

DSA PRACTICE QUESTIONS – DAY 6

Name: Jeevitha R

Reg No: 22IT040

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. 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]; 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. On the right, the Java code for the bubble sort algorithm is shown, matching the code provided in the 'Code Solution' block.

Bubble Sort

Difficulty: Easy Accuracy: 59.33% Submissions: 236K+ Points: 2

Given an array, `arr[]`. Sort the array using bubble sort algorithm.

Examples :

Input: `arr[] = [4, 1, 3, 9, 7]`
Output: `[1, 3, 4, 7, 9]`

Input: `arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1

Accuracy : 100%

```
1 // User function Template for Java
9
10
11 class Solution {
12     // Function to sort the array using bubble sort algorithm
13     public static void bubbleSort(int arr[]) {
14         // code here
15         int n=arr.length;
16         boolean swapped;
17
18         for (int i=0; i<n-1; i++) {
19             swapped=false;
20             for (int j=0; j<n-i-1; j++) {
21                 if (arr[j]>arr[j+1]) {
22                     int temp=arr[j];
23                     arr[j]=arr[j+1];
24                     arr[j+1]=temp;
25                     swapped=true;
26                 }
27             }
28             if (!swapped) break;
29         }
30     }
31 }
32
33 // Driver Code Ends
```

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. 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: [empty box]'. The 'Compilation Results' section shows 'Problem Solved Successfully' with a green checkmark. Below this, it indicates 'Test Cases Passed: 1120 / 1120' and 'Attempts: Correct / Total: 1 / 1'. The 'Accuracy' is listed as '100%'. On the right side, the code solution is shown in a dark-themed editor, matching the code provided in the previous block. The code is for a C++ class `Solution` with static methods `quickSort` and `partition`.

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

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 shows the LeetCode interface for the 'Edit Distance' problem. On the left, 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 'Problem Solved Successfully' with a green checkmark and shows '1115 / 1115' test cases passed and '1 / 1' attempts correct with 100% accuracy. On the right, a code editor displays the Java solution for the edit distance problem, which is identical to the code provided in the previous block. The editor includes line numbers from 22 to 50 and buttons for 'Compile & Run' and 'Submit'.

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 platform interface for the problem "k largest elements". On the left, 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." Examples provided are: Input: arr[] = [12, 5, 787, 1, 23], k = 2; Output: [787, 23]; Explanation: 1st largest element in the array is 787 and second largest is 23. The "Compilation Results" section shows "Problem Solved Successfully" with 1111/1111 test cases passed and 100% accuracy. On the right, a code editor shows the Java solution using a min-heap to maintain the k largest elements.

```
1  // Driver Code Starts
9
10 class Solution {
11     // Function to find the first negative integer in every window of size k
12     static List<Integer> kLargest(int arr[], int k) {
13         // write code here
14         PriorityQueue<Integer> minHeap=new PriorityQueue<>();
15         for (int num:arr) {
16             minHeap.add(num);
17             if (minHeap.size()>k) {
18                 minHeap.poll();
19             }
20         }
21         List<Integer> result=new ArrayList<>();
22         while (!minHeap.isEmpty()) {
23             result.add(minHeap.poll());
24         }
25         result.sort(Collections.reverseOrder());
26         return result;
27     }
28 }
29 // Driver Code Ends
30
```

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[]
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:

The screenshot displays a coding problem titled "Form the Largest Number" with a difficulty of Medium. The problem description states: "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 provided show an input array `[3, 30, 34, 5, 9]` resulting in the output string `349303`.

The solution is implemented in Java within a class `Solution`. The method `printLargest(int[] arr)` converts the integer array to a `String[]` using `Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new)`. It then sorts the strings using a custom comparator `(a,b) -> (b+a).compareTo(a+b)`. A check for leading zeros is performed: if the first element is "0", it returns "0". Otherwise, it uses a `StringBuilder` to concatenate all sorted strings and returns the final result.

The output window shows "Compilation Results" with a green checkmark indicating "Problem Solved Successfully". The test cases passed are 1111 / 1111, with 1 / 1 attempts correct and 100% accuracy.

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

Space Complexity: $O(n)$