

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;  
    }  
}
```

```

        }

        f++;

        l--;

    }

}

return true;

}

}

```

Output:

The screenshot displays a code editor interface. On the left, the submission status is 'Accepted' for a user named 'SomasundaramM' submitted at Nov 21, 2024 10:16. The runtime is 2 ms, beating 99.08% of other submissions, and the memory usage is 42.54 MB, beating 95.05%. The right panel shows the Java code for the solution, which is a class 'Solution' with a method 'isPalindrome' that takes a string 's' and returns a boolean. The method uses two pointers, 'f' and 'l', to traverse the string from both ends towards the center, comparing characters and returning false if they don't match, and true if they do.

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:

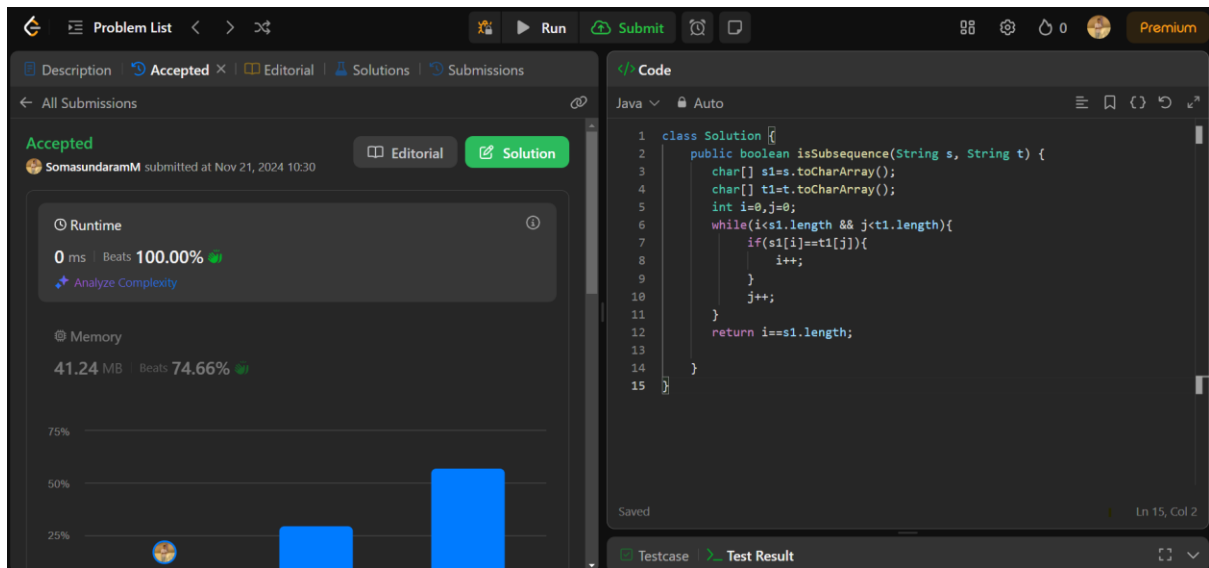
Input: s = "abc", t = "ahbgdc"

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:



```
1 class Solution {
2     public boolean isSubsequence(String s, String t) {
3         char[] s1=s.toCharArray();
4         char[] t1=t.toCharArray();
5         int i=0,j=0;
6         while(i<s1.length && j<t1.length){
7             if(s1[i]==t1[j]){
8                 i++;
9             }
10            j++;
11        }
12        return i==s1.length;
13    }
14 }
15 }
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

3. Container With Most Water

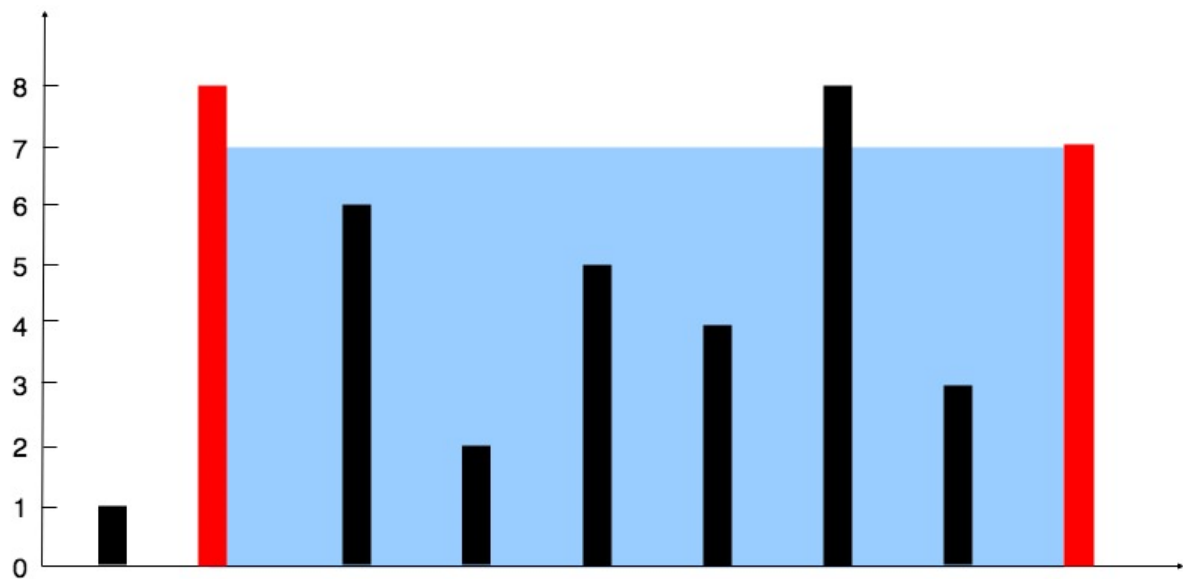
You are given an integer array height of length n . There are n vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

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:

The screenshot displays the LeetCode submission page for the 'Trapping Water' problem. The submission is marked as 'Accepted' and was submitted by 'SomasundaramM' on Nov 21, 2024, at 11:00. The runtime is 5 ms, beating 74.09% of other submissions, and the memory usage is 57.99 MB, beating 42.14%. The code is written in Java and implements a two-pointer solution.

```

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

```

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 $i \neq j$, $i \neq k$, and $j \neq k$, 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:

$\text{nums}[0] + \text{nums}[1] + \text{nums}[2] = (-1) + 0 + 1 = 0.$

$\text{nums}[1] + \text{nums}[2] + \text{nums}[4] = 0 + 1 + (-1) = 0.$

$\text{nums}[0] + \text{nums}[3] + \text{nums}[4] = (-1) + 2 + (-1) = 0.$

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++;
                }
            }
        }
        return ans;
    }
}
```

```

        k--;

        while(j<k && nums[j]==nums[j-1]) j++;

        while(j<k && nums[k]==nums[k+1]) k--;

    }

}

return ans;

}

}

```

Output:

The screenshot shows a code editor interface with a dark theme. On the left, a sidebar displays submission information for a problem titled 'Three Sum'. It indicates the submission is 'Accepted' by 'SomasundaramM' at 'Nov 21, 2024 14:03'. Performance metrics show a runtime of '29 ms' (beating 71.45% of submissions) and a memory usage of '51.17 MB' (beating 85.79%). A 'Runtime' graph is visible, showing the submission's performance relative to others. The main editor area displays the Java code for the 'threeSum' method. The code sorts the input array and uses a two-pointer approach to find all unique triplets that sum to zero. The bottom status bar shows 'Saved' and the cursor position 'Ln 20, Col 50'.

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[index₁] and numbers[index₂] where $1 \leq \text{index}_1 < \text{index}_2 \leq \text{numbers.length}$.

Return *the indices of the two numbers, index₁ and index₂, added by one as an integer array [index₁, index₂] 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, index₁ = 1, index₂ = 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++;

    }

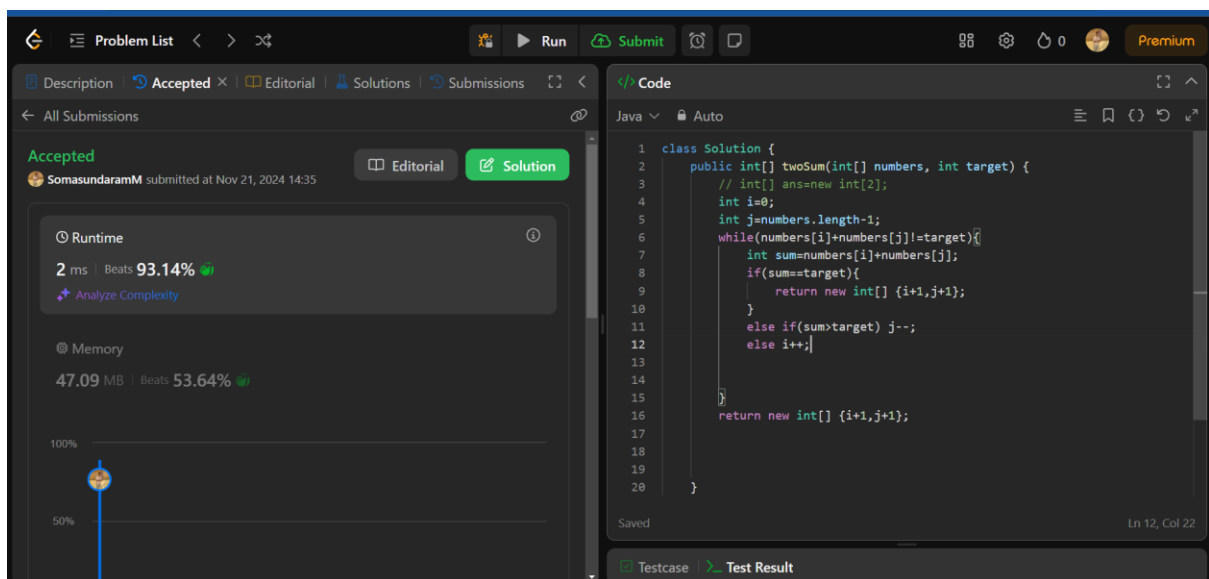
    return new int[] {i+1,j+1};

}

}

```

Output:



The screenshot shows a code editor interface with the following details:

- Problem List:** Accepted
- Editorial:** Solution
- Runtime:** 2 ms | Beats 93.14%
- Memory:** 47.09 MB | Beats 53.64%
- Code:**

```

1 class Solution {
2     public int[] twoSum(int[] numbers, int target) {
3         // int[] ans=new int[2];
4         int i=0;
5         int j=numbers.length-1;
6         while(numbers[i]+numbers[j]!=target){
7             int sum=numbers[i]+numbers[j];
8             if(sum==target){
9                 return new int[] {i+1,j+1};
10            }
11            else if(sum>target) j--;
12            else i++;
13        }
14        return new int[] {i+1,j+1};
15    }
16 }
17
18
19
20

```

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:

The screenshot shows a code editor interface with a Java solution. The code is as follows:

```

5      if(i=='{' && !sk.isEmpty() && sk.peek()=='{'){
6          sk.pop();
7      }
8      else if(i=='}' && !sk.isEmpty() && sk.peek()=='{'){
9          sk.pop();
10     }
11     else if(i=='[' && !sk.isEmpty() && sk.peek()=='['){
12         sk.pop();
13     }
14     else sk.push(i);
15 }
16 System.out.println(sk);
17 if(sk.isEmpty()) return true;
18 else return false;
19 }
20 }

```

The submission is accepted, with a runtime of 6 ms (Beats 7.82%) and memory usage of 42.14 MB (Beats 8.67%). A bar chart at the bottom shows the performance comparison against other submissions.

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:

The screenshot displays the LeetCode interface for problem 35, "Search Insert Position". The left pane shows the problem description, which asks for the index of a target value in a sorted array of distinct integers, or the index where it should be inserted. It includes three examples with inputs and outputs. The right pane shows the submission results, indicating that the solution was accepted with a runtime of 0 ms (beating 100.00% of submissions) and a memory usage of 42.77 MB (beating 69.03% of submissions).

35. Search Insert Position Solved

Easy Topics Companies

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%
[Analyze Complexity](#)

Memory: 42.77 MB | Beats 69.03%

Time Complexity: $O(\log N)$

Space Complexity: $O(1)$

8.