NAME :M SHARANYA **ID : 2300090301**

DESIGN AND ANALYSIS OF ALGORTHIMS

Sorting Algorithms

Sorting algorithms arrange data in a particular order (usually ascending or descending).

- **Selection Sort**: Repeatedly selects the smallest (or largest) element and places it in the correct position. Simple but inefficient for large datasets ($O(n^2)$).
- **Insertion Sort**: Builds the final sorted array one element at a time by comparing and inserting elements into their correct position. Efficient for small or partially sorted arrays ($O(n^2)$).
- **Merge Sort**: Divides the array into halves, recursively sorts them, and merges the sorted halves. More efficient than selection and insertion sorts (O(n log n)).
- **Quick Sort**: A divide-and-conquer algorithm that selects a "pivot" and partitions the array around it, then sorts each partition. Average case O(n log n), worst case $O(n^2)$, but practical and fast.

Searching Algorithms

Searching algorithms are used to find elements in a dataset or validate the existence of an element.

- **Binary Search**: Efficiently searches a sorted array by repeatedly dividing the search interval in half. Time complexity is O(log n).

Data Structures

Efficient data management is key to effective algorithms. Some important data structures introduced:

- **Stacks**: A last-in, first-out (LIFO) data structure. Used in algorithms like depth-first search (DFS) and function call management.
- **Queues**: A first-in, first-out (FIFO) data structure. Useful in breadth-first search (BFS) and task scheduling.
- **Priority Queues**: A queue where elements are dequeued based on priority rather than arrival order. Used in Dijkstra’s and Prim’s algorithms.

Divide and Conquer

This technique splits a problem into smaller subproblems, solves each subproblem independently, and combines the results. Both **merge sort** and **quick sort** fall into this category.

Union-Find (Dynamic Connectivity)

Union-Find is a data structure used to manage a set of elements that are partitioned into disjoint subsets. It supports two main operations:

- **Union**: Merge two sets.
- **Find**: Determine which set an element belongs to. Efficient for solving connectivity problems (e.g., determining if two nodes in a graph are connected).