**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).

**Bubble Sort**:

* **Explanation**: Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity**:
  + **Best Case**: O(n) (when the array is already sorted)
  + **Average Case**: O(n²)
  + **Worst Case**: O(n²)

**Insertion Sort**:

* **Explanation**: Insertion Sort builds the final sorted array one item at a time. It's less efficient on large lists compared to more advanced algorithms.
* **Time Complexity**:
  + **Best Case**: O(n) (when the array is already sorted)
  + **Average Case**: O(n²)
  + **Worst Case**: O(n²)

**Quick Sort**:

* **Explanation**: Quick Sort is a divide-and-conquer algorithm. It selects a 'pivot' element from the array and partitions the other elements into two sub-arrays based on whether they are less than or greater than the pivot. These sub-arrays are then sorted recursively.
* **Time Complexity**:
  + **Best Case**: O(n log n)
  + **Average Case**: O(n log n)
  + **Worst Case**: O(n²) (when the pivot selection is poor)

**Merge Sort**:

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

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

**Performance Comparison (Time Complexity)**:

* **Bubble Sort**:
  + Best Case: O(n)
  + Average Case: O(n²)
  + Worst Case: O(n²)
* **Quick Sort**:
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n²)

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

* **Efficiency**: Quick Sort is generally more efficient for large datasets, with an average time complexity of O(n log n), compared to Bubble Sort's O(n²).
* **Divide and Conquer**: Quick Sort uses a divide-and-conquer approach, making it more efficient in practice by leveraging recursion.
* **Cache Performance**: Quick Sort has better cache performance due to its in-place sorting nature, unlike Merge Sort.

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.