**Module 2 - Data Structures and Algorithms**

**Exercise 3: Sorting Customer Orders**

**1). Understand Sorting Algorithms**

* **Bubble Sort**: Repeatedly compares adjacent elements and swaps them if they are in the wrong order. Simple but inefficient.
  + Time Complexity: O(n²)
* **Insertion Sort**: Builds the sorted list one item at a time by inserting elements in the right place.
  + Time Complexity: O(n²)
* **Quick Sort**: Divides the list into smaller sub-lists (pivot-based), sorts them recursively.
  + Time Complexity: Best/Average O(n log n), Worst O(n²)
* **Merge Sort**: Splits the array into halves, sorts them, and merges them.
  + Time Complexity: O(n log n)

**2,3) Setup and Implementation**

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

                    Order temp = orders[j];

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

                    orders[j + 1] = temp;

                }

            }

        }

    }

    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 printOrders(Order[] orders) {

        for (Order order : orders) {

            System.out.println(order.orderId + " " + order.customerName + " " + order.totalPrice);

        }

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order(1, "Ram", 2500.50),

            new Order(2, "Priya", 4700.00),

            new Order(3, "Ali", 1500.75)

        };

        bubbleSort(orders);

        printOrders(orders);

        System.out.println();

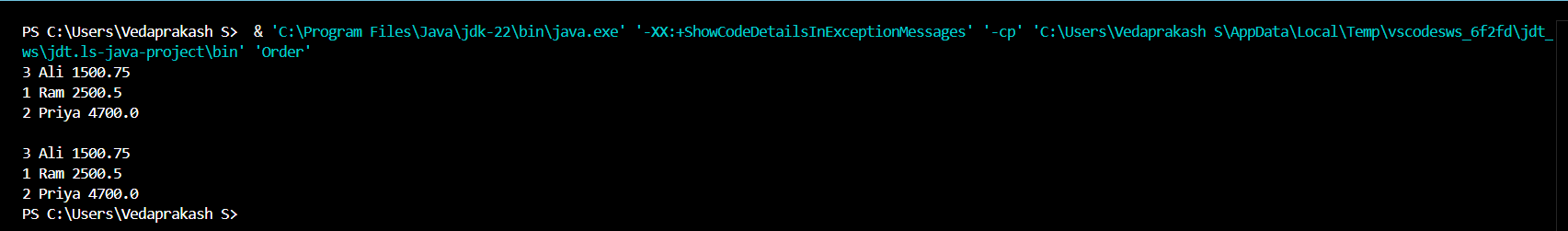
        quickSort(orders, 0, orders.length - 1);

        printOrders(orders);

    }

}

**Output:**



**4). Analysis**

* Bubble Sort is slower due to O(n²) time in most cases.
* Quick Sort is faster in average case due to its divide-and-conquer approach.
* Quick Sort is preferred for large datasets unless stability or worst-case performance is a concern.