**COGNIZANT DIGITAL NURTURE – 3.0**

**JAVA FSE**

**WEEK – 1 EXERCISES**

**DATA STRUCTURES AND ALGORITHMS**

**Exercise 3: Sorting Customer Orders**

**Step 1: Understand Sorting Algorithms:**

**I. Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

* **Bubble Sort:**

**Description:** Bubble Sort goes over the list iteratively, compares elements that are next to each other, and swaps them if they are out of order. Until the list is sorted, these steps are repeated.

**Time Complexity:**

* Best Case: O(n) (when the list is already sorted)
* Average Case: O(n^2)
* Worst Case: O(n^2)

**Characteristics:** Simple but inefficient for large lists.

* **Insertion Sort:**

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

**Time Complexity**:

* **Best Case**: O(n) (when the list is already sorted)
* **Average Case**: O(n^2)
* **Worst Case**: O(n^2)

**Characteristics**: Efficient for small data sets or nearly sorted data.

* **Quick Sort**:

**Description:** Quick Sort selects a 'pivot' element and partitions the array into two sub-arrays, according to whether the elements are less than or greater than the pivot. It then recursively sorts the sub-arrays.

**Time Complexity**:

* **Best Case**: O (n log n)
* **Average Case**: O (n log n)
* **Worst Case**: O (n^2) (when the smallest or largest element is always chosen as the pivot)

**Characteristics**: Efficient and commonly used; performs well in practice.

* **Merge Sort:**

**Description:** Merge Sort divides the list into equal halves, sorts each half, and then merges the sorted halves back together.

**Time Complexity**:

* + **Best Case**: O (n log n)
  + **Average Case**: O (n log n)
  + **Worst Case**: O (n log n)

**Characteristics**: Stable sort; good performance but requires additional memory.

**Step 2: Setup**

***Refer Program Files***

**Step 3: Implementation**

***Refer Program Files***

**Step 4: Analysis**

**I. Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

**Performance Comparison**

* **Bubble Sort**:
  + **Best Case**: O(n)
  + **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)

**II. Discuss why Quick Sort is generally preferred over Bubble Sort.**

* **Efficiency:** Since Quick Sort has a better average and best-case time complexity of O (n log n) compared to Bubble Sort's O(n^2), it is typically chosen over Bubble Sort. Because of this, Quick Sort is far more effective with bigger datasets.
* **Performance:** Because Quick Sort uses a divide-and-conquer technique, it performs better in real-world scenarios and can handle larger datasets.
* **Practical Use:** Because Bubble Sort is inefficient, it is rarely utilised in practice; in contrast, Quick Sort is frequently employed in numerous real-world scenarios.