**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.

**Steps:**

1. **Understand Sorting Algorithms:**
   * Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).
2. **Setup:**
   * Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
   * Implement **Bubble Sort** to sort orders by **totalPrice**.
   * Implement **Quick Sort** to sort orders by **totalPrice**.
4. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.
   * Discuss why Quick Sort is generally preferred over Bubble Sort.

**Concept :**

**Understand Sorting Algorithms**

**Bubble Sort**: A simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. It has a time complexity of O(n^2).

**Insertion Sort**: A simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort or merge sort. It has a time complexity of O(n^2).

**Quick Sort**: An efficient sorting algorithm that uses the divide-and-conquer approach. 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. It has an average-case time complexity of O(n log n).

**Merge Sort**: A stable, comparison-based, divide-and-conquer sorting algorithm. Most implementations produce a stable sort, meaning that the order of equal elements is the same in the input and output. It has a time complexity of O(n log n).

**Performance Comparison:**

* **Bubble Sort**:
  + **Best Case**: O(n) - when the array is already sorted.
  + **Average Case**: O(n^2) - in general.
  + **Worst Case**: O(n^2) - when the array is in reverse order.
* **Quick Sort**:
  + **Best Case**: O(n log n) - when the pivot divides the array into two equal halves.
  + **Average Case**: O(n log n) - in general.
  + **Worst Case**: O(n^2) - when the pivot is the smallest or largest element (rare with good pivot selection).

**Why Quick Sort is Generally Preferred:**

* Quick Sort is generally preferred over Bubble Sort because it has a much better average-case time complexity (O(n log n)) compared to Bubble Sort's (O(n^2)). Quick Sort is efficient for large datasets and can be optimized with different pivot selection strategies to avoid the worst-case scenario. Bubble Sort, while simple to implement, is inefficient for large datasets due to its quadratic time complexity.