**EXERCISE 3**

**Sorting Customer Orders**

**1. Understanding Sorting Algorithms**

**Bubble Sort**

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

**Insertion Sort**

Insertion Sort builds the final sorted array one item at a time by inserting each new element into its proper place. It is efficient for small datasets or nearly sorted arrays.

**Quick Sort**

Quick Sort is a divide-and-conquer algorithm that selects a 'pivot' element and partitions the other elements into two sub-arrays according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

**Merge Sort**

Merge Sort is a divide-and-conquer algorithm that divides the array into two halves, recursively sorts them, and then merges the sorted halves to produce a single sorted array.

**2. Setup**

Define a class representing an order with attributes such as orderId, customerName, and totalPrice. This class will be used to store and sort orders based on their totalPrice.

**3. Implementation**

**Bubble Sort**

Bubble Sort is implemented by repeatedly stepping through the list, comparing adjacent items, and swapping them if they are in the wrong order. This process continues until no more swaps are needed.

**Quick Sort**

Quick Sort is implemented by selecting a pivot and partitioning the array into elements less than and greater than the pivot. The sub-arrays are then sorted recursively.

**4. Analysis**

**Performance Comparison**

* **Bubble Sort:**
  + Time Complexity: O(n²)
  + Description: Inefficient for large datasets due to its quadratic time complexity, even though it’s simple to implement. It performs many redundant comparisons and swaps.
* **Quick Sort:**
  + Time Complexity: O(n log n) on average
  + Description: Generally more efficient for large datasets compared to Bubble Sort due to its logarithmic time complexity in the average case. It efficiently divides and conquers the array, with fewer comparisons and swaps.

Quick Sort is generally preferred over Bubble Sort due to:

* **Efficiency:** Quick Sort’s average time complexity of O(n log n) makes it much faster than Bubble Sort’s O(n²) for large datasets.
* **Performance**: Quick Sort typically performs fewer comparisons and swaps, leading to better overall performance.
* **Scalability:** Quick Sort scales better with increasing input sizes compared to Bubble Sort.

Bubble Sort can be useful for small or simple cases, Quick Sort is the better choice for sorting larger datasets due to its superior efficiency and performance characteristics.