**Exercise – 3 Sorting Customer Orders**

**Bubble Sort**

Compares adjacent elements and swaps them if they are in the wrong order. Repeats this process until the array is sorted.

Time Complexity:

Best Case – O(n)

Average/Worst Case – O(n^2)

Compares adjacent elements and swaps them if they are in the wrong order. Repeats this process until the array is sorted.

**Insertion Sort**

Builds the sorted array one element at a time by inserting each element into its correct position.

Time Complexity:

Best Case – O(n) (where the elements are nearly sorted)

Average/Worst Case – O(n^2)

**Quick Sort**

Picks a pivot element, partitions the array into two halves (less than and greater than the pivot), and recursively sorts the partitions.

Time Complexity:

Best Case – O(nlogn)

Average/Worst Case – O(n^2) (Where the pivot is poorly chosen)

**Merge Sort**

Divides the array into halves, recursively sorts them, and then merges the sorted halves.

Time Complexity:

All Cases – O(n logn)

**Comparison of performance (time complexity) of Bubble Sort and Quick Sort**

|  |  |  |
| --- | --- | --- |
| **Case** | **Bubble Sort** | **Quick Sort** |
| Best Case | O(n) | O(n logn) |
| Average Case | O(n^2) | O(n logn) |
| Worst Case | O(n^2) | O(n^2) |

**Quick Sort** is generally preferred because it offers much better performance, especially for large and unsorted datasets. **Bubble Sort**, while easy to understand, is inefficient and not suitable for practical use.

**Main.java**

public class Main{

    public static void main(String[] args) {

        //Orders to perform Bubble Sort

        Order[] orders1 = {

            new Order(101, "Alice", 2000),

            new Order(102, "Bob", 5000),

            new Order(105, "Charlie", 4000),

            new Order(104, "David", 3000),

            new Order(103, "Eva", 6000)

        };

        //Orders to perform Quick Sort

        Order[] orders2 = {

            new Order(101, "Alice", 2000),

            new Order(102, "Bob", 7000),

            new Order(105, "Eva", 4000),

            new Order(104, "David", 3000),

            new Order(103, "Charlie", 6000)

        };

        System.out.println("\nOrders before Bubble Sort\n");

        for(Order o: orders1){

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

        }

        bubbleSort(orders1);

        System.out.println("\nOrders after Bubble Sort\n");

        for(Order o: orders1){

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

        }

        System.out.println("\nOrders before Quick Sort\n");

        for(Order o: orders2){

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

        }

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

        System.out.println("\nOrders after Quick Sort\n");

        for(Order o: orders2){

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

        }

        System.out.println();

    }

    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;

                }

            }

        }

    }

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

        if(low<high){

            int idx = partition(orders, low, high);

            quickSort(orders, low, idx-1);

            quickSort(orders, idx+1, high);

        }

    }

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

        Order pivot = orders[high];

        int i = low-1;

        for(int j=low;j<high;j++){

            if(orders[j].totalPrice < pivot.totalPrice){

                i++;

                swap(orders, i, j);

            }

        }

        swap(orders, i+1, high);

        return i+1;

    }

    static void swap(Order[] orders, int i, int j){

        Order temp = orders[i];

        orders[i] = orders[j];

        orders[j] = temp;

    }

}

class Order{

    int orderId;

    String customerName;

    int totalPrice;

    Order(int orderId, String customerName, int totalPrice){

        this.orderId = orderId;

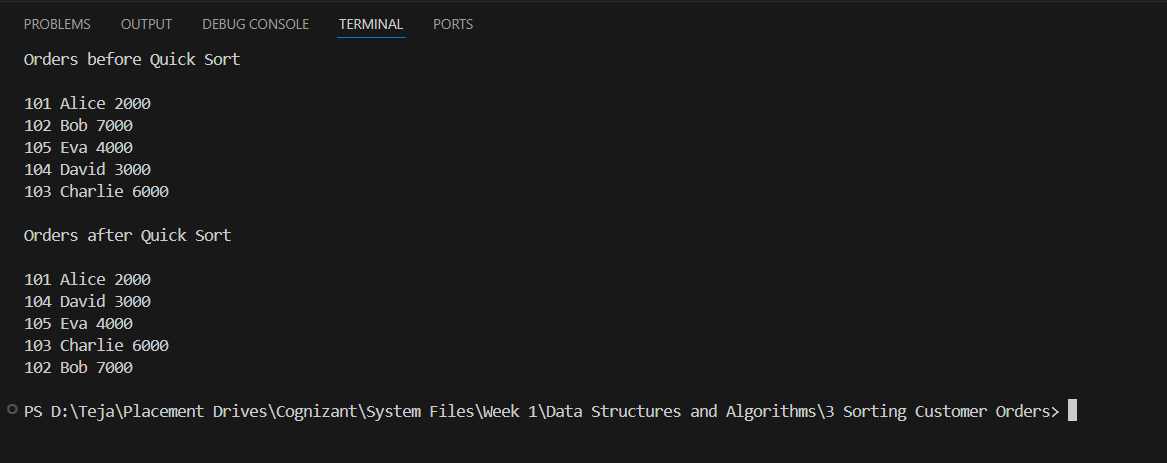
        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

}

**Output**

****