**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.

**Explanation**

**Step 1: Understand Sorting Algorithms**

**Bubble Sort:**

* Bubble Sort is a simple comparison-based 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.
* **Time Complexity:**
  + **Best Case:** O(n) (when the array is already sorted)
  + **Average Case:** O(n^2)
  + **Worst Case:** O(n^2)

**Insertion Sort:**

* Insertion Sort 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.
* **Time Complexity:**
  + **Best Case:** O(n) (when the array is already sorted)
  + **Average Case:** O(n^2)
  + **Worst Case:** 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 sub-arrays are then sorted recursively.
* **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, e.g., the smallest or largest element is always chosen)

**Merge Sort:**

* Merge Sort is also a divide-and-conquer algorithm. It divides the unsorted list into n sub-lists, each containing one element, then repeatedly merges sub-lists to produce new sorted sub-lists until there is only one sub-list remaining.
* **Time Complexity:**
  + **Best Case:** O(n log n)
  + **Average Case:** O(n log n)
  + **Worst Case:** O(n log n)

**Step 4: Analysis**

**Performance Comparison (Time Complexity) of Bubble Sort and Quick Sort:**

* **Bubble Sort:**
  + **Best Case:** O(n) (when the array 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 pivot selection is poor, e.g., the smallest or largest element is always chosen)

**Why Quick Sort is Generally Preferred Over Bubble Sort:**

* **Efficiency:** Quick Sort is generally more efficient than Bubble Sort for large datasets. Its average time complexity is O(n log n), making it significantly faster than Bubble Sort's O(n^2) for large inputs.
* **Divide and Conquer:** Quick Sort uses a divide-and-conquer approach, which can be more efficient in practice and leverages the power of recursion to handle large arrays more effectively.
* **Cache Performance:** Quick Sort has better cache performance compared to other sorting algorithms like Merge Sort due to its in-place sorting nature.

Quick Sort is preferred over Bubble Sort for sorting customer orders on an e-commerce platform due to its superior average-case time complexity and practical efficiency for large datasets.