**Exercise 3: Sorting Customer Orders**

1. **Understanding Sorting Algorithms:**
   * **Q) Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**
     + **Ans)**
       - **Bubble Sort**: A simple comparison-based sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues 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**: A comparison-based sorting algorithm that builds the final sorted array one element at a time. It is less efficient on large lists compared to more advanced algorithms like Quick Sort or Merge 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**: A highly efficient sorting algorithm based on partitioning 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**: A divide-and-conquer algorithm that 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)

1. **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)
         * Bubble Sort is inefficient for large datasets as it repeatedly goes through the list to swap adjacent elements if they are in the wrong order. The quadratic time complexity in average and worst cases makes it unsuitable for large datasets.
       - **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. Although its worst-case time complexity is O(n^2), this rarely occurs and can often be mitigated with good pivot selection strategies. Its time complexity of O(n log n) in average and best cases makes it suitable for large datasets.
   * **Q) Discuss why Quick Sort is generally preferred over Bubble Sort.**
     + **Ans)** Quick Sort is preferred over Bubble Sort mainly because it has a much faster average-case time complexity for large datasets. While Bubble Sort’s time complexity increases quadratically with the size of the input, Quick Sort’s time complexity grows logarithmically on average, making it significantly faster. Additionally, Quick Sort has better performance characteristics and is recommended for sorting large arrays or lists.