**Exercise 3: Sorting Customer Orders**

**Code:**

**SortCustomerOrders.java**

import java.util.Scanner;

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 "OrderID: " + orderId + ", Customer: " + customerName + ", Total Price: ₹" + totalPrice;

}

}

public class SortCustomerOrders {

static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {

System.out.print("Enter number of orders: ");

int n = sc.nextInt();

Order[] orders = new Order[n];

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

sc.nextLine();

System.out.print("Enter Order ID: ");

int id = sc.nextInt();

sc.nextLine();

System.out.print("Enter Customer Name: ");

String name = sc.nextLine();

System.out.print("Enter Total Price: ");

double price = sc.nextDouble();

orders[i] = new Order(id, name, price);

}

System.out.println("\nOriginal Orders:");

displayOrders(orders);

Order[] bubbleSorted = orders.clone();

long startBubble = System.nanoTime();

bubbleSort(bubbleSorted);

long endBubble = System.nanoTime();

System.out.println("\nSorted by Bubble Sort (by Total Price):");

displayOrders(bubbleSorted);

System.out.println("Bubble Sort Time: " + (endBubble - startBubble) + " nanoseconds");

Order[] quickSorted = orders.clone();

long startQuick = System.nanoTime();

quickSort(quickSorted, 0, quickSorted.length - 1);

long endQuick = System.nanoTime();

System.out.println("\nSorted by Quick Sort (by Total Price):");

displayOrders(quickSorted);

System.out.println("Quick Sort Time: " + (endQuick - startQuick) + " nanoseconds");

printAnalysis();

}

static void bubbleSort(Order[] arr) {

int n = arr.length;

boolean swapped;

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

swapped = false;

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

if (arr[j].totalPrice > arr[j + 1].totalPrice) {

Order temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break;

}

}

static void quickSort(Order[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

static int partition(Order[] arr, int low, int high) {

double pivot = arr[high].totalPrice;

int i = low - 1;

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

if (arr[j].totalPrice <= pivot) {

i++;

Order temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

Order temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

static void displayOrders(Order[] arr) {

for (Order order : arr) {

System.out.println(order);

}

}

static void printAnalysis() {

System.out.println("\n--- Time Complexity Analysis ---");

System.out.println("Bubble Sort:");

System.out.println(" - Best Case: O(n)");

System.out.println(" - Worst/Average Case: O(n^2)");

System.out.println("Quick Sort:");

System.out.println(" - Best/Average Case: O(n log n)");

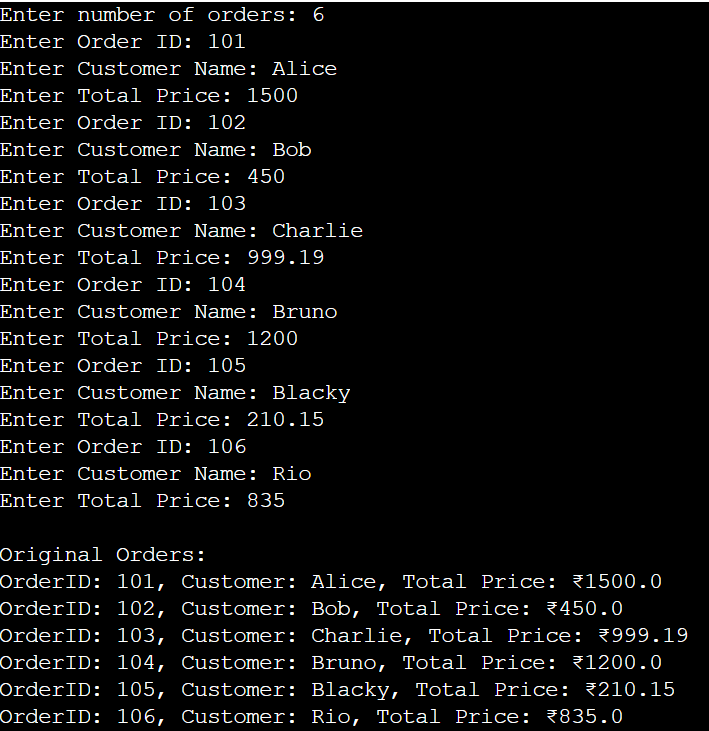
System.out.println(" - Worst Case: O(n^2)");

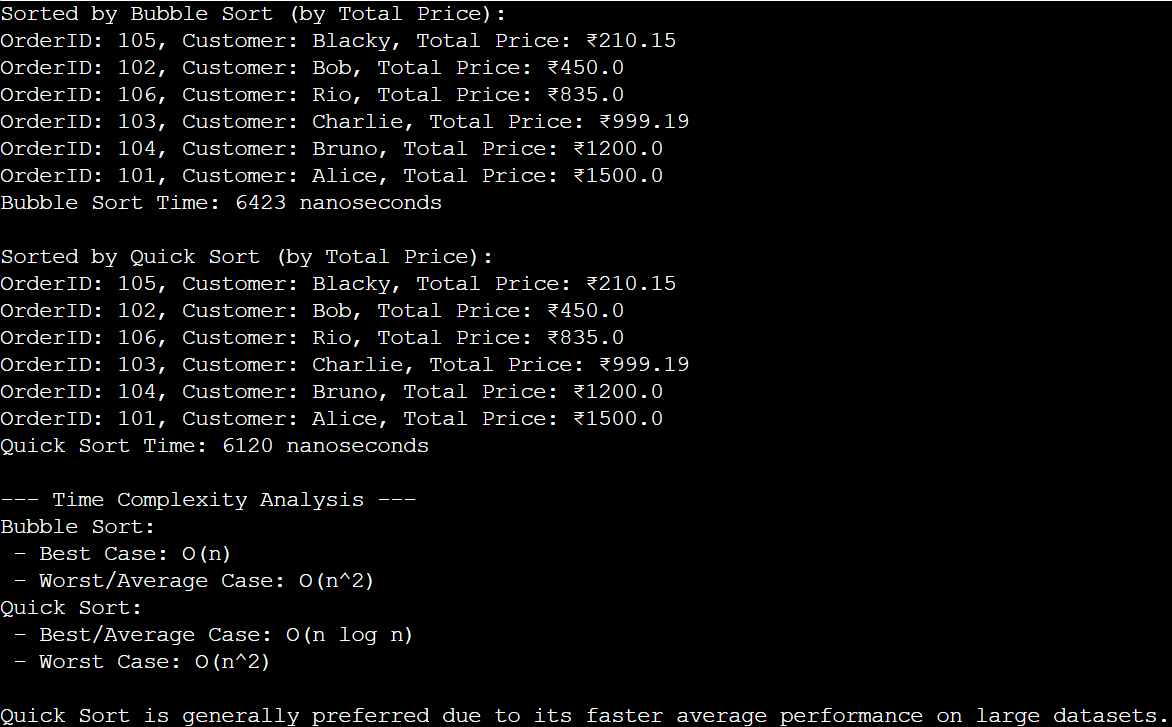
System.out.println("\nQuick Sort is generally preferred due to its faster average performance on large datasets.");

}

}

**Output:**

****

****