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.

→ Step 1: Understand Sorting Algorithms

1. Bubble Sort

- Repeatedly swaps adjacent elements if they are in the wrong order.
- **Time Complexity:**
  - Best: $O(n)$
  - Average & Worst: $O(n^2)$
- **Easy to implement**, but inefficient for large data.

2. Insertion Sort

- Builds the final sorted array one item at a time.
- **Time Complexity:**
  - Best: $O(n)$
  - Worst: $O(n^2)$
- Good for small datasets.

3. Quick Sort

- Selects a pivot, partitions the array into two halves, and recursively sorts them.

- **Time Complexity:**
  - Best & Average: $O(n \log n)$
  - Worst: $O(n^2)$ (rare, happens with poor pivot choices)
- **Efficient and widely used**.

4. **Merge Sort**

- Divides array into halves, sorts them recursively, and merges the sorted halves.
- **Time Complexity:** $O(n \log n)$
- Stable and predictable, but uses more memory (not in-place).

Step 2: Setup Order Class

```
public class Order {
  int orderId;
  String customerName;
  double totalPrice;
  public Order(int orderId, String customerName, double totalPrice) {
     this.orderId = orderId;
     this.customerName = customerName;
     this.totalPrice = totalPrice;
  }
public String toString() {
     return "Order[ID=" + orderId + ", Name=" + customerName + ", Total=$" + totalPrice + "]";
  }
}
```

Step 3: Implementation

Bubble Sort (Ascending by totalPrice)

```
 public class SortAlgorithms {
   public static void bubbleSort(Order[] orders) {
     int n = orders.length;
     for (int i = 0; i < n - 1; i++) {
        boolean swapped = false;
        for (int j = 0; j < n - 1 - i; j++) {
           if (orders[j].totalPrice > orders[j + 1].totalPrice) {
```

```
                Order temp = orders[j];
                orders[j] = orders[j + 1];
                orders[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) break; // Optimization
    }
}
```

Quick Sort (Ascending by totalPrice)

```
public static void quickSort(Order[] orders, int low, int high) {
    if (low < high) {
        int pivotIndex = partition(orders, low, high);
        quickSort(orders, low, pivotIndex - 1);
        quickSort(orders, pivotIndex + 1, high);
    }
}
private static int partition(Order[] orders, int low, int high) {
    double pivot = orders[high].totalPrice;
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (orders[j].totalPrice <= pivot) {
            i++;
            Order temp = orders[i];
            orders[i] = orders[j];
            orders[j] = temp;
        }
    }
    Order temp = orders[i + 1];
    orders[i + 1] = orders[high];
    orders[high] = temp;
```

```
        return i + 1;
    }
}
```

Main Class

```
public class Main {
    public static void main(String[] args) {
        Order[] orders = {
            new Order(1, "Alice", 99.99),
            new Order(2, "Bob", 150.75),
            new Order(3, "Charlie", 45.50),
            new Order(4, "Daisy", 120.00)
        };
        // Bubble Sort
        System.out.println("Bubble Sort:");
        SortAlgorithms.bubbleSort(orders);
        for (Order o : orders) System.out.println(o);
        // Reset and Quick Sort
        orders = new Order[]{
            new Order(1, "Alice", 99.99),
            new Order(2, "Bob", 150.75),
            new Order(3, "Charlie", 45.50),
            new Order(4, "Daisy", 120.00)
        };
        System.out.println("\nQuick Sort:");
        SortAlgorithms.quickSort(orders, 0, orders.length - 1);
        for (Order o : orders) System.out.println(o);
    }
}
```

Step 4: Analysis

Algorithm	**Time Complexity (Best / Avg / Worst)**	**In-place**	**Stable**
Bubble Sort	$O(n)$ / $O(n^2)$ / $O(n^2)$	Yes	Yes
Quick Sort	$O(n \log n)$ / $O(n \log n)$ / $O(n^2)$	Yes	No

Why Quick Sort is Preferred:

- Much **faster** on large datasets (average $O(n \log n)$).
- Requires **less memory** than Merge Sort.
- **Efficient for real-time systems** like e-commerce platforms.

OUTPUT: