Vasanthan_T_DSA_Practice-9

21-11-2024

1. Valid Palindrome

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a* ***palindrome****, or* false *otherwise*.

Example 1:

Input: s = "A man, a plan, a canal: Panama"

Output: true

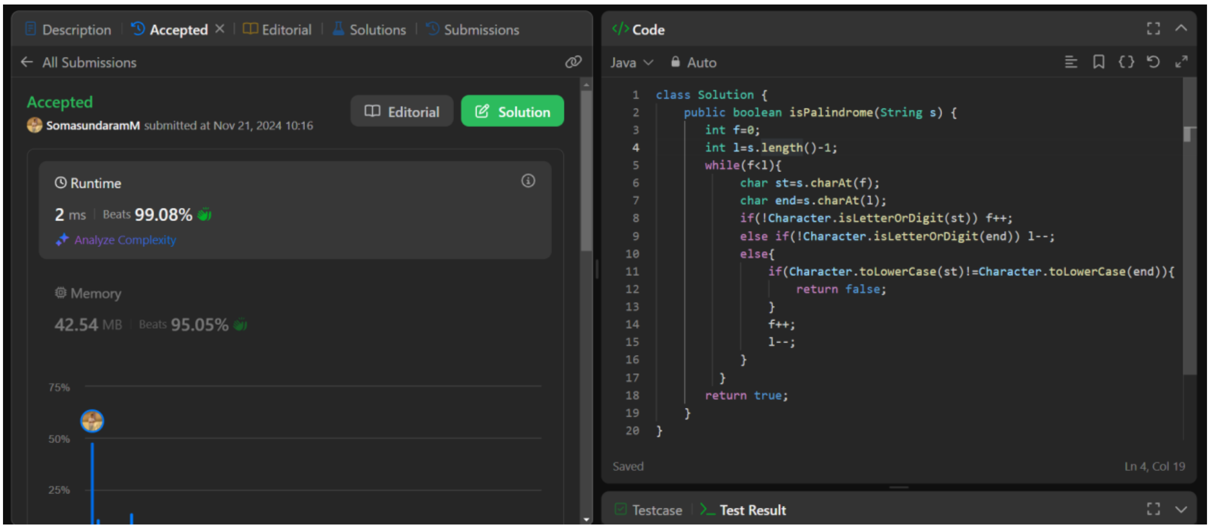
Explanation: "amanaplanacanalpanama" is a palindrome.

Code:

```
class Solution {
   public boolean isPalindrome(String s) {
      int f=0;
      int l=s.length()-1;
      while(f<l){
         char st=s.charAt(f);
         char end=s.charAt(l);
         if(!Character.isLetterOrDigit(st)) f++;
         else if(!Character.isLetterOrDigit(end)) l--;
         else{
            if(Character.toLowerCase(st)!=Character.toLowerCase(end)){
               return false;
```

```
        }
        f++;
        l--;
      }
    }
  return true;
 }
}
```

Output:



Accepted

SomasundaramM submitted at Nov 21, 2024 10:16

Runtime

2 ms | Beats 99.08%

Memory

42.54 MB | Beats 95.05%

```
class Solution {
    public boolean isPalindrome(String s) {
        int f=0;
        int l=s.length()-1;
        while(f<l){
            char st=s.charAt(f);
            char end=s.charAt(l);
            if(!Character.isLetterOrDigit(st)) f++;
            else if(!Character.isLetterOrDigit(end)) l--;
            else{
                if(Character.toLowerCase(st)!=Character.toLowerCase(end)){
                    return false;
                }
                f++;
                l--;
            }
        }
        return true;
    }
}
```

Saved

Ln 4, Col 19

Time Complexity:O(n)

Space Complexity:O(1)

2. Is Subsequence

Given two strings s and t, return true *if* s *is a subsequence of* t, *or* false *otherwise*.

A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:

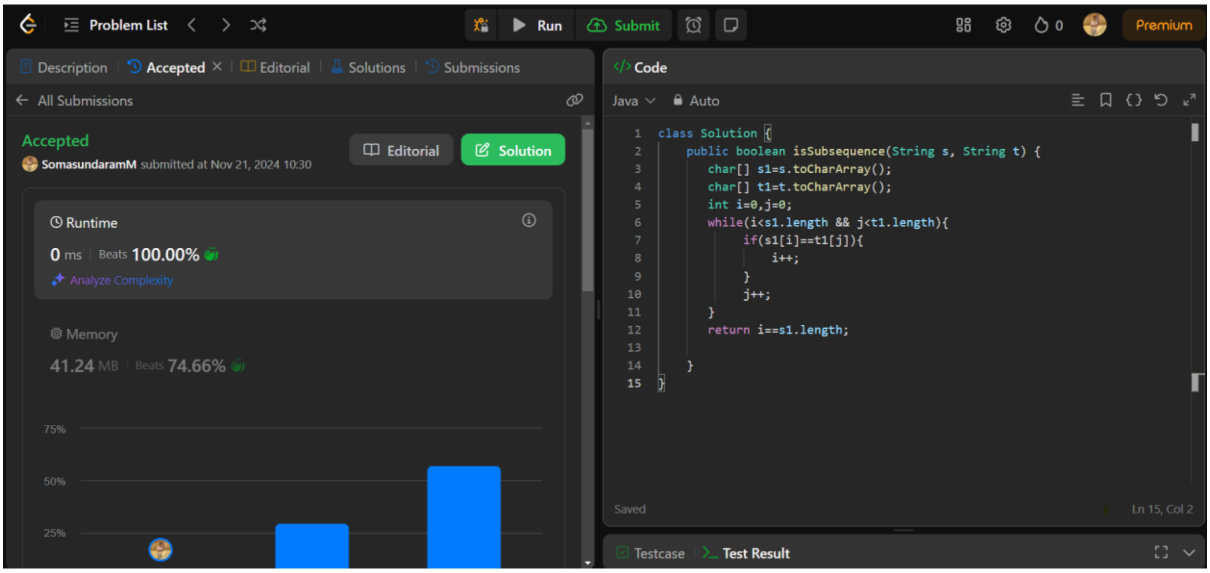
Input: s = "abc", t = "ahbgdc"

Output: true

Code:

```
class Solution {
   public boolean isSubsequence(String s, String t) {
      char[] s1=s.toCharArray();
      char[] t1=t.toCharArray();
      int i=0,j=0;
      while(i<s1.length && j<t1.length){
          if(s1[i]==t1[j]){
             i++;
          }
          j++;
      }
      return i==s1.length;

   }
}
```

Output:



Accepted

SomasundaramM submitted at Nov 21, 2024 10:30

Runtime

0 ms | Beats 100.00%

Memory

41.24 MB | Beats 74.66%

```java
class Solution {
    public boolean isSubsequence(String s, String t) {
        char[] s1=s.toCharArray();
        char[] t1=t.toCharArray();
        int i=0,j=0;
        while(i<s1.length && j<t1.length){
            if(s1[i]==t1[j]){
                i++;
            }
            j++;
        }
        return i==s1.length;

    }
}
```

Time Complexity:O(n)

Space Complexity: O(1)

3. Container With Most Water

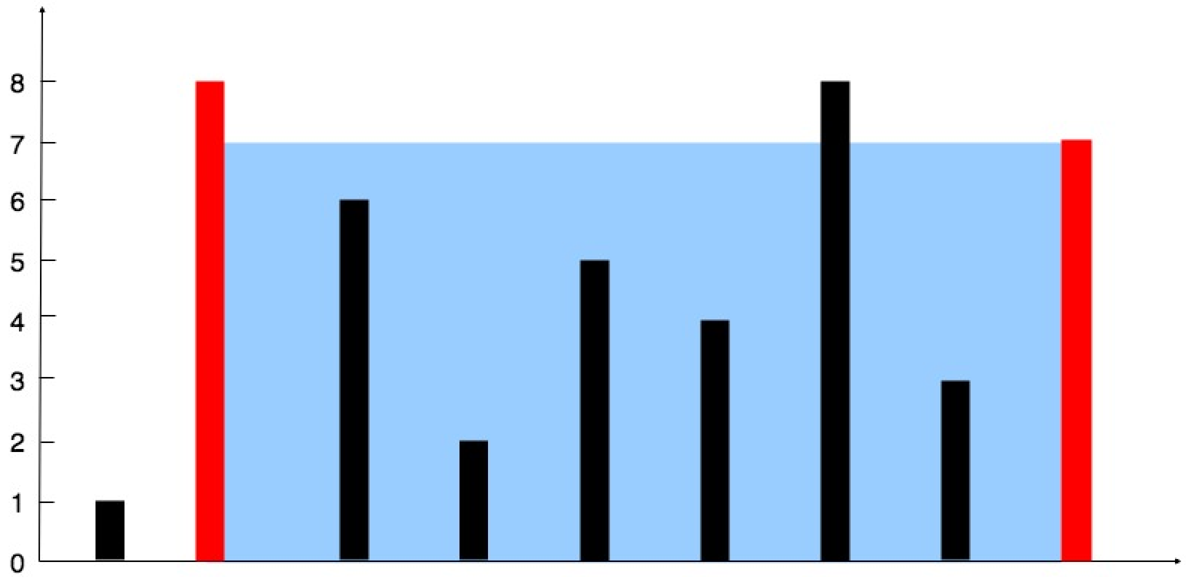
You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

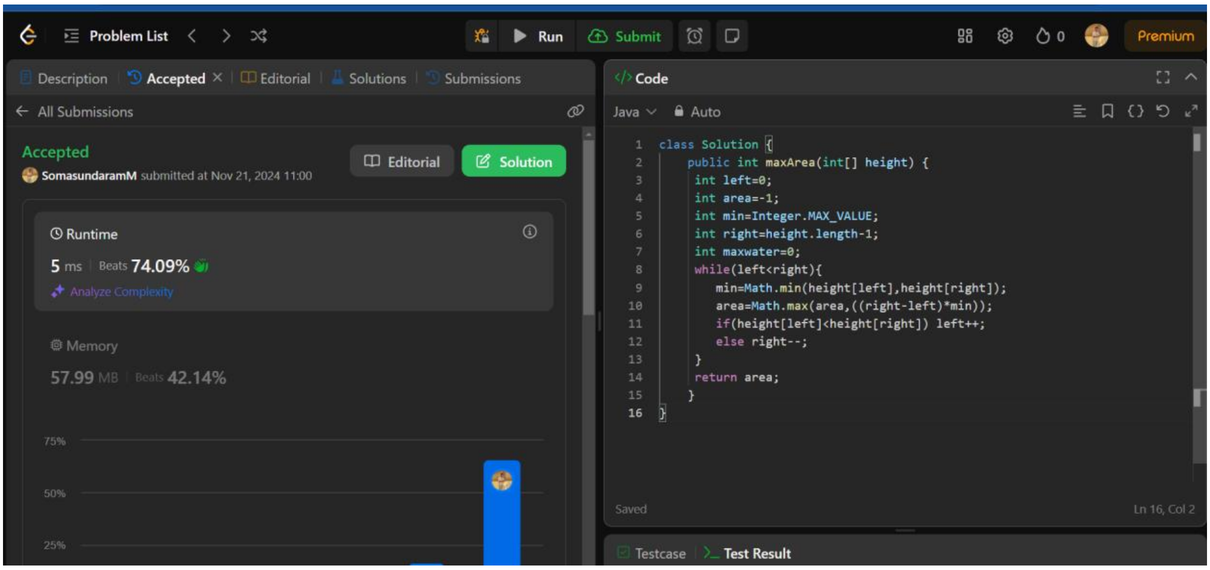
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Code:

```
class Solution {
    public int maxArea(int[] height) {
     int left=0;
     int area=-1;
     int min=Integer.MAX_VALUE;
     int right=height.length-1;
     int maxwater=0;
     while(left<right){
        min=Math.min(height[left],height[right]);
        area=Math.max(area,((right-left)*min));
```

```
      if(height[left]<height[right]) left++;
      else right--;
    }
    return area;
  }
}
```

Output:



Accepted
SomasundaramM submitted at Nov 21, 2024 11:00

Runtime
5 ms | Beats 74.09%

Memory
57.99 MB | Beats 42.14%

```
class Solution {
    public int maxArea(int[] height) {
     int left=0;
     int area=-1;
     int min=Integer.MAX_VALUE;
     int right=height.length-1;
     int maxwater=0;
     while(left<right){
        min=Math.min(height[left],height[right]);
        area=Math.max(area,((right-left)*min));
        if(height[left]<height[right]) left++;
        else right--;
     }
     return area;
    }
}
```

Saved
Ln 16, Col 2

Time Complexity:O(n)

Space Complexity:O(1)

4. 3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]

Explanation:

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

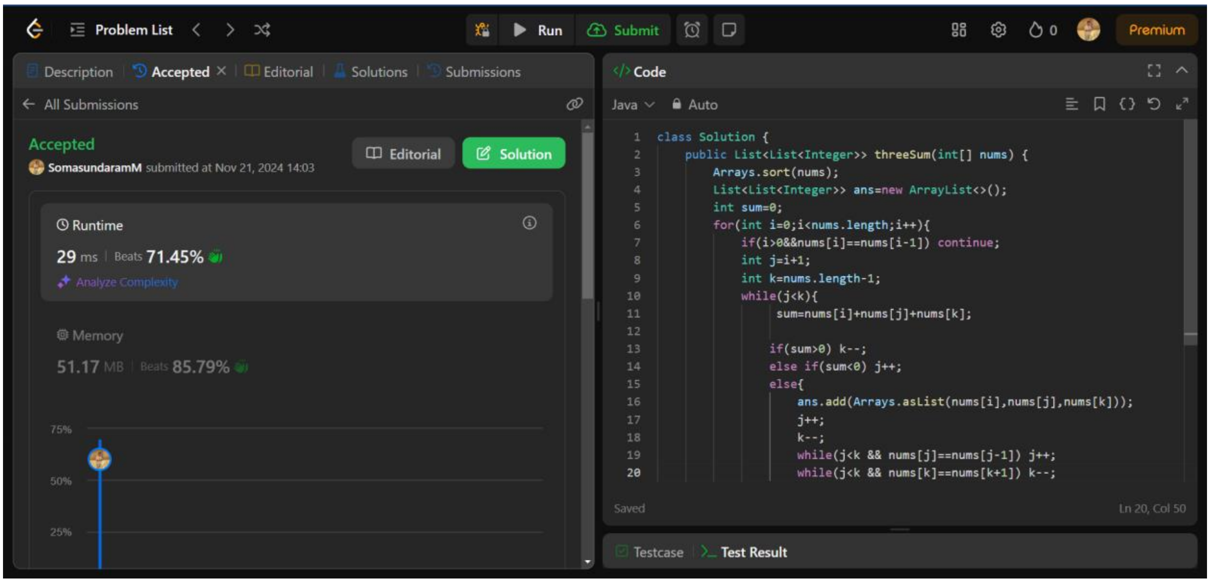
The distinct triplets are [-1,0,1] and [-1,-1,2].

Notice that the order of the output and the order of the triplets does not matter.

Code:

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ans=new ArrayList<>();
        int sum=0;
        for(int i=0;i<nums.length;i++){
            if(i>0&&nums[i]==nums[i-1]) continue;
            int j=i+1;
            int k=nums.length-1;
            while(j<k){
                 sum=nums[i]+nums[j]+nums[k];

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


        }
        return ans;
    }
}
```

Output:



Accepted

SomasundaramM submitted at Nov 21, 2024 14:03

Runtime

29 ms | Beats 71.45%

Memory

51.17 MB | Beats 85.79%

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ans=new ArrayList<>();
        int sum=0;
        for(int i=0;i<nums.length;i++){
            if(i>0&&nums[i]==nums[i-1]) continue;
            int j=i+1;
            int k=nums.length-1;
            while(j<k){
                  sum=nums[i]+nums[j]+nums[k];

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

Time Complexity:O(n^2)

Space Complexity: O(1)

5. Two Sum II - Input Array Is Sorted

Given a 1-indexed array of integers numbers that is already *sorted in non-decreasing order*, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index$_1$] and numbers[index$_2$] where 1 <= index$_1$ < index$_2$ <= numbers.length.

Return *the indices of the two numbers,* index$_1$ *and* index$_2$*, added by one as an integer array* [index$_1$, index$_2$] *of length 2.*

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

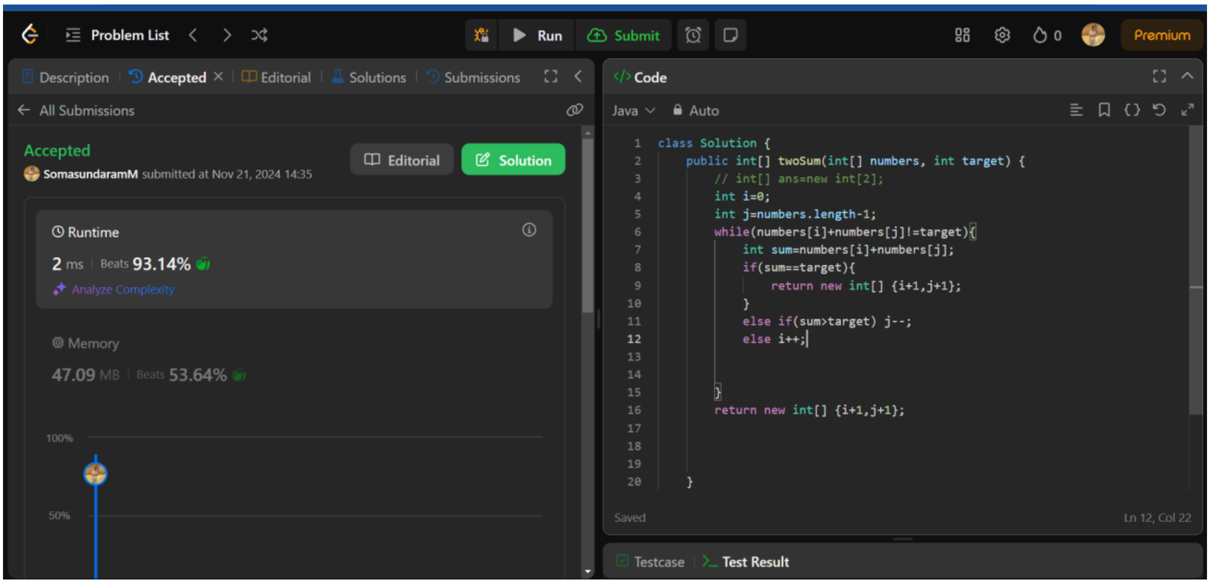
Explanation: The sum of 2 and 7 is 9. Therefore, index$_1$ = 1, index$_2$ = 2. We return [1, 2].

Code:

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        // int[] ans=new int[2];
        int i=0;
        int j=numbers.length-1;
        while(numbers[i]+numbers[j]!=target){
            int sum=numbers[i]+numbers[j];
            if(sum==target){
                return new int[] {i+1,j+1};
            }
```

```
          else if(sum>target) j--;
          else i++;



       }
       return new int[] {i+1,j+1};



    }
}
```

Output:



Accepted

SomasundaramM submitted at Nov 21, 2024 14:35

Runtime

2 ms | Beats 93.14%

Memory

47.09 MB | Beats 53.64%

```java
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        // int[] ans=new int[2];
        int i=0;
        int j=numbers.length-1;
        while(numbers[i]+numbers[j]!=target){
            int sum=numbers[i]+numbers[j];
            if(sum==target){
                return new int[] {i+1,j+1};
            }
            else if(sum>target) j--;
            else i++;


        }
        return new int[] {i+1,j+1};



    }
```

Saved

Ln 12, Col 22

Time Complexity:O(n)

Space Complexity:O(1)

6. Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

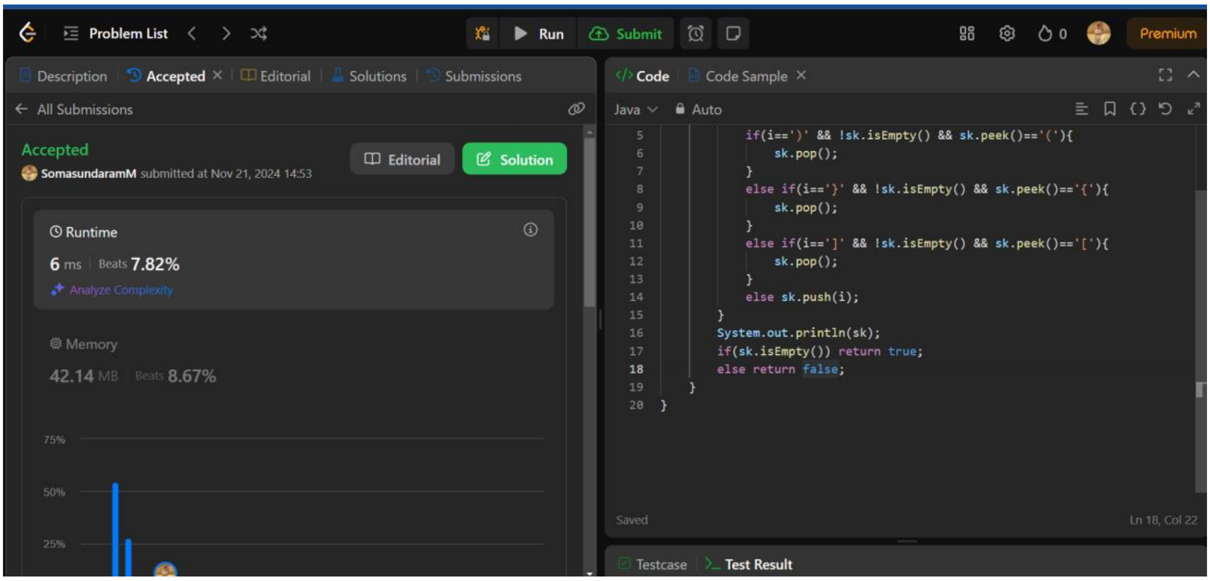
Input: s = "()"

Output: true

Code:

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> sk=new Stack<>();
        for(char i:s.toCharArray()){
            if(i==')' && !sk.isEmpty() && sk.peek()=='('){
                sk.pop();
            }
            else if(i=='}' && !sk.isEmpty() && sk.peek()=='{'){
                sk.pop();
            }
            else if(i==']' && !sk.isEmpty() && sk.peek()=='['){
                sk.pop();
            }
            else sk.push(i);
        }
        System.out.println(sk);
        if(sk.isEmpty()) return true;
        else return false;
    }
}
```

Output:



Accepted

SomasundaramM submitted at Nov 21, 2024 14:53

Runtime

6 ms | Beats 7.82%

Memory

42.14 MB | Beats 8.67%

```
            if(i==')' && !sk.isEmpty() && sk.peek()=='('){
                sk.pop();
            }
            else if(i=='}' && !sk.isEmpty() && sk.peek()=='{'){
                sk.pop();
            }
            else if(i==']' && !sk.isEmpty() && sk.peek()=='['){
                sk.pop();
            }
            else sk.push(i);
        }
        System.out.println(sk);
        if(sk.isEmpty()) return true;
        else return false;
    }
}
```

Saved

Ln 18, Col 22

Time Complexity: O(n)

Space Complexity:O(n)

7. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

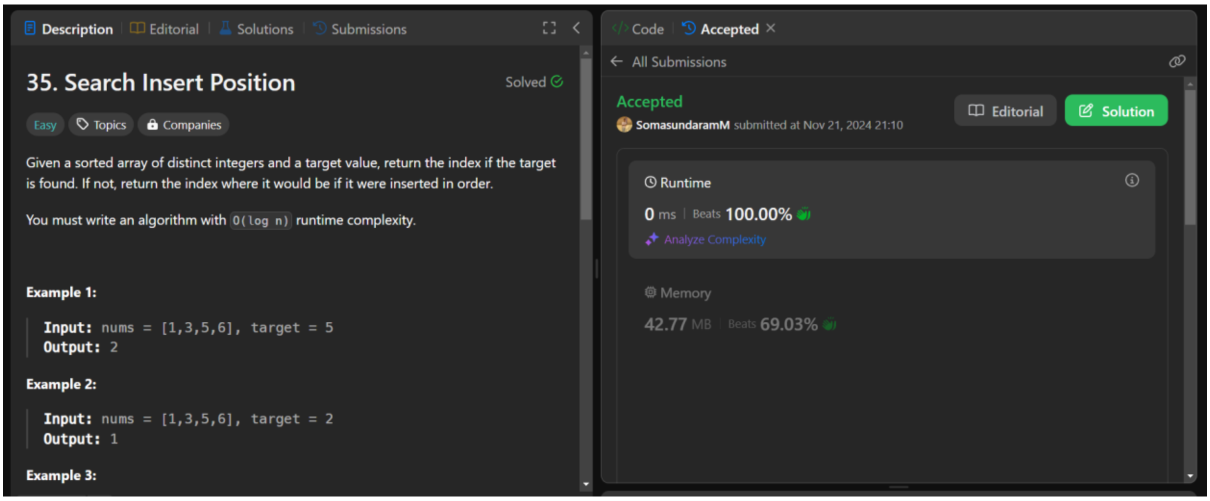
Input: nums = [1,3,5,6], target = 5

Output: 2

Code:

```
class Solution {
    public int searchInsert(int[] nums, int tar) {
        int n=nums.length;
        int l=0;
        int r=n-1;
        while(l<=r){
            int mid=l+(r-l)/2;
            if(nums[mid]==tar) return mid;
            else if(nums[mid]>tar) r=mid-1;
            else l=mid+1;
        }
        return l;
    }
}
```

Output:



35. Search Insert Position

Solved

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

Accepted

SomasundaramM submitted at Nov 21, 2024 21:10

Runtime

0 ms | Beats 100.00%

Memory

42.77 MB | Beats 69.03%

Time Complexity:O(log N)

Space Complexity: O(1)

8.