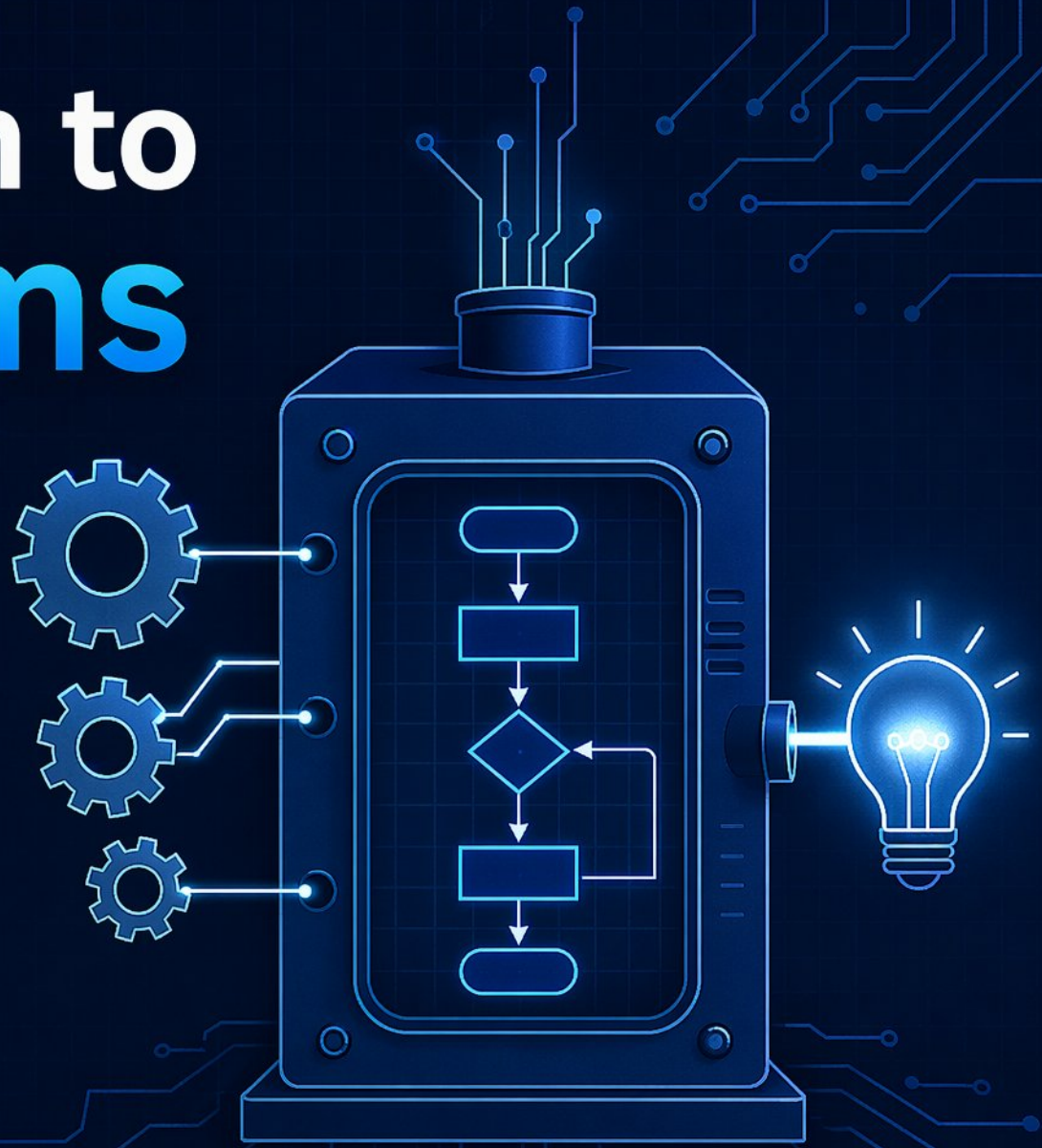
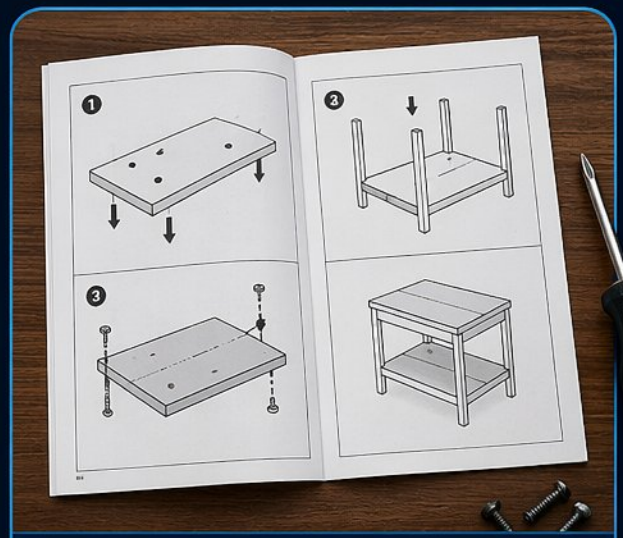
Introduction to Algorithms

- An algorithm is a finite, step-by-step procedure for solving a computing problem
- Key qualities: correctness, efficiency, clarity, and scalability
- Like a recipe: clear inputs, precise steps, guaranteed output
- This course: searching, sorting, recursion, data structures, and graph algorithms
- Real-world examples: GPS routing, social networks, and data sorting



What Is an Algorithm, Really?

A Recipe



Furniture Assembly

A GPS Route

- Formal definition: finite, step-by-step procedure with input, output, definiteness, finiteness, and effectiveness
- Everyday analogies: a recipe, furniture assembly, or a GPS route
- Algorithm vs. pseudocode vs. program: the logical plan, the structured outline, and the executable implementation

Why Algorithms Are the Core of Computing

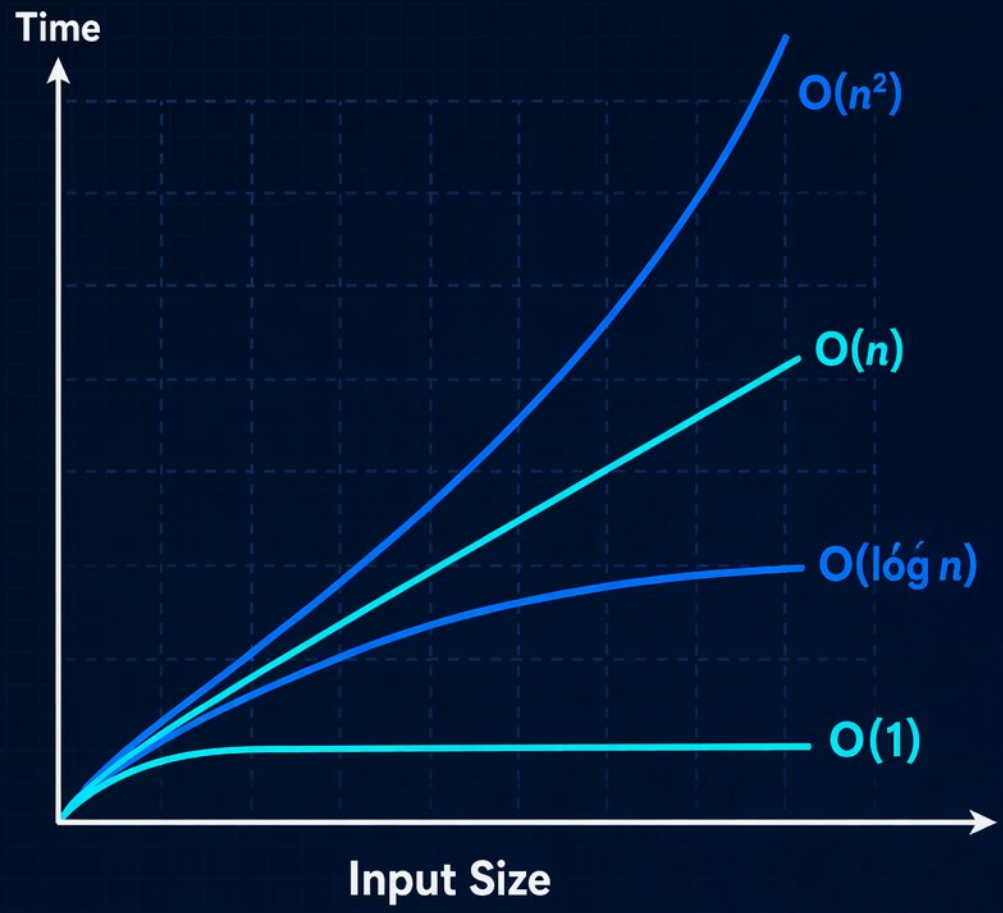
- Algorithms transform data: **input** → **steps** → **output**
- Algorithmic thinking breaks complex problems into **clear, simple steps**
- An algorithm is a **step-by-step plan**; a program is its **code implementation**
- Example: find the **maximum value** in a list of numbers

Measuring Efficiency:
Time and Space Complexity

- Big O describes how runtime grows as input size (n) increases
- O(1): Constant time, same speed regardless of data size
- O(n): Linear time, 10× more data → 10× slower
- $O(n^2)$: Quadratic time, 10× more data → 100× slower
- Space complexity tracks how memory usage grows with input

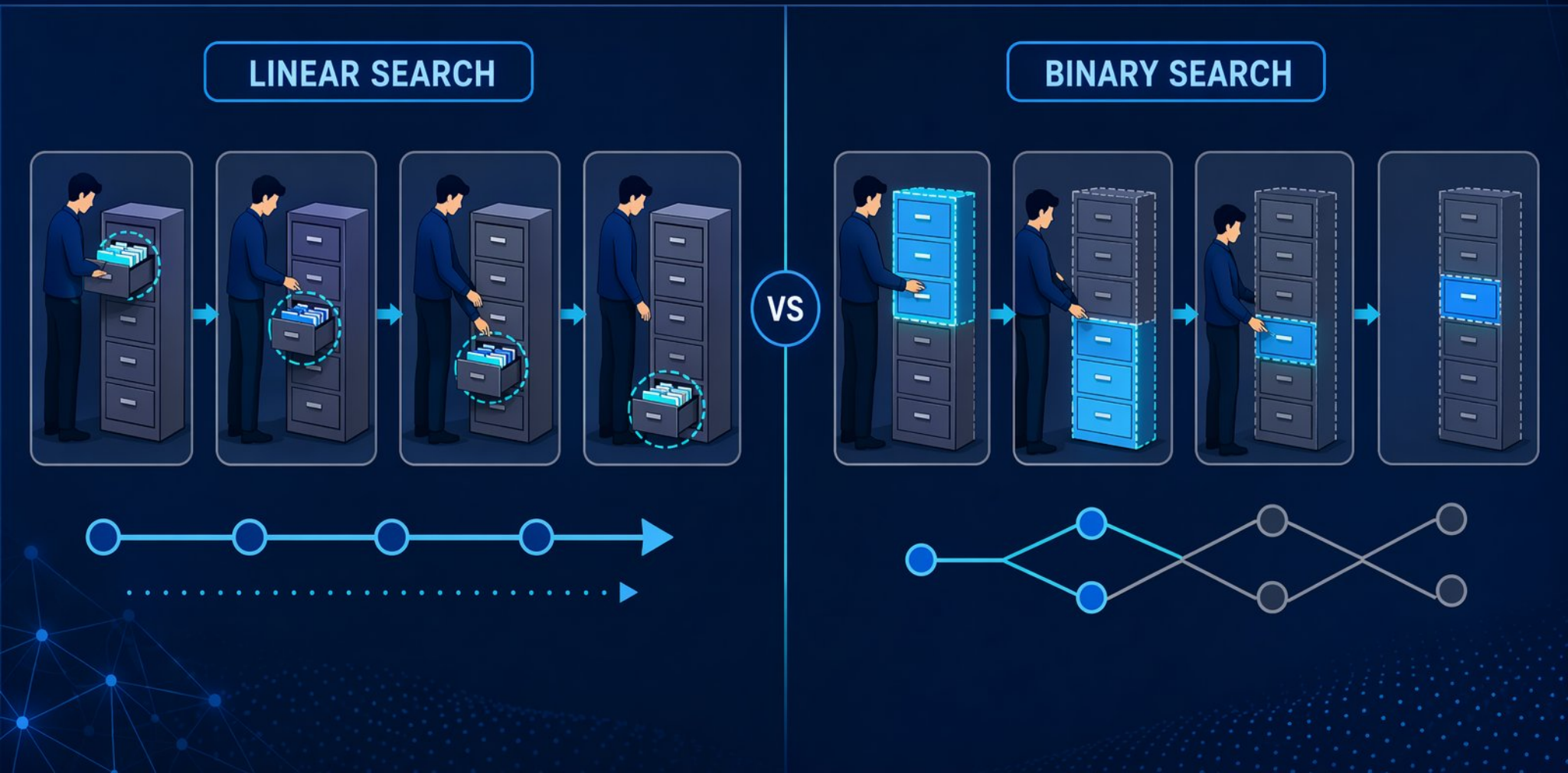
Big O in Practice: Common Growth Rates

- **$O(1)$ constant:** same time regardless of input size
- **$O(\log n)$ logarithmic:** halves the problem each step
- **$O(n)$ linear:** time grows directly with input size
- **$O(n^2)$ quadratic:** nested loops explode step count
- Drop constants and lower-order terms; keep dominant term
- Big O describes worst-case growth trend, not exact time



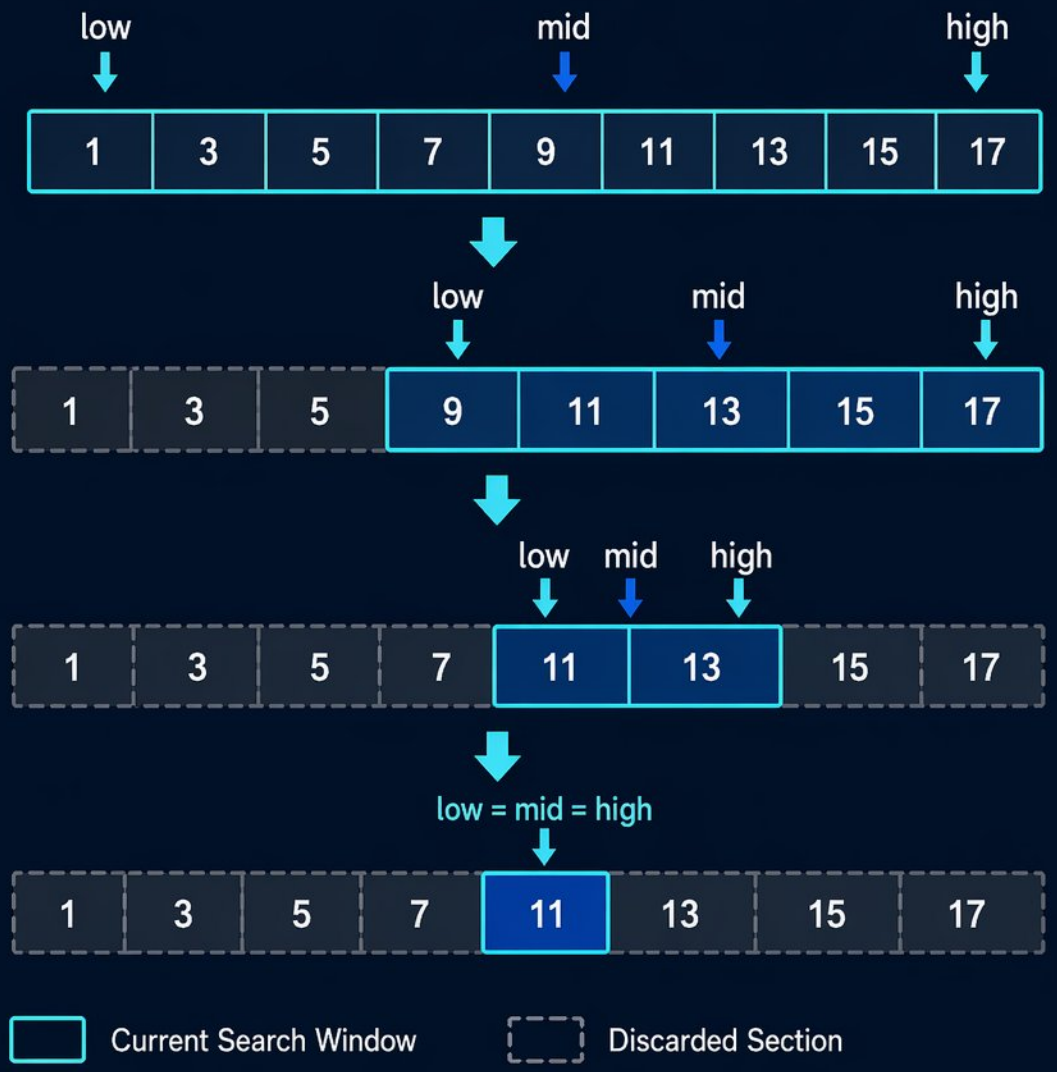
Searching: Linear and Binary Search

- Define the search problem: find a target in a collection.
- Linear search checks every element, resulting in O(n) time.
- Binary search halves the search area each step, achieving O(log n).
- Binary search requires a sorted array as a mandatory precondition.

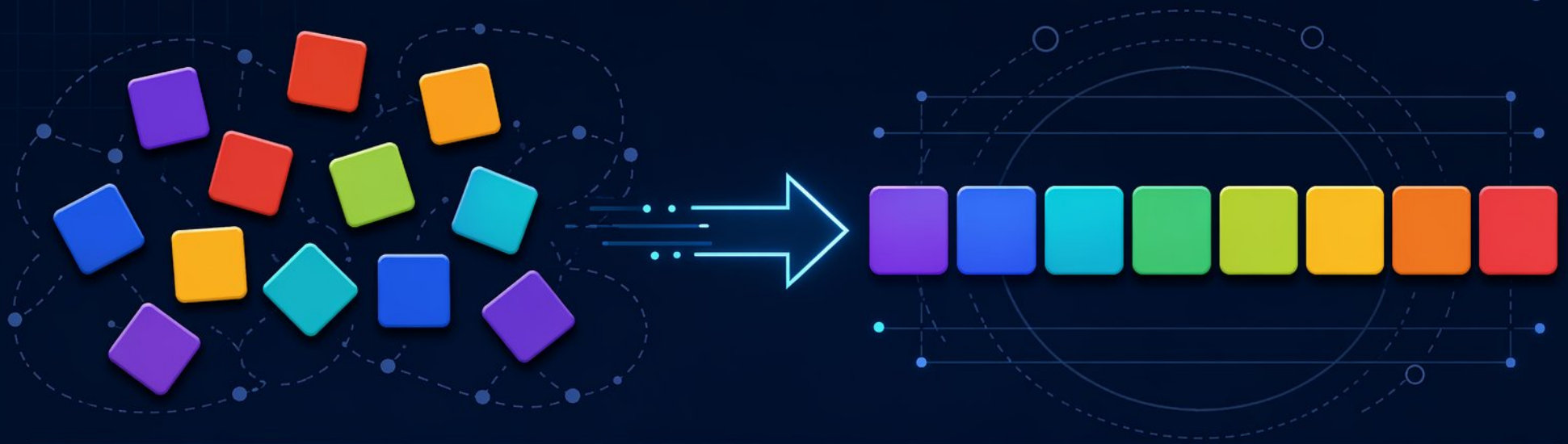


Binary Search:
Step-by-Step Execution

- Initialize low, high, and mid pointers on a sorted array
- Compare mid value to target; discard left or right half
- Each step halves the search space, guaranteeing O(log n) time
- Repeat until target found or window empty (low > high)
- Sorted input is essential; think dictionary or guessing game



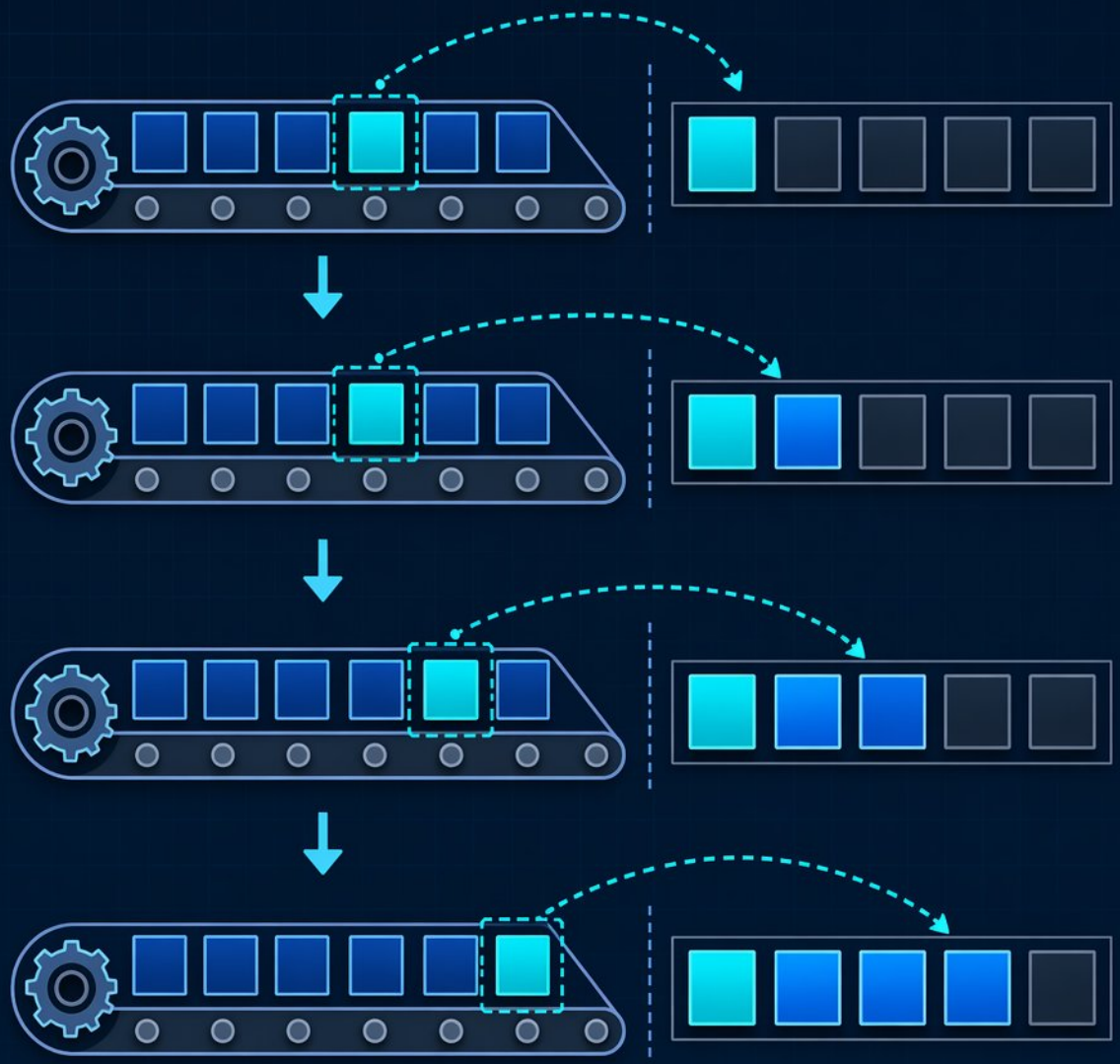
Sorting: Why Order Matters



- Sorting puts data in order, enabling faster algorithms like binary search
- Selection sort repeatedly finds the smallest unsorted element and swaps it into place
- Each pass grows a sorted region on the left until the whole array is ordered
- $O(n^2)$ comparisons but at most $n-1$ swaps — useful when writes are expensive
- Merge sort preview: uses divide-and-conquer to achieve $O(n \log n)$ runtime

Selection Sort: A Simple Quadratic Algorithm

- Each pass finds the smallest unsorted element and swaps it to the sorted boundary.
- After i passes, the first i elements are in their final sorted position (invariant).
- Always performs $O(n^2)$ comparisons, regardless of input order.
- Minimizes swaps: ȧt most $n-1$ swaps total, useful when writes are expensive.

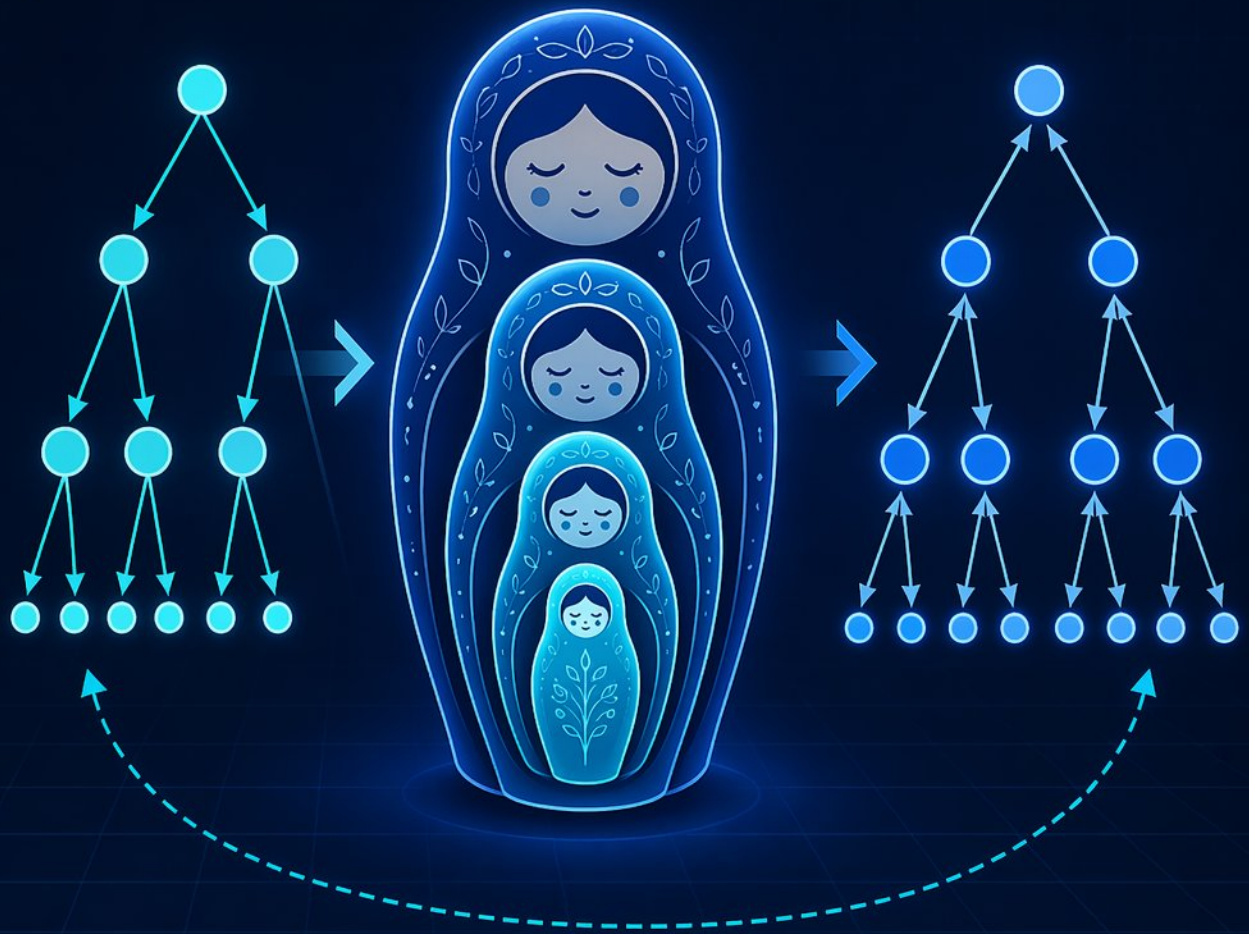


Recursion:
A Function Calling Itself

- Recursion: a function solves a problem by calling itself with a smaller input
- Every recursive function needs a base case to stop and a recursive case to continue
- Factorial example: base case n=1 returns 1; recursive case returns n * factorial(n-1)
- Each call creates a new stack frame with its own local variables, pushed onto the call stack
- When the base case returns, the stack unwinds: each frame multiplies and returns its result

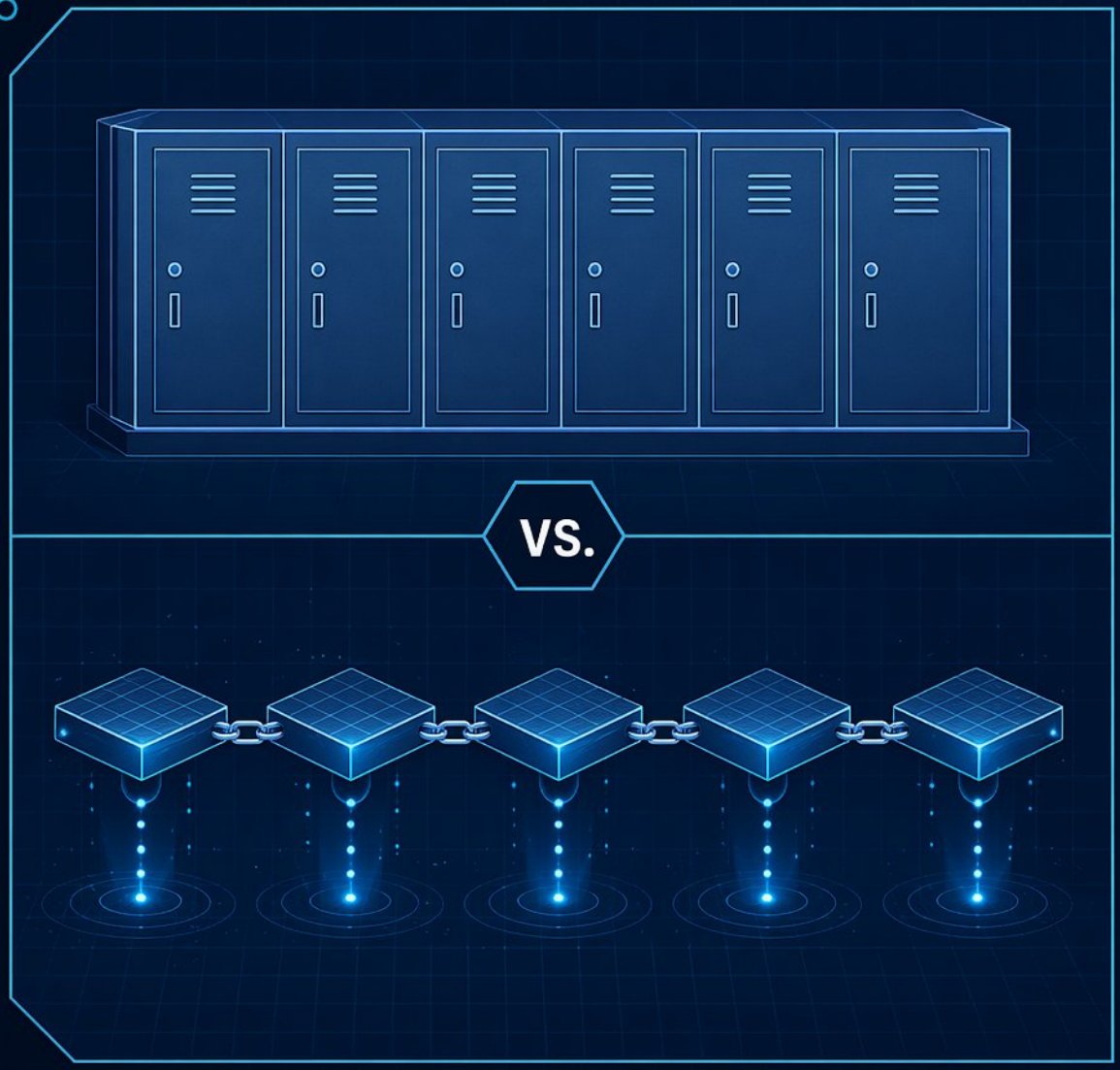
Recursion in Action:
Binary Search and Merge Sort

- Connect recursion to binary search as a divide-and-conquer strategy
- Preview merge sort: splits, recursively sorts, then merges subarrays
- A base case is essential to stop recursion and prevent stack overflow
- Each recursive call pushes a new frame; returns unwind in reverse order



Data Structures:
Organizing Data for Algorithms

- Choice of data structure directly impacts algorithm speed and memory
- Arrays: contiguous memory, O(1) access by index, slow middle insert/delete
- Linked lists: scattered nodes, O(n) access, fast insert/delete if pointer known
- Stacks use LIFO (last-in-first-out) like a stack of plates
- Queues use FIFO (first-in-first-out) like a waiting line

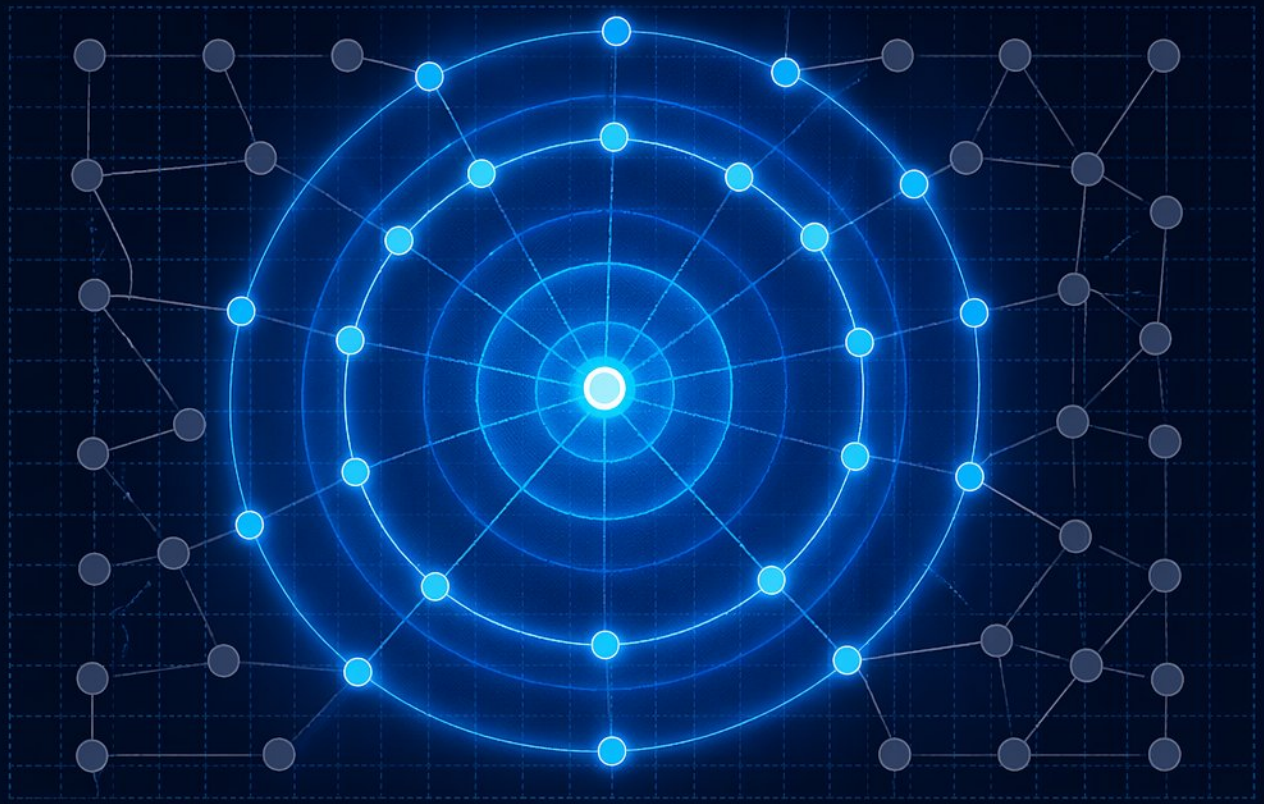


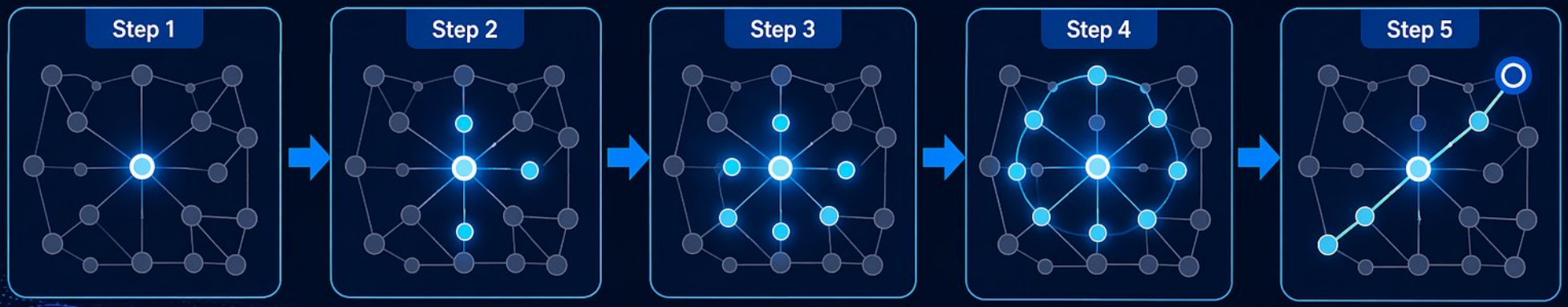
Graphs and Paths: Algorithms on Networks

- A graph consists of nodes and edges; examples include social networks and road maps
- Breadth-first search (BFS) explores level by level using a queue
- BFS finds the shortest path in unweighted graphs in O(V + E) time
- Depth-first search (DFS) goes deep first using a stack or recursion
- Use BFS for shortest paths; use DFS for complete exploration or backtracking

BFS Step-by-Step: Finding the Shortest Path

- BFS uses a queue to explore graph level-by-level from the source
- First visit to a node guarantees the shortest unweighted path
- Store parent pointers to reconstruct the actual path after search
- Reconstruct by backtracking from target to source via parents



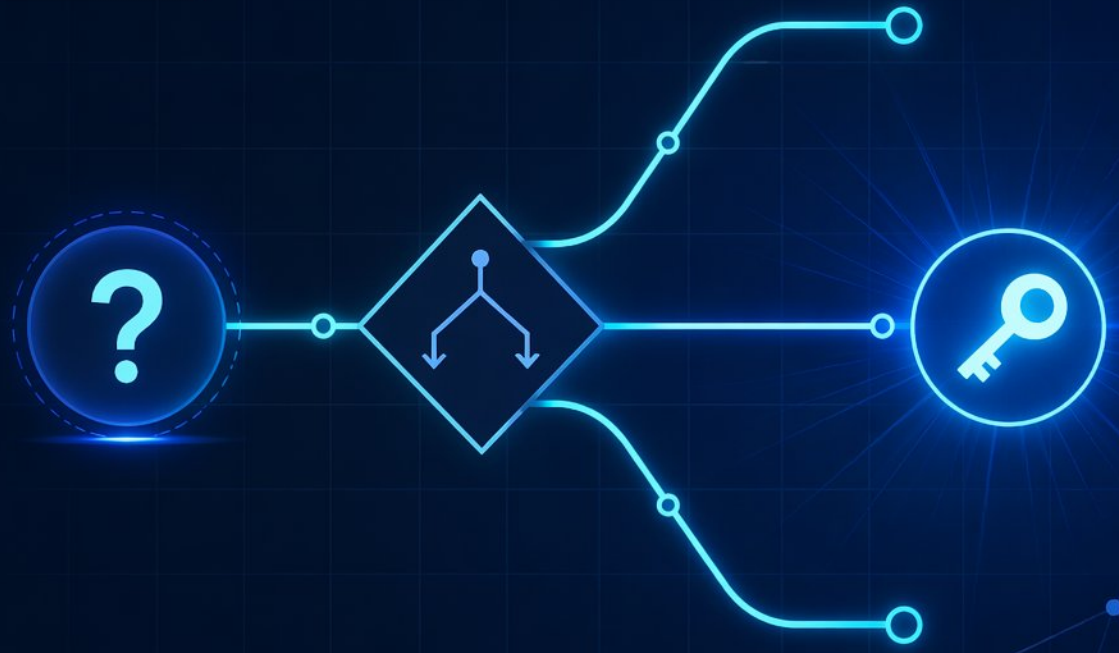


Algorithm Design Patterns

- **Brute-force:** try every possibility; simple but often slow
- **Divide-and-conquer:** split, solve, merge (e.g., merge sort)
- **Greedy:** pick the best local choice at each step
- **Dynamic programming:** break into overlapping subproblems, cache results
- **Memoization** stores computed values to avoid redundant work

From Algorithm to Code: Next Steps

- Summarize the journey from problem to algorithm selection
- Use a checklist: clarify, choose algorithm, structure data, test small
- Practice daily on platforms like LeetCode, HackerRank, or NeetCode
- Follow structured paths: HackerRank for fundamentals, LeetCode for patterns
- Track progress and verify skills with real problem-solving data



PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/30efa52f/public