

DSA PRACTICE – DAY 6

Name: Dhejan R

Reg No: 22IT022

Date: 18/11/2024

1. Bubble Sort

Code Solution:

```
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:

The screenshot displays a coding platform interface for the 'Bubble Sort' problem. On the left, the problem description states: 'Given an array, arr[]. Sort the array using bubble sort algorithm.' It provides two examples: Input: arr[] = [4, 1, 3, 9, 7] and Output: [1, 3, 4, 7, 9]; and Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]. The 'Output Window' shows 'Problem Solved Successfully' with a green checkmark. Below this, it indicates 'Test Cases Passed: 1115 / 1115' and 'Attempts: Correct / Total: 1 / 1' with an 'Accuracy: 100%'.

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

```
1 // Driver Code Ends
2 // User function Template for Java
3
4 class Solution {
5     // Function to sort the array using bubble sort algorithm
6     public static void bubbleSort(int arr[]) {
7         // code here
8         int n=arr.length;
9         boolean swapped;
10
11         for (int i=0; i<n-1; i++) {
12             swapped=false;
13             for (int j=0; j<n-i-1; j++) {
14                 if (arr[j]>arr[j+1]) {
15                     int temp=arr[j];
16                     arr[j]=arr[j+1];
17                     arr[j+1]=temp;
18                     swapped=true;
19                 }
20             }
21             if (!swapped) break;
22         }
23     }
24 }
```

Time complexity: $O(n^2)$

Space Complexity: $O(1)$

2. Quick Sort

Code Solution:

```
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:

The screenshot displays a coding platform interface for the 'Quick Sort' problem. On the left, the problem description states: 'Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, `arr[]` in ascending order. Given an array, `arr[]`, with starting index `low` and ending index `high`, complete the functions `partition()` and `quickSort()`. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it. Note: The `low` and `high` are inclusive. Examples:'. Below this, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)', indicating 'Problem Solved Successfully' with '1120 / 1120' test cases passed, '1 / 1' attempts, and '100%' accuracy.

On the right, the code editor shows the implementation of the Quick Sort algorithm, matching the code provided in the 'Code Solution' block. The code is written in C++ and includes comments like '// Function to sort an array using quick sort algorithm' and '// your code here'.

Time Complexity: $O(n^2)$

Space Complexity: $O(\log n)$

3. Non-Repeating Characters

Code Solution:

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

Output:

The screenshot displays a coding platform interface. On the left, the problem 'Non Repeating Character' is shown with its difficulty (Easy), accuracy (40.43%), and submission statistics (230K+). The problem description states: 'Given a string s consisting of lowercase Latin Letters. Return the first non-repeating character in s. If there is no non-repeating character, return '\$'. Note: When you return '\$' driver code will output -1.' Examples provided are: Input: s = "geeksforgeeks", Output: 'f'. Below this, the 'Output Window' shows 'Compilation Results' and 'Problem Solved Successfully' with a green checkmark. It also displays 'Test Cases Passed: 1130 / 1130' and 'Attempts: Correct / Total: 1 / 1' with an 'Accuracy: 100%'.

On the right, the code editor shows the solution code in Java, which is identical to the code provided in the 'Code Solution' block. The code is numbered from 1 to 49. The driver code ends at line 1, and the user function template for Java is shown from line 31 to 49. The solution code is inserted between lines 35 and 48.

Time complexity: $O(n)$

Space Complexity: $O(n)$

4. Edit Distance

Code Solution:

```
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:

The screenshot displays the LeetCode interface for the 'Edit Distance' problem. On the left, the problem details are visible: 'Edit Distance' with a difficulty of 'Hard', an accuracy of 35.14%, and 223K+ submissions. The problem description states: 'Given two strings s1 and s2. Return the minimum number of operations required to convert s1 to s2. The possible operations are permitted: 1. Insert a character at any position of the string. 2. Remove any character from the string. 3. Replace any character from the string with any other character.' Below this, it says 'Examples:' and 'Output Window'. The 'Compilation Results' section shows 'Problem Solved Successfully' with a green checkmark. At the bottom, it indicates 'Test Cases Passed: 1115 / 1115' and 'Attempts: Correct / Total: 1 / 1' with an 'Accuracy: 100%'.

On the right, the Java code solution is shown. It is a class named 'Solution' with a method 'editDistance' that takes two strings 's1' and 's2' as input. The code uses a 2D array 'dp' of size (m+1) x (n+1) to store the edit distance between substrings. It iterates over each character in both strings and calculates the minimum number of operations (insert, delete, or replace) required to convert the substring of 's1' to the substring of 's2'. The final result is stored in 'dp[m][n]' and returned.

Time Complexity: $O(m*n)$

Space Complexity: $O(m*n)$

5. k Largest Element

Code Solution:

```
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:

The screenshot displays a coding problem interface. On the left, the problem title is 'k largest elements' with a difficulty of 'Medium', an accuracy of '53.56%', and 163K+ submissions. The problem description states: 'Given an array arr[] of positive integers and an integer k. Your task is to return k largest elements in decreasing order.' An example is provided: 'Input: arr[] = [12, 5, 787, 1, 23], k = 2' and 'Output: [787, 23]'. The explanation says: '1st largest element in the array is 787 and second largest is 23.' The 'Output Window' is empty. Below, the 'Compilation Results' section shows 'Problem Solved Successfully' with a green checkmark. A table indicates 'Test Cases Passed: 1111 / 1111' and 'Attempts: Correct / Total: 1 / 1' with an 'Accuracy: 100%'. On the right, a code editor shows the Java solution for the problem, which matches the code provided in the 'Code Solution' block.

Time Complexity: $O(n \cdot \log k)$

Space Complexity: $O(k)$

6. Form the Largest Number

Code Solution:

```
class Solution {
    String printLargest(int[] arr) {
        // code here
        String[]
strArr=Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new);
        Arrays.sort(strArr, (a,b)->(b+a).compareTo(a+b));
        if (strArr[0].equals("0")) {
            return "0";
        }
        StringBuilder result=new StringBuilder();
        for (String num:strArr) {
            result.append(num);
        }
        return result.toString();
    }
}
```

Output:

Form the Largest Number

Difficulty: Medium Accuracy: 37.82% Submissions: 162K+ Points: 4

Given an array of integers `arr[]` representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the **largest** possible number. Since the result may be very large, return it as a string.

Note: There are no leading zeros in each array element.

Examples:

Input: `arr[] = [3, 30, 34, 5, 9]`

Output: `3459303`

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully [Suggest Feedback](#)

Test Cases Passed: **1111 / 1111**

Attempts: Correct / Total: **1 / 1**

Accuracy: 100%

```
1 // User function Template for Java
28
29
30
31
32 class Solution {
33     String printLargest(int[] arr) {
34         // code here
35         String[] strArr=Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new);
36         Arrays.sort(strArr, (a,b)->(b+a).compareTo(a+b));
37         if (strArr[0].equals("0")) {
38             return "0";
39         }
40         StringBuilder result=new StringBuilder();
41         for (String num:strArr) {
42             result.append(num);
43         }
44         return result.toString();
45     }
46 }
```

Time Complexity: $O(n * k \log n)$

Space Complexity: $O(n)$