Design and Analysis of Algorithms - Unit-wise Summaries

Unit-I: Introduction to Algorithms

1. Definition of Algorithm: A finite set of well-defined instructions that solve a problem.
2. Pseudocode Conventions: A way to express algorithms using a mix of code and human language.
3. Recursive Algorithms: Algorithms that call themselves to solve subproblems.
4. Time and Space Complexity: Measures of the amount of time and memory an algorithm uses.
5. Big-O Notation: Describes the worst-case scenario for time/space complexity.
6. Practical Complexities: Real-world performance considerations for algorithms.
7. Randomized Algorithms: Algorithms that use random numbers to influence decisions.
8. Repeated Element Testing: Checking if an element appears multiple times in a set.
9. Primality Testing: Algorithms for determining if a number is prime.
10. Divide and Conquer: Breaking down a problem into smaller subproblems, solving them independently, then combining the results (e.g., Merge Sort).

Unit-II: Advanced Divide and Conquer

1. Quicksort: A divide-and-conquer sorting algorithm that selects a pivot and partitions the array.
2. Strassen's Matrix Multiplication: An algorithm for multiplying matrices more efficiently.
3. Greedy Method: An algorithm that makes the locally optimal choice at each step (e.g., job scheduling).
4. Knapsack Problem: Optimization problem where you maximize value while staying within a weight limit.
5. Tree Vertex Splitting: Dividing a tree's vertices in certain ways to solve problems.
6. Job Sequencing with Deadlines: Scheduling tasks with deadlines to maximize profit.

7. Optimal Storage on Tapes: Algorithms to minimize storage costs when reading/writing data.

Unit-III: Dynamic Programming

1. General Method: Breaking problems into simpler subproblems, solving each once, and storing the solutions.

2. Multistage Graphs: Graphs with multiple layers, solved using dynamic programming.

3. All-Pairs Shortest Paths: Algorithms to find shortest paths between all pairs of vertices (e.g., Floyd-Warshall).

4. Single-Source Shortest Paths: Algorithms to find the shortest path from one vertex to others (e.g., Dijkstra's).

5. String Editing: Finding the minimum number of edits needed to transform one string into another.

6. 0/1 Knapsack: A variation where items can either be taken or left.

7. Search Techniques for Graphs: Depth-First Search (DFS), Breadth-First Search (BFS) for exploring graphs.

8. Connected Components: Subgraphs where any two vertices are connected by paths.

9. Biconnected Components: Components that remain connected even if any one vertex is removed.

Unit-IV: Backtracking and Branch and Bound

1. Backtracking: A problem-solving method that explores all possibilities by building a solution incrementally (e.g., 8-Queens Problem).

2. Sum of Subsets: Finding subsets of numbers that sum up to a target value.

3. Graph Coloring: Assigning colors to vertices such that no two adjacent vertices share the same color.

4. Hamiltonian Cycles: A cycle that visits each vertex of a graph exactly once.

5. Branch and Bound: Algorithm for optimization problems like the Traveling Salesman Problem

(TSP) that prunes unnecessary solutions.

Unit-V: Lower Bound Theory and NP Problems

1. Lower Bound Theory: Proving that no algorithm can solve a problem faster than a certain limit.

2. Oracles and Advisory Arguments: Hypothetical tools used to solve decision problems in complexity theory.

3. NP-Hard and NP-Complete:

- NP-Hard: Problems as hard as the hardest problems in NP (nondeterministic polynomial time).

- NP-Complete: Problems that are both in NP and NP-Hard, meaning they can be verified in polynomial time and are the most difficult problems in NP.