25 Algorithms Every Programmer Should Know

1. Binary Search

How it works: Searches for an element in a sorted array by repeatedly dividing the search interval in half.

Example:

Array: [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
Find: 23

Step 1: Check middle (16) → 23 > 16, search right half
Step 2: Check middle of right half (56) → 23 < 56, search left half

Step 3: Check middle (23) → Found!

---

2. Bubble Sort

How it works: Repeatedly compares adjacent elements and swaps them if they're in the wrong order, pushing the largest element to the end in each pass.

Example:

Array: [5, 1, 4, 2, 8]

Pass 1: [1, 4, 2, 5, 8] (8 bubbles to end)
Pass 2: [1, 2, 4, 5, 8] (5 bubbles to position)

Pass 3: [1, 2, 4, 5, 8] (already sorted)

---

3. Merge Sort

How it works: Divides array into halves, recursively sorts each half, then merges the sorted halves back together.

Example:

Array: [38, 27, 43, 3]

Divide:
[38, 27] [43, 3]
[38] [27] [43] [3]

Merge:
[27, 38] [3, 43]

[3, 27, 38, 43]

---

4. Quick Sort

How it works: Picks a pivot element, partitions array so elements smaller than pivot are on left and larger on right, then recursively sorts both partitions.

Example:

Array: [10, 7, 8, 9, 1, 5]
Pivot: 5

Partition: [1] 5 [10, 7, 8, 9]
Continue sorting left and right partitions

Result: [1, 5, 7, 8, 9, 10]

---

5. Depth-First Search (DFS)

How it works: Explores a graph/tree by going as deep as possible down each branch before backtracking.

Example:

Tree: 1
  / \
 2 3
/ \
4 5

DFS Order: 1 → 2 → 4 → 5 → 3

(Go deep into left branch first)

---

6. Breadth-First Search (BFS)

How it works: Explores a graph/tree level by level, visiting all neighbors before moving to the next level.

Example:

Tree: 1
  / \
 2 3
/ \
4 5

BFS Order: 1 → 2 → 3 → 4 → 5

(Level by level)

---

7. Dijkstra's Algorithm

How it works: Finds the shortest path from a starting node to all other nodes in a weighted graph by always exploring the closest unvisited node.

Example:

Graph:
A --4-- B
| |
2 1
| |
C --5-- D

Shortest path A to D:

A → C (2) → B (3) → D (4) = Total: 4

---

8. Dynamic Programming

How it works: Solves complex problems by breaking them into smaller subproblems, storing results to avoid recalculation.

Example (Fibonacci):

Fib(5) without DP: Calculates Fib(3) multiple times
Fib(5) with DP: Stores Fib(3) result, reuses it

Fib(0)=0, Fib(1)=1

Fib(2)=1, Fib(3)=2, Fib(4)=3, Fib(5)=5

---

9. Greedy Algorithm

How it works: Makes the locally optimal choice at each step, hoping to find a global optimum.

Example (Coin Change):

Make 30 cents using coins: [25¢, 10¢, 5¢, 1¢]

Greedy approach:
1. Take 25¢ (largest) → Remaining: 5¢
2. Take 5¢ → Remaining: 0¢

Result: 2 coins (25¢ + 5¢)

---

10. Hashing

How it works: Uses a hash function to convert keys into array indices for fast data storage and retrieval.

Example:

Hash Function: key % 10

Store "apple" (hash: 3) → array[3] = "apple"
Store "banana" (hash: 7) → array[7] = "banana"

Search "apple" → Calculate hash (3) → Check array[3] → Found!

---

11. Insertion Sort

How it works: Builds sorted array one element at a time by inserting each element into its correct position.

Example:

Array: [5, 2, 4, 6, 1]

Step 1: [5] 2, 4, 6, 1 → [2, 5] 4, 6, 1
Step 2: [2, 5] 4, 6, 1 → [2, 4, 5] 6, 1
Step 3: [2, 4, 5] 6, 1 → [2, 4, 5, 6] 1
Step 4: [2, 4, 5, 6] 1 → [1, 2, 4, 5, 6]

---

12. Selection Sort

How it works: Finds the minimum element from unsorted portion and places it at the beginning, repeat for remaining array.

Example:

Array: [64, 25, 12, 22, 11]

Step 1: Find min (11) → [11, 25, 12, 22, 64]
Step 2: Find min (12) → [11, 12, 25, 22, 64]
Step 3: Find min (22) → [11, 12, 22, 25, 64]
Step 4: Find min (25) → [11, 12, 22, 25, 64]

---

13. Counting Sort

How it works: Counts occurrences of each value, then uses counts to place elements in correct position.

Example:

Array: [4, 2, 2, 8, 3, 3, 1]

Count array:
1→1 time, 2→2 times, 3→2 times, 4→1 time, 8→1 time

Output: [1, 2, 2, 3, 3, 4, 8]

---

14. Heap Sort

How it works: Builds a max heap from array, repeatedly extracts maximum element and places at end.

Example:

Array: [4, 10, 3, 5, 1]

Build Max Heap: [10, 5, 3, 4, 1]
Extract 10: [5, 4, 3, 1] | 10
Extract 5: [4, 1, 3] | 5, 10

Continue... → [1, 3, 4, 5, 10]

---

15. Binary Tree Traversal

How it works: Three ways to visit all nodes in a binary tree.

Example:

Tree: 1
  / \
 2 3
/ \
4 5

Inorder (Left-Root-Right): 4, 2, 5, 1, 3
Preorder (Root-Left-Right): 1, 2, 4, 5, 3

Postorder (Left-Right-Root): 4, 5, 2, 3, 1

---

16. Recursion

How it works: Function calls itself with smaller input until reaching a base case.

Example (Factorial):

factorial(5):
  5 × factorial(4)
    4 × factorial(3)
      3 × factorial(2)
        2 × factorial(1)
          1 (base case)

Result: 5 × 4 × 3 × 2 × 1 = 120

---

17. Backtracking

How it works: Tries different solutions, abandons a path when it determines it won't lead to a solution.

Example (N-Queens, 4×4 board):

Try Queen at (0,0) → (1,2) → Can't place (2,?) → Backtrack
Try Queen at (0,1) → (1,3) → (2,0) → (3,2) → Success!

Q . . .
. . . Q
Q . . .

. . Q .

---

18. Divide and Conquer

How it works: Breaks problem into smaller subproblems, solves them independently, combines results.

Example (Find Max in Array):

Array: [3, 7, 2, 9, 1, 5]

Divide: [3, 7, 2] and [9, 1, 5]
Divide: [3] [7, 2] [9] [1, 5]
Conquer: Max([3])=3, Max([7,2])=7, Max([9])=9, Max([1,5])=5
Combine: Max(3,7)=7, Max(9,5)=9

Result: Max(7,9)=9

---

19. String Pattern Matching (KMP)

How it works: Searches for pattern in text efficiently by using information from previous matches to skip unnecessary comparisons.

Example:

Text: "ABABCABAB"
Pattern: "ABAB"

Step 1: Match "ABAB" at position 0 → Found
Step 2: Use pattern info, skip to position 5

Step 3: Match "ABAB" at position 5 → Found

---

20. Graph Cycle Detection

How it works: Uses DFS to detect if visiting a node that's already in current path means there's a cycle.

Example:

Graph: A → B → C → A (cycle!)

DFS from A:
Visit A → Visit B → Visit C → Try to visit A again

A is already in current path → Cycle detected!

---

21. Topological Sort

How it works: Orders vertices in directed graph so for every edge u→v, u comes before v in ordering.

Example:

Tasks with dependencies:
A → B (A must finish before B)
A → C

B → D
C → D

Valid order: A, B, C, D or A, C, B, D

---

22. Bit Manipulation

How it works: Performs operations directly on binary representation of numbers.

Example:

Check if number is even: n & 1 == 0
  6 (110 in binary) & 1 = 0 → Even
  7 (111 in binary) & 1 = 1 → Odd

Set 3rd bit: n | (1 << 2)
  5 (101) | 100 = 101 → 5

Toggle bit: n ^ (1 << position)

---

23. Kadane's Algorithm

How it works: Finds maximum sum of contiguous subarray by tracking current and maximum sum.

Example:

Array: [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Current sum: -2, 1, -2, 4, 3, 5, 6, 1, 5
Max sum: -2, 1, 1, 4, 4, 5, 6, 6, 6

Maximum subarray: [4, -1, 2, 1] = 6

---

24. Two Pointer Technique

How it works: Uses two pointers moving through data structure to solve problems efficiently.

Example (Find pair with sum):

Sorted array: [1, 2, 3, 4, 6, 8]
Target sum: 10

Left=0, Right=5: 1+8=9 (too small, move left→)

Left=1, Right=5: 2+8=10 → Found!

---

25. Sliding Window

How it works: Maintains a window of elements and slides it through the array to solve problems.

Example (Max sum of 3 consecutive elements):

Array: [2, 1, 5, 1, 3, 2]

Window 1: [2, 1, 5] → sum = 8
Window 2: [1, 5, 1] → sum = 7
Window 3: [5, 1, 3] → sum = 9 (max)

Window 4: [1, 3, 2] → sum = 6