**Sorting Algorithms**

**Bubble Sort**

- **Description**: Repeatedly compares adjacent elements and swaps them if they are in the wrong order. Continues until no swaps are needed.

- **Time Complexity**:

- Best Case: O(n) (already sorted)

- Average/Worst Case: O(n^2)

- Space Complexity: O(1)

**Insertion Sort**

- **Description**: Builds the sorted array one item at a time by inserting each element into its correct position within the already sorted portion.

- **Time Complexity**:

-Best Case: O(n) (already sorted)

- Average/Worst Case: O(n^2)

- Space Complexity: O(1)

**Quick Sort**

- **Description**: A divide-and-conquer algorithm that selects a pivot, partitions the array into elements less than and greater than the pivot, and recursively sorts the partitions.

- **Time Complexity**:

- Best/Average Case: O(n log n)

- Worst Case: O(n^2) (with poor pivot choices)

- Space Complexity: O(log n) (recursive stack)

**Merge Sort**

- **Description**: Divides the array into halves, recursively sorts each half, and then merges the sorted halves.

**- Time Complexity**:

-Best/Average/Worst Case: O(n log n)

- Space Complexity: O(n) (additional space for merging)

**Performance Comparison: Bubble Sort vs. Quick Sort**

**Bubble Sort:**

- Best Case: O(n) (when already sorted)

- Average Case: O(n^2)

- Worst Case: O(n^2)

- Space Complexity: O(1)

**Quick Sort:**

- Best Case: O(n log n)

- Average Case: O(n log n)

- Worst Case: O(n^2) (rare, with poor pivot choices)

- Space Complexity: O(log n) (due to recursion stack)

**Why Quick Sort is Preferred**

**Quick Sort** is generally preferred over Bubble Sort due to its significantly better average-case time complexity (O(n log n) vs. O(n^2)) and its efficiency for large datasets. Bubble Sort’s quadratic time complexity makes it impractical for large or complex datasets, while Quick Sort offers faster performance and is more scalable.