**Data Structures Algorithms**

**EXERCISE 3: Sorting Customer Orders**

**Source Code**

// Main.java

import java.util.\*;

// Step 2: Order class

class Order {

String orderId;

String customerName;

double totalPrice;

public Order(String orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public void display() {

System.out.println("Order ID: " + orderId +

", Customer: " + customerName +

", Total Price: ₹" + totalPrice);

}

}

public class Main {

// Step 3a: Bubble Sort implementation

public static void bubbleSort(List<Order> orders) {

int n = orders.size();

for (int i = 0; i < n - 1; i++) {

boolean swapped = false;

for (int j = 0; j < n - i - 1; j++) {

if (orders.get(j).totalPrice > orders.get(j + 1).totalPrice) {

// swap

Order temp = orders.get(j);

orders.set(j, orders.get(j + 1));

orders.set(j + 1, temp);

swapped = true;

}

}

if (!swapped) break;

}

}

// Step 3b: Quick Sort implementation

public static void quickSort(List<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(List<Order> orders, int low, int high) {

double pivot = orders.get(high).totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders.get(j).totalPrice < pivot) {

i++;

Collections.swap(orders, i, j);

}

}

Collections.swap(orders, i + 1, high);

return i + 1;

}

// Display all orders

public static void displayOrders(List<Order> orders) {

for (Order o : orders) {

o.display();

}

System.out.println();

}

public static void main(String[] args) {

List<Order> orders = new ArrayList<>();

orders.add(new Order("O001", "Swathi", 3458.99));

orders.add(new Order("O002", "Mani", 1390.50));

orders.add(new Order("O003", "Rishi", 6749.00));

orders.add(new Order("O004", "Prani", 8712.25));

System.out.println("Original Orders:");

displayOrders(orders);

// Bubble Sort

List<Order> bubbleSortedOrders = new ArrayList<>(orders);

bubbleSort(bubbleSortedOrders);

System.out.println("Orders Sorted by Bubble Sort:");

displayOrders(bubbleSortedOrders);

// Quick Sort

List<Order> quickSortedOrders = new ArrayList<>(orders);

quickSort(quickSortedOrders, 0, quickSortedOrders.size() - 1);

System.out.println("Orders Sorted by Quick Sort:");

displayOrders(quickSortedOrders);

// Step 4: Analysis

System.out.println("Analysis:");

System.out.println("- Bubble Sort: O(n^2) time complexity, inefficient for large datasets.");

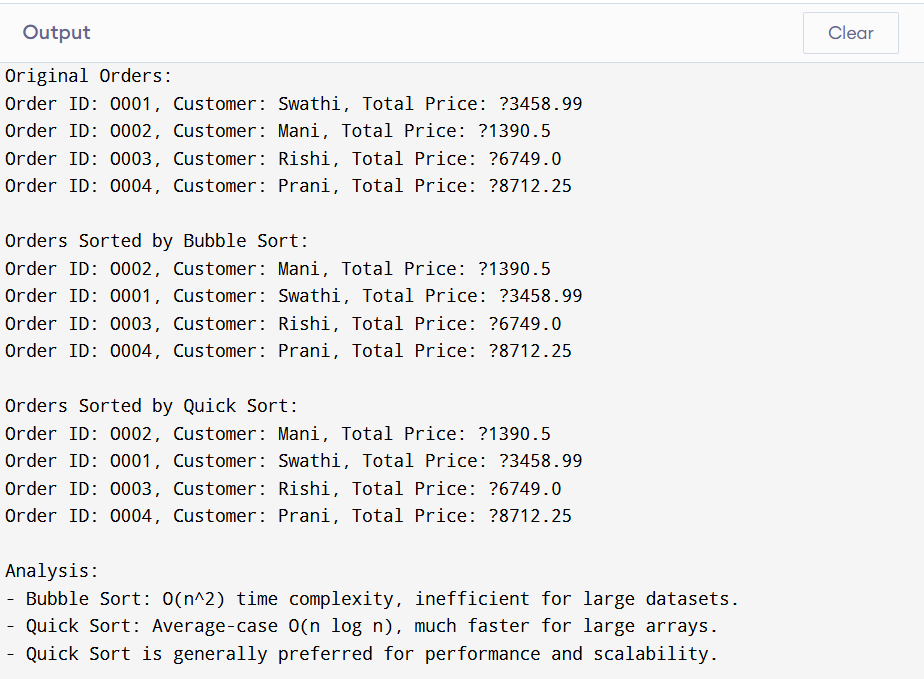
System.out.println("- Quick Sort: Average-case O(n log n), much faster for large arrays.");

System.out.println("- Quick Sort is generally preferred for performance and scalability.");

}

}

**OUTPUT**

****