

# DSA QUESTIONS PRACTICE

21/11/24

## TWO POINTERS:

### 1. VALID PALINDROME: $O(N)$

Accepted

submitted at Nov 21, 2024 13:00

Editorial

Solution

Runtime

2 ms | Beats 99.08%

Analyze Complexity

Memory

42.67 MB | Beats 90.25%

75%

```
1 class Solution {
2     public boolean isPalindrome(String s) {
3         int left = 0, right = s.length() - 1;
4         while (left < right) {
5             while (left < right && !Character.isLetterOrDigit(s.charAt(left))) left++;
6             while (left < right && !Character.isLetterOrDigit(s.charAt(right))) right--;
7             if (Character.toLowerCase(s.charAt(left)) != Character.toLowerCase(s.charAt(right))) return false;
8             left++;
9             right--;
10        }
11        return true;
12    }
13 }
14
```

### 2. IS SUBSEQUENCE: $O(N+M)$

Accepted

submitted at Nov 21, 2024 13:08

Editorial

Solution

Runtime

2 ms | Beats 63.46%

Analyze Complexity

Memory

41.12 MB | Beats 86.88%

```
1 class Solution {
2     public boolean isSubsequence(String s, String t) {
3         int i = 0, j = 0;
4         while (i < s.length() && j < t.length()) {
5             if (s.charAt(i) == t.charAt(j)) {
6                 i++;
7             }
8             j++;
9         }
10        return i == s.length();
11    }
12 }
13
```

### 3. TWO SUM II: O(N)

**Accepted**

submitted at Nov 21, 2024 20:16

Editorial Solution

**Runtime**

3 ms | Beats 26.29%

Analyze Complexity

**Memory**

47.00 MB | Beats 67.41%

100%

```
1 class Solution {
2     public int[] twoSum(int[] nums, int target) {
3         int i=0,j=nums.length-1;
4         while(i<nums.length && i<j){
5             if(nums[i]+nums[j]==target){
6                 return new int[]{i+1,j+1};
7             }
8             else if(nums[i]+nums[j]>target){
9                 j--;
10            }
11            else{ i++;}
12        }
13        return new int[]{-1,-1};
14    }
15 }
16 }
17 }
```

### 4.CONTAINER WITH MOST WATER: O(N)

**Accepted**

submitted at Nov 21, 2024 20:19

Editorial Solution

**Runtime**

5 ms | Beats 74.09%

Analyze Complexity

**Memory**

58.43 MB | Beats 9.15%

75%

50%

```
1 class Solution {
2     public int maxArea(int[] height) {
3         int left = 0;
4         int right = height.length - 1;
5         int maxArea = 0;
6
7         while (left < right) {
8             int currentArea = Math.min(height[left], height[right]) *
9             (right - left);
10            maxArea = Math.max(maxArea, currentArea);
11
12            if (height[left] < height[right]) {
13                left++;
14            } else {
15                right--;
16            }
17        }
18        return maxArea;
19    }
20 }
```

## 5. 3 SUM:

Accepted

submitted at Nov 21, 2024 20:36

Editorial

Solution

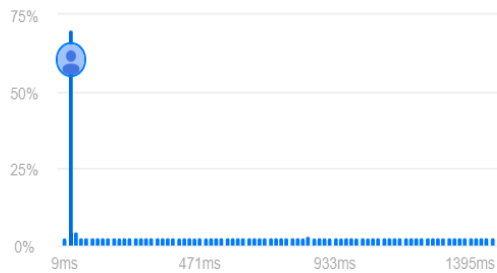
Runtime

30 ms | Beats 66.71%

Analyze Complexity

Memory

51.41 MB | Beats 54.36%



```
1 import java.util.*;
2
3 class Solution {
4     public List<List<Integer>> threeSum(int[] nums) {
5         List<List<Integer>> res = new ArrayList<>();
6         Arrays.sort(nums);
7
8         for (int i = 0; i < nums.length - 2; i++) {
9             if (i > 0 && nums[i] == nums[i - 1]) continue;
10
11             int j = i + 1, k = nums.length - 1;
12
13             while (j < k) {
14                 int sum = nums[i] + nums[j] + nums[k];
15
16                 if (sum == 0) {
17                     res.add(Arrays.asList(nums[i], nums[j], nums[k]));
18                     while (j < k && nums[j] == nums[j + 1]) j++;
19                     while (j < k && nums[k] == nums[k - 1]) k--;
20                     j++;
21                     k--;
22                 } else if (sum < 0) {
23                     j++;
24                 } else {
25                     k--;
26                 }
27             }
28         }
29         return res;
30     }
31 }
```

## SLIDING WINDOW:

### 1. Minimum Size Subarray Sum:

Accepted

submitted at Nov 21, 2024 21:11

Editorial

Solution

Runtime

1 ms | Beats 99.95%

Analyze Complexity

Memory

58.04 MB | Beats 43.00%

```
1 class Solution {
2     public int minSubArrayLen(int target, int[] nums) {
3         int left=0, right=0, sum =0;
4         int ans = Integer.MAX_VALUE;
5         for(right=0; right<nums.length; right++){
6             sum +=nums[right];
7             while(sum>=target){
8                 ans=Math.min(ans, right-left+1);
9                 sum -=nums[left++];
10            }
11        }
12        return ans == Integer.MAX_VALUE ? 0:ans;
13    }
14 }
15 }
```

