**Exercise 3: Sorting Customer Orders**

**1. Understand Sorting Algorithms**

**Bubble Sort:**

**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.**

**Insertion Sort:**

**Builds the final sorted array one item at a time. It picks the next item and inserts it into its correct position among the previously sorted items.**

**Quick Sort:**

**A divide-and-conquer algorithm that selects a pivot element and partitions the array into elements less than the pivot and elements greater than the pivot, then recursively sorts the partitions.**

**Merge Sort:**

**A divide-and-conquer algorithm that divides the list into two halves, recursively sorts each half, and then merges the two sorted halves.**

**2. Setup**

**// Order.java**

**public 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;**

**}**

**// Getters and Setters**

**public String getOrderId() {**

**return orderId;**

**}**

**public String getCustomerName() {**

**return customerName;**

**}**

**public double getTotalPrice() {**

**return totalPrice;**

**}**

**@Override**

**public String toString() {**

**return "Order{" +**

**"orderId='" + orderId + '\'' +**

**", customerName='" + customerName + '\'' +**

**", totalPrice=" + totalPrice +**

**'}';**

**}**

**}**

**3. Implementation**

**// BubbleSort.java**

**public 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 - i - 1; j++) {**

**if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {**

**// Swap orders[j] and orders[j + 1]**

**Order temp = orders[j];**

**orders[j] = orders[j + 1];**

**orders[j + 1] = temp;**

**}**

**}**

**}**

**}**

**}**

**// QuickSort.java**

**public 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) {**

**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:**

* **Time Complexity:**

**Best Case: O(n) (if the list is already sorted)**

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

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

**Quick Sort:**

* **Time Complexity:**

**Best Case: O(n log n)**

**Average Case: O(n log n)**

**Worst Case: O(n^2) (if pivot selection is poor, e.g., always picking the smallest or largest element)**

**Why Quick Sort is Generally Preferred:**

**Efficiency: Quick Sort is generally faster than Bubble Sort for large datasets due to its O(n log n) average time complexity.**

**Divide and Conquer: Quick Sort’s divide-and-conquer approach efficiently reduces the problem size in each step.**

**Adaptive: Quick Sort performs better on average than Bubble Sort, especially with large datasets.**