Exercise 3: Sorting Customer Orders

1. Understand Sorting Algorithms:

- **Bubble Sort:** Repeatedly swaps adjacent elements if they are in the wrong order. Time Complexity: O(n^2).

- Insertion Sort: Builds the sorted array one element at a time. Time Complexity: O(n^2).

- **Quick Sort:** Divides the array using a pivot and recursively sorts. Average Time Complexity: O(n log n).

- **Merge Sort:** Divides array into halves, sorts, and merges. Time Complexity: O(n log n).

2. Setup:

Creating a class Order with attributes like orderId, customerName, and totalPrice.

```
public class Order {
  private int orderId;
  private String customerName;
  private double totalPrice;

  public Order(int orderId, String customerName, double totalPrice) {
    this.orderId = orderId;
    this.customerName = customerName;
    this.totalPrice = totalPrice;
  }

  public int getOrderId() { return orderId; }
  public String getCustomerName() { return customerName; }
  public double getTotalPrice() { return totalPrice; }

  @Override
  public String toString() {
    return "Order[ID=" + orderId + ", Customer=" + customerName + ", Price=" + totalPrice
+ "]";
  }}
```

3. Implementation:

Implementing Bubble Sort and Quick Sort to sort the orders by totalPrice.

```
public static void bubbleSort(Order[] orders) {
    int n = orders.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {
                Order temp = orders[j];
                orders[j] = orders[j + 1];
                orders[j + 1] = temp;
            }
        }
    }
}
```

```
public static void quickSort(Order[] orders, int low, int high) {
    if (low < high) {
        int pi = partition(orders, low, high);
        quickSort(orders, low, pi - 1);
        quickSort(orders, pi + 1, high);
    }
}
```

```
private static int partition(Order[] orders, int low, int high) {
    double pivot = orders[high].getTotalPrice();
    int i = low - 1;


    for (int j = low; j < high; j++) {
        if (orders[j].getTotalPrice() <= 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;

}
```

4. Analysis:

- Bubble Sort has a time complexity of O(n^2), making it inefficient for large datasets.

- Quick Sort has an average time complexity of O(n log n), which is significantly faster.

- Quick Sort is generally preferred due to better performance and efficiency in most practical scenarios.