

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:

The screenshot displays a coding platform interface for the 'Bubble Sort' problem. The problem description states: 'Given an array, arr[]. Sort the array using bubble sort algorithm.' Examples provided are: 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 'Compilation Results' section shows 'Problem Solved Successfully' with '1115 / 1115' test cases passed, '1 / 1' attempts correct, and '100%' accuracy. The code editor on the right shows the Java implementation of the bubble sort algorithm, which matches the code provided in the previous block.

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:

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' is visible. The 'Compilation Results' section shows 'Problem Solved Successfully' with a green checkmark. The 'Test Cases Passed' section shows '1120 / 1120'. The 'Attempts' section shows '1 / 1' and 'Accuracy: 100%'. On the right, the code editor shows the solution code, which matches the code provided in the first 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

```
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:

Non Repeating Character

Difficulty: Easy Accuracy: 40.43% Submissions: 230K+ Points: 2

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:

Input: s = "geeksforgeeks"
Output: 'f'

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed: **1130 / 1130**

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

Accuracy: 100%

```
1 // Driver Code Ends
29
30
31 // User function Template for Java
32
33 class Solution {
34     // Function to find the first non-repeating character in a string.
35     static char nonRepeatingChar(String s) {
36         // Your code here
37         Map<Character, Integer> charCountMap=new LinkedHashMap<>();
38         for (char c:s.toCharArray()) {
39             charCountMap.put(c, charCountMap.getOrDefault(c, 0)+1);
40         }
41         for (Map.Entry<Character, Integer> entry:charCountMap.entrySet()) {
42             if (entry.getValue()==1) {
43                 return entry.getKey();
44             }
45         }
46         return '$';
47     }
48 }
49
```

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:

Edit Distance

Difficulty: Hard Accuracy: 35.14% Submissions: 223K+ Points: 8

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.

Examples:

Output Window

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

Problem Solved Successfully [Suggest Feedback](#)

Test Cases Passed	Attempts : Correct / Total
1115 / 1115	1 / 1
	Accuracy : 100%

```

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++) {
            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];
    }
}

```

Custom Input **Compile & Run** **Submit**

Time Complexity: $O(m \cdot n)$

Space Complexity: $O(m \cdot n)$

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:

The screenshot displays a coding platform interface. On the left, the problem 'k largest elements' is shown with a difficulty of Medium, 53.56% accuracy, 163K+ submissions, and 4 points. 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 'Output Window' shows 'Compilation Results' as 'Problem Solved Successfully' with 1111/1111 test cases passed, 1/1 attempts correct, and 100% accuracy. On the right, the code editor shows the Java solution for the problem, which uses a min-heap to maintain the k largest elements.

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

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:

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	Attempts : Correct / Total
1111 / 1111	1 / 1
	Accuracy : 100%

```

1 // Driver Code starts
28
29
30 // User function Template for Java
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: $O(n \cdot k \log n)$

Space Complexity: $O(n)$