**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**

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.

* **Best Case**: O(n)
* **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, heapsort, or merge sort.

* **Best Case**: O(n)
* **Average Case**: O(n^2)
* **Worst Case**: O(n^2)

**Quick Sort**

Quick Sort is an efficient, divide-and-conquer, recursive 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.

* **Best Case**: O(n log n)
* **Average Case**: O(n log n)
* **Worst Case**: O(n^2) (rarely occurs, can be mitigated with good pivot selection)

**Merge Sort**

Merge Sort is an efficient, stable, 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.

* **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.

**Time Complexity Comparison**

* **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., always picking the smallest or largest element)
  + Discuss why Quick Sort is generally preferred over Bubble Sort.

Quick Sort is generally preferred over Bubble Sort for several reasons:

1. **Efficiency**: Quick Sort has an average-case time complexity of O(n log n), making it much faster than Bubble Sort's O(n^2) for large datasets.
2. **Performance**: Quick Sort is a divide-and-conquer algorithm, making it more efficient in handling and sorting large datasets compared to the simple comparison-based approach of Bubble Sort.
3. **Memory Usage**: Quick Sort is an in-place sorting algorithm, meaning it requires a constant amount of additional memory space, whereas Bubble Sort can be less efficient with memory due to excessive swapping.