Design and Analysis of Algorithms

UNIT I: Sorting

- Selection Sort: A simple sorting algorithm that divides the list into a sorted and unsorted part, repeatedly selecting the smallest element from the unsorted portion and placing it in the sorted portion.
- Insertion Sort: Builds the final sorted array one item at a time by repeatedly picking the next element and inserting it into its correct position.
- Bubble Sort: Compares adjacent elements and swaps them if they are in the wrong order, repeatedly iterating until the list is sorted.
- Heap Sort: A comparison-based sorting algorithm that uses a binary heap data structure to sort elements.
- Linear Time Sorting: Sorting algorithms that operate in O(n) time, such as Counting Sort, Radix Sort, and Bucket Sort.
- Running Time Analysis and Correctness: Discusses the time complexity (best, worst, average case) and ensures algorithms correctly sort the input.

UNIT II: Graphs

- Graph Traversals:
  - Depth-First Search (DFS): Explores as far as possible along each branch before backtracking.
  - Breadth-First Search (BFS): Explores neighbors level by level, visiting nodes in layers.
- Graph Connectivity: Determines whether all vertices in the graph are connected.
- Testing Bi-partiteness: Verifying if a graph can be colored using two colors such that no two adjacent vertices have the same color.
- Directed Acyclic Graphs (DAGs): A graph with directed edges and no cycles, essential for tasks like scheduling.

- Topological Ordering: A linear ordering of vertices such that for every directed edge uv, vertex u appears before v in the ordering.

UNIT III: Divide and Conquer & Intractability

- Divide and Conquer:
  - Breaks a problem into smaller sub-problems, solves them recursively, and combines the solutions.
  - Algorithms:
    - Merge Sort: Splits the array, sorts the halves, and merges them.
    - Quick Sort: Partitions the array and sorts partitions recursively.
    - Maximum Subarray Problem: Finds the contiguous subarray with the maximum sum using a divide-and-conquer approach.
- Intractability:
  - Decision vs. Optimization Problems: Decision problems have yes/no answers, while optimization problems seek the best solution.
  - NP as a Class of Problems: Focuses on problems solvable in polynomial time by a nondeterministic machine.
  - NP-hardness and NP-completeness: Concepts related to computational complexity, identifying problems that are as hard as the hardest problems in NP.

UNIT IV: Greedy and Dynamic Programming

- Greedy Algorithms:
  - Builds a solution step by step, choosing the most optimal choice at each step.
  - Applications:
    - Minimum Spanning Trees: Algorithms like Kruskal's and Prim's.
    - Fractional Knapsack Problem: Solves using a greedy strategy by selecting items based on their value-to-weight ratio.

- Dynamic Programming (DP):
  - Solves problems by breaking them into overlapping sub-problems, solving each once, and storing the results.
  - Applications:
    - Subset Sum Problem: Finds a subset of numbers that sums up to a target.
    - Integer Knapsack Problem: Maximizes the value of items in a knapsack with a weight limit.
  - Optimized for problems where sub-problems recur.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. Introduction to Algorithms. 3rd Edition. Pearson Education, 2010.
2. Kleinberg, J., Tardos, E. Algorithm Design. 1st Edition. Pearson, 2013.