**Exercise 3: Sorting Customer Orders**

Scenario:

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

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

Bubble Sort : It is a comparison-based sorting algorithm. It 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. The worst case time complexity for bubble sort is *O(n^2).*

Insertion sort: Insertion Sorts the final array one item at a time. It is less efficient on large lists than more advanced algorithms like Quick Sort or Merge Sort but is more efficient than Bubble Sort. The worst case time complexity for insertion sort is *O(n^2).*

Quick sort: Quick Sort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The worst case time complexity for the quick sort algorithm is *O(n^2).*

Merge sort: Merge Sort is a divide-and-conquer algorithm that splits the array into halves, recursively sorts each half, and then merges the sorted halves to produce a sorted array. The time complexity for merge sort is O(n log n).

Analysis:

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

**Time Complexity Comparison: Bubble Sort vs. Quick Sort**

**1. Bubble Sort**

* **Best Case Time Complexity**: O(n)  
  Occurs when the array is already sorted. Bubble Sort can achieve this if an optimization is used to check if any swaps were made during the pass.
* **Average Case Time Complexity**: O(n^2)  
  Bubble Sort performs n-1 passes over the array, with each pass involving up to n comparisons and potentially n-1 swaps. Thus, the time complexity is quadratic on average.
* **Worst Case Time Complexity**: O(n^2)  
  The worst case occurs when the array is sorted in reverse order. Bubble Sort will still make n-1 passes, each requiring up to n comparisons and swaps.

**2. Quick Sort**

* **Best Case Time Complexity**: O(n log n)  
  Occurs when the pivot divides the array into two nearly equal halves. Quick Sort performs log n levels of recursion, each requiring O(n) comparisons.
* **Average Case Time Complexity**: O(n log n)  
  On average, Quick Sort also divides the array into two roughly equal parts, leading to log n levels of recursion with O(n) comparisons at each level.
* **Worst Case Time Complexity**: O(n^2)  
  Occurs when the pivot is the smallest or largest element in the partitioned array, leading to unbalanced partitions.

o Discuss why Quick Sort is generally preferred over Bubble Sort.

**. Time Complexity**

* **Quick Sort:** On average, Quick Sort has a time complexity of O(nlogn). This efficiency arises because Quick Sort partitions the array into smaller sub-arrays, sorts them recursively, and then combines the results. Even in the worst case, where the time complexity can be O(n2), careful implementations with random pivots or median-of-three strategies can mitigate this.
* **Bubble Sort:** Bubble Sort has a time complexity of O(n2) in both the average and worst cases. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This approach results in many redundant comparisons and swaps, leading to its poor performance for large datasets.

**2. Efficiency with Large Datasets**

* **Quick Sort:** Due to its O(nlogn) average time complexity, Quick Sort performs significantly better with large datasets compared to Bubble Sort. It reduces the problem size more quickly through partitioning, leading to fewer overall comparisons and swaps.
* **Bubble Sort:** For large datasets, Bubble Sort becomes impractically slow because it has to perform O(n2) comparisons and swaps. This inefficiency makes it unsuitable for sorting large amounts of data.

**3. Number of Comparisons and Swaps**

* **Quick Sort:** Generally performs fewer comparisons and swaps compared to Bubble Sort. The partitioning process helps quickly sort sub-arrays, reducing the total number of operations needed.
* **Bubble Sort:** In the worst case, Bubble Sort can makeO(n2) comparisons and swaps, making it less efficient. Even with optimized versions that track if the list is already sorted, it still performs more operations compared to Quick Sort.