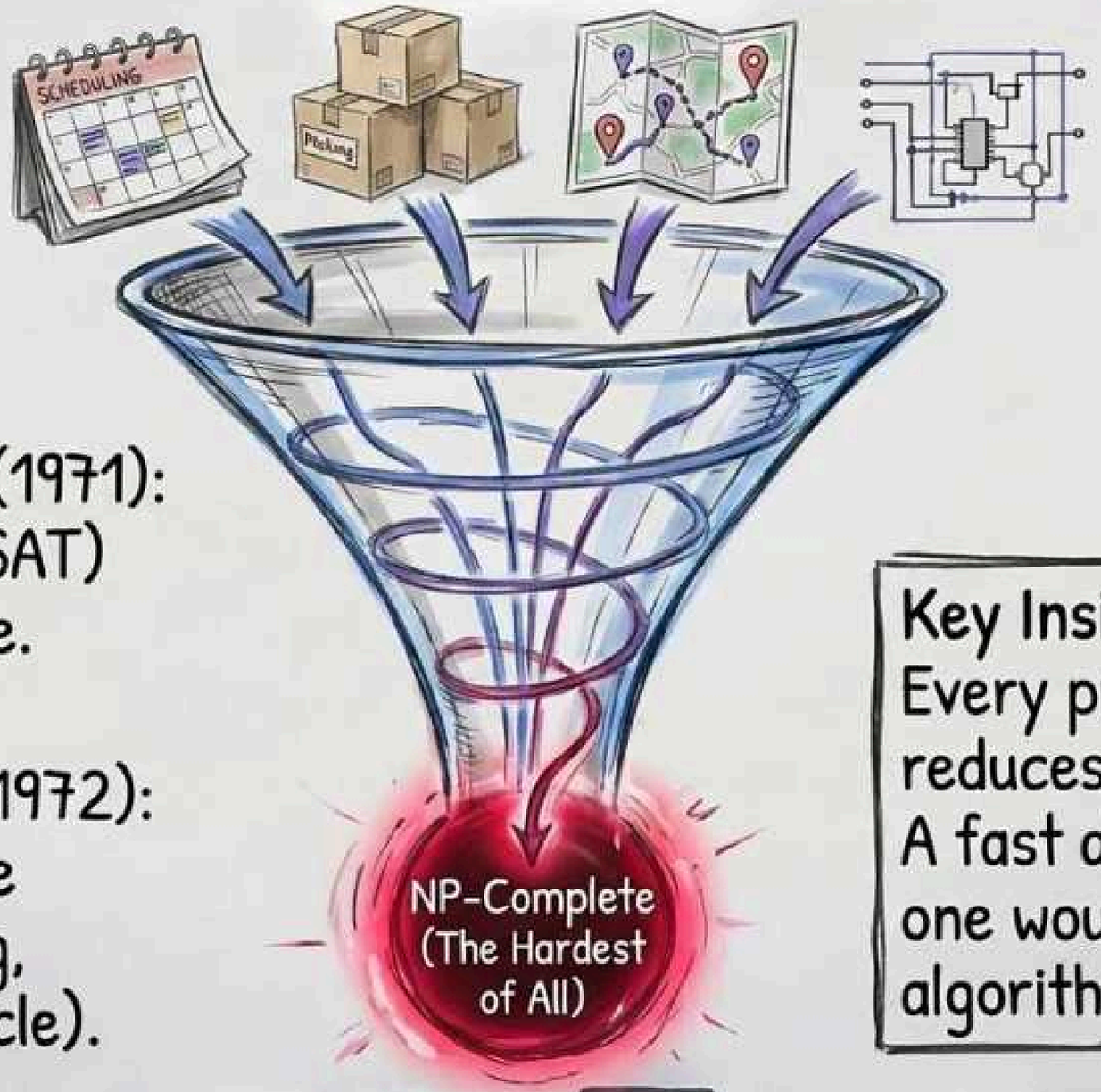
NP (Easy to Solve)

Examples:
Sorting,
shortest paths
(polynomial time).



NP (Easy to Check)

Finding the right key is incredibly hard, but verifying a key works is instantaneous (polynomial time checking).



- Stephen Cook (1971): Satisfiability (SAT) is NP-Complete.

- Richard Karp (1972): Proved 21 more (Graph coloring, Hamiltonian cycle).

Key Insight:
Every problem in NP reduces to these.
A fast algorithm for one would mean a fast algorithm for all.

Gemini Notebook

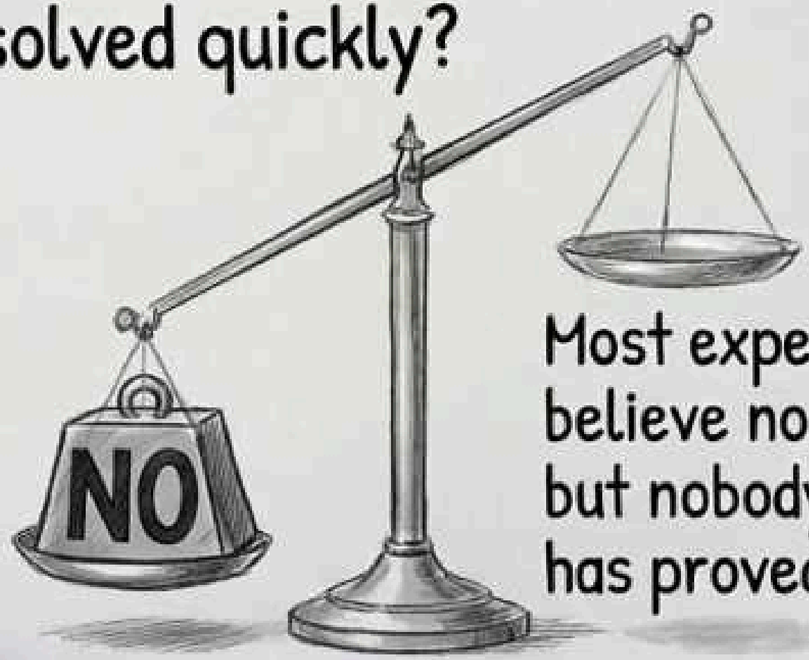
¶ 6.11

P =? NP

Can every problem whose solution is checked quickly also be solved quickly?



Clay Mathematics Institute
- Millennium Prize Problem (Year 2000)
- $1,000,000 Reward.

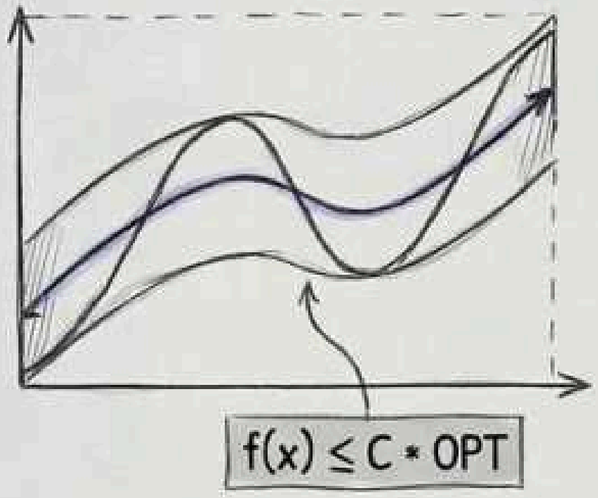


Most experts believe no, but nobody has proved it.

Gemini Notebook

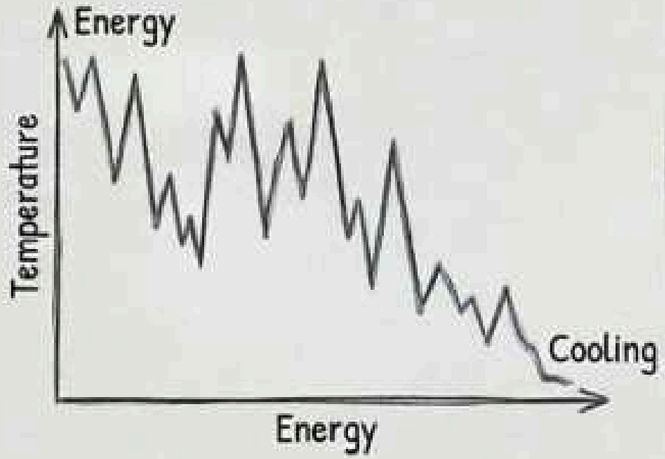
Solving the Unsolvable in Practice

Approximation Algorithms



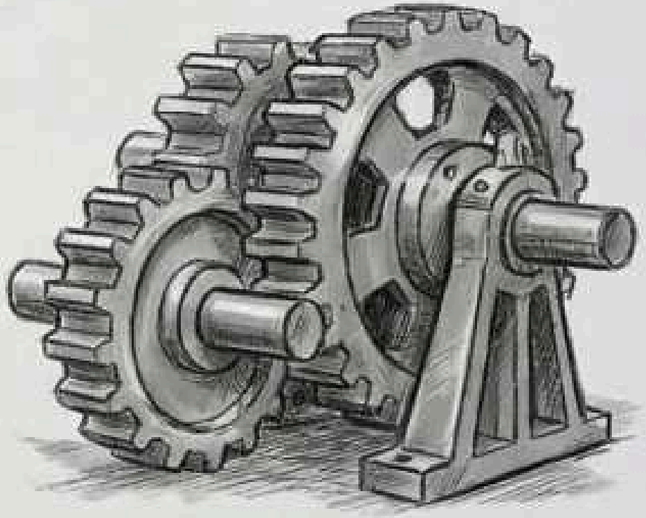
Guarantees answer within a known factor of the best. (e.g., Christofides, 1976: 1.5x shortest tour for metric TSP).

Heuristics



Simulated annealing and local search. Finds good answers without mathematical guarantees.

Specialized Solvers

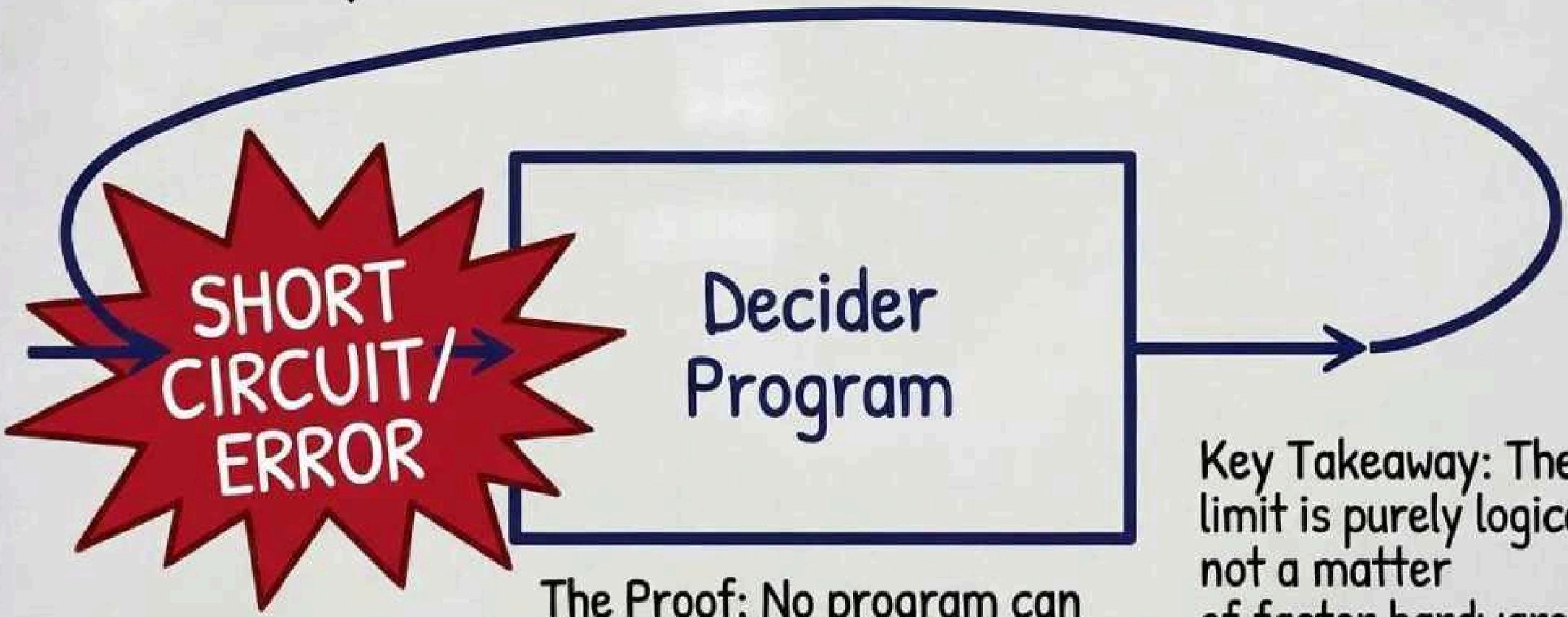


Handles massive instances that arise in industry.

Practical Lesson: Recognize hard problems early. Don't waste months searching for a perfect fast algorithm that probably doesn't exist.

Gemini Notebook

Undecidability: Worse Than Hard



Context:
Alan Turing, 1936:
The Halting Problem.

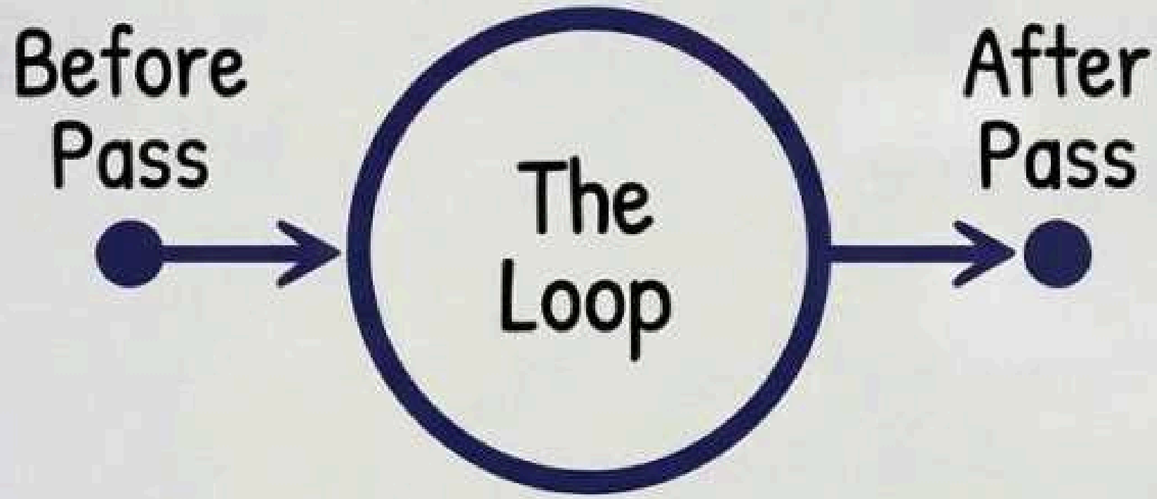
The Proof: No program can decide, for every program and input, whether that program eventually stops.

Key Takeaway: The limit is purely logical, not a matter of faster hardware. Some questions about software are beyond algorithms entirely.

Gemini Notebook

Mathematical Correctness

Loop Invariant



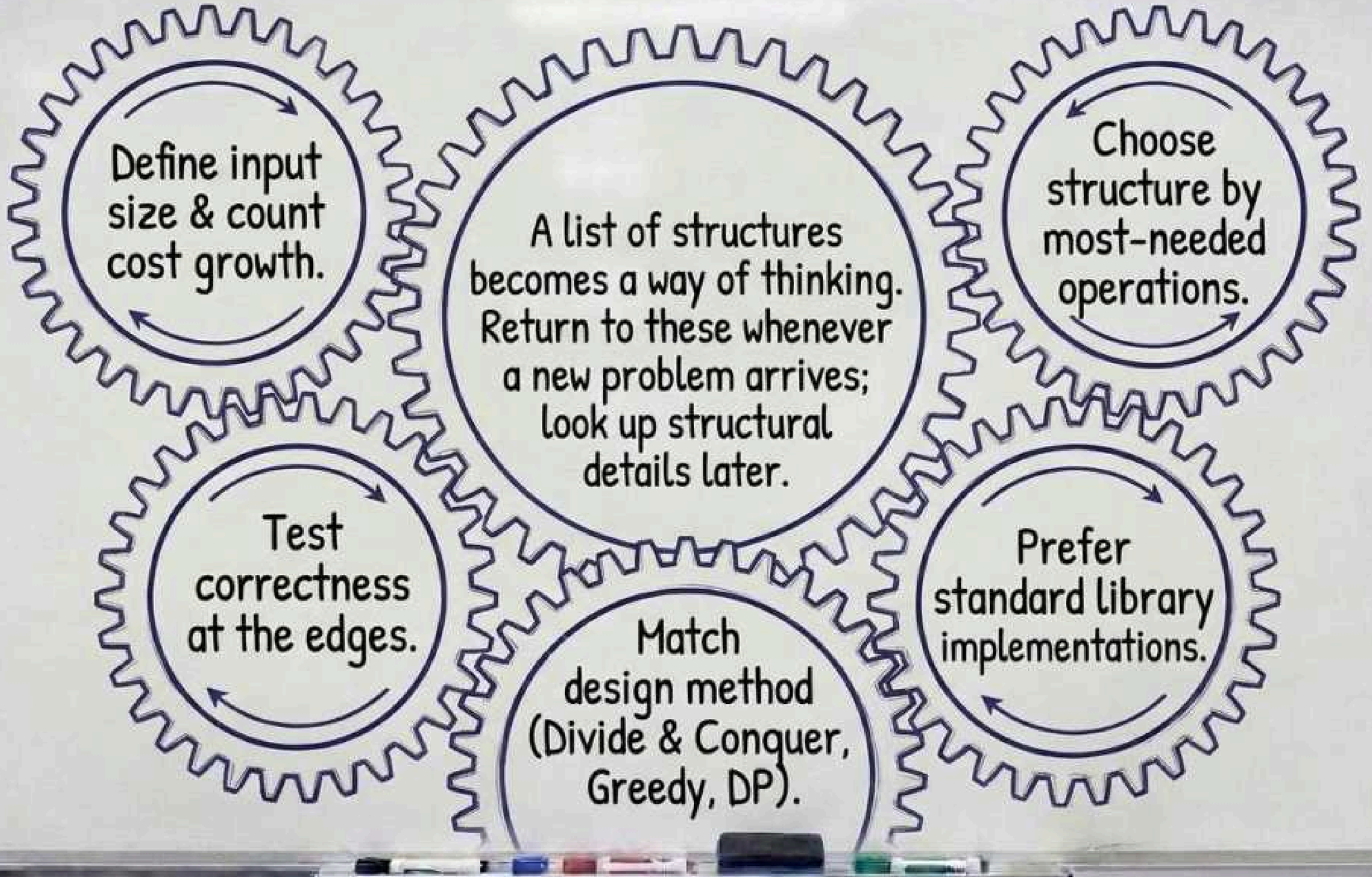
Prove by induction to show an algorithm is correct.

Practical Testing



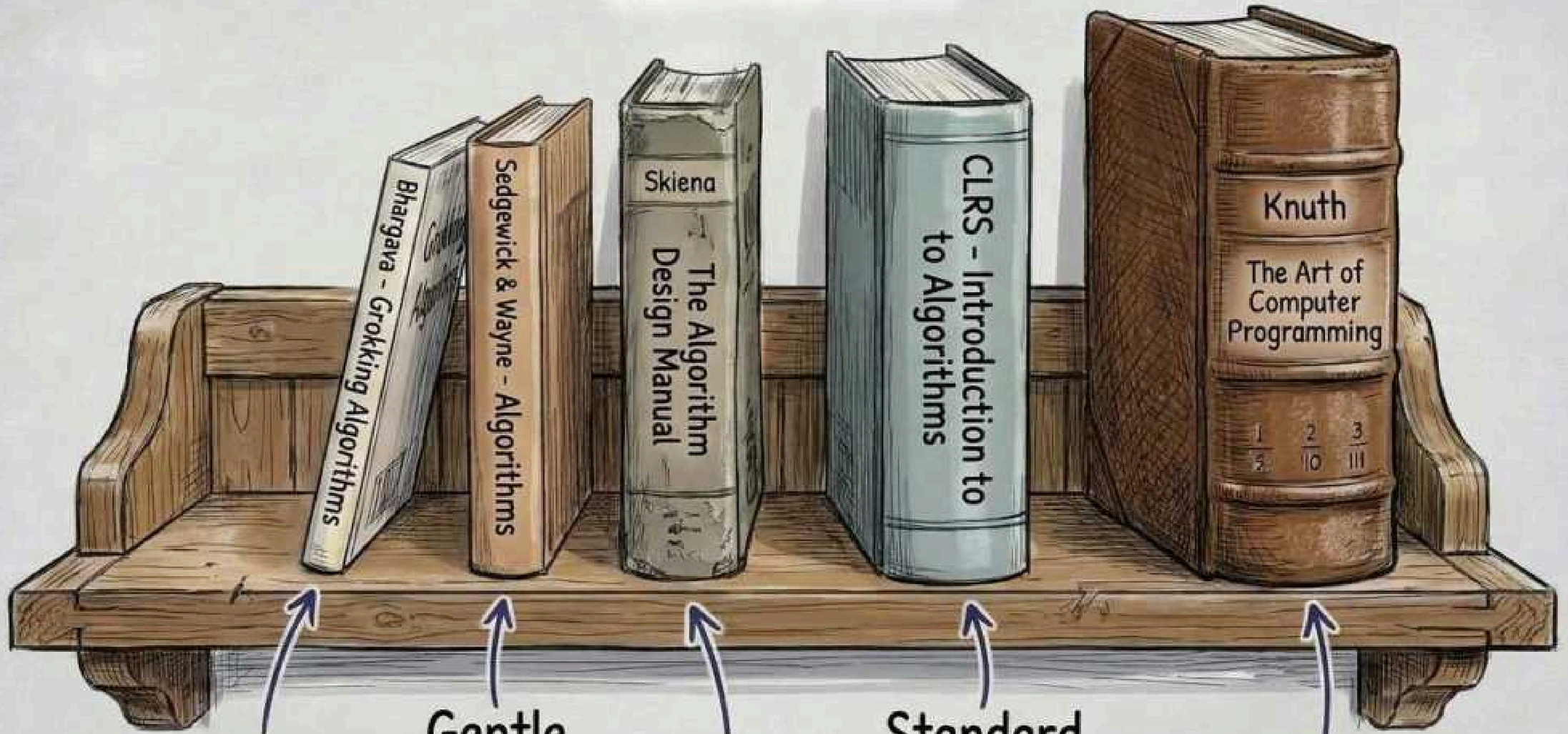
Testing shows the presence of bugs, never their absence. — Edsger W. Dijkstra

¶ 6.15



Gemini Notebook

¶ 6.16


Bhargava - Grokking Algorithms
Sedgewick & Wayne - Algorithms
Skiena
The Algorithm Design Manual
CLRS - Introduction to to Algorithms
Knuth
The Art of Computer Programming
1/2
2/10
3/111
A friendly first book
Gentle
Practical
Standard
Monumental

¶ 6.17

Closing Exercise

1. Take any program you have written that runs slowly.
2. Find which loop dominates the execution.
3. State its Big O notation mathematically.
4. Ask: Would a different data structure change it?

That gap between a working program and a fast program is what this course teaches. Thank you for studying Computer Science 201 B with Anderson Academy.

¶ 6.18

One last reminder.

Questions or feedback: garyanderson@mail.grokbot.com
X: @garyanderson

Gemini Notebook