DSA Notes

1. Introduction to Data Structures and Algorithms

Data Structures refer to the way data is stored, organized, and managed for efficient access and modification.
Algorithms are step-by-step procedures or formulas for solving a problem.

Understanding data structures and algorithms is essential for solving problems efficiently and improving the performance of software.

---

2. Types of Data Structures

2.1 Linear Data Structures

- **Array**: A collection of elements stored in contiguous memory locations.
  - **Advantages**: Fast access using indices, simple structure.
  - **Disadvantages**: Fixed size, costly insertions and deletions.
- **Linked List**: A collection of nodes, where each node contains data and a reference (or link) to the next node.
  - **Types**: Singly Linked List, Doubly Linked List, Circular Linked List.
  - **Advantages**: Dynamic size, efficient insertions and deletions.
  - **Disadvantages**: Random access is not possible.
- **Stack**: A linear data structure that follows Last In, First Out (LIFO) order.
  - **Operations**: `push`, `pop`, `peek`, `is_empty`
- **Queue**: A linear data structure that follows First In, First Out (FIFO) order.
  - **Types**: Simple Queue, Circular Queue, Priority Queue, Deque.
  - **Operations**: `enqueue`, `dequeue`, `peek`, `is_empty`

2.2 Non-Linear Data Structures

- **Tree**: A hierarchical data structure consisting of nodes connected by edges.
  - **Types**: Binary Tree, Binary Search Tree (BST), AVL Tree, Heap, etc.
  - **Operations**: Traversal (in-order, pre-order, post-order), insertion, deletion, searching.
- **Graph**: A collection of nodes (vertices) and edges that connect pairs of nodes.
  - **Types**: Directed, Undirected, Weighted, Unweighted, Cyclic, Acyclic.
  - **Algorithms**: BFS (Breadth First Search), DFS (Depth First Search), Dijkstra’s algorithm, Kruskal’s and Prim’s algorithms.

2.3 Hashing

- **Hash Table**: A data structure that maps keys to values using a hash function.
  - **Operations**: Insertion, deletion, search.
  - **Collision resolution**: Chaining, Open Addressing.

---

3. Algorithms

3.1 Searching Algorithms

- **Linear Search**: Sequential search through an array.
  - **Time Complexity**: O(n)
- **Binary Search**: Efficient search on a sorted array by repeatedly dividing the search interval in half.
  - **Time Complexity**: O(log n)

3.2 Sorting Algorithms

- **Bubble Sort**: Repeatedly compares adjacent elements and swaps them if they are in the wrong order.
  - **Time Complexity**: O(n^2)
- **Selection Sort**: Finds the minimum element in the unsorted part and swaps it with the first unsorted element.
  - **Time Complexity**: O(n^2)
- **Insertion Sort**: Builds the sorted list one element at a time by repeatedly picking the next element and inserting it in the correct position.
  - **Time Complexity**: O(n^2)
- **Merge Sort**: A divide-and-conquer algorithm that splits the array into halves, sorts them, and then merges them.
  - **Time Complexity**: O(n log n)
- **Quick Sort**: A divide-and-conquer algorithm that picks a pivot and partitions the array around the pivot.
  - **Time Complexity**: O(n log n) on average, O(n^2) in the worst case
- **Heap Sort**: A comparison-based sorting algorithm that uses a binary heap.
  - **Time Complexity**: O(n log n)

3.3 Greedy Algorithms

- A greedy algorithm builds a solution step by step by choosing the locally optimal solution at each step with the hope of finding a global optimum.
  - **Examples**: Fractional Knapsack Problem, Huffman Encoding, Prim's and Kruskal's algorithms for Minimum Spanning Tree (MST).

3.4 Dynamic Programming

- A technique used for solving optimization problems by breaking them down into smaller subproblems and storing their results.
  - **Examples**: Fibonacci Sequence, 0/1 Knapsack Problem, Longest Common Subsequence (LCS), Matrix Chain Multiplication.

3.5 Backtracking

- A technique for solving problems incrementally and abandoning a solution as soon as it determines it cannot be completed.

        - **Examples**: N-Queens Problem, Sudoku Solver, Subset Sum Problem.

---

4. Time and Space Complexity Analysis

- **Big O Notation**: Represents the upper bound (worst-case) time complexity of an algorithm.
    - **Common Complexities**: O(1), O(log n), O(n), O(n log n), O(n^2), O(2^n)
- **Best Case**: The best scenario for the algorithm.
- **Worst Case**: The worst scenario for the algorithm.
- **Average Case**: The average scenario for the algorithm.

Time Complexity Example:

- **Bubble Sort**:
    - Worst Case: O(n^2)
    - Best Case: O(n)

Space Complexity: Measures the amount of extra memory required by the algorithm in addition to the input data.

---

5. Advanced Data Structures and Concepts

5.1 Trie

- A tree-like data structure used for storing strings efficiently, enabling fast retrieval of words and prefixes.
    - **Operations**: Insertion, search, prefix matching.

5.2 Segment Tree

- A tree-like structure used for storing intervals or segments. It allows querying and updating range-based information efficiently.
    - **Operations**: Range sum, range minimum, range maximum.

5.3 Disjoint Set (Union-Find)

- A data structure that keeps track of a partition of a set into disjoint subsets, supporting operations like find and union.
    - **Operations**: `find`, `union`
    - **Applications**: Kruskal's algorithm for MST.

---

6. Graph Algorithms

6.1 Breadth-First Search (BFS)

- An algorithm for traversing or searching a graph in level order.
    - **Time Complexity**: O(V + E) where V is the number of vertices and E is the number of edges.

6.2 Depth-First Search (DFS)

- An algorithm for traversing or searching a graph by going as deep as possible down one branch before backtracking.
  - **Time Complexity**: O(V + E)

6.3 Dijkstra's Algorithm

- A shortest-path algorithm for weighted graphs with non-negative edge weights.
  - **Time Complexity**: O(V^2) (can be improved with min-heap to O(E + V log V))

6.4 Floyd-Warshall Algorithm

- An algorithm for finding shortest paths between all pairs of vertices in a graph.
  - **Time Complexity**: O(V^3)

---

7. Practice Problems

7.1 Arrays

- Find the maximum subarray sum (Kadane's Algorithm).
- Rotate an array.
- Move all zeroes to the end of an array.

7.2 Linked Lists

- Reverse a singly linked list.
- Detect a cycle in a linked list.
- Merge two sorted linked lists.

7.3 Trees

- Check if a binary tree is balanced.
- Find the height of a binary tree.
- Perform in-order, pre-order, and post-order traversal.

7.4 Graphs

- Find the shortest path between two nodes in an unweighted graph (BFS).
- Detect a cycle in a directed graph (DFS).
- Find the Minimum Spanning Tree (MST) using Prim's or Kruskal's algorithm.

---

8. Conclusion

Mastering **Data Structures** and **Algorithms** is key to excelling in technical interviews and problem-solving. Understanding the time and space complexity of algorithms and choosing the appropriate data structure for a given problem will help you write efficient and scalable code.