Day 1 – Introduction & Sorting Basics

Topics Covered

1. **Introduction to Analysis of Algorithms**
   - Meaning of an algorithm and why we analyze them.
   - Parameters for analysis: **time complexity**, **space complexity**, and **efficiency**.
   - Role of algorithm analysis in optimizing performance.
2. **Importance of Algorithm Analysis**
   - Helps in selecting the best algorithm among alternatives.
   - Determines scalability for large inputs.
   - Guides in optimizing software and system design.
3. **Time Complexity & Big-O Notation Basics**
   - **Time complexity:** The computational time taken by an algorithm as a function of input size.
   - **Big-O Notation:** Describes **upper bound** or worst-case performance.
     - Examples:
       - O(1) – Constant time
       - O(n) – Linear time
       - O(n^2) – Quadratic time
       - O(log n), O(n log n) – Logarithmic/Log-linear time

Lab Work

- **Bubble Sort (ascending order):**
  Compare adjacent elements and swap if out of order. Repeat till sorted.
  Time Complexity: O(n^2)
- **Quick Sort (ascending order):**
  Uses **Divide and Conquer**. Choose a pivot, partition array, recursively sort subarrays.
  Time Complexity: Average O(n log n)

Day 2 – Advanced Sorting & Graph Basics

Topics Covered

1. **Merge Sort (Divide & Conquer Approach)**
   - Split array into halves, sort each recursively, and merge.
   - Stable sorting algorithm.
     Time Complexity: O(n log n)
2. **Introduction to Graphs**
   - Graph Terminology: vertices, edges, degree, connected components.
   - Types: Directed, Undirected, Weighted, Unweighted.
   - **Graph Representation:**
     - **Adjacency Matrix** – 2D array representation.
     - **Adjacency List** – Linked list for each vertex showing connected nodes.

Lab Work

- Implement **Merge Sort**
- Practice graph creation using both representations.

Day 3 – Graph Traversal Algorithms

Topics Covered

1. **DFS (Depth First Search)**
   - Explore as far as possible along each branch before backtracking.
   - Uses **stack** (recursion or explicit).
   - **Applications:** Cycle detection, Topological sorting, Maze solving.
2. **BFS (Breadth First Search)**
   - Explore all neighbors at current level before moving deeper.
   - Uses **queue**.
   - **Applications:** Shortest path in unweighted graph, Web crawling.

Lab Work

- Program for **DFS**
- Program for **BFS**

Day 4 – Backtracking (Part 1)

Topics Covered

1. **Concept of Backtracking**
   - A **refined brute-force** approach.
   - Builds solution incrementally and abandons if it violates constraints (pruning).
   - Used in constraint satisfaction problems.
2. **N-Queens Problem**
   - Place N queens on an N×N chessboard such that no two queens attack each other.
   - **Approach:** Try placing a queen row by row and backtrack if conflict occurs.

Lab Work

- Implement **N-Queens** using recursion and backtracking.

Day 5 – Backtracking (Part 2)

Topics Covered

1. **Applications of Backtracking**
   - Puzzle solving, combinatorial optimization, pathfinding.
2. **Sum of Subsets Problem**
   - Find subsets of a set that sum to a given value using backtracking.
3. **Hamiltonian Circuit Problem**
   - Find a path visiting each vertex exactly once and returning to the start.

Lab Work

- Implement **Sum of Subsets**
- Implement **Hamiltonian Circuit**

Day 6 – Greedy Algorithms (Part 1)

Topics Covered

1. **Introduction to Greedy Strategy**
   - Builds up a solution piece by piece, choosing the **locally optimal** choice at each step.
   - Doesn’t guarantee global optimum for all problems.
2. **Job Sequencing with Deadlines**
   - Schedule jobs with deadlines and profits to maximize total profit.
   - Select job with **highest profit** that fits in available time.

Lab Work

- Implement **Job Sequencing using Greedy Approach**

Day 7 – Greedy Algorithms (Part 2)

Topics Covered

1. **Single Source Shortest Path Problem**
   - Find shortest path from one source to all vertices.
2. **Algorithms:**
   - **Dijkstra’s Algorithm:** Works on **weighted, non-negative edges**.
   - **Prim’s Algorithm:** Constructs **Minimum Spanning Tree (MST)** by adding smallest edges connecting new vertices.

Lab Work

- Implement **Dijkstra’s Algorithm**
- Implement **Prim’s Algorithm**

Day 8 – Greedy + MST (Continuation)

Topics Covered

1. **Minimum Spanning Tree (MST)**
   - Spanning tree with minimum total edge weight.
   - Used in network design, clustering, etc.
2. **Kruskal's Algorithm**
   - Sort edges by weight and pick smallest edge that doesn't form a cycle.
3. **Prim's vs Kruskal's**
   - **Prim's:** Grows MST from a vertex.
   - **Kruskal's:** Grows MST edge by edge (using Disjoint Set).

Lab Work

- Implement **Kruskal's Algorithm**
- Compare **Prim's vs Kruskal's**

Day 9 – Dynamic Programming (Part 1)

Topics Covered

1. **Introduction to Dynamic Programming (DP)**
   - Solves problems by breaking them into overlapping subproblems.
   - Uses **Memoization (Top-Down)** or **Tabulation (Bottom-Up)**.
2. **0/1 Knapsack Problem**
   - Choose items with given weight & value to maximize profit within weight limit.
   - Each item can be either included or excluded (0/1).

Lab Work

- Implement **0/1 Knapsack using DP approach**

Day 10 – Dynamic Programming (Part 2) & Wrap-up

Topics Covered

1. **Optimal Binary Search Tree (OBST)**
   - Construct a BST with minimum search cost given frequencies of keys.
2. **Greedy vs DP Knapsack**
   - **Greedy:** Works for **Fractional Knapsack** (items can be divided).
   - **DP:** Works for **0/1 Knapsack** (items are indivisible).
3. **Floyd's Algorithm (All Pairs Shortest Path)**
   - Finds shortest paths between all pairs of vertices in a weighted graph.

Lab Work

- Implement **Optimal Binary Search Tree**
- Implement **Floyd's Algorithm**
