

DSA PRACTICE – 6

Name:Deepak S

Register No: 22IT018

Non Repeating Character

Code:

```
class Solution {  
  
    // Function to find the first non-repeating character in a string.  
    static char nonRepeatingChar(String s) {  
  
        HashMap<Character, Integer> map = new HashMap<>();  
  
        for(char ch : s.toCharArray())  
            map.put(ch, map.getOrDefault(ch,0)+1);  
  
        for(char ch : s.toCharArray())  
        {  
            if(map.get(ch)==1)  
                return ch;  
        }  
        return '$';  
    }  
}
```

output:

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for a problem solved successfully. It indicates that all 1130 test cases passed, with 1 attempt correct out of 1 total, achieving 100% accuracy. The user scored 2 out of 2 points, and the time taken was 0.54 seconds. The 'Solve Next' section lists 'Reverse Words' and 'Longest substring with distinct characters'. The main editor on the right shows the Java code for the 'Non Repeating Character' problem, which matches the code provided in the text. The code is in Java 1.8 and includes a 'Start Timer' button. The bottom of the interface has a 'Custom Input' field and 'Compile & Run' and 'Submit' buttons.

k largest elements

output:

Class Solution {

// Function to find the first negative integer in every window of size k

static List<Integer> kLargest(int arr[], int k) {

Arrays.sort(arr);

int n = arr.length;

int[] brr = new int[k];

int i = 0;

while (k != 0) {

brr[i] = arr[n - 1];

n--;

i++;

k--;

}

List<Integer> result = new ArrayList<>();

for (int num : brr) {

result.add(num);

}

return result;}

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Problem Solved Successfully' with a green checkmark. Below this, it lists 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Your Total Score: 6'. At the bottom, it suggests 'Solve Next' problems: 'Merge k Sorted Arrays', 'Maximum Sum Combination', and 'Optimal Array'. On the right, the code editor shows the Java code for the 'k largest elements' problem, which matches the code provided in the text above. The code is in Java 1.8 and includes a 'Start Timer' button. The code is as follows:

```
1 class Solution {
2
3 // Function to find the first negative integer in every window of size k
4 static List<Integer> kLargest(int arr[], int k) {
5     Arrays.sort(arr);
6     int n = arr.length;
7     int[] brr = new int[k];
8
9     int i = 0;
10    while (k != 0) {
11        brr[i] = arr[n - 1];
12        n--;
13        i++;
14        k--;
15    }
16
17    List<Integer> result = new ArrayList<>();
18    for (int num : brr) {
19        result.add(num);
20    }
21    return result;
22 }
```

Bubble Sort

Code:

```
class Solution {  
    // Function to sort the array using bubble sort algorithm.  
    public static void bubbleSort(int arr[]) {  
  
        for(int i=0;i<arr.length;i++){  
            for(int j=0;j<arr.length-i-1;j++){  
  
                if(arr[j]>arr[j+1]){  
  
                    int temp= arr[j];  
                    arr[j]=arr[j+1];  
                    arr[j+1]=temp;  
  
                }  
  
            }  
        }  
    }  
}
```

Output:

The screenshot displays a code editor interface with a dark theme. On the left, the 'Output Window' is open, showing 'Compilation Results' for a problem titled 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. The results show 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 2 / 2', and 'Accuracy: 100%'. The 'Time Taken' is listed as '0.64'. A note at the bottom states: 'You get marks only for the first correct submission if you solve the problem without viewing the full solution.' On the right, the code editor shows the Java code for the Bubble Sort algorithm, which matches the code provided in the 'Code' section. The code is for a class 'Solution' with a static method 'bubbleSort'. The editor also shows a 'Start Timer' button and an 'Average Time: 15m' indicator. At the bottom right, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

QUITE SORT

Code:

```
class Solution {  
    // Function to sort an array using quick sort algorithm.  
    static void quickSort(int arr[], int low, int high) {  
        if(low<high){  
            int pi = partition(arr,low,high);  
            quickSort(arr,low,pi-1);  
            quickSort(arr,pi+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 - 1; j++) {  
            if (arr[j] < pivot) {  
                i++;  
                swap(arr, i, j);  
            }  
        }  
        swap(arr, i + 1, high);  
        return i + 1;  
    }  
  
    static void swap(int[] arr, int i, int j) {  
        int temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
    }  
}
```

```

}

}

```

Output:

The screenshot shows a coding platform interface. On the left, the 'Output Window' displays 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1120 / 1120', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 0 / 4', and 'Time Taken: 0.64'. A message at the bottom states: 'No marks, because you solved this problem after visiting Full solution on 2024-11-18.' On the right, the code editor shows a Java solution for the Quick Sort algorithm, including a recursive function and a partition function.

Edit Distance

CODE:

```

class Solution {
    public int editDistance(String s1, String s2) {
        // Code here
        int m=s1.length();
        int n=s2.length();
        int[][] dp=new int[m+1][n+1];

        for(int i=0;i<=m;i++)
            dp[i][0]=i;

        for(int j=0;j<=n;j++)
            dp[0][j]=j;

        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(s1.charAt(i-1)==s2.charAt(j-1))
                    dp[i][j]=dp[i-1][j-1];

                else
                    dp[i][j]=Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]))+1;
            }
        }
    }
}

```

```

    }
    return dp[m][n];
}
}

```

Output:

The screenshot shows a coding platform interface with a dark theme. On the left, the 'Output Window' is open, displaying 'Compilation Results' for a problem solved successfully. The statistics are as follows:

Test Cases Passed	Attempts : Correct / Total
1115 / 1115	1 / 1

Additional statistics shown include Accuracy: 100%, Points Scored: 8 / 8, and Time Taken: 0.16. The 'Your Total Score' is 16. Below these, there are buttons for 'Solve Next' and 'Interleaved Strings', 'Shortest Common Supersequence', and 'Form a palindrome'.

On the right, the code editor shows a Java solution for the edit distance problem. The code is as follows:

```

22
23
24 class Solution {
25     public int editDistance(String s1, String s2) {
26         // Code here
27         int m=s1.length();
28         int n=s2.length();
29         int[][] dp=new int[m+1][n+1];
30
31         for(int i=0;i<=m;i++)
32             dp[i][0]=i;
33
34         for(int j=0;j<=n;j++)
35             dp[0][j]=j;
36
37         for(int i=1;i<=m;i++){
38             for(int j=1;j<=n;j++){
39                 if(s1.charAt(i-1)==s2.charAt(j-1))
40                     dp[i][j]=dp[i-1][j-1];
41                 else
42                     dp[i][j]=Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]))
43             }
44         }
45         return dp[m][n];
46     }
47 }
48
49
50

```