Sure! Algorithms are step-by-step procedures or formulas for solving problems or performing tasks. They are fundamental in computer science, mathematics, and programming.

◆ Types of Algorithms

1. **Brute Force Algorithm**
   - Tries all possible solutions.
   - Simple but inefficient for large input.
   - Example: Trying all passwords.
2. **Divide and Conquer**
   - Breaks the problem into smaller subproblems, solves them recursively, and combines the results.
   - Example: Merge Sort, Quick Sort, Binary Search.
3. **Greedy Algorithm**
   - Makes the locally optimal choice at each step.
   - Doesn't always give the best overall result.
   - Example: Kruskal's MST, Dijkstra’s (without priority queue), Huffman coding.
4. **Dynamic Programming (DP)**
   - Solves problems by breaking them into overlapping subproblems and storing the results (memoization or tabulation).
   - Example: Fibonacci, Knapsack, Longest Common Subsequence (LCS).
5. **Backtracking**
   - Tries all solutions and backtracks if one doesn't lead to a solution.
   - Example: N-Queens, Sudoku Solver.
6. **Recursion**
   - Function calls itself with a smaller input.
   - Used in divide and conquer, backtracking, and dynamic programming.
   - Example: Factorial, Tower of Hanoi.
7. **Graph Algorithms**
   - BFS, DFS
   - Dijkstra’s, Bellman-Ford
   - Floyd-Warshall, A*
   - Kruskal’s and Prim’s for MST
   - Tarjan/Kosaraju for SCC
8. **Sorting Algorithms**
   - Bubble, Insertion, Selection
   - Merge Sort, Quick Sort, Heap Sort
   - Counting, Radix, and Bucket Sort (non-comparison sorts)
9. **Searching Algorithms**
   - Linear Search
   - Binary Search
   - Ternary Search (less common)
10. **Bit Manipulation Algorithms**
    - XOR tricks, Bit masking
    - Used in problems involving subsets or toggling bits.

◆ Algorithm Design Techniques

- **Recursion and Memoization**
- **Greedy Choice Property**
- **Optimal Substructure**
- **State Space Tree (used in Backtracking)**
- **Sliding Window**
- **Two Pointers**
- **Union-Find / Disjoint Set**
- **Monotonic Stack/Queue**

◆ Popular Algorithm Problems

Category	Examples
Sorting	Merge Sort, Quick Sort, Counting Sort
Searching	Binary Search, BFS, DFS
DP	Knapsack, LCS, LIS
Graph	Dijkstra, Kruskal, Floyd-Warshall
Greedy	Activity Selection, Fractional Knapsack
Backtracking	Sudoku Solver, N-Queens
Bit Manipulation	XOR of subsets, Counting set bits

Would you like:

- A **roadmap** for learning algorithms?
- A list of **problems with increasing difficulty**?
- Or help with a specific **algorithm topic or problem**?

Let me know how you'd like to proceed!

You said:

"DS" commonly stands for **Data Structures** in computer science and programming. It refers to ways of organizing and storing data so that it can be accessed and modified efficiently.

◆ Common Data Structures:

1. Linear Data Structures

- **Array**
  Fixed-size, index-based collection of elements.
- **Linked List**
  Nodes connected by pointers (Singly/Doubly/ Circular).
- **Stack**
  LIFO (Last-In-First-Out) structure. Operations: `push`, `pop`, `peek`.
- **Queue**
  FIFO (First-In-First-Out) structure. Variants: `Queue`, `Deque`, `Priority Queue`.

2. Non-Linear Data Structures

- **Tree**
  Hierarchical structure. Types: Binary Tree, Binary Search Tree (BST), AVL Tree, Segment Tree, etc.
- **Graph**
  Collection of nodes (vertices) and edges. Types: Directed, Undirected, Weighted, Unweighted.

3. Hash-based Structures

- **Hash Table / HashMap**
  Key-value storage with average O(1) access time.

4. Advanced Structures

- **Trie (Prefix Tree)** – For efficient string searches.
- **Heap** – For priority queue (min-heap, max-heap).
- **Disjoint Set (Union-Find)** – For tracking connected components.

◆ Why Learn Data Structures?

- Efficient **algorithm design**.
- Crucial for **interviews** at companies like Google, Amazon, etc.
- Helps build **scalable applications**.

◆ Common Operations to Practice:

- Insert, Delete, Search

- Traversal (Inorder, Preorder, Postorder for trees)
- Graph traversals (DFS, BFS)
- Sorting and Searching algorithms

✅ 1. Basics of Graph Theory

- What is a Graph?
- Types of Graphs:
  - Directed vs Undirected
  - Weighted vs Unweighted
  - Cyclic vs Acyclic
  - Connected vs Disconnected
- Representation:
  - Adjacency Matrix
  - Adjacency List
  - Edge List

✅ 2. Graph Traversals

- **Depth First Search (DFS)**
- **Breadth First Search (BFS)**
- Applications:
  - Path Finding
  - Connected Components
  - Cycle Detection (in DFS)

✅ 3. Topological Sorting

- Definition & Applications
- **Kahn's Algorithm (BFS based)**
- **DFS Based Topo Sort**

✅ 4. Shortest Path Algorithms

- **Dijkstra's Algorithm** (Greedy)
- **Bellman-Ford Algorithm** (Handles negative weights)
- **Floyd-Warshall Algorithm** (All-pairs shortest path)
- **0-1 BFS** (with deque)
- **SPFA** (Shortest Path Faster Algorithm - Optimized Bellman-Ford)

✅ 5. Minimum Spanning Tree (MST)

- **Prim's Algorithm**
- **Kruskal's Algorithm (with Union-Find)**

✅ 6. Union-Find (Disjoint Set Union - DSU)

- Basic DSU
- Path Compression
- Union by Rank
- Applications:
    - Cycle Detection
    - Kruskal’s Algorithm
    - Connected Components

✅ 7. Cycle Detection

- In **Undirected Graph** using DFS or DSU
- In **Directed Graph** using DFS (back edges) or Kahn's Algorithm (if topo sort fails)

✅ 8. Strongly Connected Components (SCC)

- **Kosaraju’s Algorithm**
- **Tarjan’s Algorithm**

✅ 9. Bridges and Articulation Points

- Tarjan’s Algorithm (DFS-based)
- Applications in network design and fault-tolerant systems

✅ 10. Bipartite Graph

- Checking using BFS/DFS
- Applications:
    - Graph Coloring
    - Matching Problems

✅ 11. Graph Coloring

- 2-coloring, m-coloring problem
- Greedy coloring algorithms

- Applications in scheduling, register allocation

✅ 12. Advanced Graph Concepts

- **Topological Sort with Cycle Detection**
- **Directed Acyclic Graph (DAG) Longest Path**
- **Tree DP (Dynamic Programming on Trees)**
- **Lowest Common Ancestor (LCA)**
- **Centroid Decomposition**
- **Euler Tour**

✅ 13. Grid-Based Graphs

- Graph on 2D Grid (Maze Solving)
- BFS/DFS on grid
- Flood Fill Algorithm

✅ 14. Flow Algorithms (Advanced)

- **Ford-Fulkerson Algorithm**
- **Edmonds-Karp Algorithm**
- **Dinic's Algorithm**
- Applications:
    - Max Flow
    - Bipartite Matching
    - Network Design

✅ 15. Geometry + Graph

- Line Intersection as Graph
- Delaunay Triangulation
- Voronoi Diagrams

□ Practice Tip:

For interviews and CP, focus first on:

- BFS/DFS
- Dijkstra/Kruskal/Prim

- Union-Find
- Topo Sort
- SCC (Kosaraju)

🔰 1. Basics of Trees

- Definition of a tree
- Terminology:
  - Node, Edge, Root, Leaf, Parent, Child, Sibling
  - Height, Depth, Level, Degree
- Types of Trees:
  - Binary Tree
  - Binary Search Tree (BST)
  - N-ary Tree
  - Full/Complete/Perfect Trees
  - Balanced Trees (e.g., AVL)
- Applications of Trees

🌿 2. Tree Representations

- Using Nodes and Pointers
- Using Arrays (for Binary Trees)
- Adjacency List (for generic trees)

🔁 3. Tree Traversals

Recursive

- Inorder (LNR)
- Preorder (NLR)
- Postorder (LRN)

Iterative (Using Stack)

- Inorder, Preorder, Postorder (via two stacks)

Level Order

- Using Queue (BFS)
- Reverse Level Order

🌲 4. Binary Search Tree (BST)

- Insert, Search, Delete
- Validate BST
- Floor and Ceil

- Lowest Common Ancestor (LCA)
- Kth Smallest/Largest Element
- Range Sum in BST

🛠️ 5. Tree Problems (Intermediate)

- Height/Depth of Tree
- Diameter of Tree
- Balanced Tree Check
- Path from Root to Node
- Root to Leaf Paths
- Path Sum Problems
- Leaf Nodes Count

□ 6. Advanced Tree Concepts

- LCA using Binary Lifting
- Euler Tour Technique
- Tree Flattening
- Binary Indexed Tree on Trees
- Heavy-Light Decomposition (HLD)

📌 7. Special Trees

- AVL Tree (Self-balancing BST)
- Red-Black Tree
- Segment Tree (used for range queries)
- Fenwick Tree (Binary Indexed Tree)
- Trie (Prefix Tree)
- B-Tree, B+ Tree (used in DBMS)

□ 8. Tree Algorithms

- Tree Diameter (DFS or DP)
- Tree DP (Dynamic Programming on Trees)
- Subtree Sizes
- Centroid Decomposition
- Rerooting Technique

🎯 9. Practice Problems

- LeetCode: Easy to Hard Tree problems
- Binary Tree Zigzag Level Order Traversal
- Serialize and Deserialize Tree
- Construct Tree from Inorder + Preorder/Postorder
- Tree Isomorphism

If you'd like, I can also give:

- ✅ Topic-wise problems for each section
- ✅ C++/Java code templates for each topic
- ✅ Visual diagrams to understand concepts better

Just let me know what you need next!