**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**Steps:**

1. **Understand Sorting Algorithms:**
   * Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).
2. **Setup:**
   * Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
   * Implement **Bubble Sort** to sort orders by **totalPrice**.
   * Implement **Quick Sort** to sort orders by **totalPrice**.
4. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.
   * Discuss why Quick Sort is generally preferred over Bubble Sort.

**1: Understand Sorting Algorithms**

**Bubble Sort**

* **Compares adjacent elements and swaps them if they’re in the wrong order.**
* **Repeats until no swaps are needed.**
* **Time Complexity:**
  + **Best: O(n) (already sorted)**
  + **Average/Worst: O(n²)**
* **Simple but slow for large datasets.**

**Insertion Sort**

* **Builds the sorted array one element at a time.**
* **Efficient for small or nearly sorted datasets.**
* **Time Complexity: O(n²)**

**Quick Sort**

* **Selects a *pivot*, partitions the array around it, and recursively sorts the partitions.**
* **Time Complexity:**
  + **Best/Average: O(n log n)**
  + **Worst: O(n²) (rare with good pivot choice)**
* **Efficient and commonly used in practice.**

**Merge Sort**

* **Divides array into halves, recursively sorts, then merges them.**
* **Time Complexity: O(n log n)**
* **Uses extra space for merging.**

**CODE:-**

public class CustomerOrderSorting {

    // Order class

    static 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);

        }

    }

    // Bubble Sort

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

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

            boolean swapped = false;

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

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

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                    swapped = true;

                }

            }

            if (!swapped) break; // Optimization to stop early if already sorted

        }

    }

    // Quick Sort

    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].totalPrice;

        int i = low - 1;

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

            if (orders[j].totalPrice < 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;

    }

    // Display orders

    public static void displayOrders(String message, Order[] orders) {

        System.out.println("\n" + message);

        for (Order order : orders) {

            order.display();

        }

    }

    // Main method

    public static void main(String[] args) {

        // Sample orders

        Order[] orders1 = {

            new Order("O101", "Alice", 250.75),

            new Order("O102", "Bob", 150.00),

            new Order("O103", "Charlie", 400.25),

            new Order("O104", "David", 320.50),

            new Order("O105", "Eva", 100.00)

        };

        // Create a copy for quick sort

        Order[] orders2 = new Order[orders1.length];

        for (int i = 0; i < orders1.length; i++) {

            orders2[i] = new Order(orders1[i].orderId, orders1[i].customerName, orders1[i].totalPrice);

        }

        // Display original

        displayOrders("Original Orders", orders1);

        // Bubble sort

        bubbleSort(orders1);

        displayOrders("Orders Sorted by Bubble Sort (Low to High)", orders1);

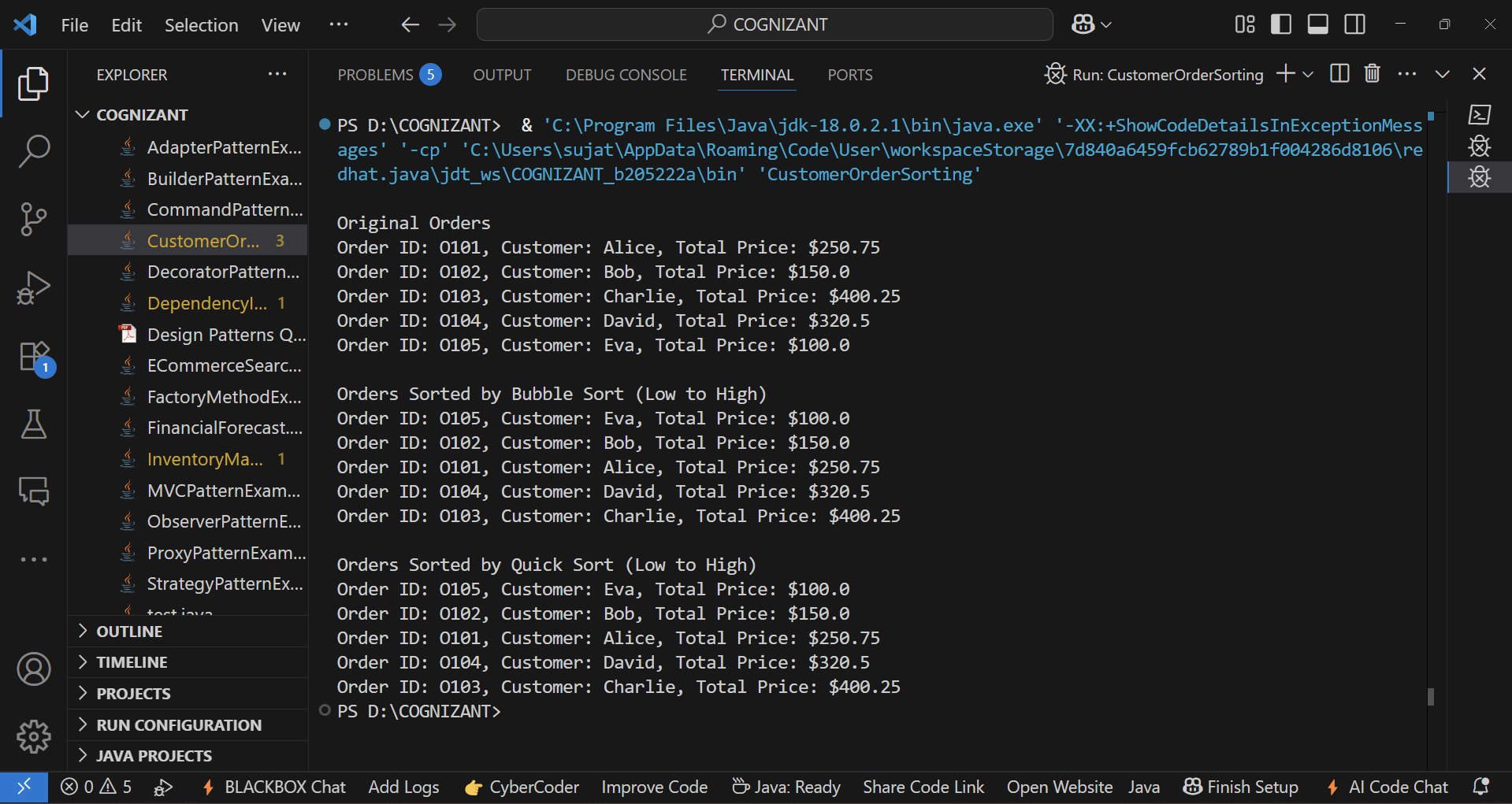
        // Quick sort

        quickSort(orders2, 0, orders2.length - 1);

        displayOrders("Orders Sorted by Quick Sort (Low to High)", orders2);

    }

}

**OUTPUT:-**

4: Analysis

| **Algorithm** | **Time Complexity (Avg)** | **Space** | **Stable** | **Notes** |
| --- | --- | --- | --- | --- |
| **Bubble Sort** | O(n²) | O(1) | Yes | Simple, slow, best for small data |
| **Quick Sort** | O(n log n) | O(log n) | No | Fast, preferred for large data |

**Why Quick Sort is Preferred:**

* Much faster on large datasets.
* Better average-case performance.
* Uses less memory than merge sort.