



Experiment-2.3

Student Name: Himanshu

UID: 20BCS7944

Branch: CSE

Section: 905/A

Semester: 6

Date of Performance: 30/03/2023

Subject Name: Competitive Coding-II

Subject Code: 20CSP-351

1. Aim:

To demonstrate the concept of Divide and Conqueror algorithm.

2. Objective:

- The objective is to build problem solving capability and to learn the basic concepts of divide and conqueror algorithm.
- The implementation of Median of two sorted arrays which shows and brushes up the concept of Tree.
- The implementation of Maximum subarray.

3. LeetCode code and output:

- Median of two sorted arrays -

```
class Solution {  
    public boolean hasPathSum(TreeNode root, int sum) {  
        return root == null ? false : DFS(root, 0, sum);  
    }  
    public boolean DFS(TreeNode node, int tmp, int sum){  
        boolean flag = false;  
        tmp += node.val;
```

```

if(node.left == null && node.right == null){
    return tmp == sum;
}
if(node.left != null){
    flag = flag || DFS(node.left, tmp, sum);
}
if(!flag && node.right != null){
    flag = flag || DFS(node.right, tmp, sum);
}

return flag;
}
}

```

OUTPUT:

4. Median of Two Sorted Arrays

Hard 22.9K 2.6K

Companies

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays.

The overall run time complexity should be $O(\log(m+n))$.

Example 1:

Input: `nums1 = [1,3], nums2 = [2]`
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: `nums1 = [1,2], nums2 = [3,4]`
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is $(2 + 3) / 2 = 2.5$.

```

2 public double findMedianSortedArrays(int[] nums1, int[] nums2) {
3     int n1 = nums1.length;
4     int n2 = nums2.length;
5     int n = n1 + n2;
6     int[] new_arr = new int[n];
7
8     int i=0, j=0, k=0;
9
10    while (i<=n1 && j<=n2) {
11        if (i == n1) {
12            while(j<n2) new_arr[k++] = nums2[j++];
13            break;
14        } else if (j == n2) {
15            while (i<n1) new_arr[k++] = nums1[i++];
16            break;
17        }
18        if (nums1[i] <= nums2[j]) new_arr[k++] = nums1[i++];
19        else new_arr[k++] = nums2[j++];
20    }
21    return (n%2==0) ? (new_arr[k-1]+new_arr[k])/2.0 : new_arr[k-1];
22 }

```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums1 =

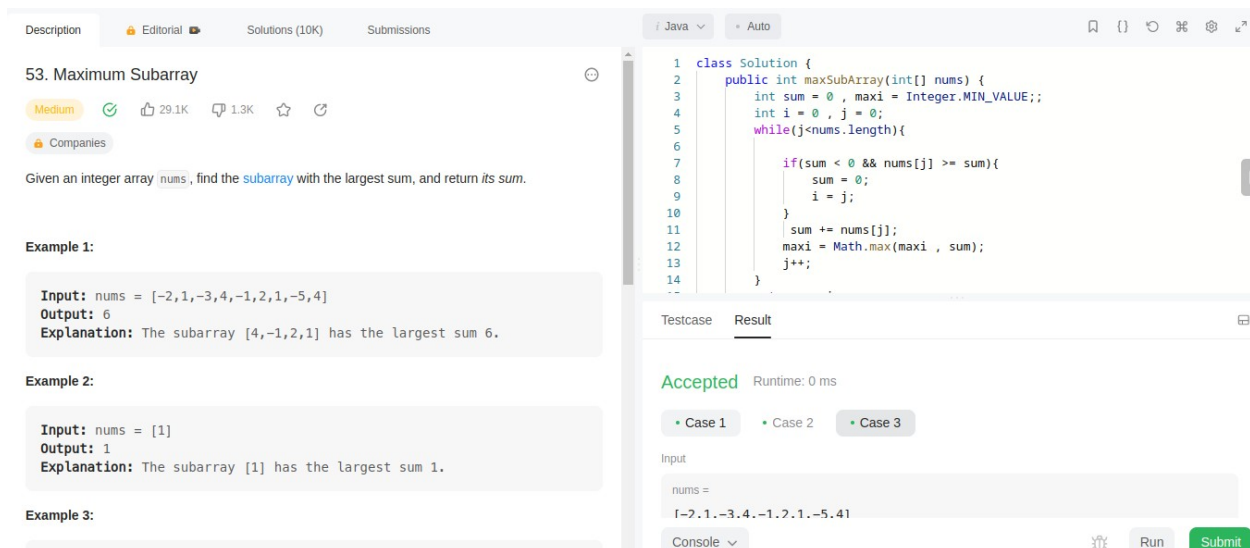
[1,3]

Console Run Submit

- Median of two sorted arrays -

```
class Solution {
    public int maxSubArray(int[] nums) {
        int sum = 0 , maxi = Integer.MIN_VALUE;
        int i = 0 , j = 0; while(j < nums.length){
            if(sum < 0 && nums[j] >= sum){
                sum = 0;
                i = j;
            }
            sum += nums[j];
            maxi = Math.max(maxi , sum);
            j++;
        }
        return maxi;
    }
}
```

OUTPUT:



The screenshot displays a coding interface for the 'Maximum Subarray' problem (LeetCode 53). On the left, the problem description states: 'Given an integer array `nums`, find the subarray with the largest sum, and return its sum.' It includes three examples: Example 1 with input `[-2,1,-3,4,-1,2,1,-5,4]` and output 6; Example 2 with input `[1]` and output 1; and Example 3 with input `[-2,1,-3,4,-1,2,1,-5,4]` and output 1. The right side shows the Java code for the solution, which implements Kadane's algorithm. The code is as follows:

```
1 class Solution {
2     public int maxSubArray(int[] nums) {
3         int sum = 0 , maxi = Integer.MIN_VALUE;;
4         int i = 0 , j = 0;
5         while(j < nums.length){
6
7             if(sum < 0 && nums[j] >= sum){
8                 sum = 0;
9                 i = j;
10            }
11            sum += nums[j];
12            maxi = Math.max(maxi , sum);
13            j++;
14        }
15    }
16 }
```

Below the code, the 'Testcase' tab shows the result 'Accepted' with a runtime of 0 ms. The 'Input' field contains the array `[-2,1,-3,4,-1,2,1,-5,4]`. At the bottom, there are buttons for 'Console', 'Run', and 'Submit'.