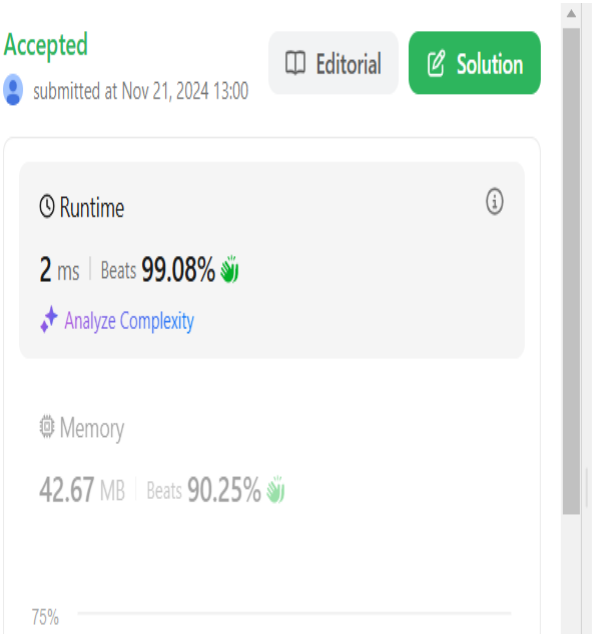
DSA QUESTIONS PRACTICE

21/11/24

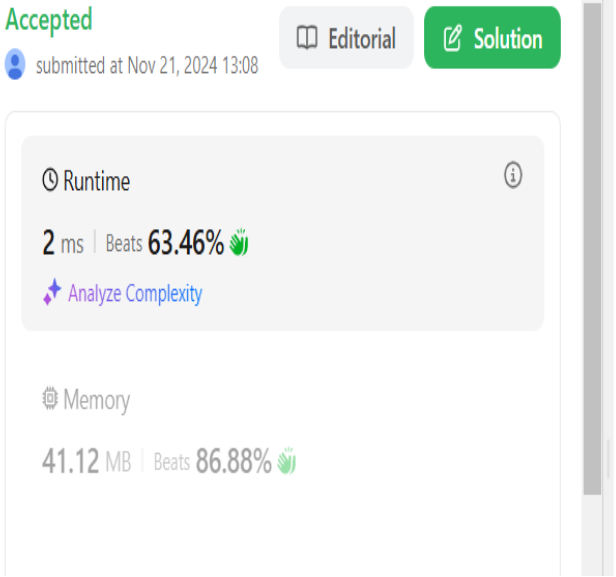
TWO POINTERS:

1.VALID PALINDROME: O(N)



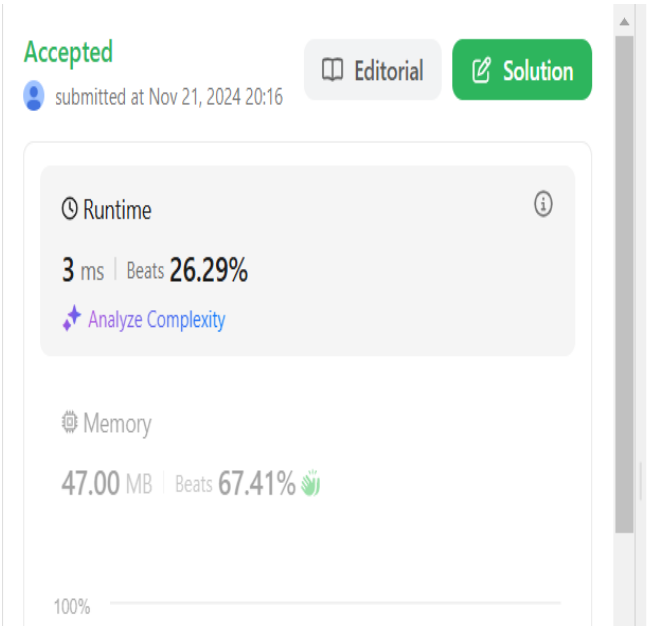

```java
class Solution {
    public boolean isPalindrome(String s) {
        int left = 0, right = s.length() - 1;
        while (left < right) {
            while (left < right && !Character.isLetterOrDigit(s.charAt(left))) left++;
            while (left < right && !Character.isLetterOrDigit(s.charAt(right))) right--;
            if (Character.toLowerCase(s.charAt(left)) != Character.toLowerCase(s.charAt(right))) return false;
            left++;
            right--;
        }
        return true;
    }
}
```

2.IS SUBSEQUENCE: O(N+M)



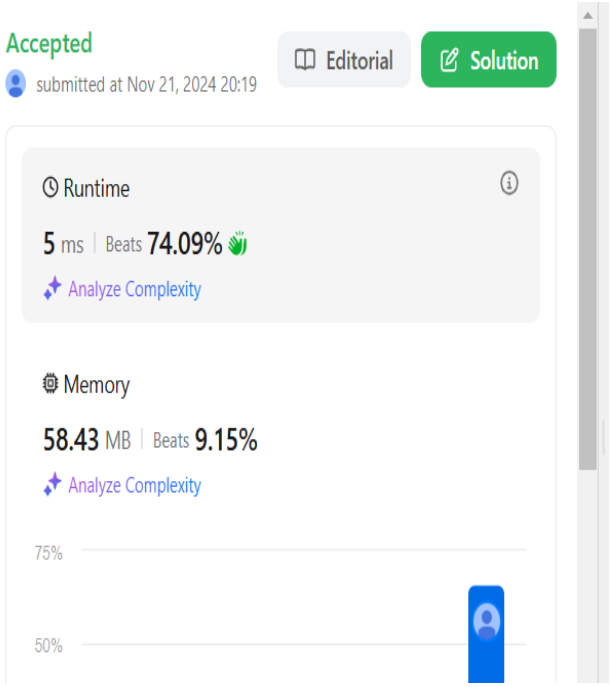

```java
class Solution {
    public boolean isSubsequence(String s, String t) {
        int i = 0, j = 0;
        while (i < s.length() && j < t.length()) {
            if (s.charAt(i) == t.charAt(j)) {
                i++;
            }
            j++;
        }
        return i == s.length();
    }
}
```

3. TWO SUM II: O(N)



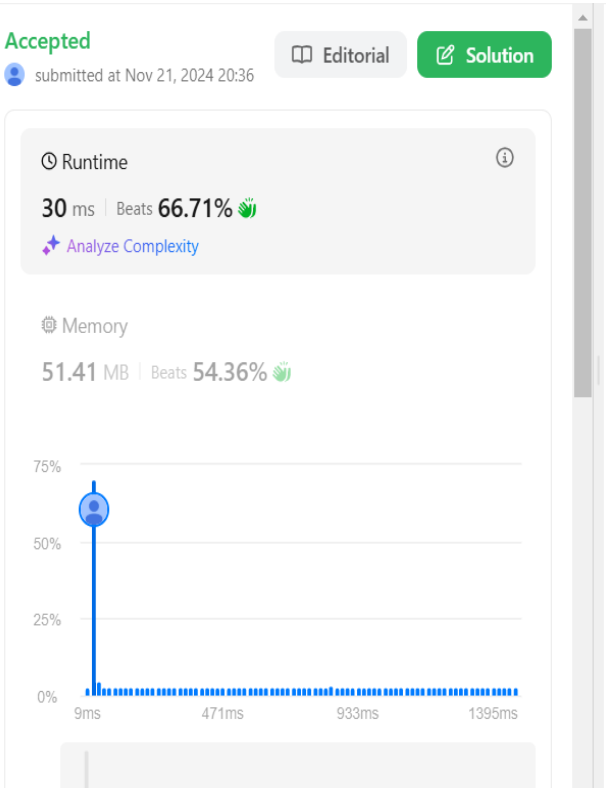

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int i=0,j=nums.length-1;
        while(i<nums.length && i<j){
            if(nums[i]+nums[j]==target){
                return new int[]{i+1,j+1};
            }
            else if(nums[i]+nums[j]>target){
                j--;
            }
            else{ i++;}


        }return new int[]{-1,-1};

    }
}
```

4.CONTAINER WITH MOST WATER: O(N)




```java
class Solution {
    public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int maxArea = 0;

        while (left < right) {
            int currentArea = Math.min(height[left], height[right]) * (right - left);
            maxArea = Math.max(maxArea, currentArea);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return maxArea;
    }
}
```

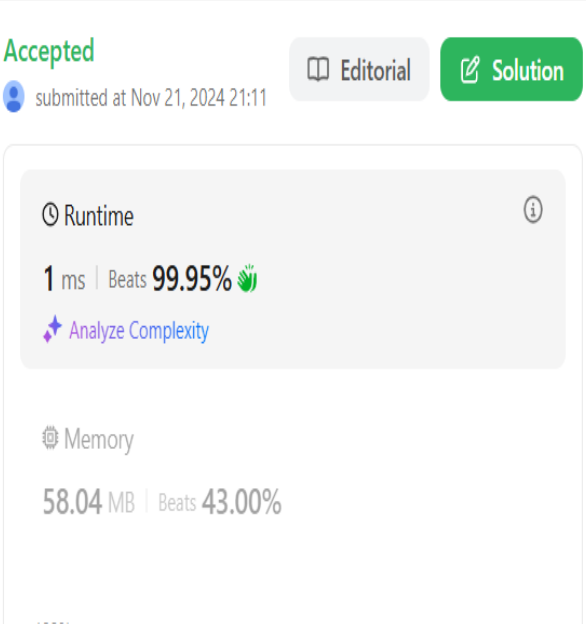
5. 3 SUM:




```java
import java.util.*;

class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            int j = i + 1, k = nums.length - 1;

            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];

                if (sum == 0) {
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    while (j < k && nums[j] == nums[j + 1]) j++;
                    while (j < k && nums[k] == nums[k - 1]) k--;
                    j++;
                    k--;
                } else if (sum < 0) {
                    j++;
                } else {
                    k--;
                }
            }
        }
        return res;
```

SLIDING WINDOW:

1. Minimum Size Subarray Sum:




```java
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int left=0,right=0,sum =0;
        int ans = Integer.MAX_VALUE;
        for(right=0;right<nums.length;right++){
            sum +=nums[right];
            while(sum>=target){
                ans=Math.min(ans,right-left+1);
                sum -=nums[left++];
            }

        }
        return ans == Integer.MAX_VALUE ? 0:ans;
    }
}
```