**Exercise 3: Sorting Customer Orders**

1)Bubble Sort

Compares adjacent elements and swaps them if they are in the wrong order.

Repeatedly passes through the list until no swaps occur.

Inefficient for large datasets.

Time complexity: O(n^2).

2) Insertion Sort

Builds the sorted array one element at a time.

Compares the current element with the sorted portion and inserts it in the correct position.

Efficient for small datasets or partially sorted data.

Time complexity: O(n^2).

3)Quick Sort

A divide-and-conquer algorithm.

Picks a pivot element and partitions the array around it.

Recursively sorts the sub-arrays.

Generally efficient for most datasets.

Time complexity: O(n log n) on average.

4)Merge Sort

A divide-and-conquer algorithm.

Divides the array into halves, recursively sorts them, and merges the sorted halves.

Efficient for large datasets.

Time complexity: O(n log n).

5)Analysis of Time Complexity:

Bubble Sort: O(n^2)

Quick Sort: O(n log n) on average

6)Why Quick Sort is Preferred:

Quick Sort is generally faster than Bubble Sort for larger datasets due to its average-case time complexity of O(n log n) compared to O(n^2) for Bubble Sort. Quick Sort also has better cache performance and is more efficient in practice.