**Sorting Customer Orders**

**Step 1: Understanding Sorting Algorithms**

* **Bubble Sort**:
  + Compares adjacent elements and swaps if they are in the wrong order.
  + Simple but inefficient for large datasets.
  + Time Complexity:
    - Worst & Average: O(n²)
    - Best: O(n) (already sorted)
* **Insertion Sort**:
  + Builds the sorted array one item at a time by inserting each new element into the correct position.
  + Efficient for small or nearly sorted datasets.
  + Time Complexity:
    - Worst & Average: O(n²)
    - Best: O(n)
* **Quick Sort**:
  + Divide-and-conquer algorithm that selects a pivot, partitions the array, and recursively sorts subarrays.
  + Fast and commonly used in practice.
  + Time Complexity:
    - Average: O(n log n)
    - Worst: O(n²) (rare, unbalanced partition)
* **Merge Sort**:
* Recursively divides the list in half, sorts, and merges.
* Very stable and predictable performance.
* Time Complexity: O(n log n) in all cases

**Step 2: Setup 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);

}

}

**Step 3: Implementation**

**Bubble Sort**

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 BubbleSortOrders {

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

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                }

            }

        }

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order("O101", "Elsa", 2500.00),

            new Order("O102", "Anna", 9000.00),

            new Order("O103", "Meghana", 2300.00),

            new Order("O104", "Siri", 1200.00)

        };

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

        for (Order order : orders) order.display();

        bubbleSort(orders);

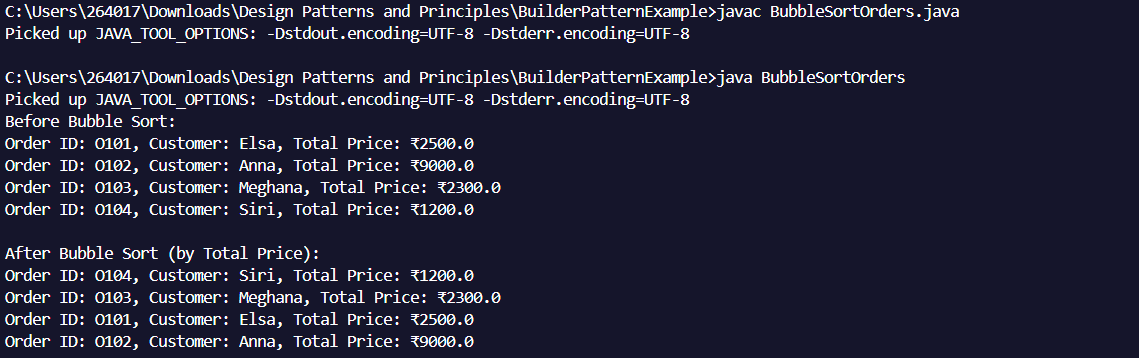
        System.out.println("\nAfter Bubble Sort (by Total Price):");

        for (Order order : orders) order.display();

    }

}

**Output:**

****

**Quick Sort:**

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 QuickSortOrders {

    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;

    }

    public static void main(String[] args) {

        Order[] orders = {

              new Order("O101", "Elsa", 2500.00),

            new Order("O102", "Anna", 9000.00),

            new Order("O103", "Meghana", 2300.00),

            new Order("O104", "Siri", 1200.00)

        };

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

        for (Order order : orders) order.display();

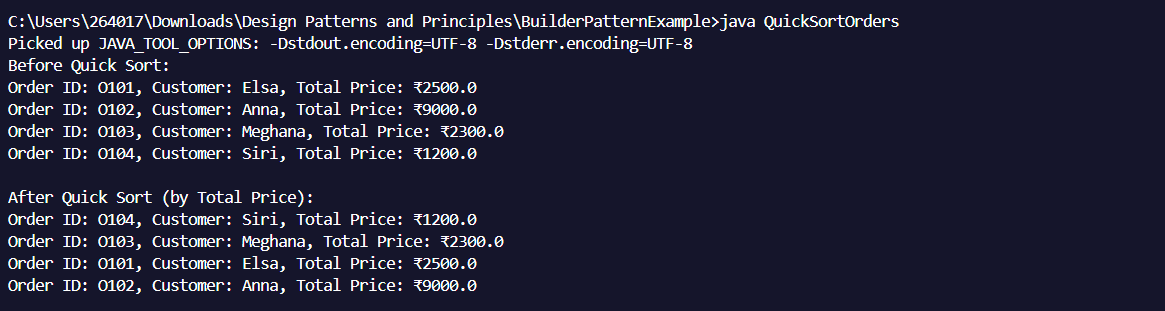
        quickSort(orders, 0, orders.length - 1);

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

        for (Order order : orders) order.display();

    }

}

**Output:**

**Step 4: Analysis**

| Algorithm | Best Case | Average Case | Worst Case |
| --- | --- | --- | --- |
| Bubble Sort | O(n) | O(n²) | O(n²) |
| Quick Sort | O(n log n) | O(n log n) | O(n²) |

* Why Quick Sort is preferred:
  + Much faster on average due to efficient partitioning.
  + Better suited for large datasets.
  + Less memory usage compared to Merge Sort.