**Week-I**

**Algorithms & DSA**

**Exercise-3:**

**Different Sorting Algorithms:**

1. **Bubble Sort**: Compares adjacent elements, and swaps them if they are in the wrong order.

**Time Complexity**: O(n^2) in the average and worst-case scenarios, O(n) in the best case (when the list is already sorted).

**Space Complexity**: O (1).

1. **Insertion Sort**: Builds the sorted array one item at a time by repeatedly picking the next item and inserting it into its correct position.

**Time Complexity**: O(n^2) in the average and worst-case scenarios, O(n) in the best case (when the list is already sorted).

**Space Complexity**: O (1).

1. **Quick Sort**: Divides the array into two smaller sub-arrays (low and high) based on a pivot element and recursively sorts the sub-arrays.

**Time Complexity**: O(n log n) - average, O(n^2) -worst case **Space Complexity**: O(log n) for the recursive stack.

1. **Merge Sort**: Divides the array into two halves, recursively sorts them, and then merges the two sorted halves.

**Time Complexity**: O(n log n) in all cases.

**Space Complexity**: O(n) due to the additional space required for the temporary arrays.

**Performance Comparison:**

**Bubble Sort**:

Best Case: O(n) (when the array is already sorted).

Average Case: O(n^2).

Worst Case: O(n^2).

**Quick Sort**:

Best Case: O(n log n).

Average Case: O(n log n).

Worst Case: O(n^2)

**Why Quick Sort is Generally Preferred Over Bubble Sort:**

Quick Sort has a better average-case time complexity of O(n log n) compared to Bubble Sort's O(n^2). This makes Quick Sort more efficient for larger datasets.

Bubble Sort, while simple to implement, is inefficient for large datasets due to its quadratic time complexity. Quick Sort is much faster for larger arrays and is widely used in practice due to its efficiency and adaptability.