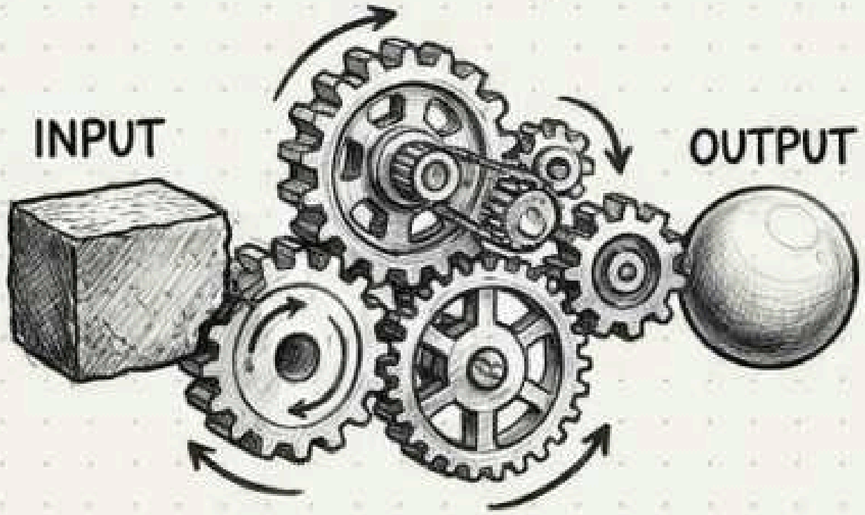
Algorithms, Complexity, and Big O

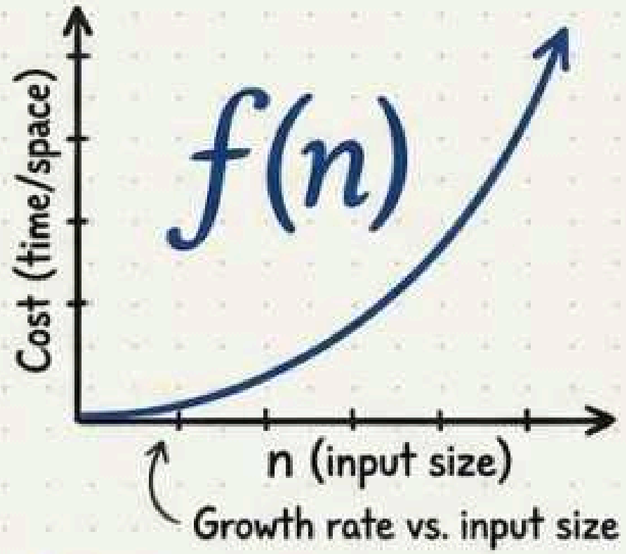
¶ 1.1

ALGORITHMS & EFFICIENCY

What is an algorithm?

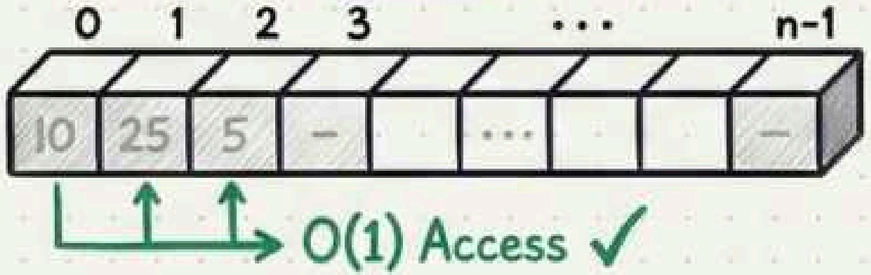


How do we measure cost?

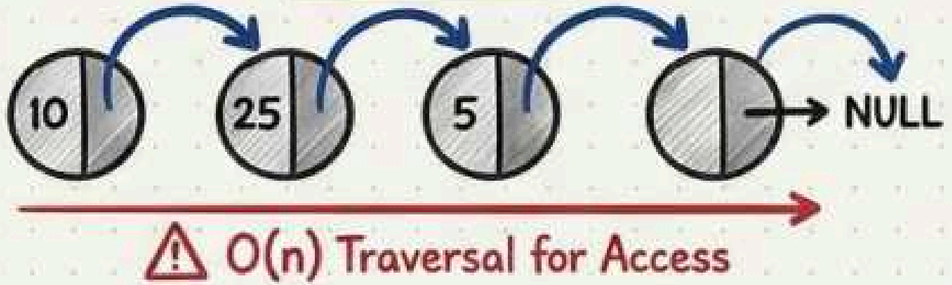


Arrays vs. Linked Lists

ARRAY

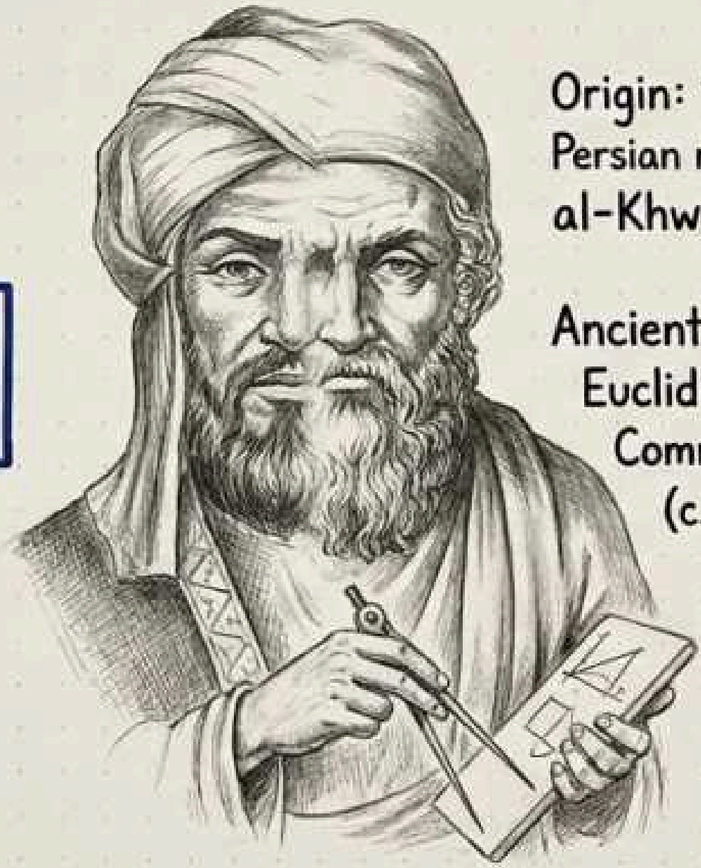


LINKED LIST



Defining the Algorithm Through History and Practice

¶ 1.2



Origin: 9th-century Persian mathematician al-Khwarizmi

Ancient usage: Euclid's Greatest Common Divisor (c. 300 BCE)



- Everyday Algorithms:
  - Recipe
  - Directions
  - Checkout procedure



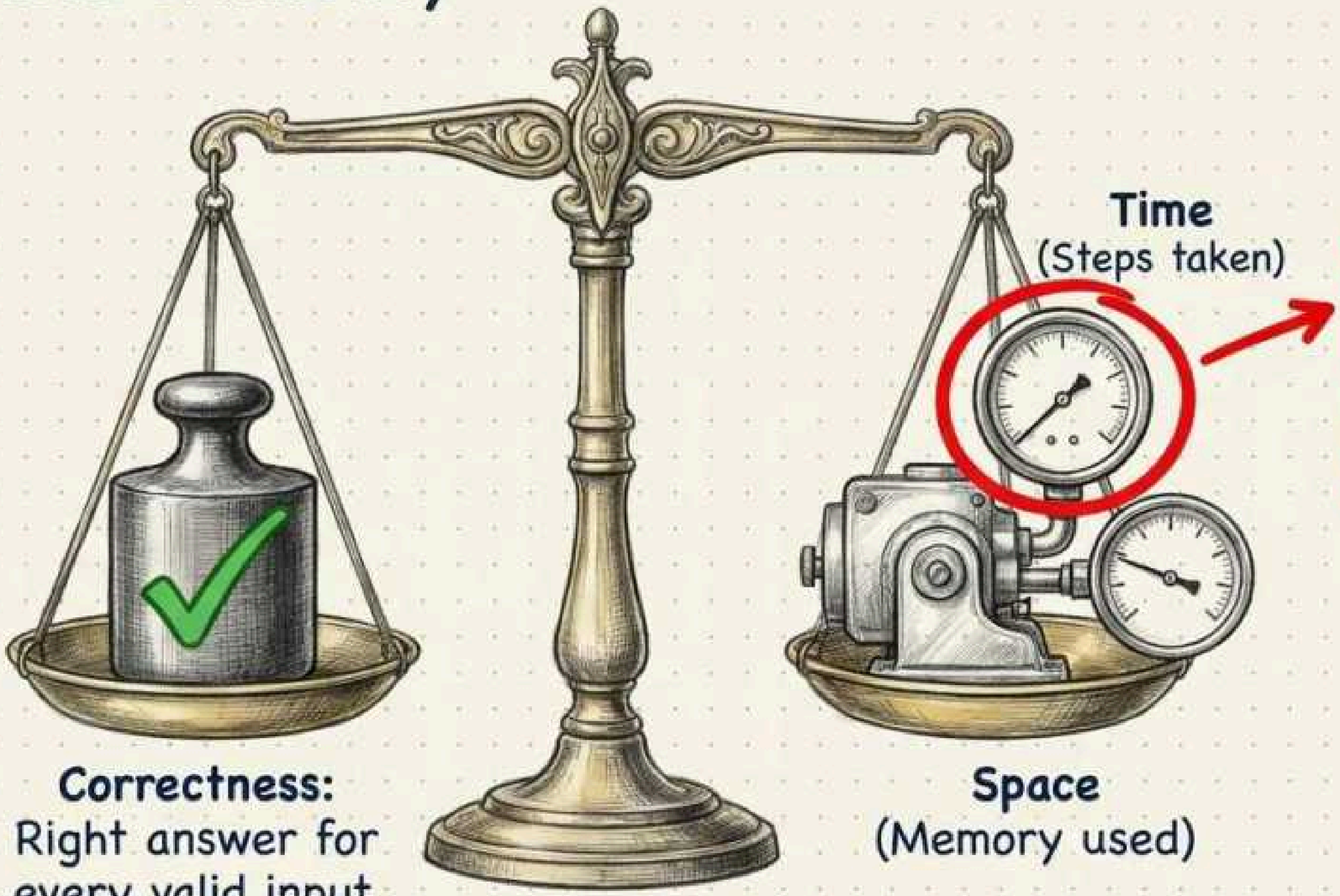




Gemini Notebook

Evaluating Algorithms by Correctness and Efficiency

¶ 1.3



Gemini Notebook

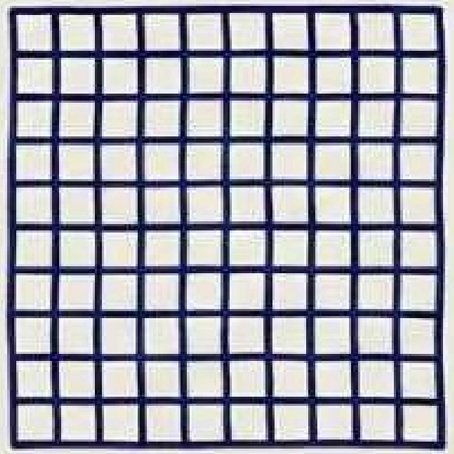
Measuring Cost by Growth Rate, Not Stopwatches

¶ 1.4



Depends on machine, language, and day.

Count basic operations as a function of the input size (n).



n × n
(comparing every pair)

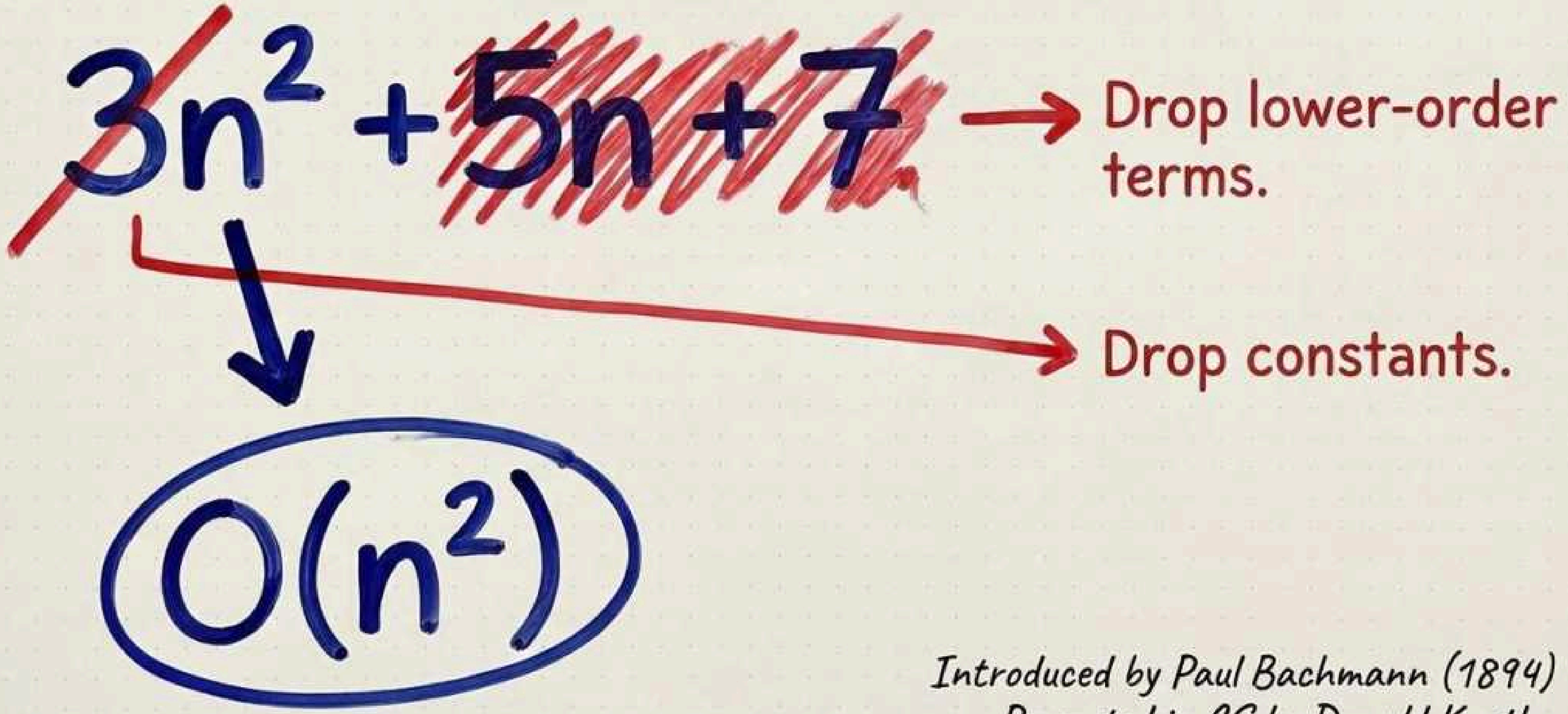


n
(scanning once)

Gemini Notebook

Simplifying Cost into Big O Notation

¶ 1.5

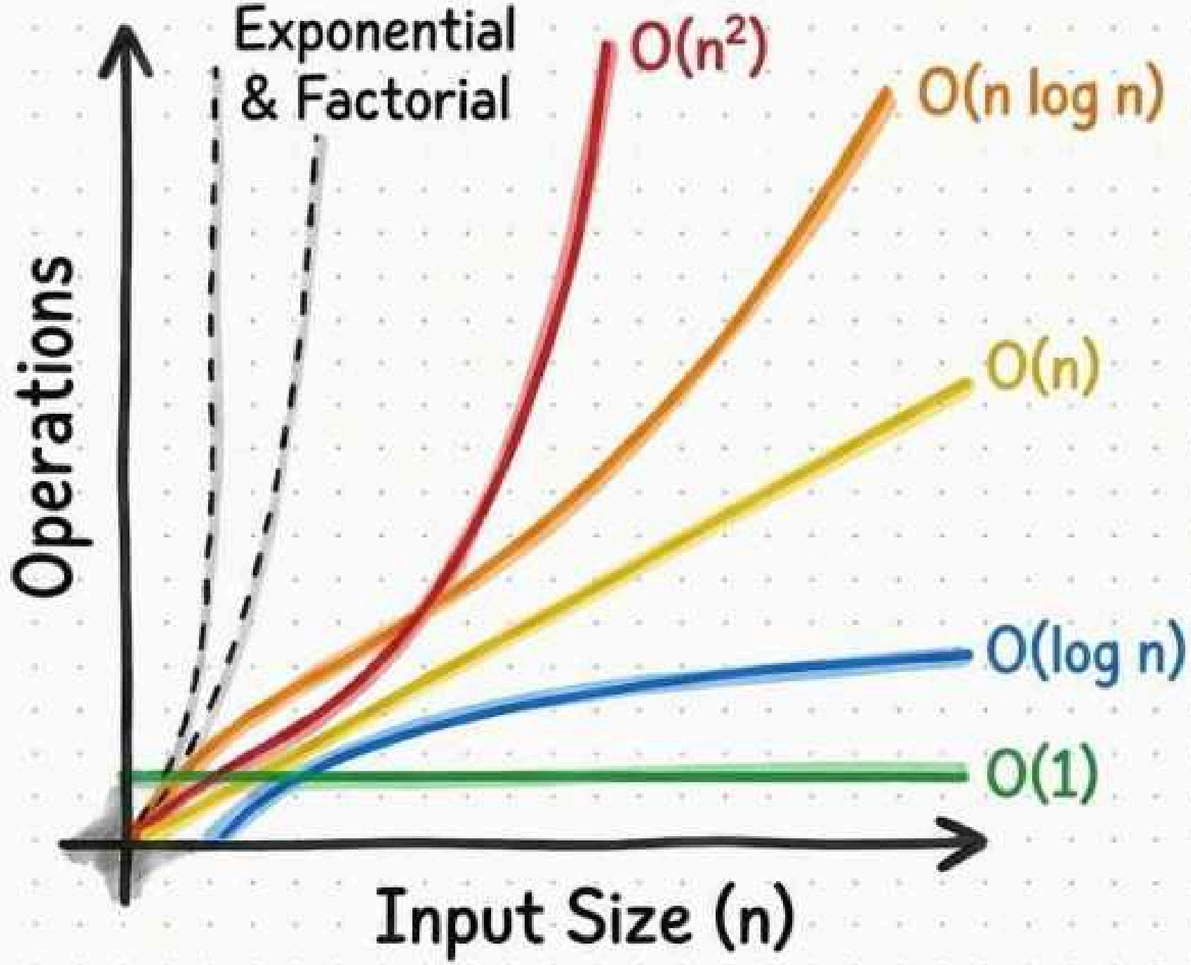


Introduced by Paul Bachmann (1894)
— Promoted in CS by Donald Knuth.

Gemini Notebook

The Hierarchy of Algorithmic Growth Rates

¶ 1.6



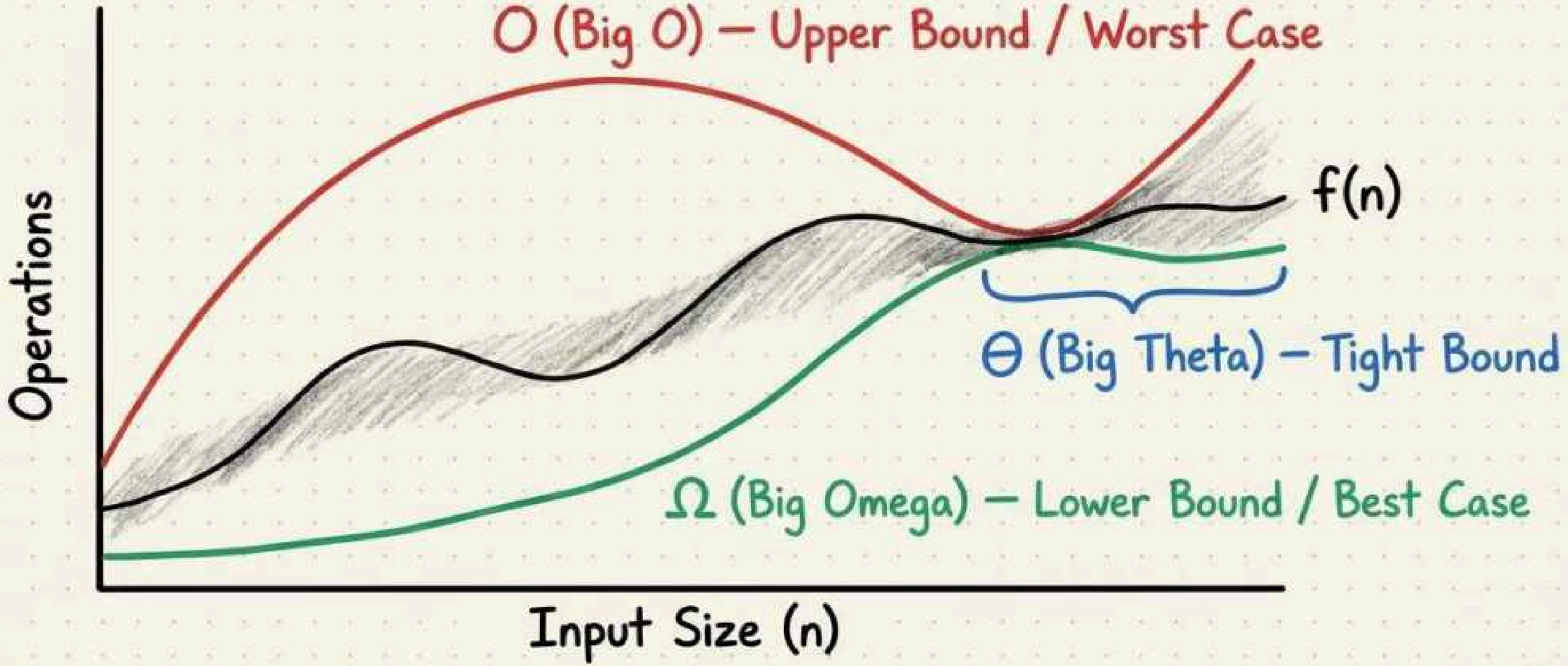
For 1,000,000 items:
Linear O(n) = 1 Million steps
O(n log n) ≈ 20 Million steps
Quadratic $O(n^2)$ = 1 Trillion steps

Best Practice: Estimate before you code. Ask how large n can become.

Gemini Notebook

Defining Upper, Lower, and Tight Bounds

¶ 1.7



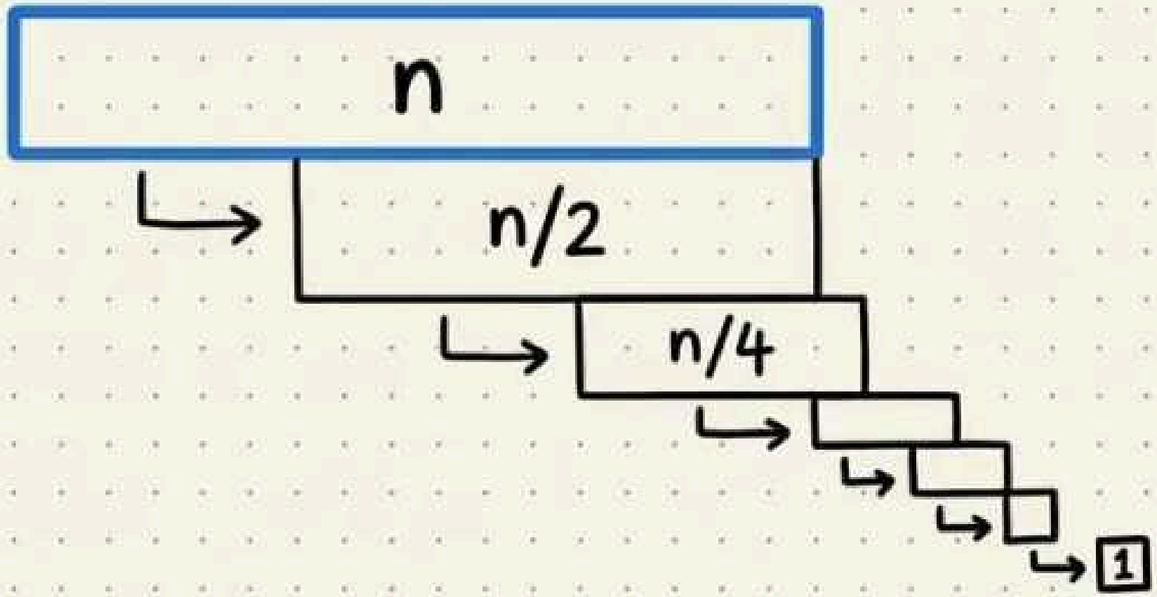
We can also measure average case or amortized cost over a sequence.

✱ When unstated, assume the worst case (Big O).

Gemini Notebook

The Unrivaled Speed of Logarithmic Time

¶ 1.8



Base-two logarithm of n:
The number of times you can halve n before reaching 1.
(e.g., Binary Search)

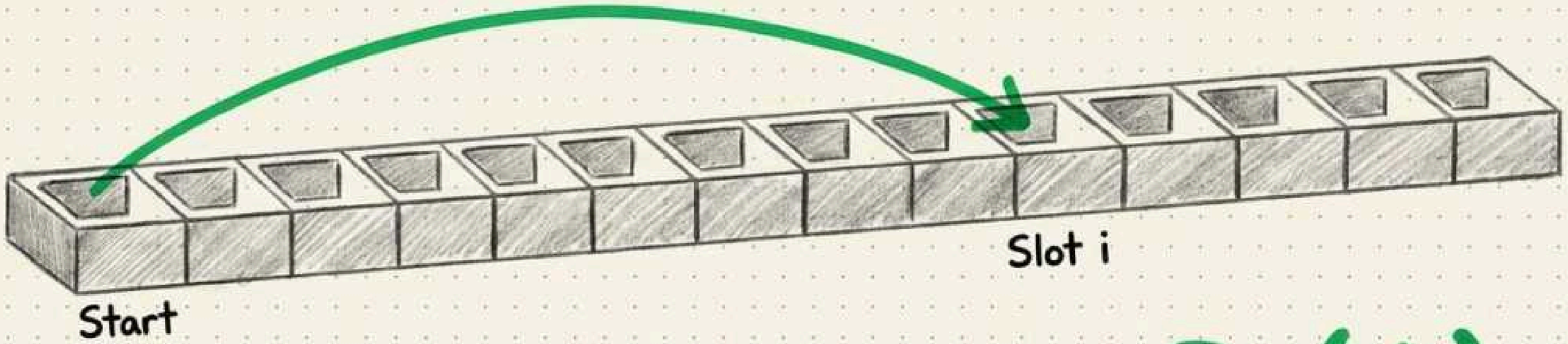
1,000,000 sorted items	→ max 20 comparisons
1,000,000,000 sorted items	→ max 30 comparisons

"Astronomically large inputs barely slow them down."

Gemini Notebook

Arrays Provide Instant Access via Contiguous Memory

¶ 1.9



Items stored in consecutive memory locations of identical size.

Address = Start + (i × slot_size)

O(1)

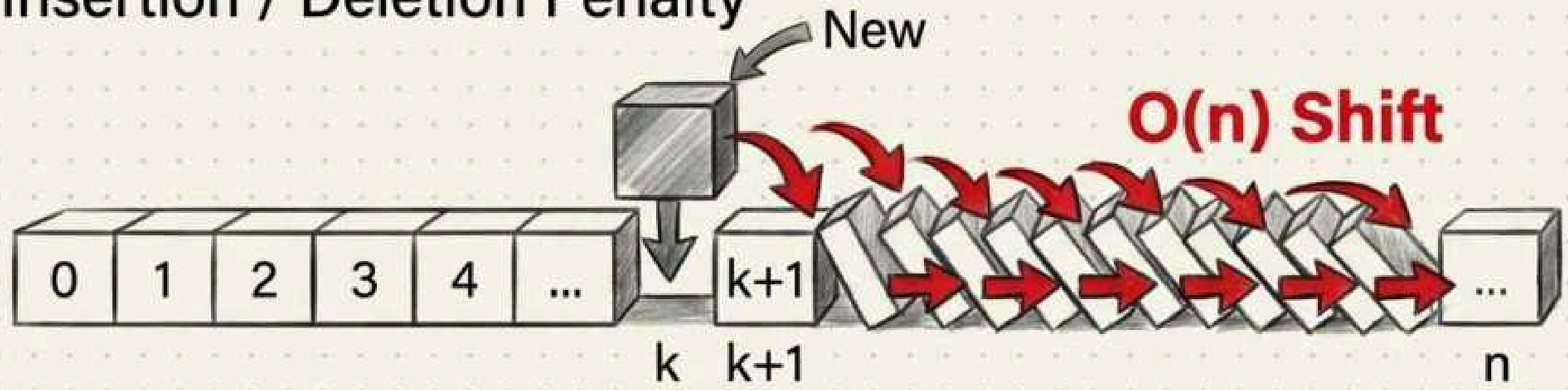
Jumping to the 1,000,000th item is exactly as fast as jumping to the 1st.

Gemini Notebook

The Structural Penalties of Fixed Arrays

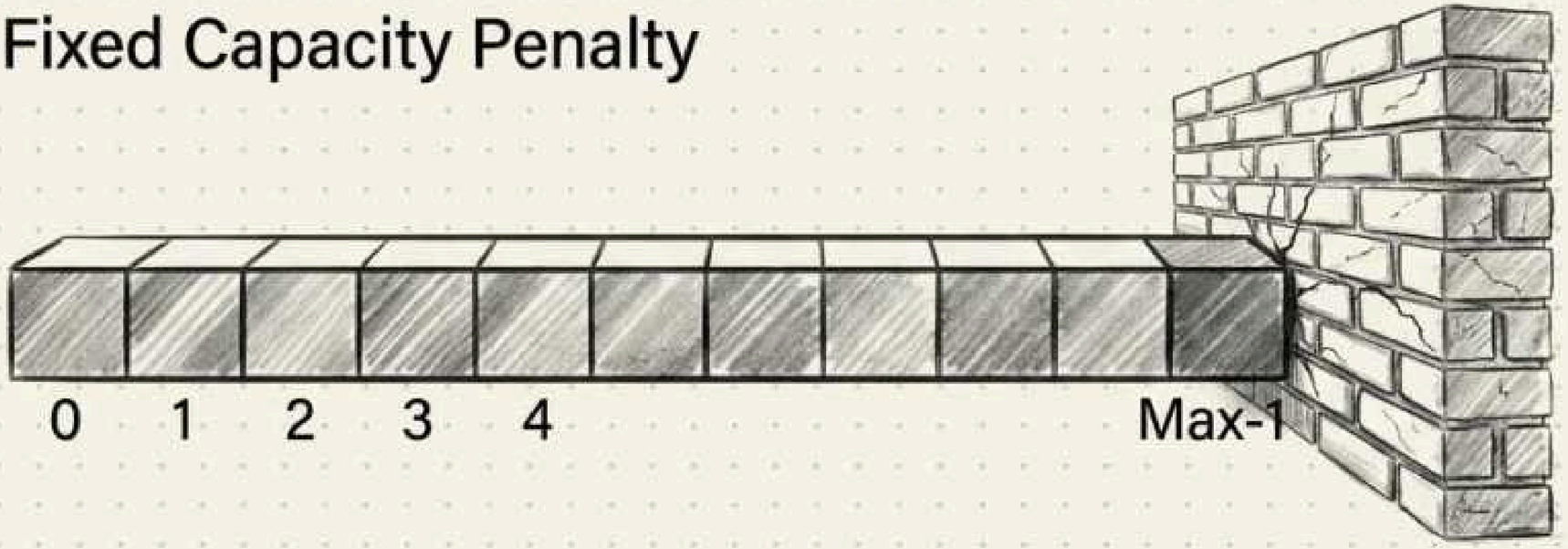
¶ 1.10

Insertion / Deletion Penalty



Searching an unsorted array also costs O(n).

Fixed Capacity Penalty

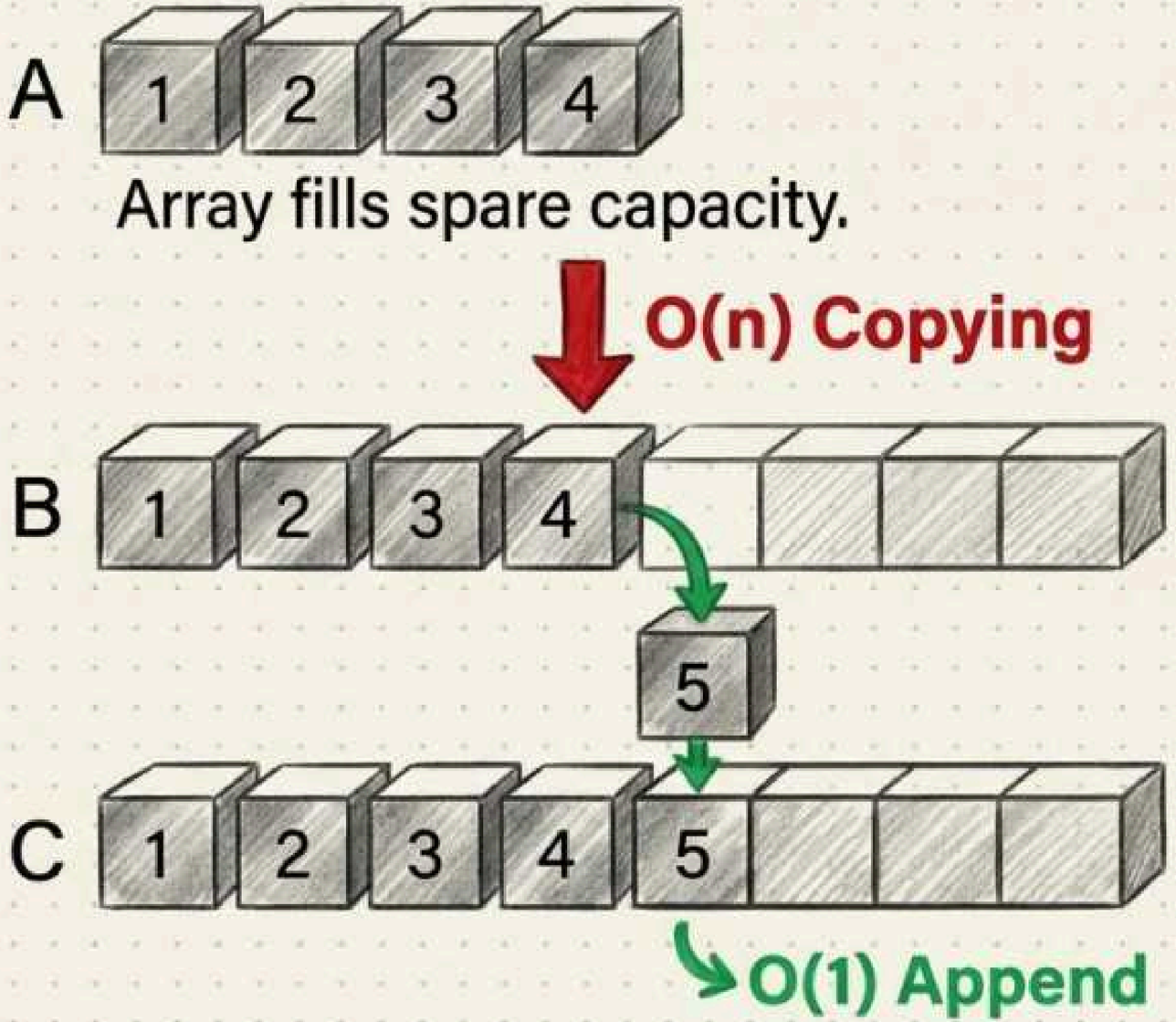


Growing requires allocating a brand new, larger array and copying everything over.

Gemini Notebook

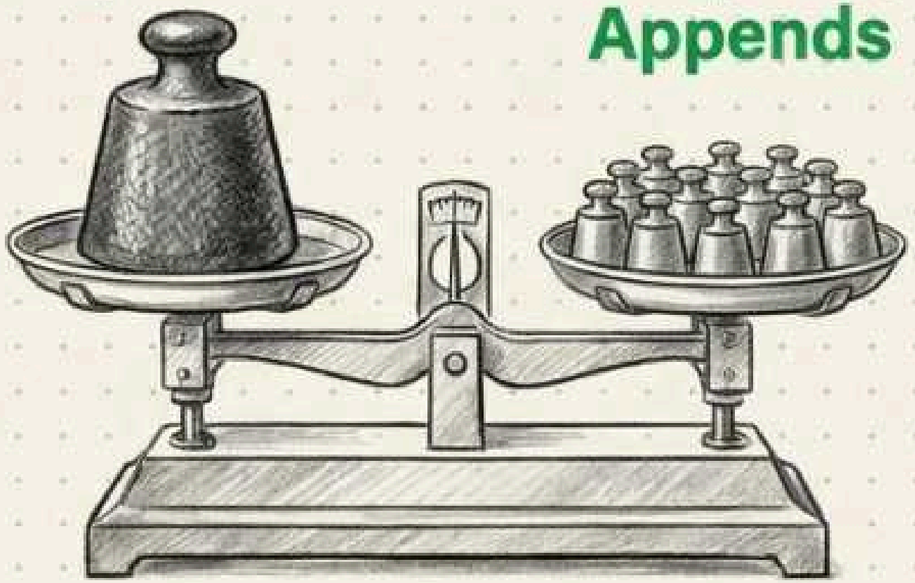
Dynamic Arrays and Amortized Analysis

¶ 1.11



One **O(n)** **Expansion**

Many **O(1)** **Appends**

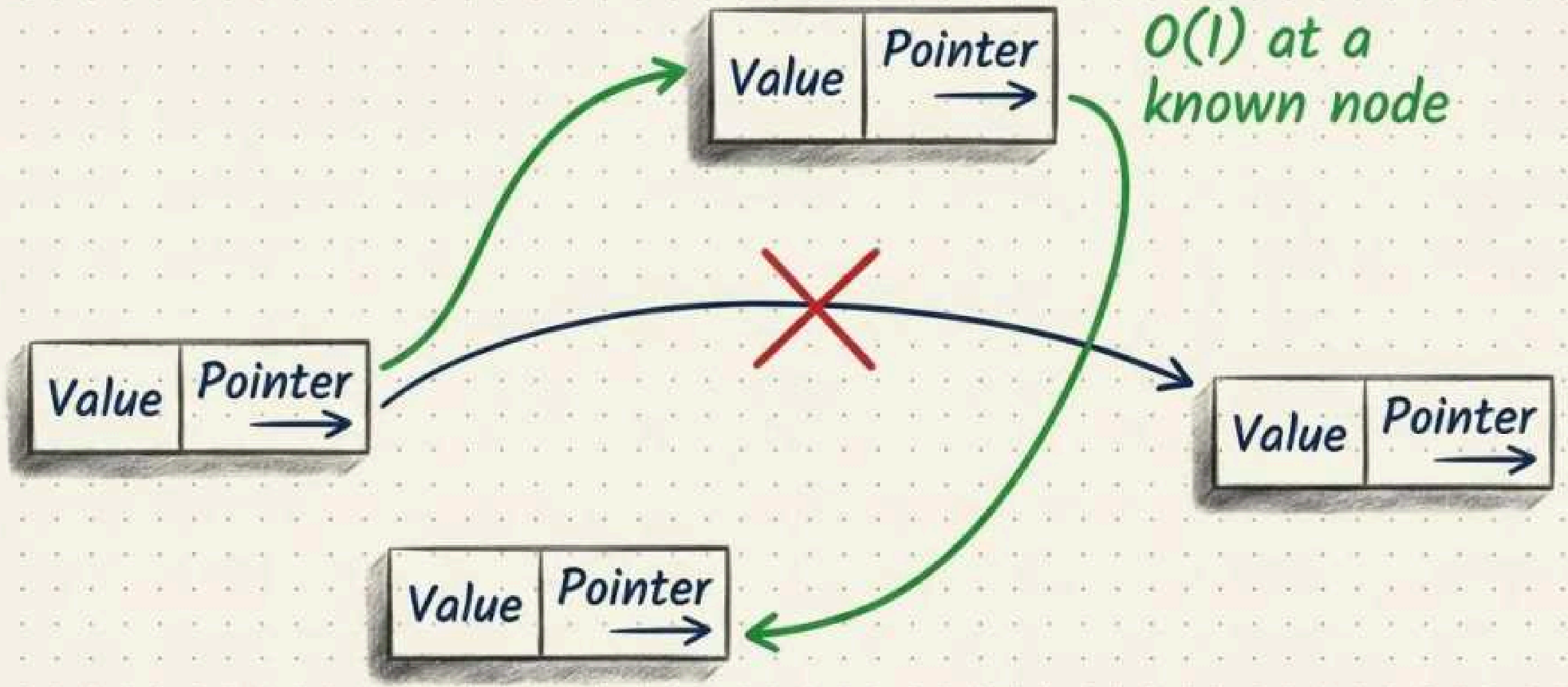


Amortized Analysis: Rare expensive steps are paid for by many cheap ones, yielding **an average cost of O(1).**

Gemini Notebook

Linked Lists Allow Instant Insertion Anywhere

¶ 1.12



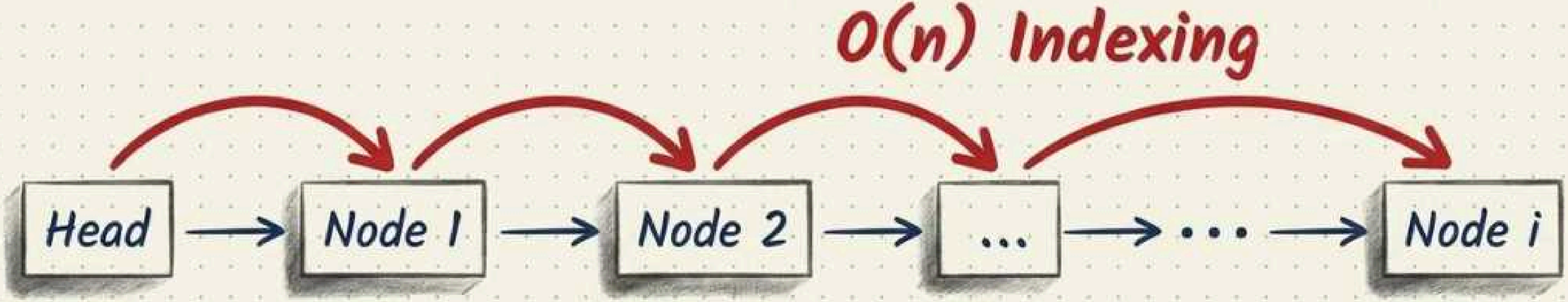
Zero shifting required. No copying ever.

Ideal for a stack growing one item at a time. Traces back to Lisp (1958).

Gemini Notebook

The Navigation Penalty of Linked Lists

¶ 1.13



To reach position i, you must follow i individual pointers.

Variant: Doubly Linked List



- Allows **O(1) removal** of a **known node**.
- **Cost**: Incurs **extra memory overhead** for the previous pointer in **every single** node.

Gemini Notebook

The Ultimate Space-Time Trade-Off Matrix

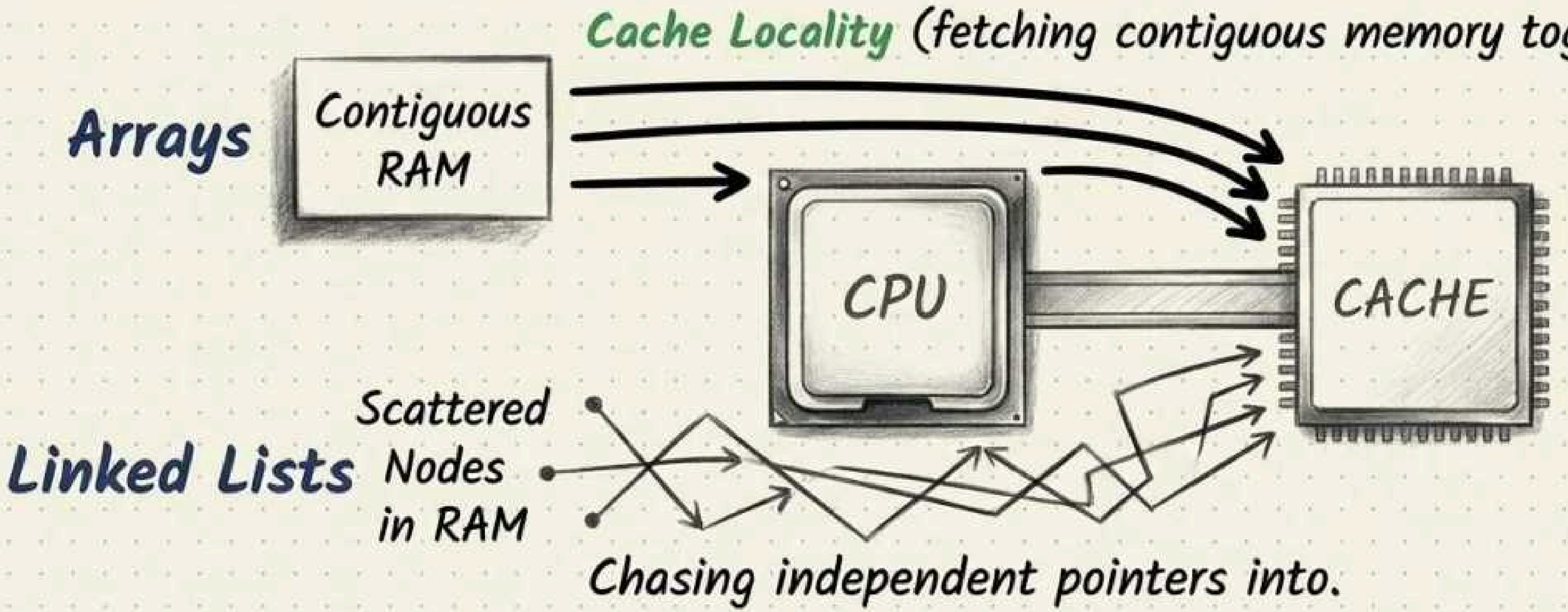
	Pros	Cons
Arrays	*Instant* indexing **O(1)**, compact contiguous storage.	*Slow mid-insertion* **O(n)**, fixed physical size.
Linked Lists	*Instant mid-insertion* **O(1)**, easy infinite growth.	*Slow* indexing **O(n)**, extra memory overhead for pointers.

***Neither wins everywhere* — the ultimate trade-off.**

Gemini Notebook

Hardware Realities: Why Constants Matter

¶ 1.15

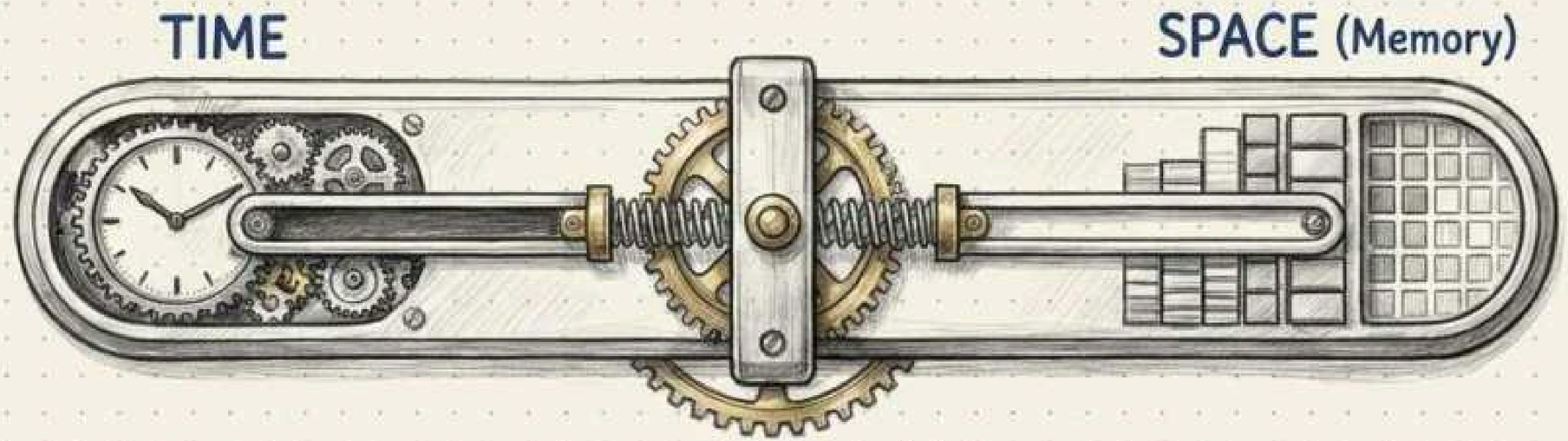


Big O ignores constants, but hardware processors don't. Arrays are the default choice in practice; **linked lists** earn their keep only when frequent mid-insertion is the primary need.

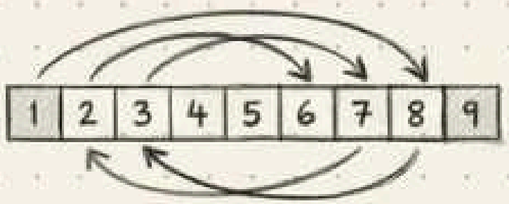
Gemini Notebook

Balancing the Space-Time Scale

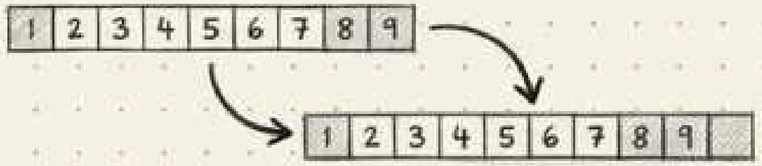
¶ 1.16



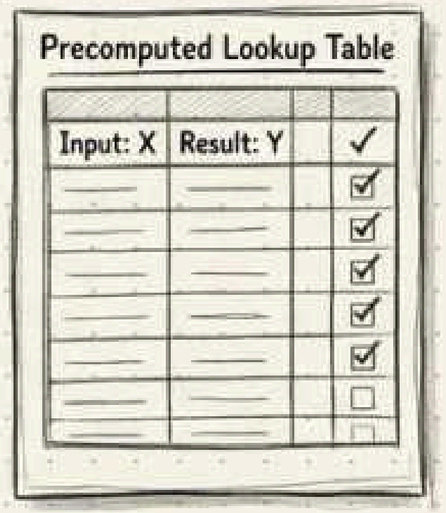
- Space Efficient: In-place sorting (Uses O(1) extra memory, but might consume more time).



- Time Efficient: Copying data (Uses O(n) extra memory to drastically speed up processing).



Memoization



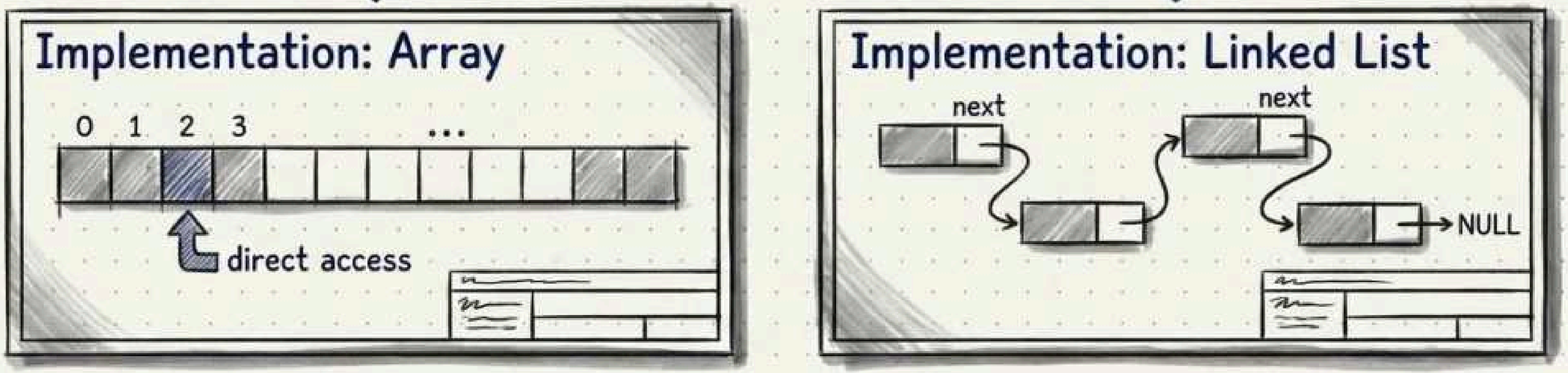
Storing answers we have already computed (spending Space) to avoid repeating expensive work (saving Time).

Gemini Notebook

Abstract Data Types vs. Implementation

¶ 1.17

Abstract Data Type (Contract)
List supports: get, insert, remove

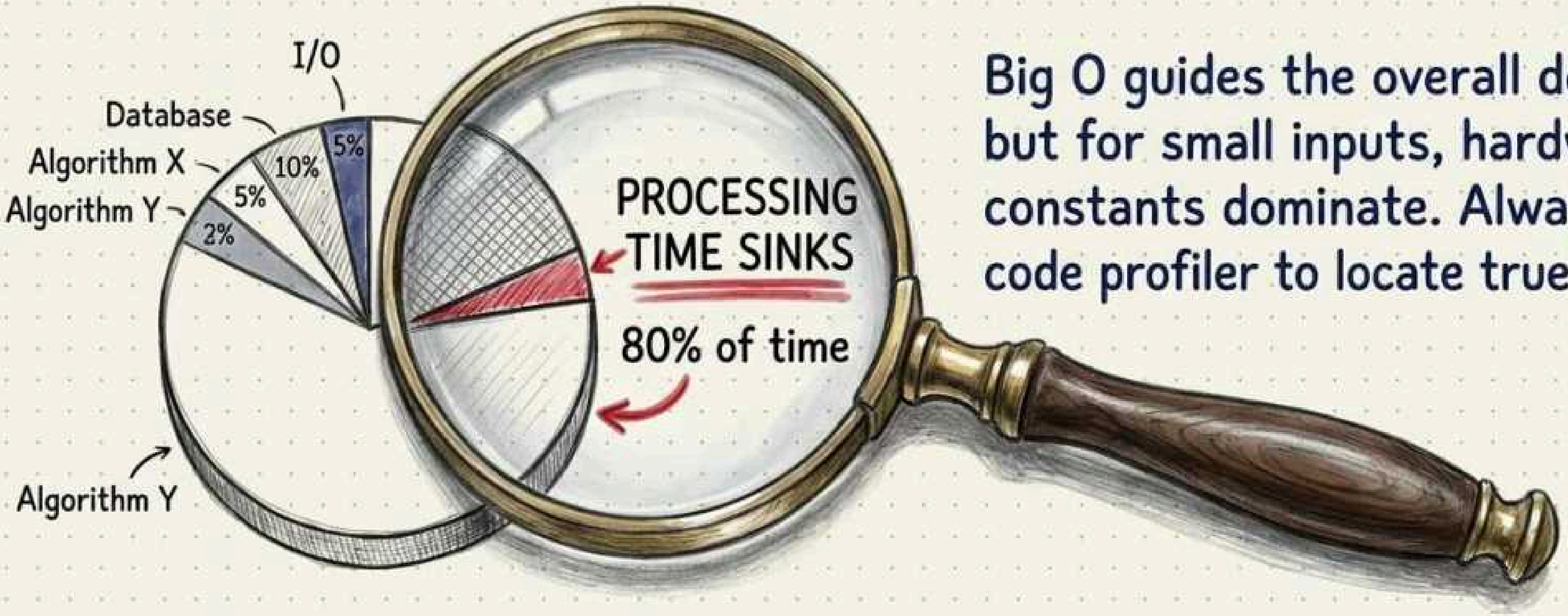


Programmers pick the implementation based on operation frequency. This strict separation allows swapping the underlying machinery later without rewriting the entire program.

Gemini Notebook

Measure Before You Optimize

¶ 1.18



Big O guides the overall design, but for small inputs, hardware constants dominate. Always use a code profiler to locate true time sinks.

"Premature optimization is the root of all evil."
— Donald Knuth, 1974

Gemini Notebook

Closing Exercise and Next Steps


```
function findLargest(list of n items):
    max = list[0]   // Assume first is largest
    for i from 1 to n-1:
        if list[i] > max:  // Comparison
            max = list[i]
    return max
```

Exercise 1: Find the largest value in a list of n items. Count comparisons. State the Big O.

Exercise 2: Change the loop to find the largest AND second largest. Does the Big O change?

Lecture 2 Teaser: Stacks, Queues, and Hash Tables

Gemini Notebook