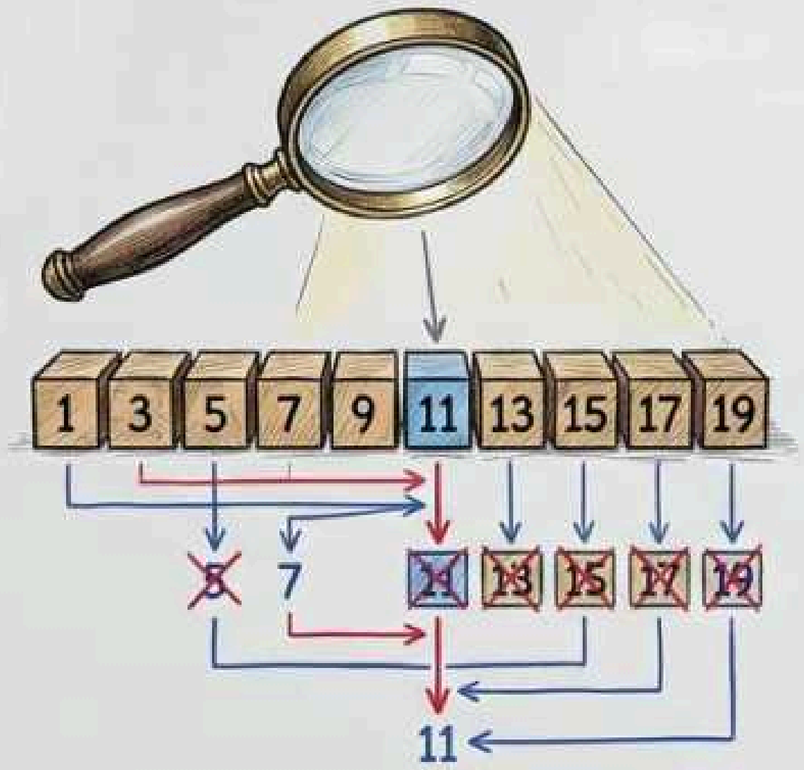
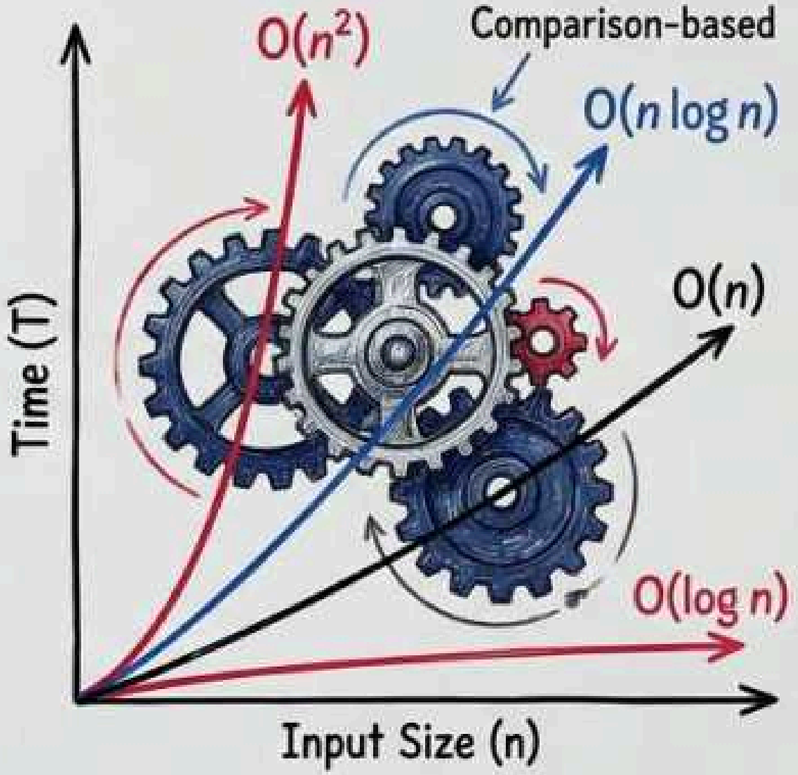
¶ 5.1



How does binary search find items so quickly?

By repeatedly dividing the search interval in half. Start with the whole range.
If the target value is lower than the middle element, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeat until the target is found or the interval is empty.
Time Complexity: O(log n).



How do sorting algorithms work & what do they cost?

Sorting algorithms rearrange elements in a specific order. Common examples include Bubble Sort, Selection Sort, Insertion Sort (typically $O(n^2)$), Merge Sort, Quick Sort, and Heap Sort (typically O(n log n)).
Their "cost" is measured by time complexity, indicating how their runtime grows with input size.

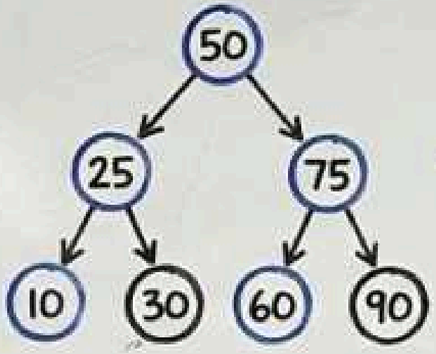


Is there a limit on how fast sorting can be?

Yes. For comparison-based sorting algorithms, the theoretical lower bound for average and worst-case time complexity is Ω(n log n). This is due to the minimum number of comparisons required to distinguish between all possible orderings (permutations) of n elements, often visualized using a decision tree.

Gemini Notebook

¶5.2



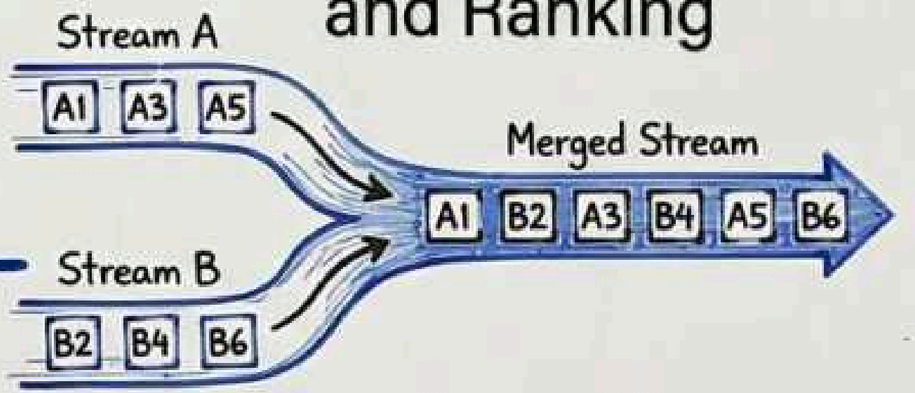
Enables
Binary Search

SORTING



Groups
Duplicates

Simplifies Merging
and Ranking

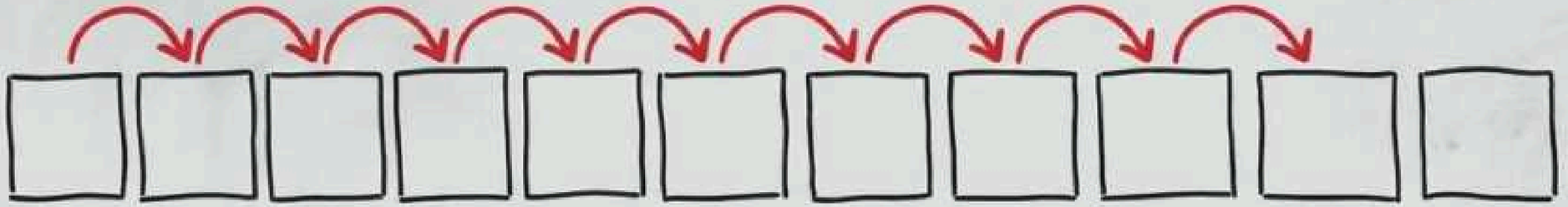


> *More than 25% of all computing time spent sorting.*
> *— Donald Knuth (1970s)*

Gemini Notebook

Linear Search: O(n)

¶ 5.3

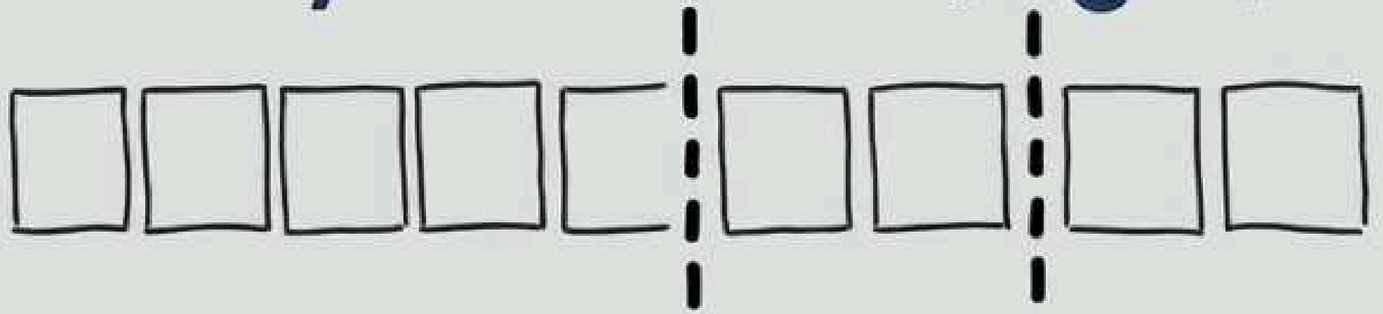


Searching 1 million items:
1,000,000 steps (Linear) vs. ~20 steps (Binary).



2006 Java Bug:
Integer overflow on (low + high)

Binary Search: O(log n)



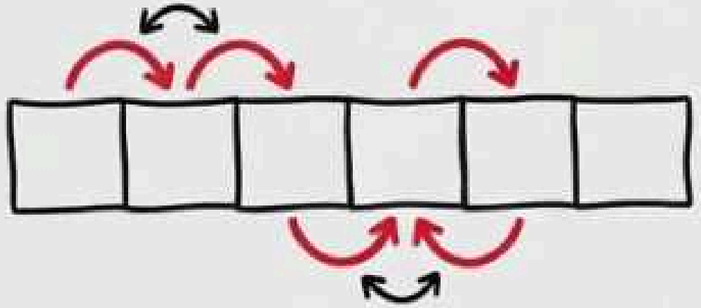
Requisite: The array MUST be sorted first.

Gemini Notebook

The Elementary Sorts: $O(n^2)$ Worst Case

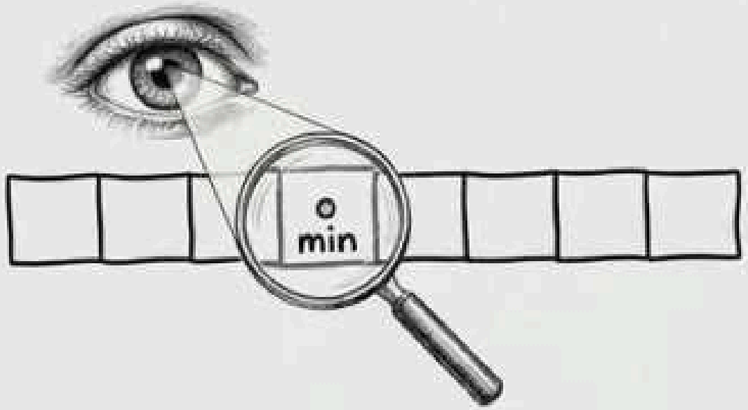
¶ 5.4

Bubble Sort



- Repeatedly swaps adjacent out-of-order items.

Selection Sort



- Scans remaining array to pluck out the absolute smallest item.

Insertion Sort



- Builds a sorted prefix by sliding new items into place.

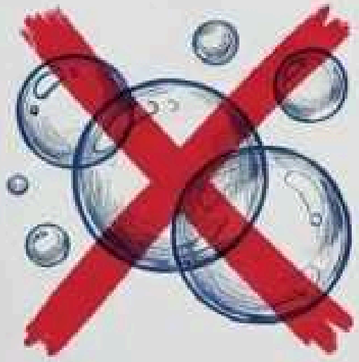
Easy to write and understand → Taught first.

The Case for Insertion Sort

¶ 5.5



Low overhead for small arrays. Used for the small pieces in real progrrmming libraries.



Bubble Sort:

Mostly a teaching example.

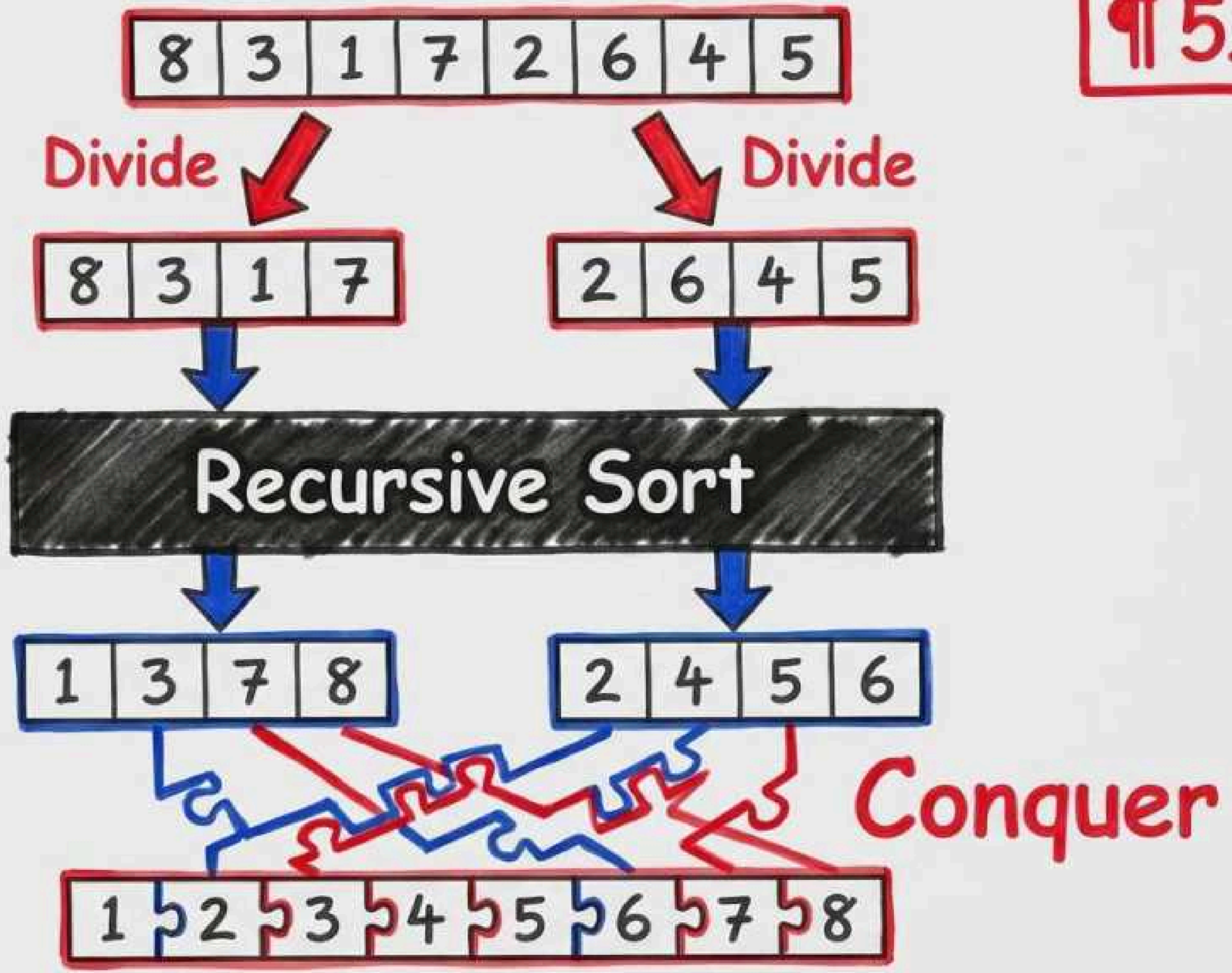
Gemini Notebook



John von Neumann, 1945

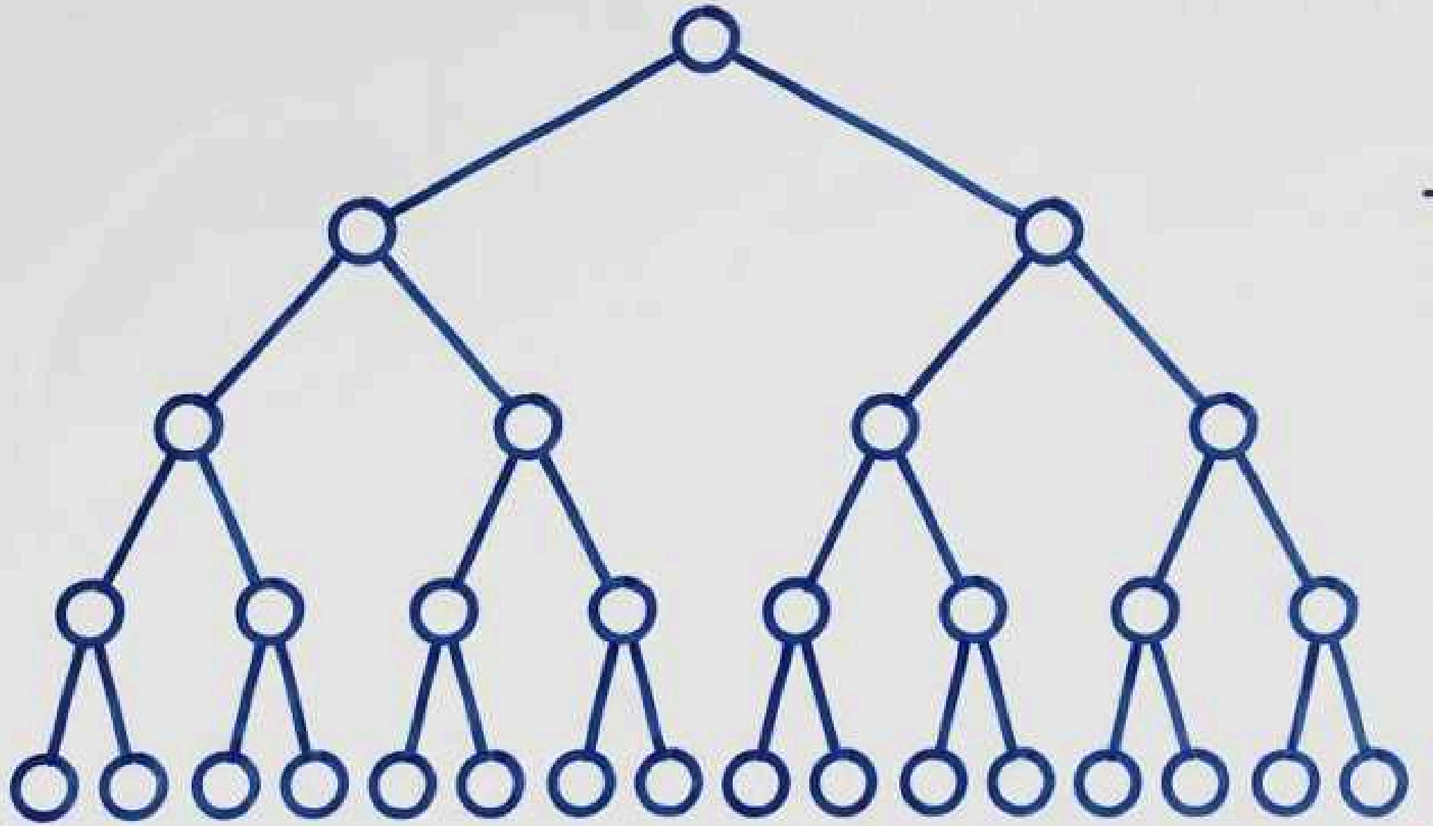
Split. Solve. Combine.

Mechanics: Repeatedly take the smaller front item.



¶ 5.6

¶ 5.7



Virtues	Drawbacks
Stable	Requires O(n) extra space for merging
Predictable	
Great for linked lists & disk data	
No bad case	

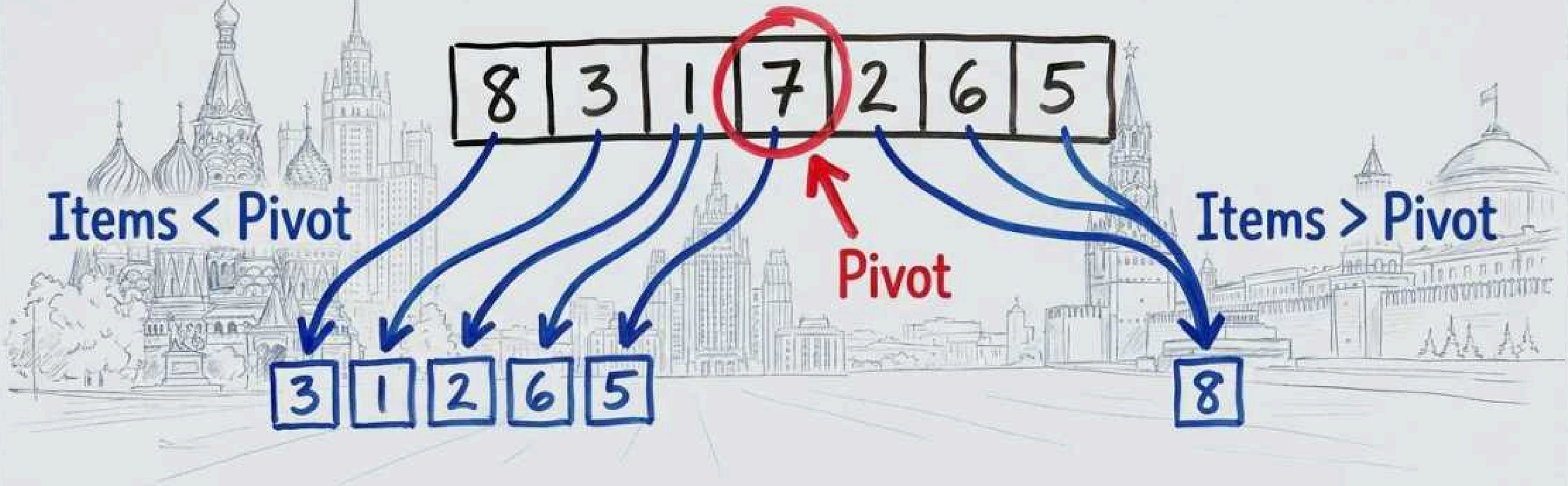
O(n) work merging per level

Total O(n log n) in every case

Gemini Notebook

¶5.8

Quicksort



Sorts in-place. Average time O(n log n).

Degrades to $O(n^2)$ if pivots are chosen badly (e.g., first item of an already sorted array).

Gemini Notebook

Defeating the Worst Case (Pivot Strategies)

¶ 5.9

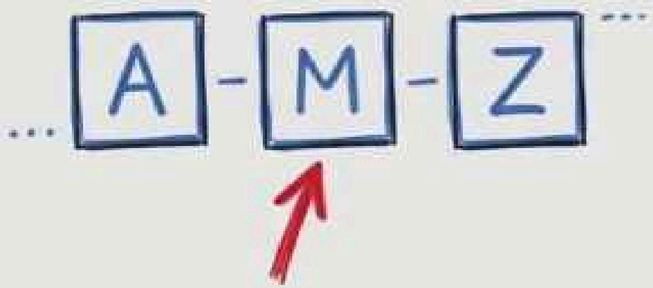
Goal: Guarantee $O(n \log n)$

Random



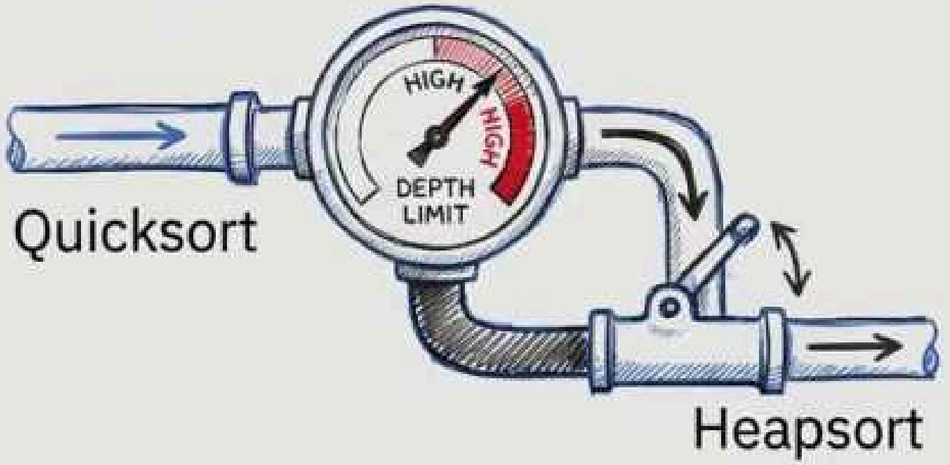
Pick any element at random as the pivot.

Median of Three



Choose the median of the first, middle, and last elements.

Introsort (Musser, 1997)



Switch to Heapsort if recursion depth becomes too large.

Variants of Introsort are used in the C++ standard library.

Gemini Notebook

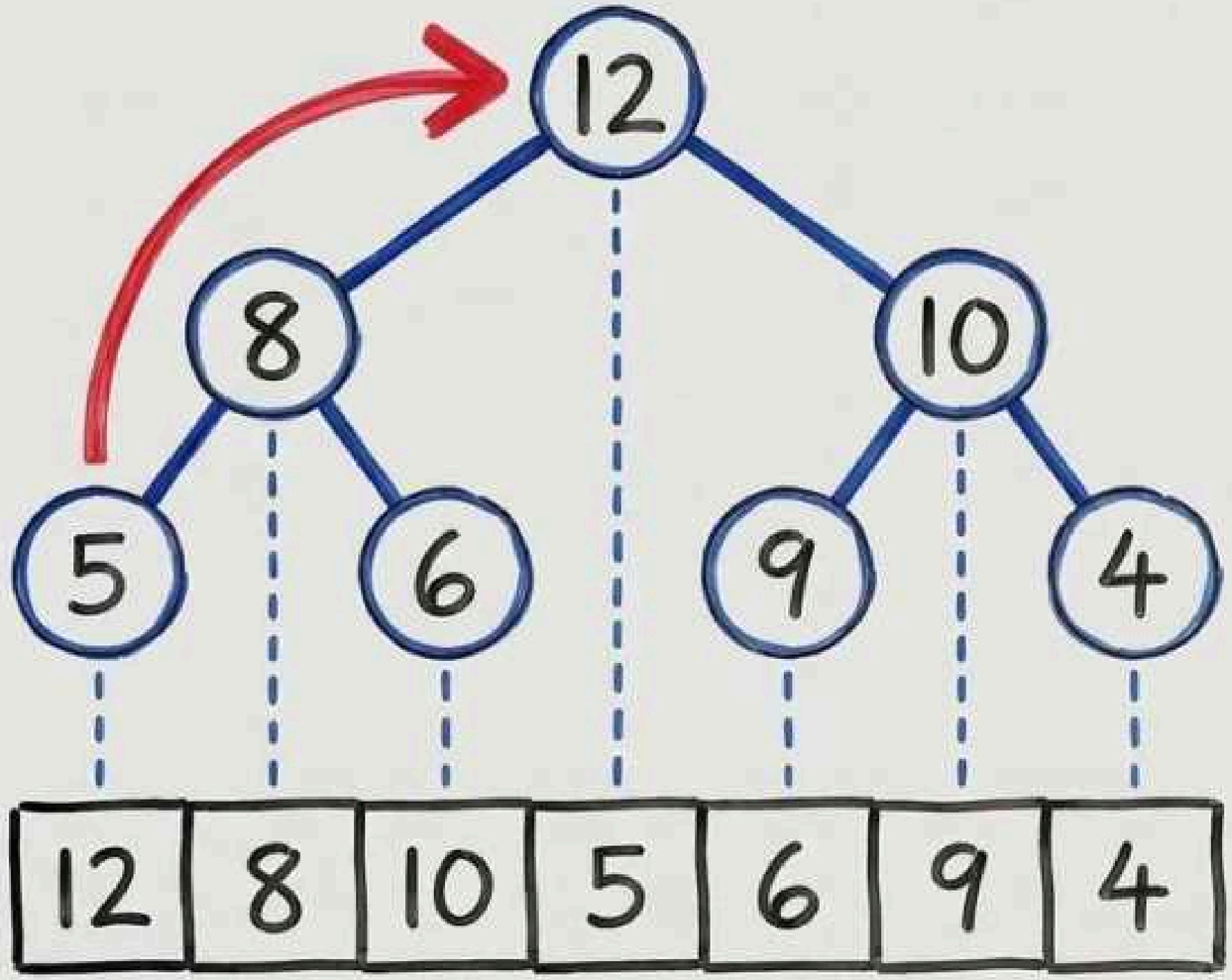
Heapsort

¶ 5.10

Strengths

Guarantees $O(n \log n)$ worst-case.

Uses absolutely zero extra space.



Drawbacks

Usually slower in practice than quicksort (jumps around memory).

NOT stable.

Gemini Notebook

The Concept of Stability

¶ 5.11

Stable: Items with equal keys stay in their original order.

Step 1

	A	B
1	**Name**	**City**
2	Alice	Chicago
3	Bob	Boston
4	Charlie	Chicago
5	Dave	Atlanta
6	Eve	Boston
7	Frank	Chicago

Step 2

	A	B
1	**Name**	**City**
2	Dave	Atlanta
3	Bob	Boston
4	Eve	Boston
5	Alice	Chicago
6	Charlie	Chicago
7	Frank	Chicago

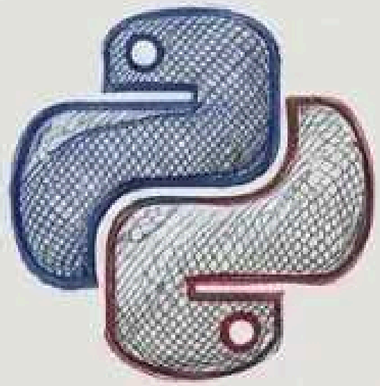
Original relative order: Bob, then Eve

Original relative order: Alice, then Charlie, then Frank

- Stable: Merge, Insertion.
- Unstable: Quick, Heap.

Gemini Notebook

¶ 5.12



[Merge Sort] + [Insertion Sort] + + [Ordered Runs] = TIMSORT



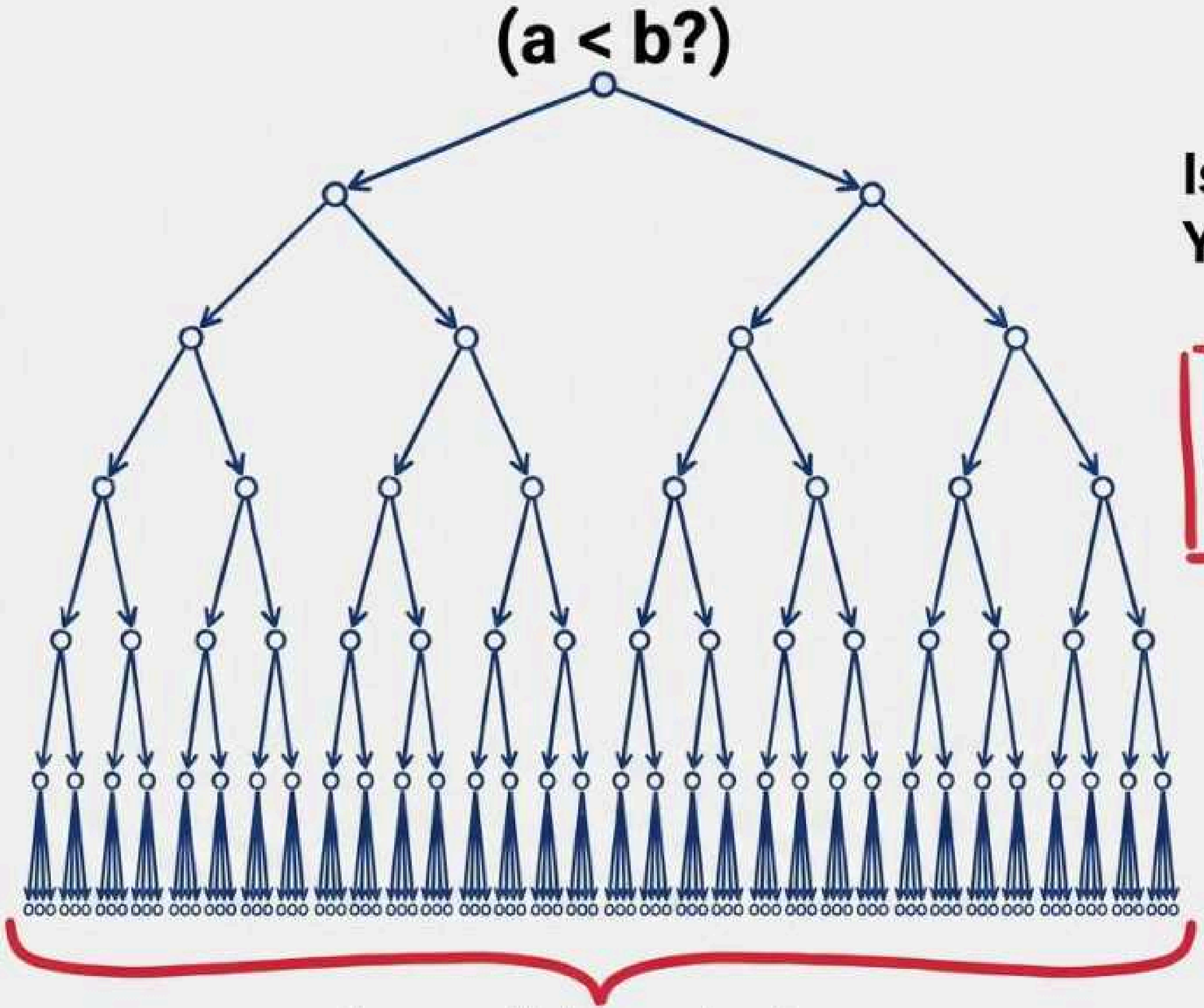
Created by Tim Peters (2002).

Exploits existing ordered runs to achieve O(n) on sorted input.

Sorting programs in practice should call the library sort, not write one.

Gemini Notebook

¶ 5.13



Is O(n log n) the best possible?
YES, for comparison sorts.

$$\text{Height} \geq \log_2(n!) \approx n \log n$$

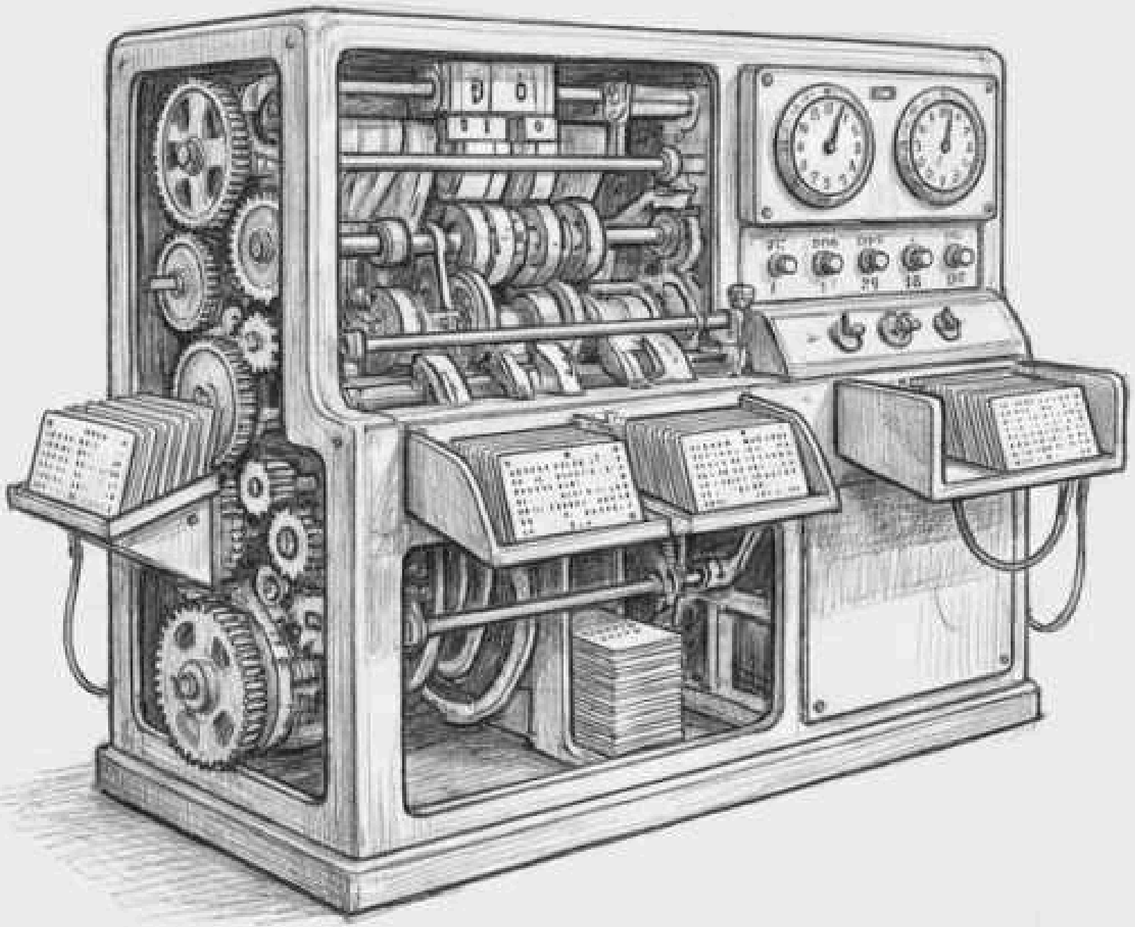
Merge and Heapsort are mathematically optimal.

Claims of faster comparison sorts must be wrong or use extra knowledge about the data.

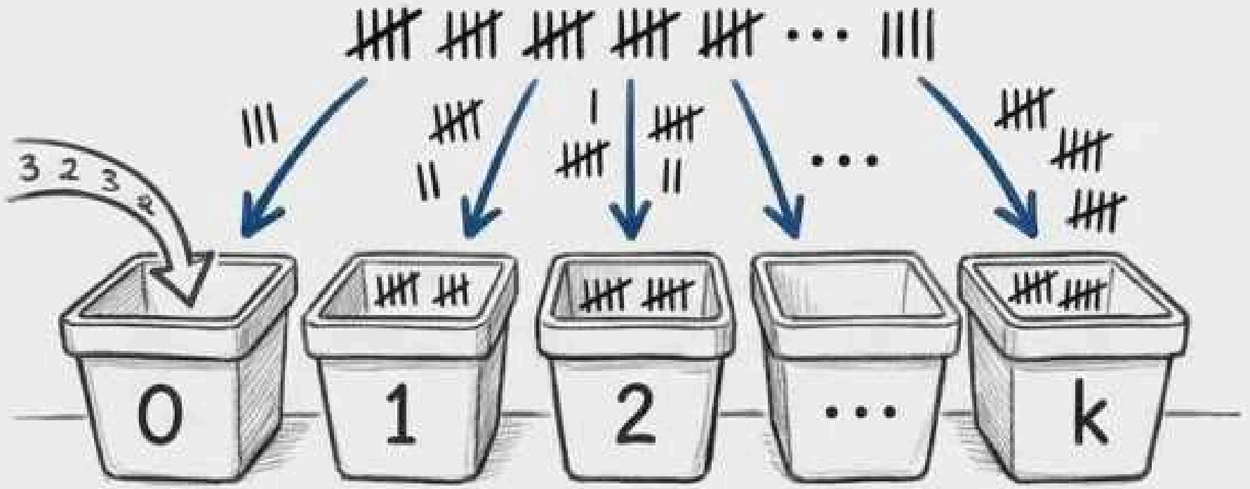
Gemini Notebook

Breaking the Bound (Non-Comparison Sorts)

¶ 5.14



Mechanical Punched-Card Sorter (e.g., Hollerith)

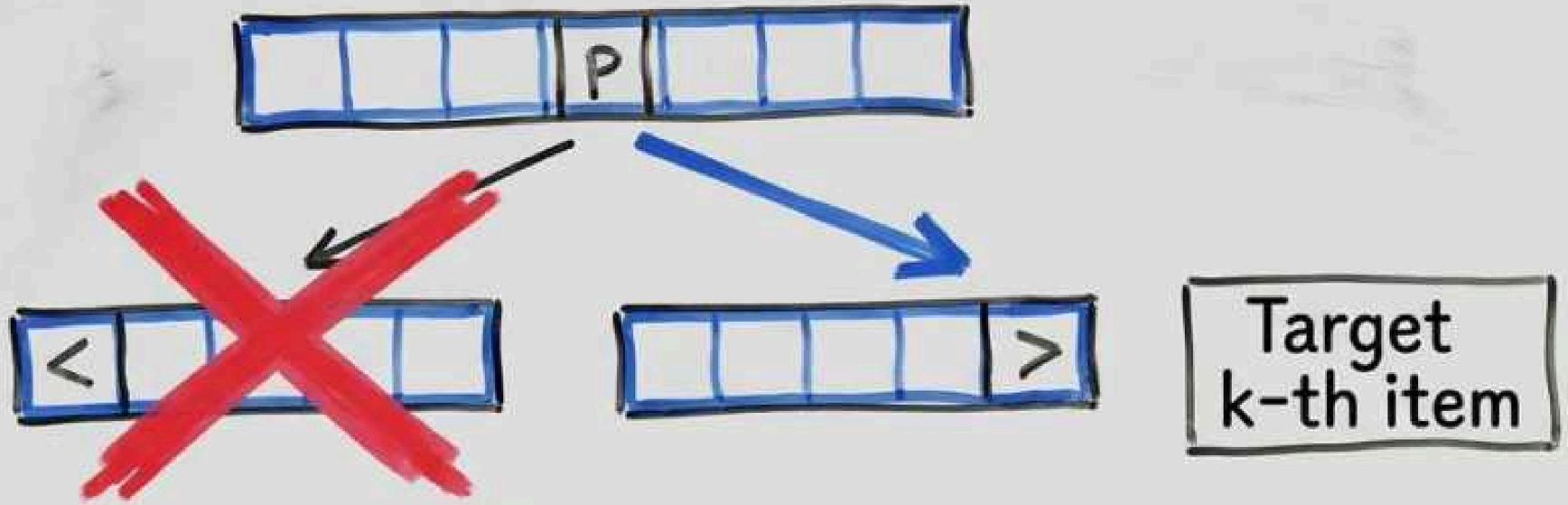


- **Counting Sort:** O(n + k) time for whole numbers.
- **Radix Sort:** Applies stable counting sort one digit at a time.
- **Ideal Use Cases:** Bounded ranges (ages, scores, postal codes).

Gemini Notebook

¶ 5.15

Quickselect: When you only need the middle value.



Partitions like quicksort, but only follows one side.

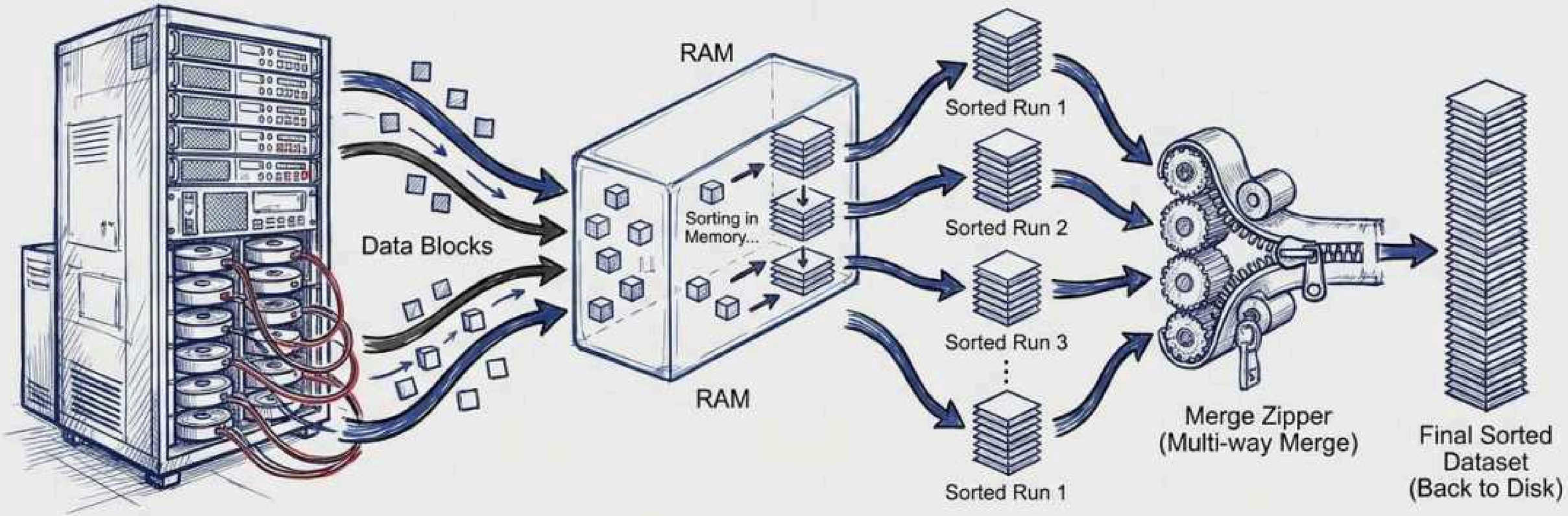
Average time: O(n).

1973 breakthrough: “Median of Medians” (Blum, Floyd, Pratt, Rivest, Tarjan) guarantees O(n) in the worst case.

Gemini Notebook

External Sorting: When Data > Memory

¶ 5.16



Essential for big databases and search engines.

Core Engineering Questions:

- How much data?
- How much memory?
- Does stability matter?

5, 3, 8, 1, 9, 2

¶ 5.17

1. Insertion Sort (Count the Swaps)

2. Merge Sort (Draw Splits & Merges)

Compare the work.

Lecture 6: Design methods and the limits of computation

Gemini NotebookLM