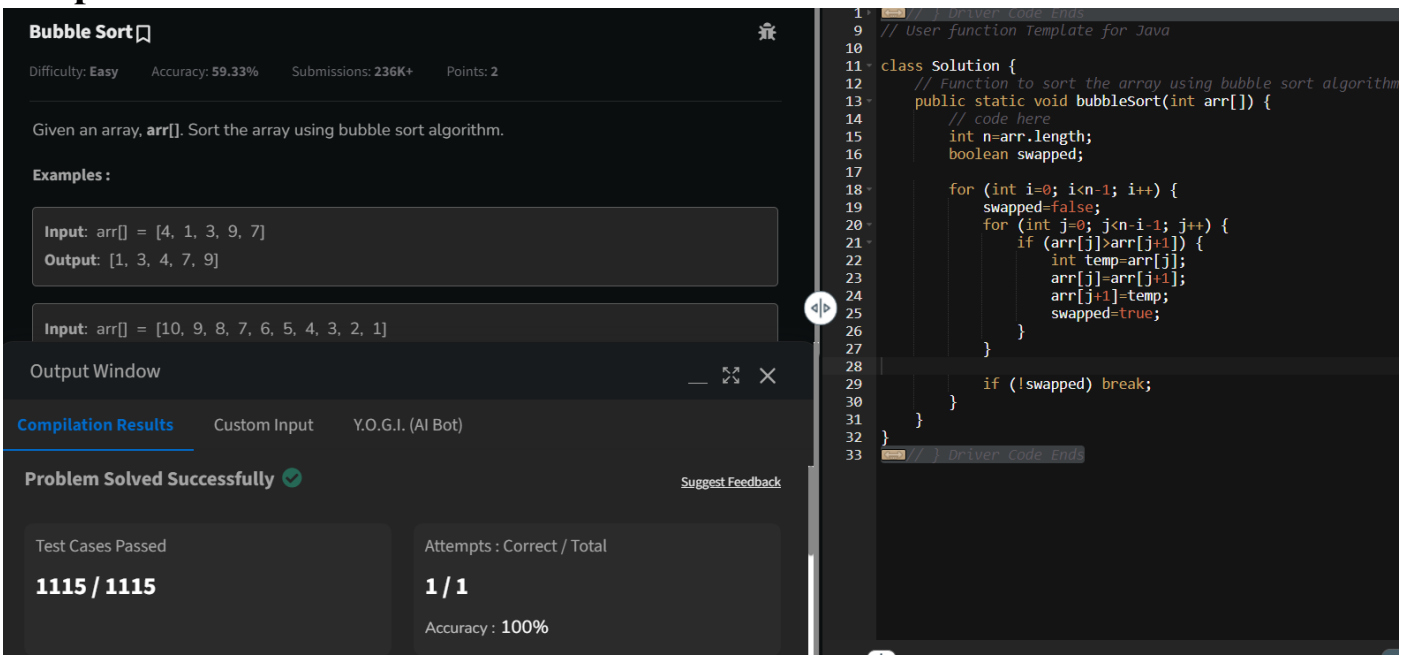
DSA PRACTICE QUESTIONS – DAY 6

Name: Jeevitha R
Reg No: 22IT040
Date: 18/11/2024

1. Bubble Sort

```
class Solution {
    public static void bubbleSort(int 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]>arr[j+1]) {
                    int temp=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                    swapped=true;
                }
            }
            if (!swapped) break;
        }
    }
}
```

Output:

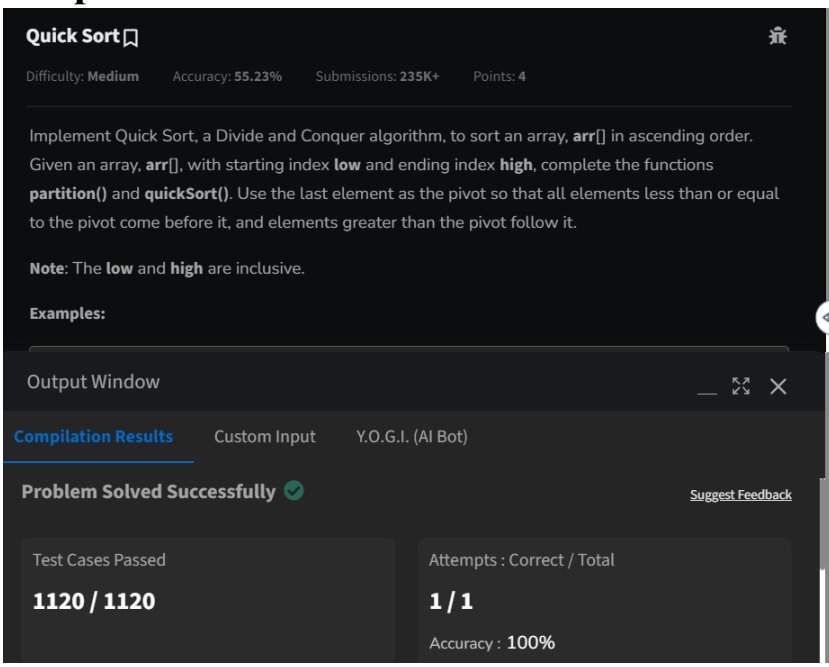


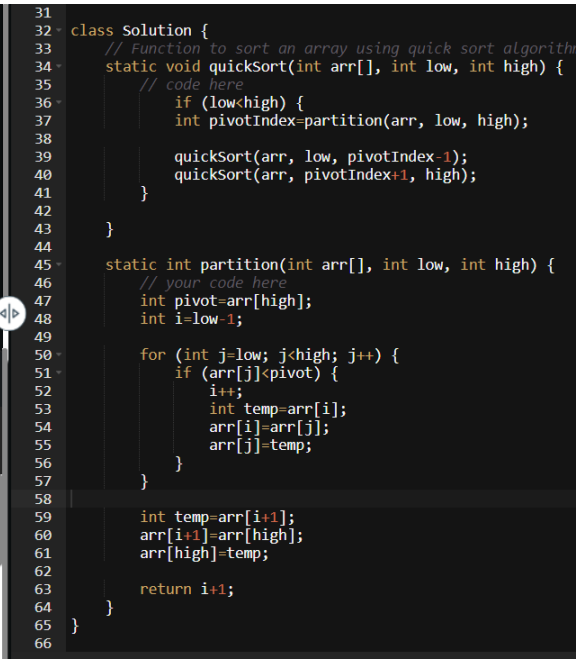
Time complexity: O (n^2)
Space Complexity: O (1)

2. Quick Sort

```
class Solution {
    static void quickSort(int arr[], int low, int high) {
        if (low<high) {
        int pivotIndex=partition(arr, low, high);
        quickSort(arr, low, pivotIndex-1);
        quickSort(arr, pivotIndex+1, high);}
    }
    static int partition(int arr[], int low, int high) {
        int pivot=arr[high];
        int i=low-1;
        for (int j=low; j<high; j++) {
            if (arr[j]<pivot) {
                i++;
                int temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;}}
        int temp=arr[i+1];
        arr[i+1]=arr[high];
        arr[high]=temp;
        return i+1;
    }
}
```

Output:



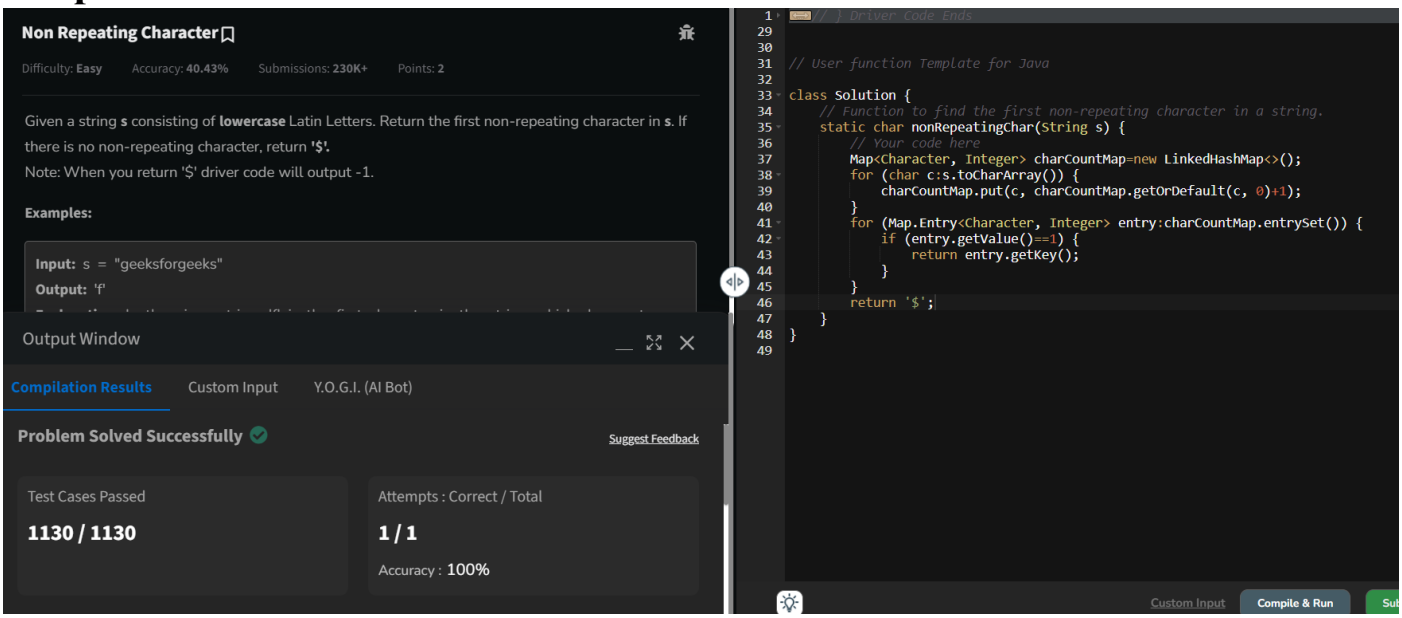


Time Complexity: O (n^2)
Space Complexity: O (logn)

3. Non-Repeating Characters

```
class Solution {
    // Function to find the first non-repeating character in a string.
    static char nonRepeatingChar(String s) {
        // Your code here
        Map<Character, Integer> map=new LinkedHashMap<>();
        for (char c:s.toCharArray()) {
            map.put(c, map.getOrDefault(c, 0)+1);
        }
        for (Map.Entry<Character, Integer> entry:map.entrySet()) {
            if (entry.getValue()==1) {
                return entry.getKey();
            }
        }
        return '$';
    }
}
```

Output:



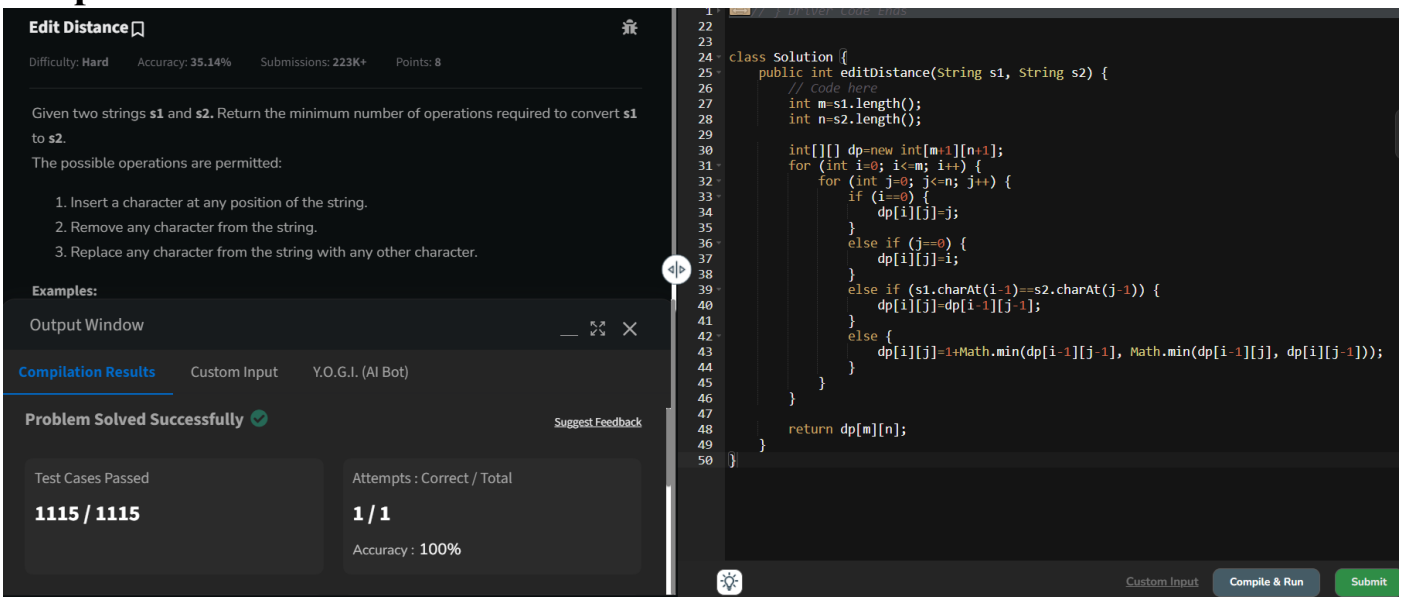
Time complexity: O (n)
Space Complexity: O (n)

4. Edit Distance

```
class Solution {
    public int editDistance(String s1, String s2) {
        int m=s1.length();
        int n=s2.length();
```

```
        int[][] dp=new int[m+1][n+1];
        for (int i=0; i<=m; i++) {
            for (int j=0; j<=n; j++) {
                if (i==0) {
                    dp[i][j]=j;
                }
                else if (j==0) {
                    dp[i][j]=i;
                }
                else if (s1.charAt(i-1)==s2.charAt(j-1)) {
                    dp[i][j]=dp[i-1][j-1];
                }
                else {
                    dp[i][j]=1+Math.min(dp[i-1][j-1], Math.min(dp[i-1][j], dp[i][j-1]));
                }
            }
        }
        return dp[m][n];
    }
}
```

Output:

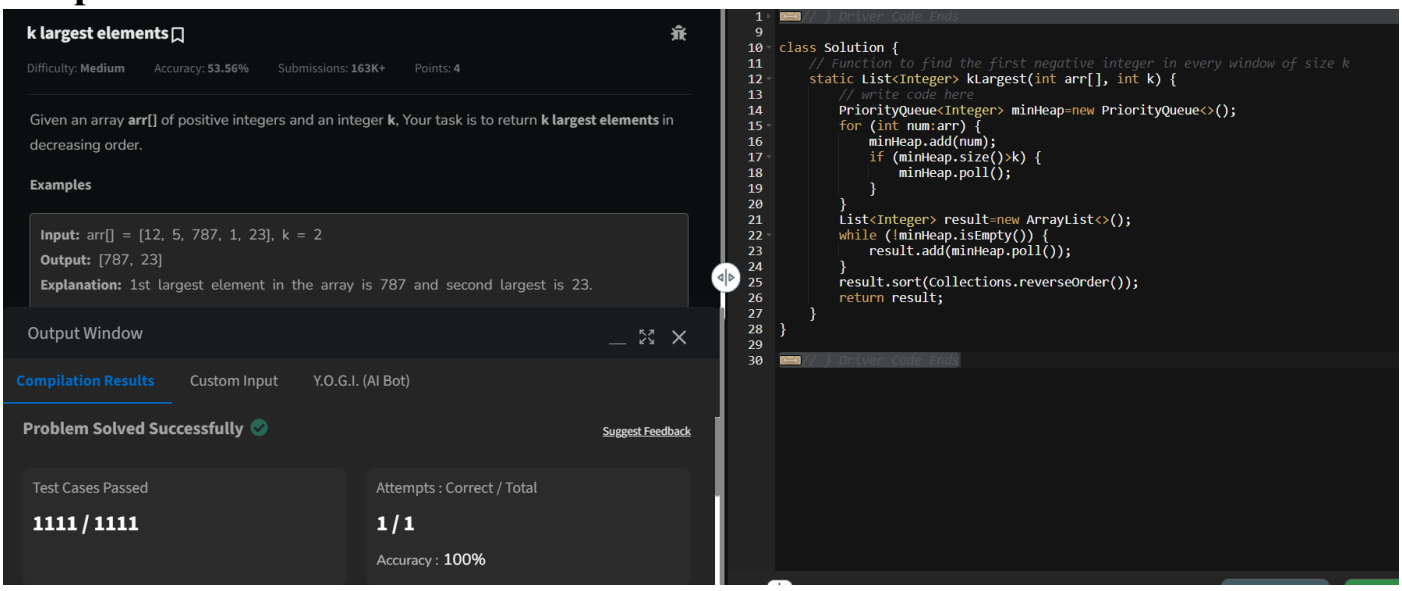


Time Complexity: O (m*n)
Space Complexity: O (m*n)

5. k Largest Element

```
class Solution {
    // Function to find the first negative integer in every window of size k
    static List<Integer> kLargest(int arr[], int k) {
        // write code here
        PriorityQueue<Integer> minHeap=new PriorityQueue<>();
        for (int num:arr) {
            minHeap.add(num);
            if (minHeap.size()>k) {
                minHeap.poll();
            }
        }
        List<Integer> result=new ArrayList<>();
        while (!minHeap.isEmpty()) {
            result.add(minHeap.poll());
        }
        result.sort(Collections.reverseOrder());
        return result;
    }
}
```

Output:



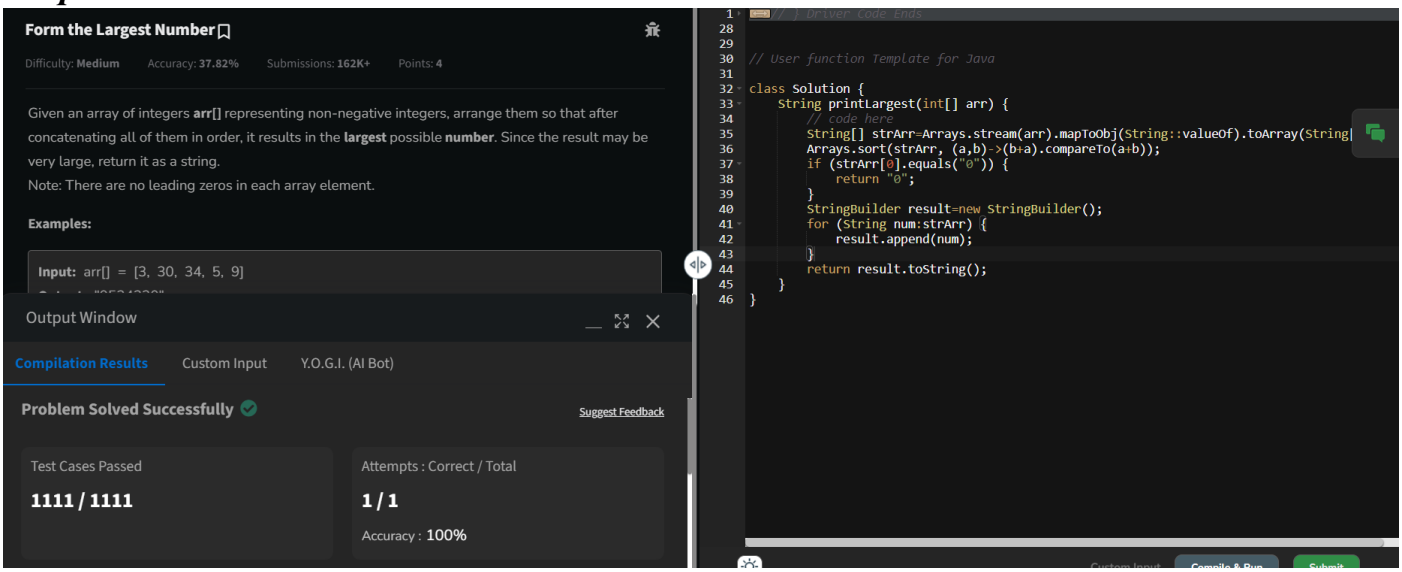
Time Complexity: O (n*logk)
Space Complexity: O (k)

6. Form the Largest Number

```
class Solution {
    String printLargest(int[] arr) {
        // code here
        String[] nums=Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new);
```

```
        Arrays.sort(nums, (a,b)->(b+a).compareTo(a+b));
        if (nums[0].equals("0")) {
            return "0";
        }
        StringBuilder result=new StringBuilder();
        for (String num:nums) {
            result.append(num);
        }
        return result.toString();
    }
}
```

Output:



Time Complexity: O (n*klogn)
Space Complexity: O (n)