**Exercise 3: Sorting Customer Orders**

1. **Understand Sorting Algorithms:**

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

Ans) **Bubble Sort: -** Bubble Sort is a simple comparison-based sorting algorithm. It works by repeatedly stepping through the list to be sorted, comparing adjacent elements, and swapping them if they are in the wrong order. This process is repeated until the list is sorted.

**Time Complexity**

* Best-case: O(n) (when the array is already sorted)
* Average-case: O(n^2)
* Worst-case: O(n^2)

**Space Complexity**

* O(1) (in-place sort)

**Insertion Sort: -** Insertion Sort is a comparison-based sorting algorithm that builds the final sorted array one element at a time. It is much less efficient on large lists than more advanced algorithms such as Quick Sort, Merge Sort, or Heap Sort.

**Time Complexity**

* Best-case: O(n) (when the array is already sorted)
* Average-case: O(n^2)
* Worst-case: O(n^2)

**Space Complexity**

* O(1) (in-place sort)

**Quick Sort: -** Quick Sort is a highly efficient sorting algorithm and is based on partitioning of an array into smaller sub-arrays. A pivot element is chosen, and the array is partitioned so that elements less than the pivot are on the left and elements greater than the pivot are on the right. This process is recursively applied to the sub-arrays.

**Time Complexity**

* Best-case: O(n log n)
* Average-case: O(n log n)
* Worst-case: O(n^2) (when the pivot selection is poor)

**Space Complexity**

* O(log n) (due to the recursive call stack)

**Merge Sort: -** Merge Sort is a divide-and-conquer algorithm. It divides the input array into two halves, sorts each half, and then merges the two sorted halves to produce the final sorted array. It is known for its stable sorting.

**Time Complexity**

* Best-case: O(n log n)
* Average-case: O(n log n)
* Worst-case: O(n log n)

**Space Complexity**

* O(n) (due to the temporary arrays used in merging)

4. **Analysis:**

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

Ans) **Bubble Sort:**

Best case: O(n)

Average case: O(n^2)

Worst case: O(n^2)

This makes Bubble sort inefficient for large data sets. It repeatedly goes through the list to swap the adjacent elements if they are in the wrong order. The time complexity is, hence, quadratic in the average and worst cases, which makes this sorting method unsuitable for large data sets.

**Quick Sort:**

Best case: O(n log n)

Average case: O(n log n)

Worst case: O(n^2)

Quick Sort is generally more efficient than Bubble Sort. It uses a divide-and-conquer approach to sort the list. Though its worst-case time complexity is O(n^2), it rarely happens and can often be mitigated with good pivot selection strategies. Its time complexity of O(n log n) at average and best cases makes it suitable for large datasets.

Q) Discuss why Quick Sort is generally preferred over Bubble Sort.

Ans) The main reasons Quick Sort is preferred over Bubble Sort include that it has an average-case time complexity much faster for large datasets. While the time complexity of Bubble Sort increases quadratically with respect to the size of the input, that of Quick Sort grows logarithmically on average, hence it is much faster. Besides, it has far better performance characteristics and is recommended for sorting large arrays or lists.