Sorting Customer Orders-EX 3

1. Understanding Sorting Algorithms

Sorting algorithms arrange the elements of a list in a particular order (typically ascending or descending). Here are some common sorting algorithms:

- **Bubble Sort**:
  - **Description**: A simple comparison-based algorithm where each pair of adjacent elements is compared and swapped if they are in the wrong order. This process is repeated until the list is sorted.
  - **Time Complexity**: O(n^2) in the worst and average cases. It performs well only for small datasets.
- **Insertion Sort**:
  - **Description**: Builds the sorted array one item at a time, picking the next element and inserting it into the correct position in the already sorted part of the array.
  - **Time Complexity**: O(n^2) in the worst and average cases, but O(n) in the best case when the array is already mostly sorted.
- **Quick Sort**:
  - **Description**: A divide-and-conquer algorithm that picks a "pivot" element, partitions the array around the pivot such that elements less than the pivot come before it and elements greater than the pivot come after it. This process is recursively applied to the sub-arrays.
  - **Time Complexity**: O(n log n) on average, but O(n^2) in the worst case (when the pivot selections are poor, such as when the smallest or largest element is consistently chosen as the pivot). However, with good pivot selection (e.g., using median-of-three), it usually performs well.
- **Merge Sort**:
  - **Description**: Also a divide-and-conquer algorithm that divides the array into halves, sorts each half, and then merges the sorted halves back together.
  - **Time Complexity**: O(n log n) in all cases, making it very efficient. It requires additional space for the merging process, which is O(n).

2. Setup

The Order class is defined to represent customer orders with attributes like orderId, customerName, and totalPrice.

```
class Order {

  private String orderId;
```

```
    private String customerName;
    private double totalPrice;

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

    public String getOrderId() {
        return orderId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public double getTotalPrice() {
        return totalPrice;
    }
}
```

3. Implementation

The implementation includes two sorting algorithms: Bubble Sort and Quick Sort, focusing on sorting the orders by totalPrice.

Bubble Sort Implementation:

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

Quick Sort Implementation:

```
class QuickSort {
    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) {
        Order pivot = orders[high];
        int i = (low - 1);
        for (int j = low; j < high; j++) {
            if (orders[j].getTotalPrice() > pivot.getTotalPrice()) {
                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

Time Complexity Comparison:

- **Bubble Sort**:
  - Best Case: O(n) (when the list 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)

Why Quick Sort is Generally Preferred Over Bubble Sort:

1. **Efficiency**: Quick Sort has an average and best-case time complexity of O(n log n), making it much more efficient than Bubble Sort's O(n^2) in most practical scenarios. This efficiency is crucial for large datasets, which are common in e-commerce platforms.
2. **In-place Sorting**: Quick Sort is an in-place sorting algorithm, meaning it requires only a small, constant amount of additional memory space, whereas Bubble Sort also operates in-place but does not scale well in terms of time complexity.
3. **Practical Performance**: Despite its worst-case scenario, Quick Sort often outperforms other O(n log n) algorithms like Merge Sort due to its good cache performance and lower overhead, provided a good pivot selection strategy is used.
4. **Adaptability**: Quick Sort can be optimized with various strategies, such as choosing a better pivot, switching to insertion sort for small sub-arrays, and more.