**Exercise 3: Sorting Customer Orders**

1. **Understand Sorting Algorithms:**
   * **Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).**

* **Bubble Sort:** Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.

**Steps:**

* + - Traverse the list from the beginning to the end.
    - Compare adjacent elements and swap them if the first element is greater than the second.
    - After each pass, the largest unsorted element "bubbles up" to its correct position.
    - Repeat the process for the remaining unsorted portion of the list.
* **Insertion Sort:** Insertion Sort builds the final sorted array one item at a time. It picks the next item from the input data and inserts it into the correct position in the already sorted portion of the list.

**Steps:**

* Start with the second element and compare it to the elements before it.
* Shift all elements greater than the current element to the right.
* Insert the current element into its correct position.
* Repeat for each element in the list.
* **Quick Sort:** Quick Sort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element and partitioning the array into two sub-arrays—elements less than the pivot and elements greater than the pivot. It then recursively sorts the sub-arrays.

**Steps:**

* Choose a pivot element from the array.
* Partition the array so that elements less than the pivot are on the left, and elements greater than the pivot are on the right.
* Recursively apply the same process to the sub-arrays.
* **Merge Sort:** Merge Sort is a divide-and-conquer algorithm that divides the array into halves, sorts each half, and then merges the sorted halves to produce a fully sorted array.

**Steps:**

* Divide the array into two halves.
* Recursively sort each half.
* Merge the two sorted halves into a single sorted array.

1. **Analysis:**
   * **Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

**Ans:**

* Bubble Sort:

Best Case: O(n) (when the list is already sorted)

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) (when the smallest or largest element is always chosen as the pivot)

* + **Discuss why Quick Sort is generally preferred over Bubble Sort.**
* Quick Sort is generally preferred over Bubble Sort due to its superior efficiency in handling large datasets.
* Quick Sort has an average-case time complexity of O(n log n), which makes it significantly faster than Bubble Sort's average-case time complexity of O(n²).
* While Bubble Sort performs numerous redundant comparisons and swaps, leading to poor performance with large or unsorted arrays, Quick Sort efficiently divides the array into smaller partitions and sorts them recursively. This divide-and-conquer approach minimizes unnecessary operations, resulting in faster sorting times, especially for larger datasets.
* Quick Sort's efficiency, despite its worst-case time complexity of O(n²) (which can be mitigated with good pivot selection strategies), makes it a more scalable and practical choice compared to Bubble Sort.