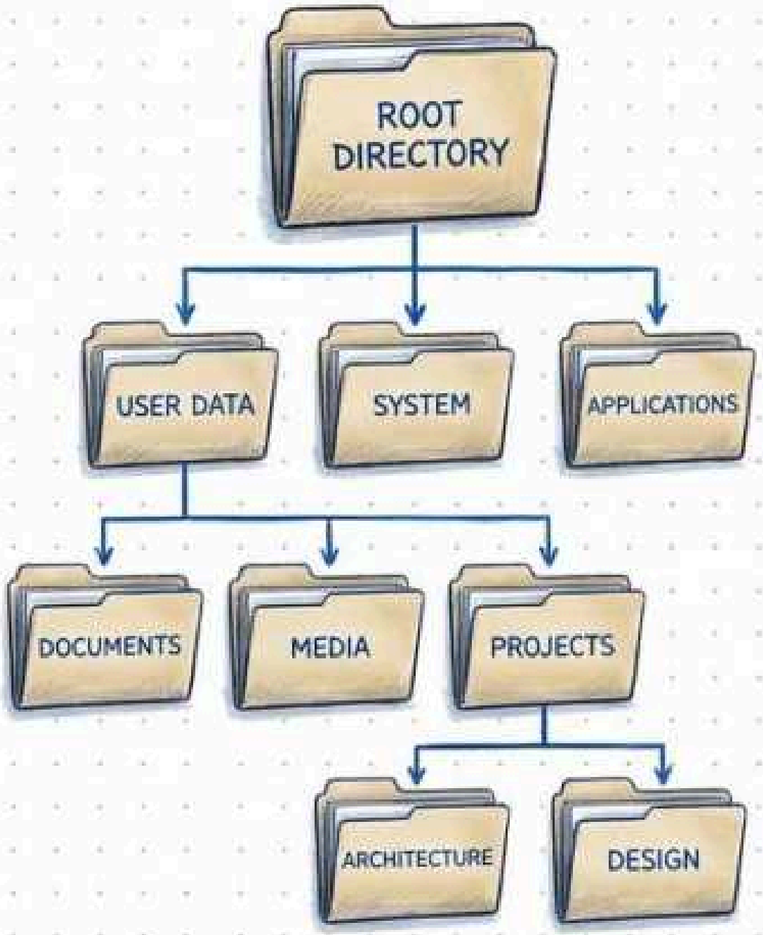
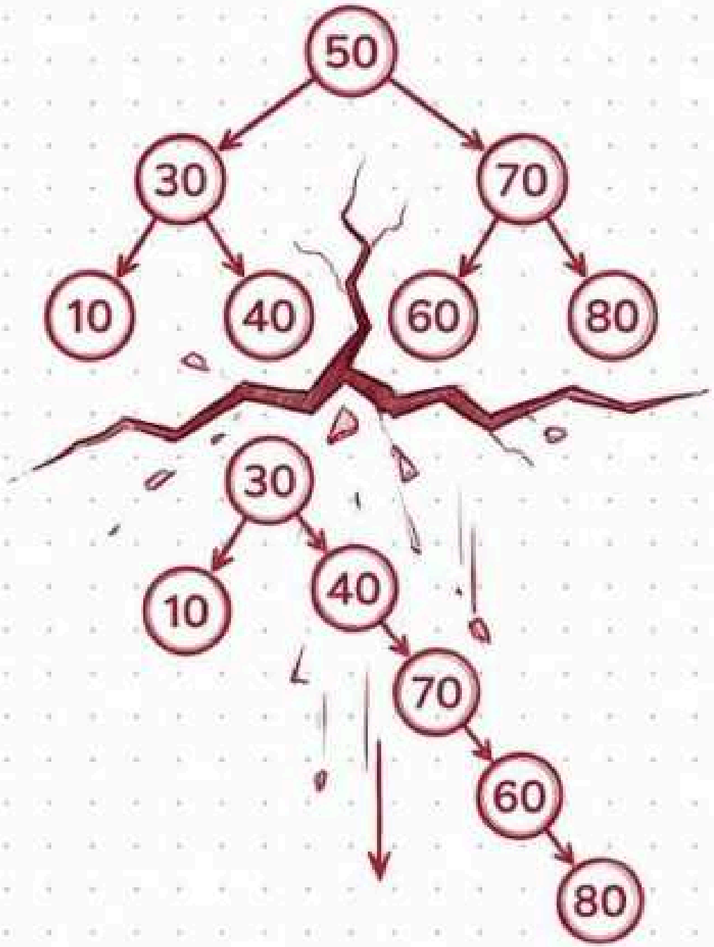
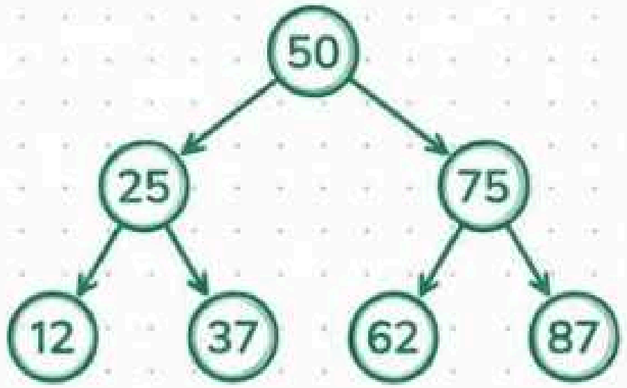
Three Architectural Inquiries

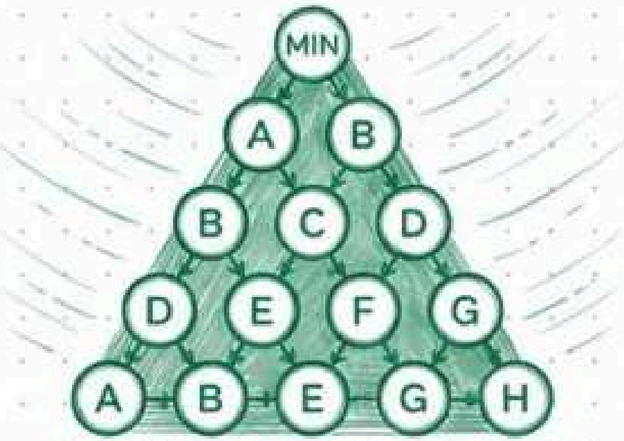


Hierarchy: Storing Data



Order & Failure: Binary Search Trees

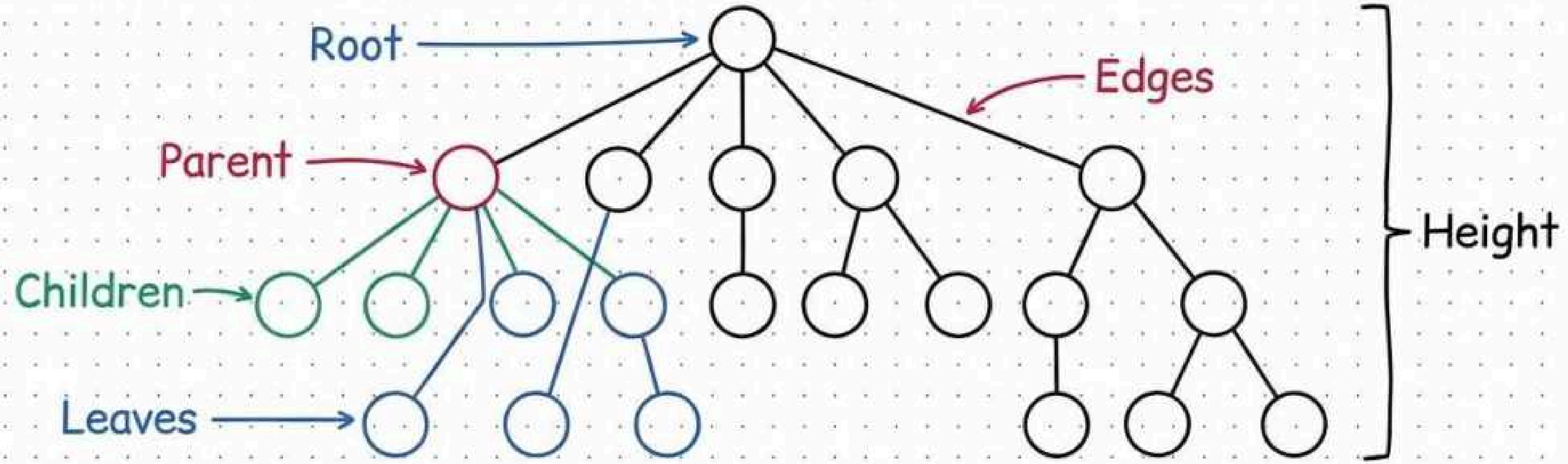




Speed Guarantees: Heaps & Balancing

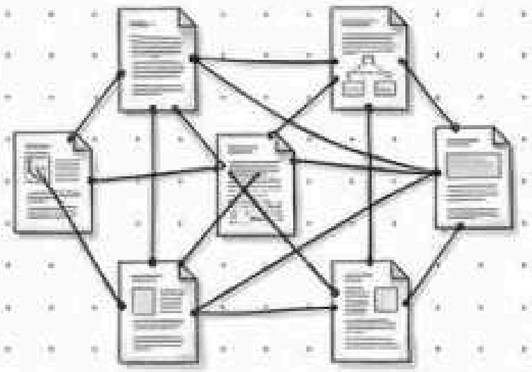
Gemini Notebook

The First Structure That Is Not a Line





File Systems



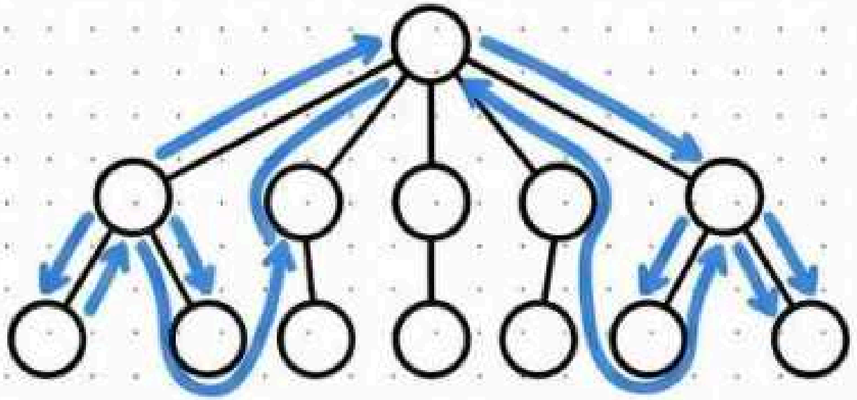
Sitemaps



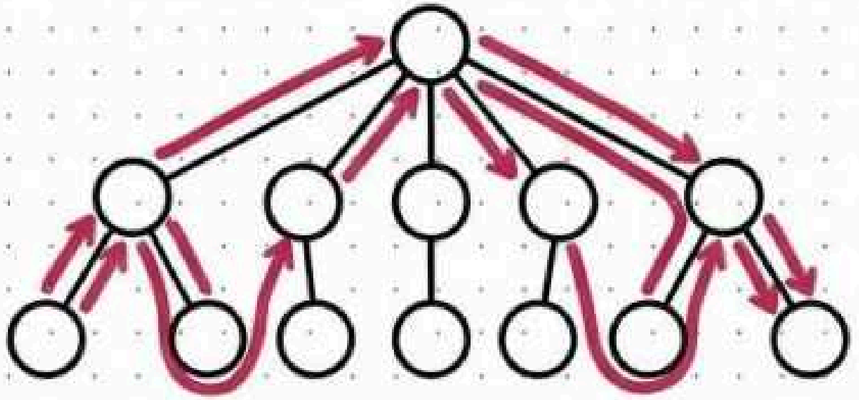
Org Charts

Gemini Notebook

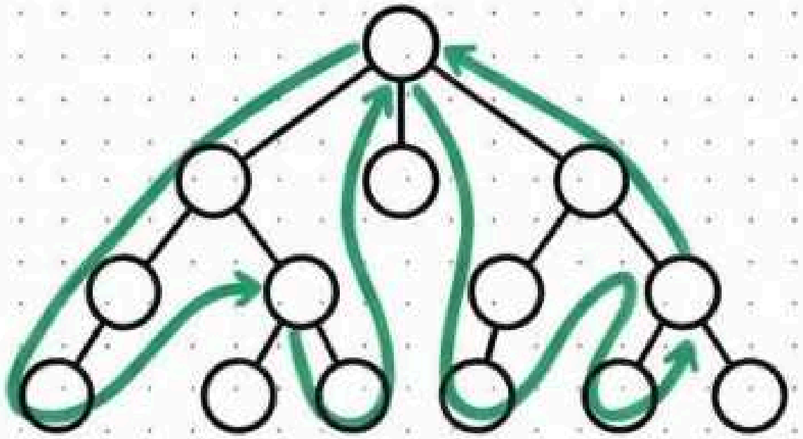
Traversing the Binary Tree in O(n) Time



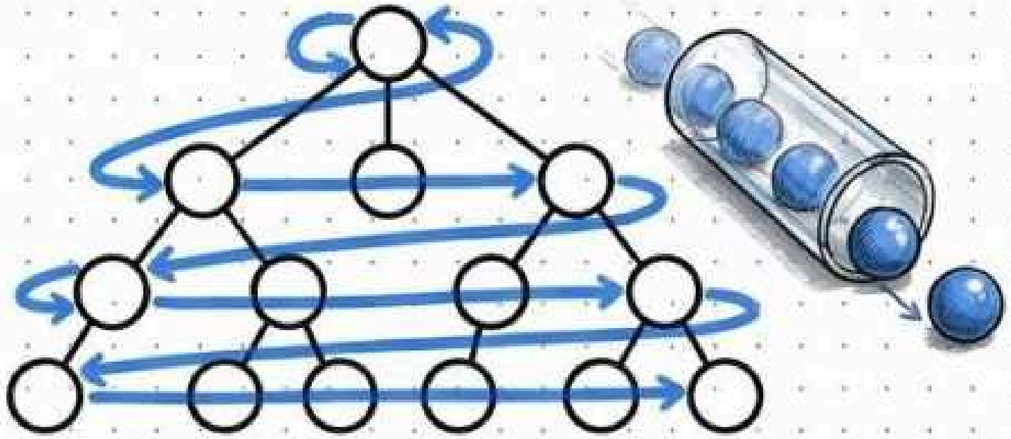
Pre-order: Root → Left → Right



In-order: Left → Root → Right



Post-order: Left → Right → Root

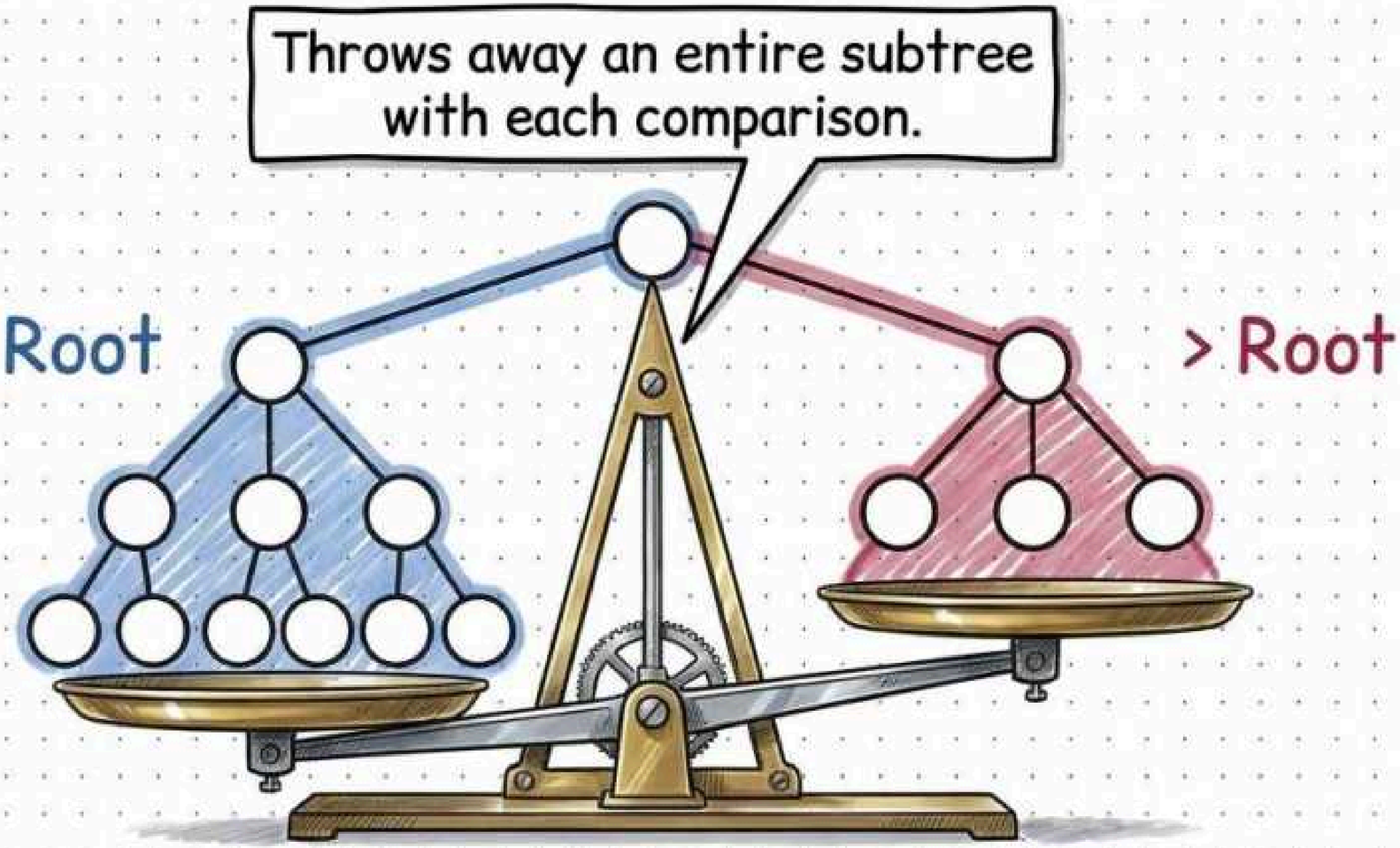


Level-order: Row by row

Every traversal visits each node exactly once, guaranteeing O(n) complexity.

Gemini Notebook

The Ordering Rule



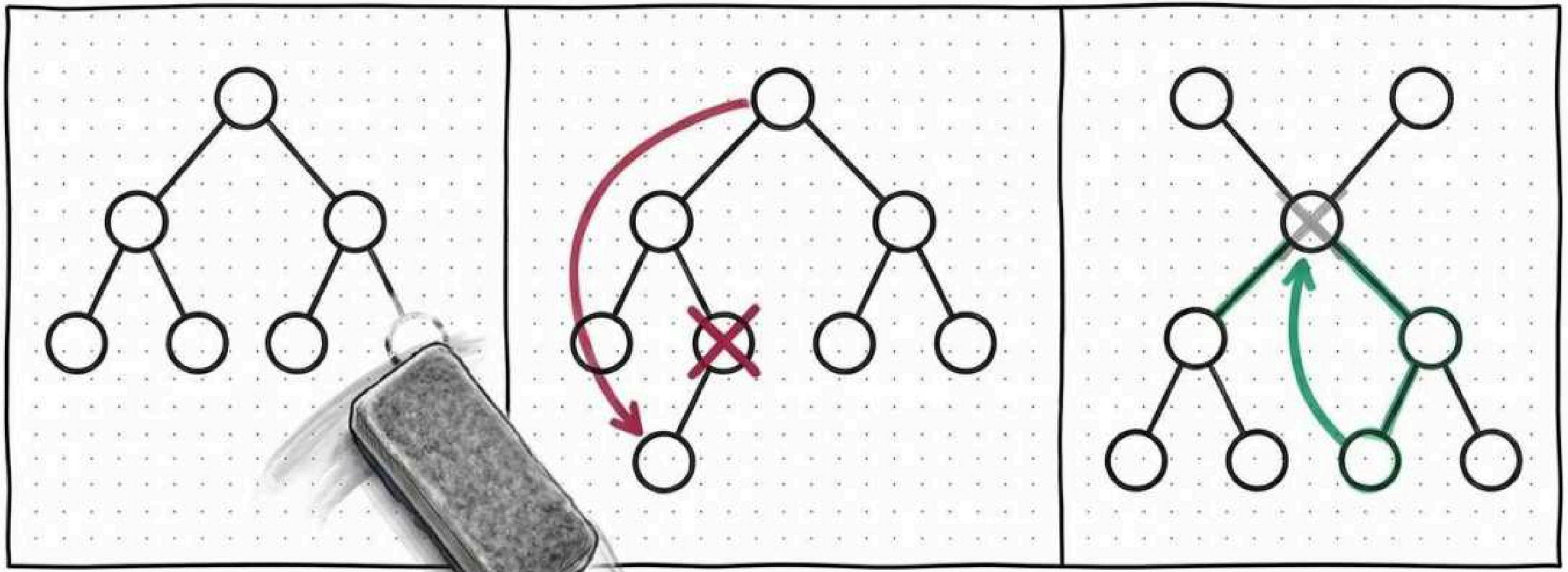
Search Cost ∝ Tree Height

Searching 1,000,000 keys ≈ 20 comparisons

Identical logic to Binary Search on an array.

Gemini Notebook

Deletion Mechanics: Three Cases



Case 1: Leaf
Simply removed.

Case 2: One Child
Replaced by child.

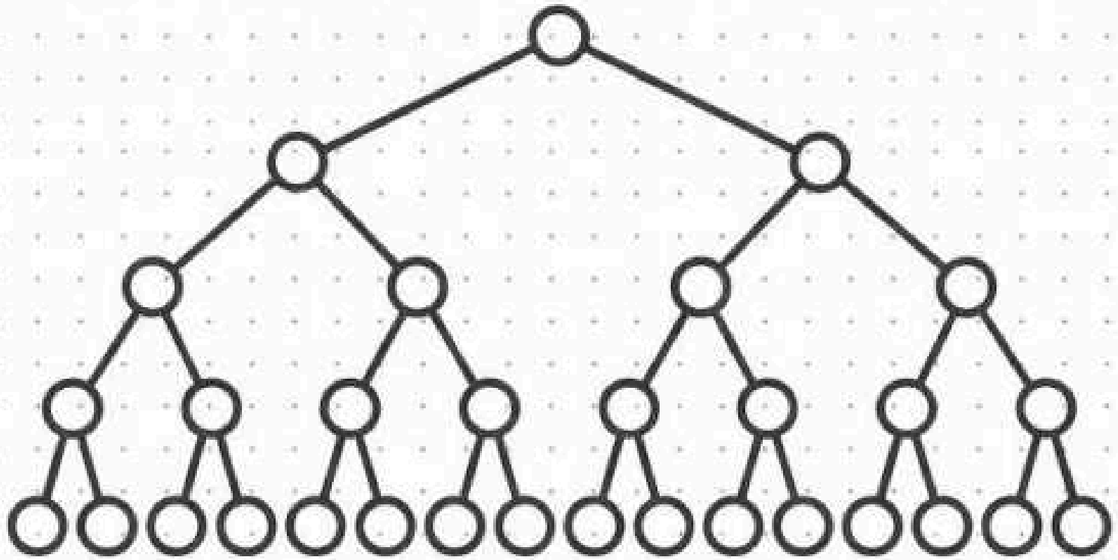
Case 3: Two Children
Replaced by in-order successor.

Gemini Notebook

The Shape Trap

¶ 3.6

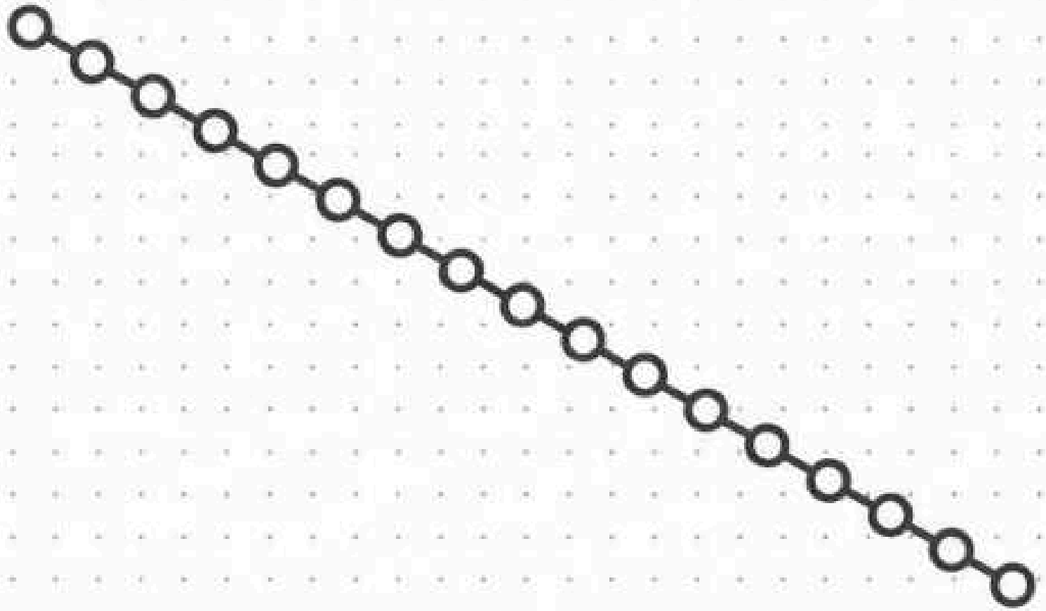
Random Insertion



Height ∝ O(log n)

Fast.

Sorted Insertion



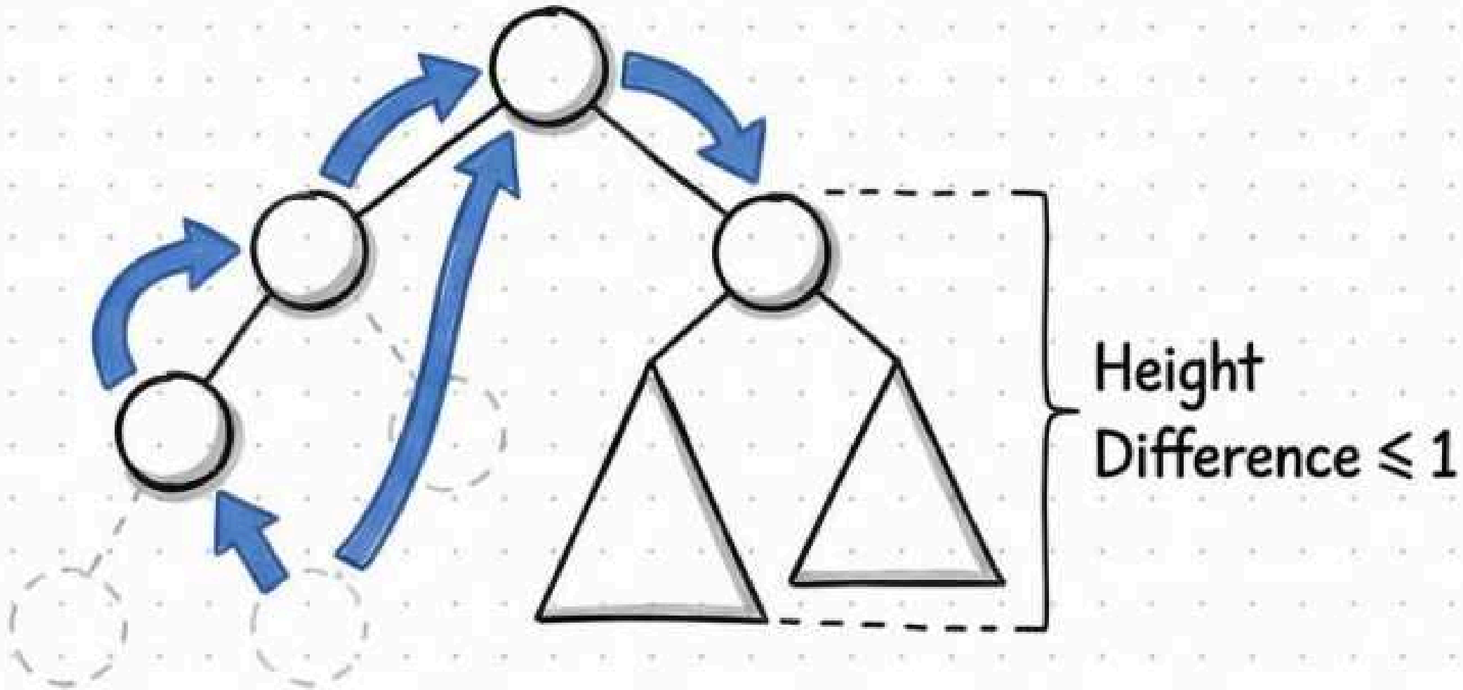
Height = n

Operations degrade to O(n).

A trap for production systems.
The tree degenerates into a linked list.

Gemini Notebook

Strict Equilibrium: The AVL Tree (1962)



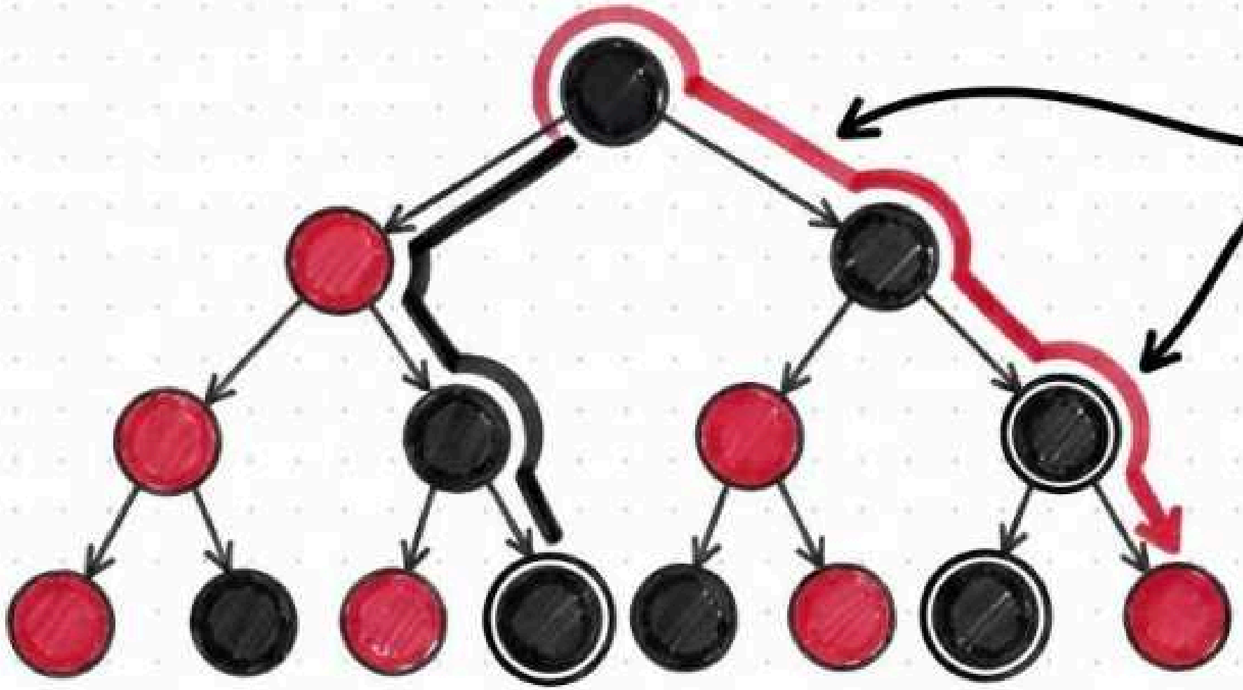
- The Mechanism: Local rotations restore order immediately after insertion or deletion.
- Attribution: Adelson-Velsky and Landis.

Search, Insert, and Delete guaranteed at O(log n)

Gemini Notebook

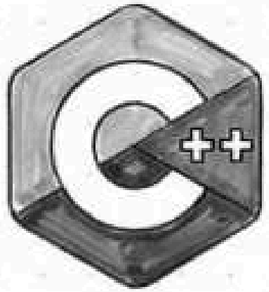
Looser Guarantees: The Red-Black Tree

¶ 3.8



$$\text{Longest Path} \leq 2 \times \text{Shortest Path}$$

The Trade-off: Looser balancing rule = less rebalancing work.



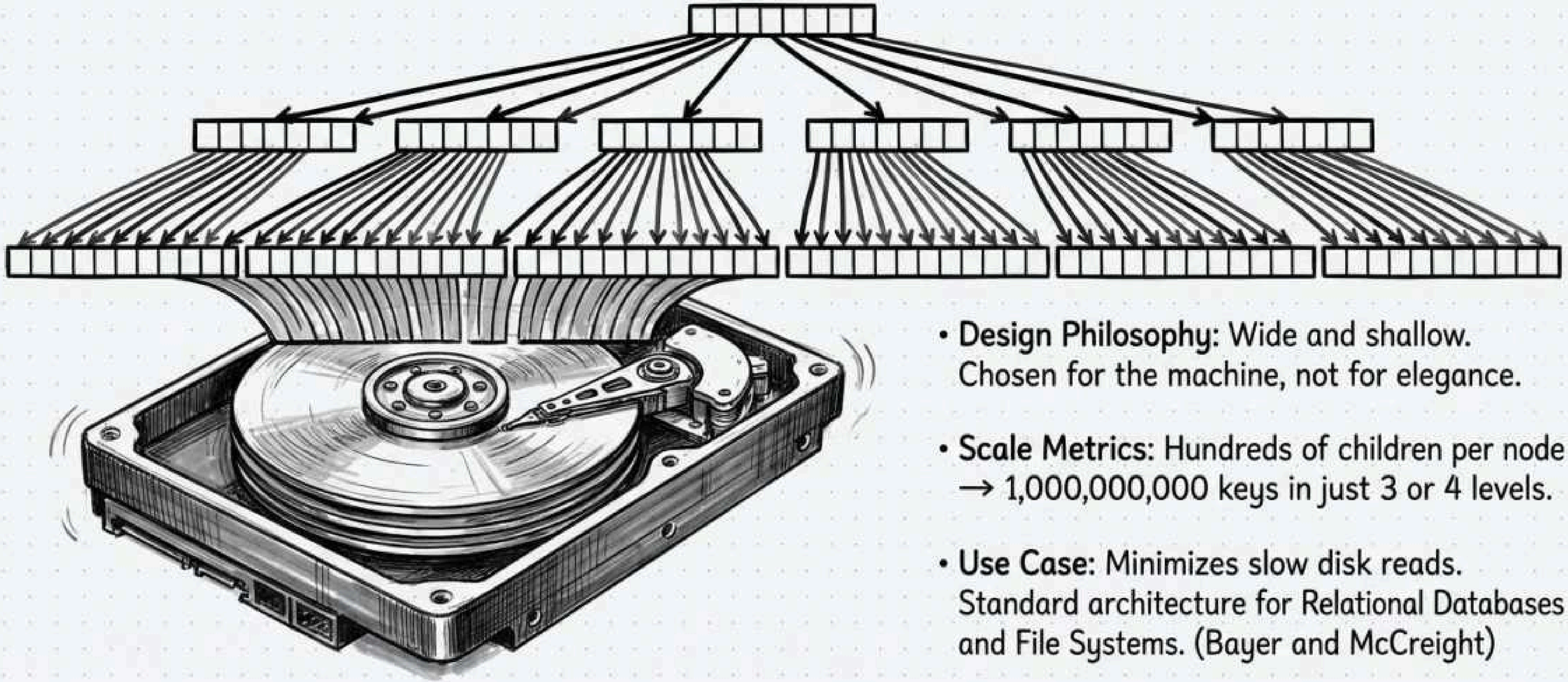




Underlies ordered maps and core kernels.

Gemini Notebook

Optimized for the Machine: B-Trees (1970)



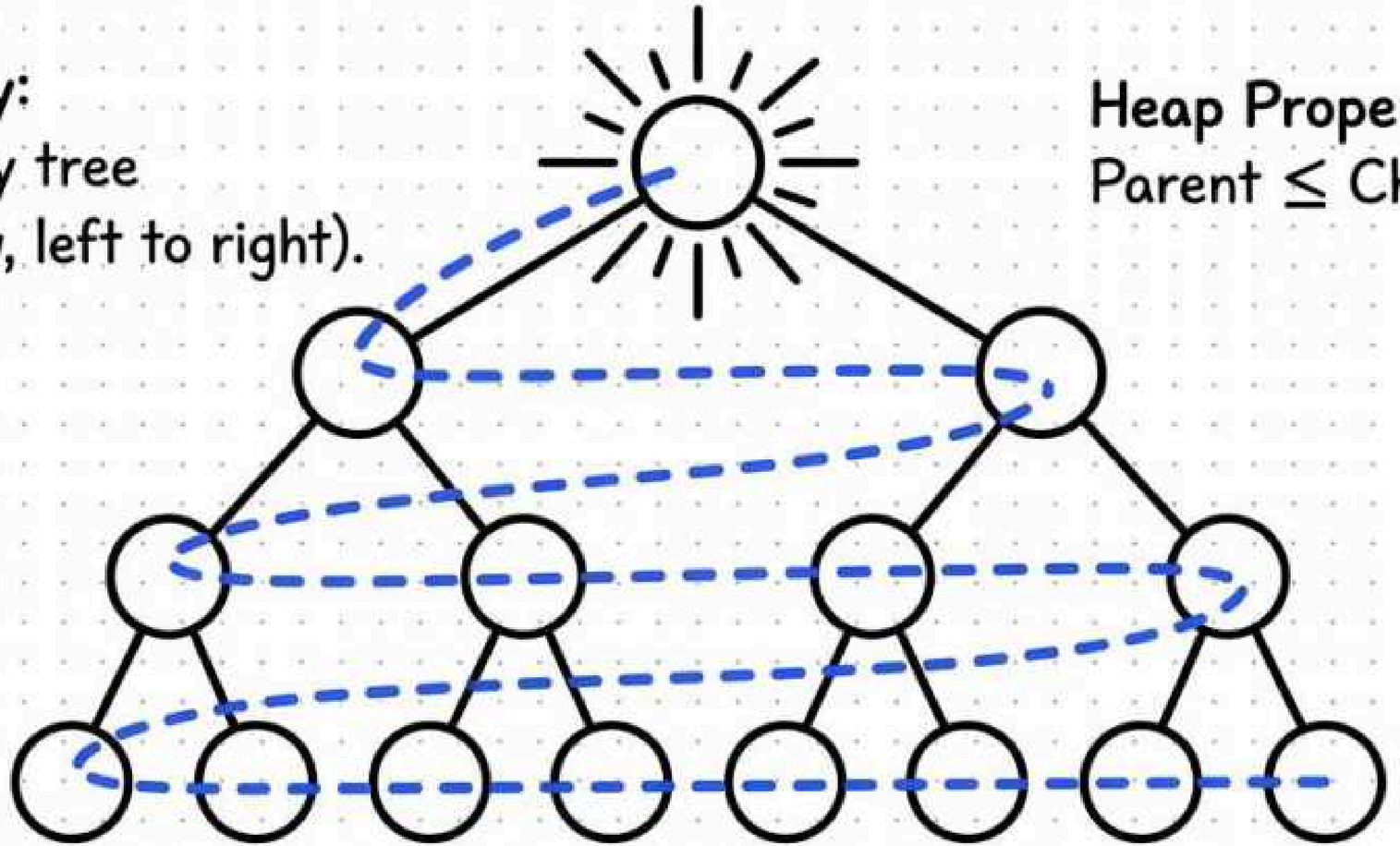
- **Design Philosophy:** Wide and shallow. Chosen for the machine, not for elegance.
- **Scale Metrics:** Hundreds of children per node → 1,000,000,000 keys in just 3 or 4 levels.
- **Use Case:** Minimizes slow disk reads. Standard architecture for Relational Databases and File Systems. (Bayer and McCreight)

Gemini Notebook

The Heap Paradigm

Shape Property:
Complete binary tree
(fills row by row, left to right).

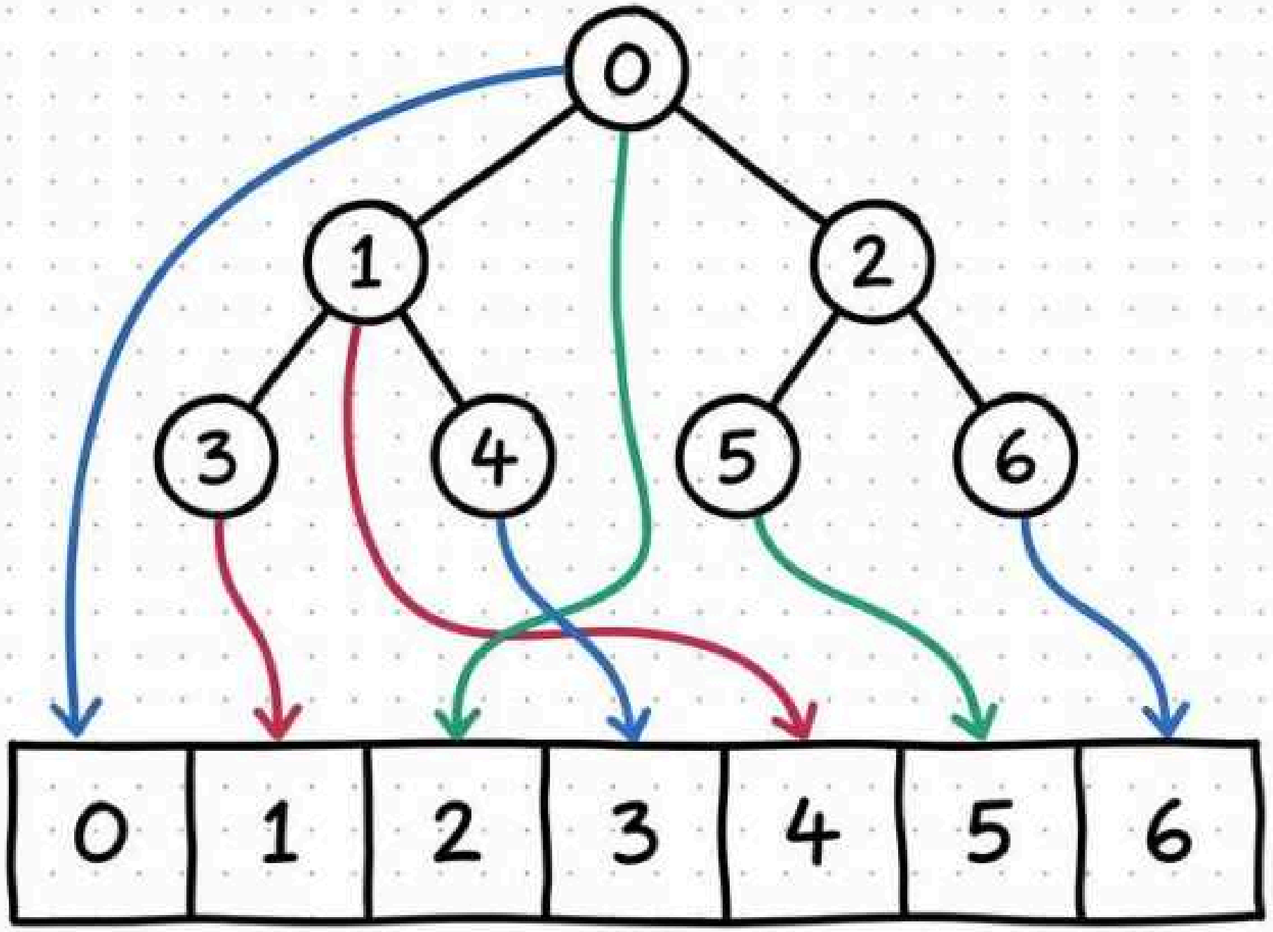
Heap Property (Min-Heap):
Parent ≤ Children.



The Result: The smallest item is always trapped at the root.
Direct access in O(1).

Gemini Notebook

Zero Pointers: Array Mapping



Left Child: $2i + 1$

Right Child: $2i + 2$

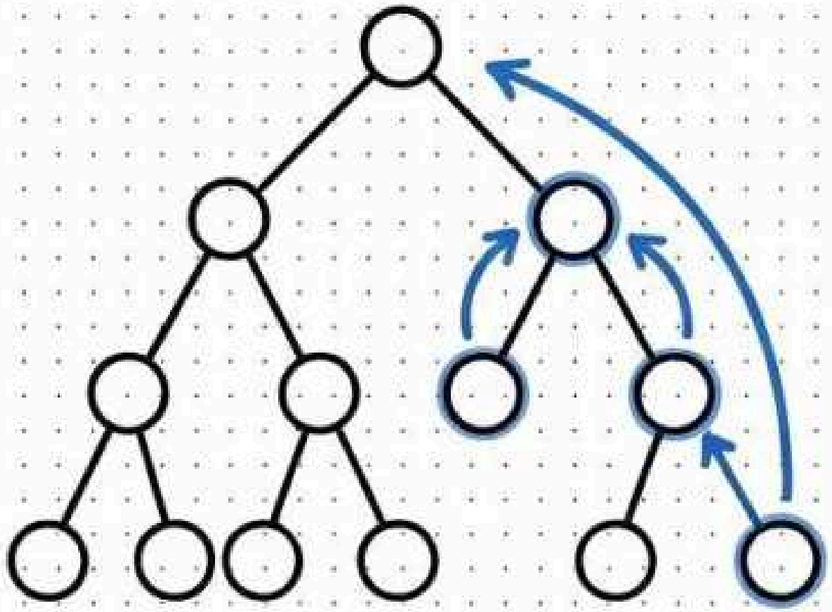
Parent: $\left\lfloor \frac{i-1}{2} \right\rfloor$

Compact and cache-friendly. Zero memory wasted on pointers.

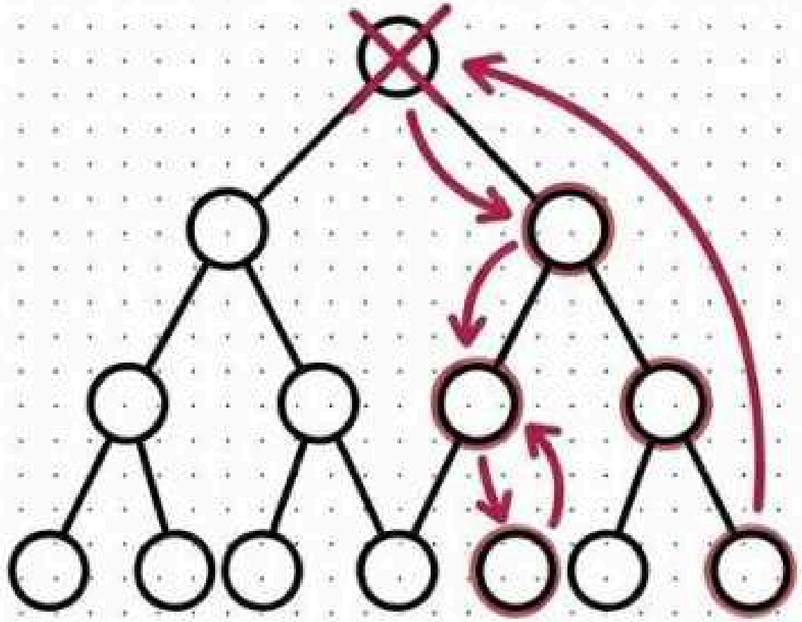
Gemini Notebook

¶ 3.12

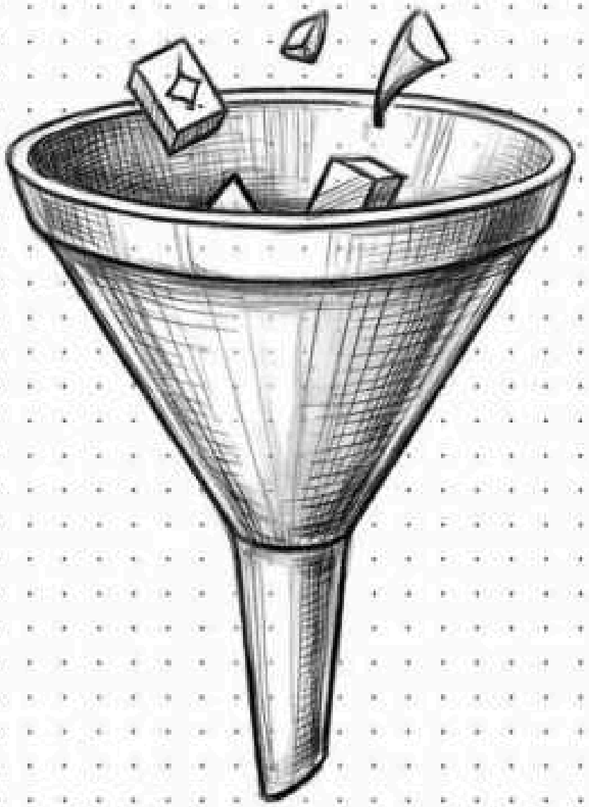
Maintaining Partial Order: O(log n) Operations



Insert: Place at end → Sift Up



Extract Min: Remove root, move last to root → Sift Down



Keeps only as much order as it needs, which is why it is cheap.

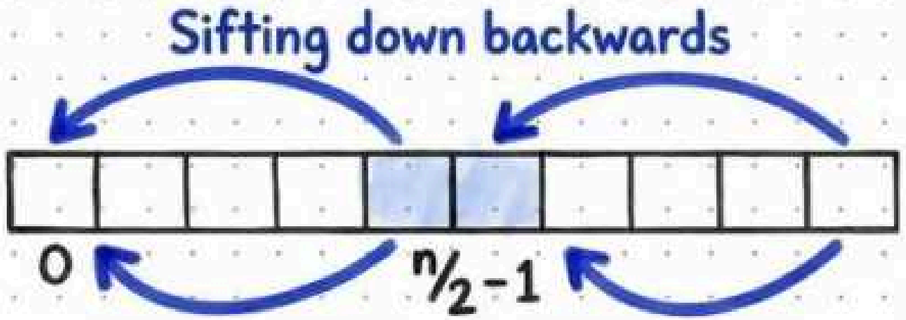
Gemini Notebook

Building and Sorting

¶ 3.13

[Unsorted Array] → [Heap] → [Sorted Array]

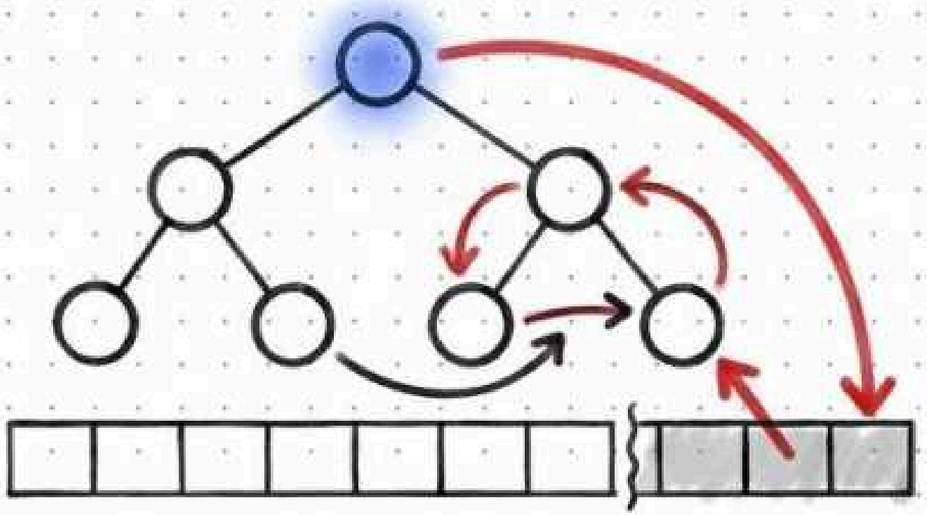
Phase 1: Building a Heap (Floyd, 1964)



O(n) time.

Not O(n log n).

Phase 2: Heapsort (Williams, 1964)

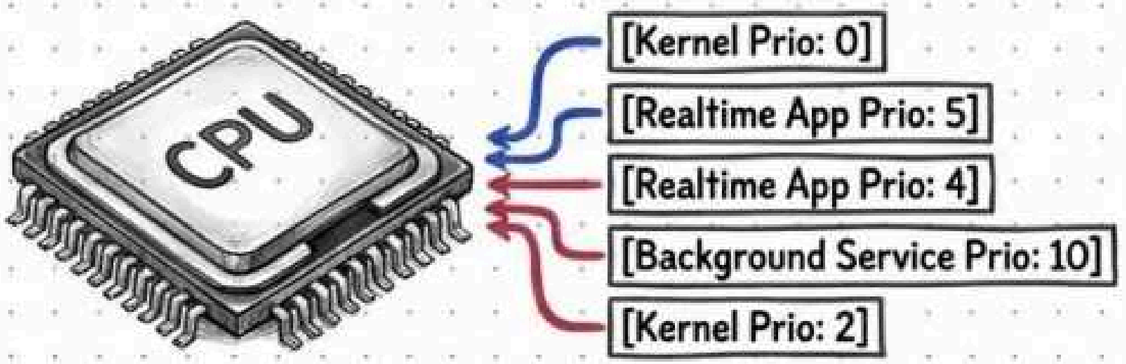


O(n log n) time, entirely in-place.

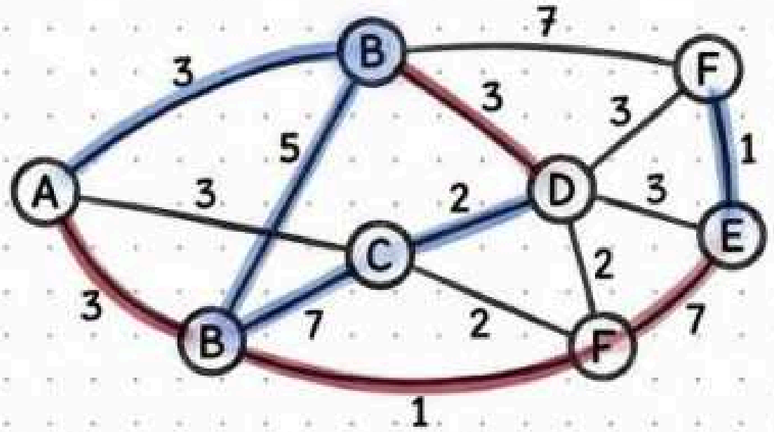
Gemini Notebook

Heaps in the Wild

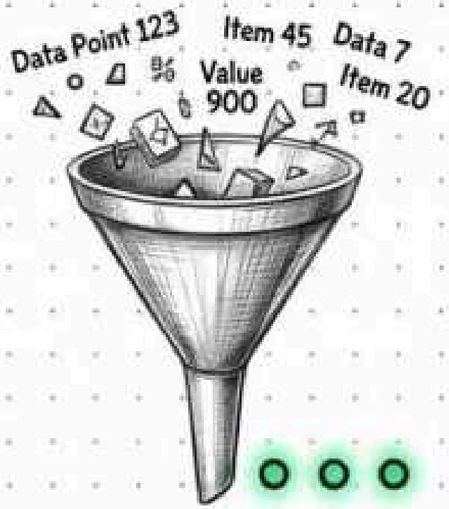
¶ 3.14



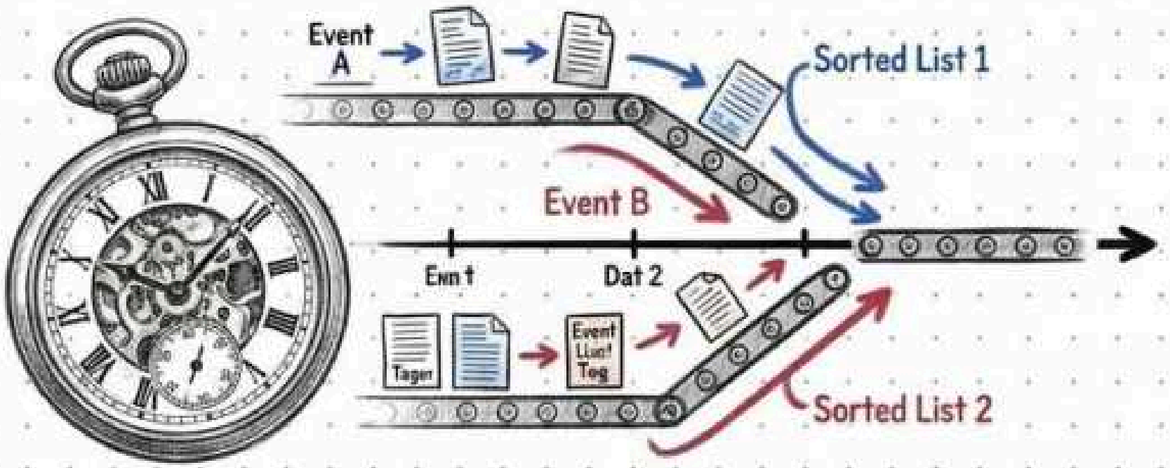
OS Task Scheduling: By priority.



Dijkstra's Algorithm: Finding shortest paths (Lecture 4).



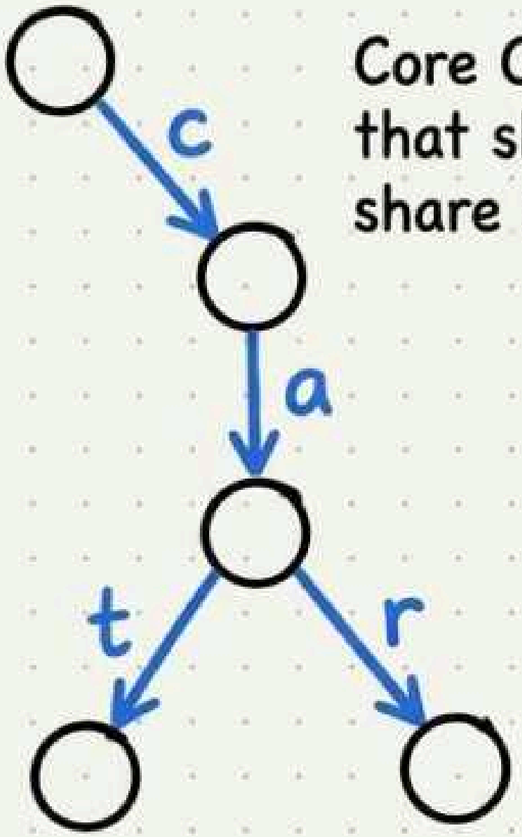
Data Streams: Finding top k items in n stream costs O(n log k).



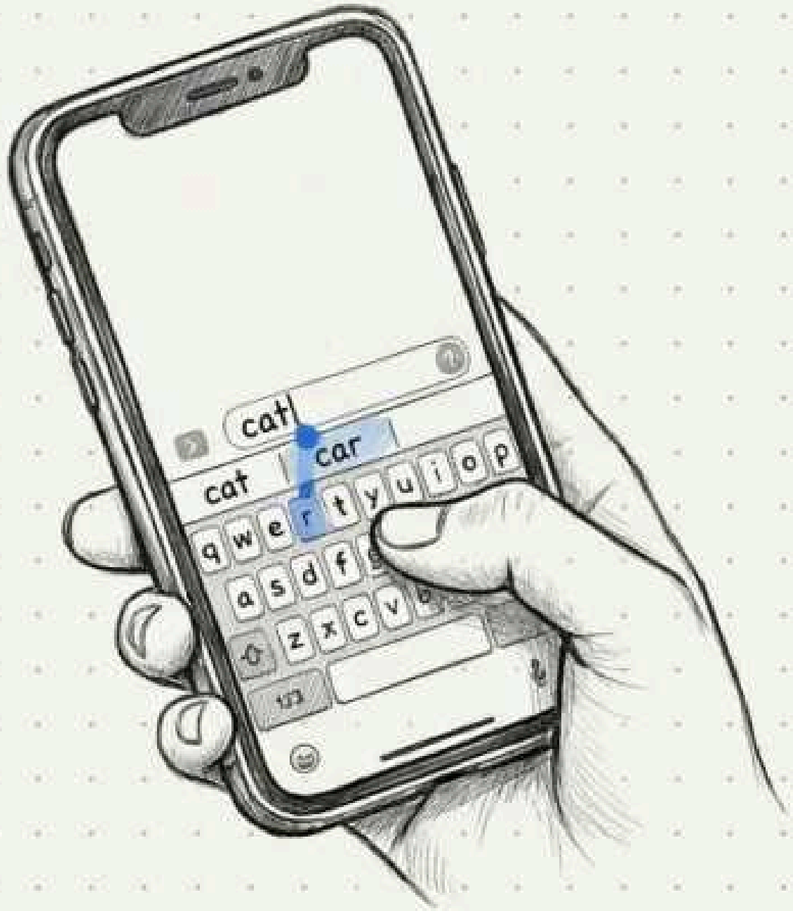
Other Uses: Event simulations, merging sorted lists.

Gemini Notebook

Prefix Sharing: The Trie



Core Concept: Words that share a prefix share a physical path.



Speed: Lookup time is proportional to word length, not the total number of words.

Applications: Autocomplete, spell checkers, routing tables.

Tradeoff: Requires considerable memory.

Gemini Notebook

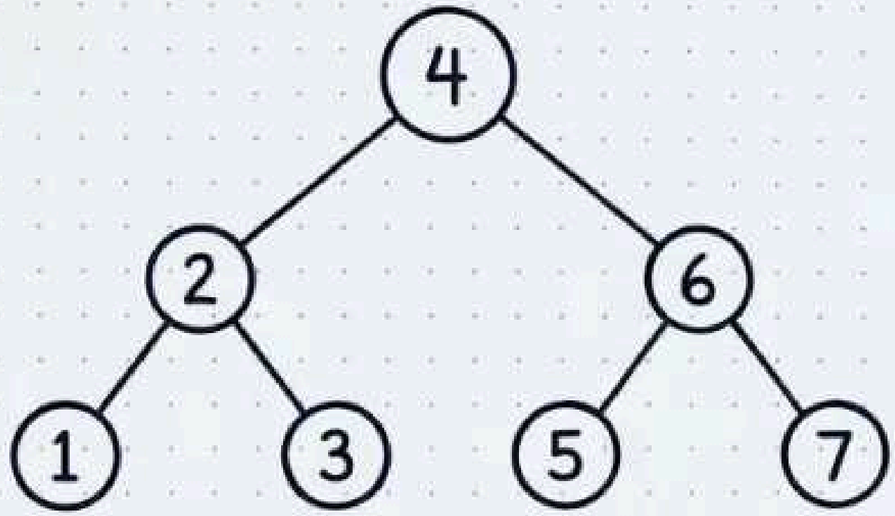
The Architecture Matrix

¶ 3.16

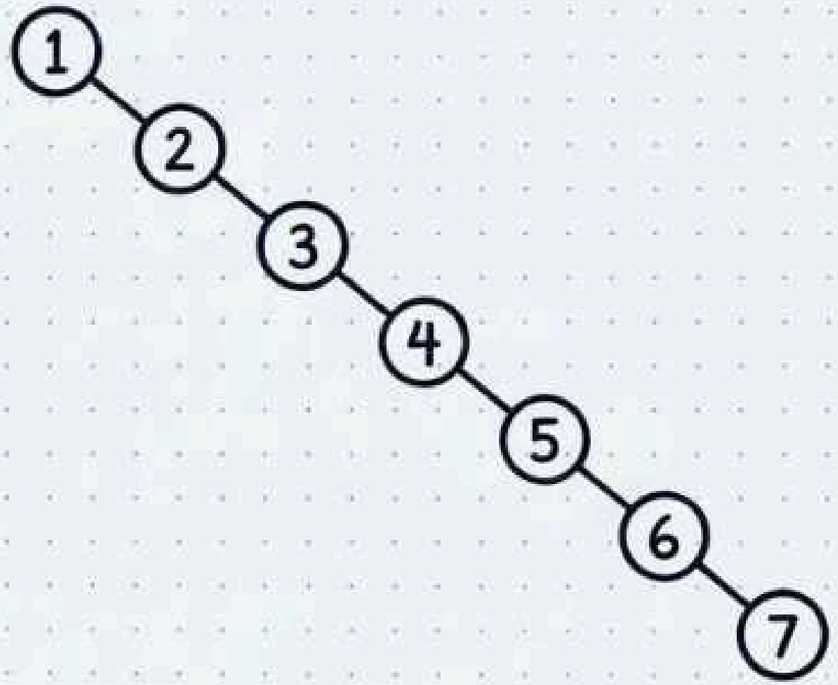
Structure	Performance	Properties
Balanced Search Tree	$O(\log n)$ Search, Insert, Delete.	Maintains sorted order, allows range queries.
Hash Table	$O(1)$ on average.	Fast, but no inherent order.
Heap	Min/Max in $O(1)$, Insert in $O(\log n)$.	Extremely cheap, but cannot search for arbitrary keys quickly.

Gemini Notebook

The Shape of the Same Data



Inserting in order {4, 2, 6, 1, 3, 5, 7} yields Height 2.



Inserting in order {1, 2, 3, 4, 5, 6, 7} yields Height 6.

The exact same data can create vastly different structures.

Next Up: Graphs
(Lec

Gemini Notebook