

# DSA PRACTICE QUESTIONS

13/11/2024

## 1. Kth Smallest element: TC: $O(N)$

The screenshot displays the 'Compilation Results' for the 'Kth Smallest element' problem. The status is 'Problem Solved Successfully' with a green checkmark. The test cases passed are 1110 / 1110. The attempts are 3 / 3, and the accuracy is 100%. The time taken is 0.25. The code is written in Java and uses a PriorityQueue to solve the problem.

```
43 // User function template for Java
44
45 class Solution {
46     public static int kthSmallest(int[] arr, int k) {
47         PriorityQueue<Integer> queue = new PriorityQueue<>();
48         for(int i : arr){
49             queue.add(i);
50         }
51         while(k-1>0){
52             queue.remove();
53             k--;
54         }
55         return queue.remove();
56     }
57 }
58
59
```

## 2. PARENTHESES CHECKER: TC: $O(N)$

The screenshot displays the 'Compilation Results' for the 'PARENTHESES CHECKER' problem. The status is 'Problem Solved Successfully' with a green checkmark. The test cases passed are 1111 / 1111. The attempts are 2 / 2, and the accuracy is 100%. The code is written in Python and uses a stack to check if the parentheses are balanced.

```
1 // User function template for Python
2
3 class Solution:
4     def isParenthesisBalanced(self, s):
5         stk = []
6         for ch in s:
7             if ch == '(':
8                 stk.append('(')
9             elif ch == '{':
10                 stk.append('{')
11             elif ch == '[':
12                 stk.append '[')
13             else:
14                 if not stk or stk.pop() != ch:
15                     return False
16         return not stk
17
18
19
```

## 3. EQUILIBRIUM POINT: TC: $O(N)$

The screenshot displays the 'Compilation Results' for the 'EQUILIBRIUM POINT' problem. The status is 'Problem Solved Successfully' with a green checkmark. The test cases passed are 1111 / 1111. The attempts are 2 / 2, and the accuracy is 100%. The code is written in Java and finds the equilibrium point by calculating the total sum and then iterating through the array to find the point where the left sum equals the total sum minus the current element.

```
40
41 class Solution {
42     public static int equilibriumPoint(int arr[]) {
43         int totalSum = 0;
44         for (int num : arr) totalSum += num;
45
46         int leftSum = 0;
47         for (int i = 0; i < arr.length; i++) {
48             totalSum -= arr[i];
49             if (leftSum == totalSum) return i + 1;
50             leftSum += arr[i];
51         }
52         return -1;
53     }
54 }
55
56
```

## 4. NEXT GREATER ELEMENT:

Output Window

Compilation Results

Custom Input

Y.O.G.I. (AI Bot)

Problem Solved Successfully

Test Cases Passed

1110 / 1110

Attempts: Correct / Total

2 / 3

Accuracy: 66%

Time Taken

1.68

1 // Driver Code Ends

36

37 class Solution {

38 public ArrayList<Integer> nextLargerElement(int[] arr) {

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

40 Stack<Integer> stack = new Stack<>();

41

42 for (int i = arr.length - 1; i >= 0; i--) {

43 while (!stack.isEmpty() && stack.peek() <= arr[i]) {

44 stack.pop();

45

46 if (stack.isEmpty()) {

47 result.add(0, -1);

48 } else {

49 result.add(0, stack.peek());

50

51 stack.push(arr[i]);

52

53 }

54

55 return result;

56 }

57 }

## 5. BINARY SEARCH: TC: $O(N \log N)$

Binary Search

Difficulty: Easy

Accuracy: 44.32%

Submissions: 530K+

Points: 2

Given a sorted array `arr` and an integer `k`, find the position(0-based indexing) at which `k` is present in the array using binary search.

Note: If multiple occurrences are there, please return the smallest index.

**Examples:**

**Input:** `arr[] = [1, 2, 3, 4, 5]`, `k = 4`

**Output:** 3

**Explanation:** 4 appears at index 3.

**Input:** `arr[] = [11, 22, 33, 44, 55]`, `k = 445`

**Output:** -1

**Explanation:** 445 is not present.

*Note: Try to solve this problem in constant space i.e  $O(1)$*

**Constraints:**

- $1 \leq \text{arr.size}() \leq 10^5$
- $1 \leq \text{arr}[i] \leq 10^6$
- $1 \leq k \leq 10^6$

1 // Driver Code Ends

32

33

34 // User function Template for Java

35

36 class Solution {

37 public int binarysearch(int[] nums, int k) {

38 int n = nums.length;

39 int low = 0;

40 int high = n - 1;

41 while (low <= high) {

42 int mid = (low + high) / 2;

43 if (nums[mid] == k) {

44 return mid;

45

46 } else if (nums[mid] > k) {

47 high = mid - 1;

48

49 } else {

50 low = mid + 1;

51

52 }

53 return -1;

54 }

55 }