**WEEK1\_ALGORITHMS\_DATA\_STRUCTURES**

**EXERCISE-3**

**Explanation of Sorting Algorithms**

**Bubble Sort**

* **Description:** A simple comparison-based sorting algorithm where each pair of adjacent elements is compared and swapped if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity:** O(n^2) in the worst and average case, O(n) in the best case.

**Insertion Sort**

* **Description:** Builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
* **Time Complexity**: O(n^2) in the worst and average case, O(n) in the best case.

**Quick Sort**

* **Description**: An efficient sorting algorithm that uses a divide-and-conquer approach. It selects a 'pivot' element from the array and partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.
* **Time Complexity:** O(n log n) on average, O(n^2) in the worst case.

**Merge Sort**

* **Description:** A divide-and-conquer algorithm that splits the array into halves, recursively sorts them, and then merges the sorted halves.
* **Time Complexity:** O(n log n) in all cases.

**Analysis**

**Time Complexity**

* **Bubble Sort**: O(n^2)
* **Quick Sort**: O(n log n) on average, O(n^2) in the worst case

**Comparison**

* **Bubble Sort**: Simple but inefficient for large datasets due to its O(n^2) complexity.
* **Quick Sort**: More efficient for larger datasets, generally preferred over Bubble Sort despite its worst-case complexity.