

# DSA PRACTICE PROBLEMS- DAY 9

NAME: Dhejan R

REG NO: 22IT022

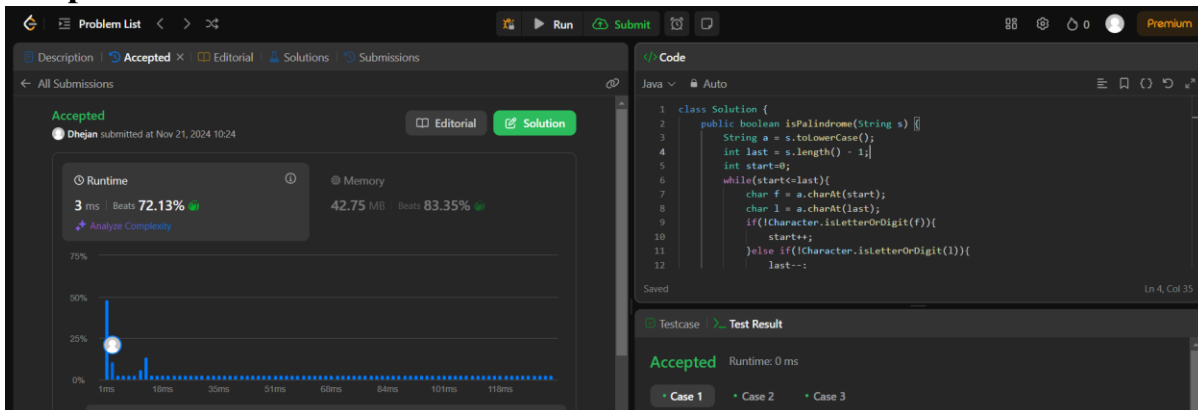
DATE: 21/11/2024

## 1. Valid Palindrome

### Code Solution

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

### Output



**Time complexity:  $O(n)$**

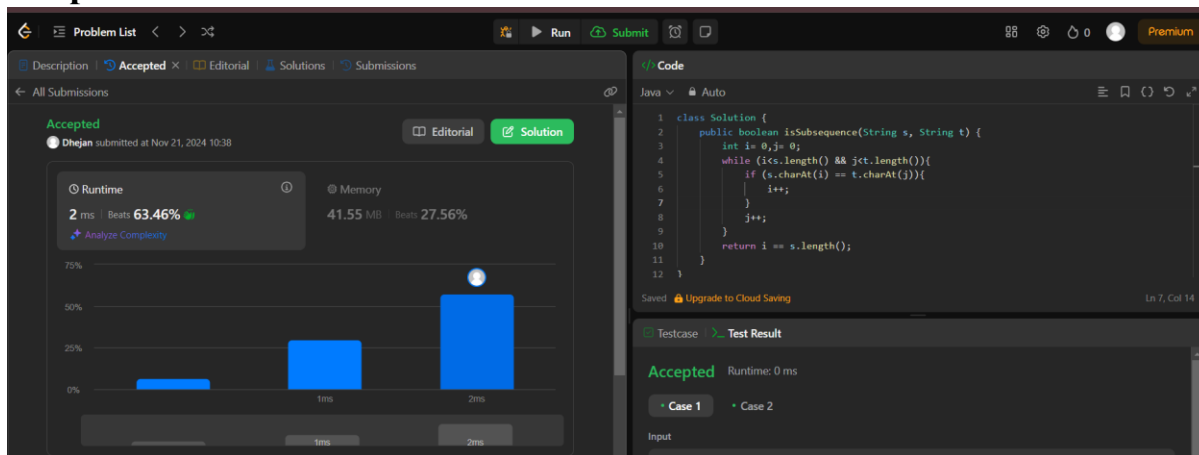
**Space complexity:  $O(1)$**

## 2. Is Subsequence

### Code Solution

```
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();  
    }  
}
```

### Output



**Time complexity:**  $O(n+m)$

**Space complexity:**  $O(1)$

## 3. Two Sum II- Input array is sorted

### Code Solution

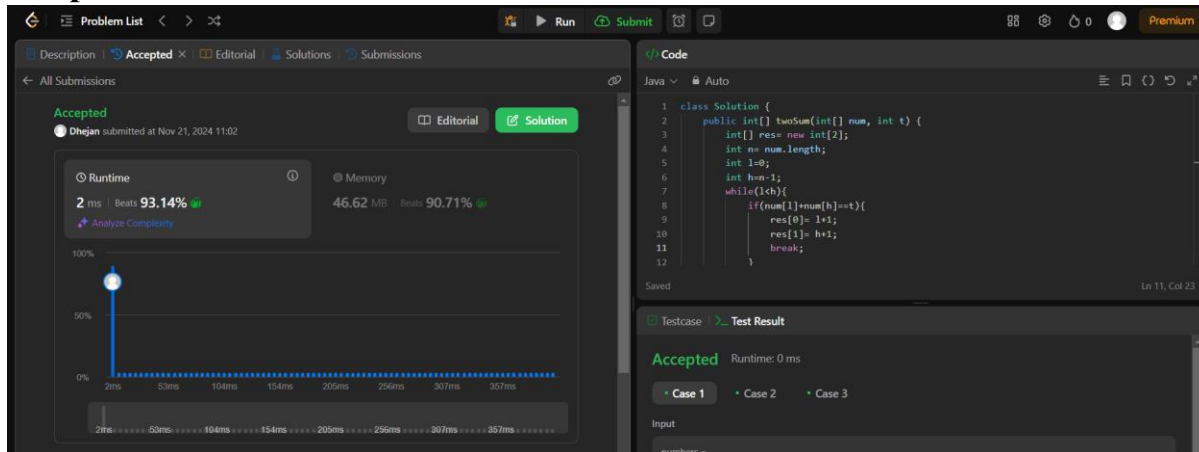
```
class Solution {  
    public int[] twoSum(int[] num, int t) {  
        int[] res= new int[2];  
        int n= num.length;  
        int l=0;  
        int h=n-1;  
        while(l<h){  
            if(num[l]+num[h]==t){  
                res[0]= l+1;  
                res[1]= h+1;  
                break;  
            }  
            else if(num[l]+num[h]>t){  
                h--;  
            }  
            else if(num[l]+num[h]<t){  
                l++;  
            }  
        }  
        return res;  
    }  
}
```

```

        h--;
    }else{
        l++;
    }
}
return res;
}
}

```

## Output



**Time complexity:  $O(n)$**

**Space complexity:  $O(1)$**

## 4. Container with most water

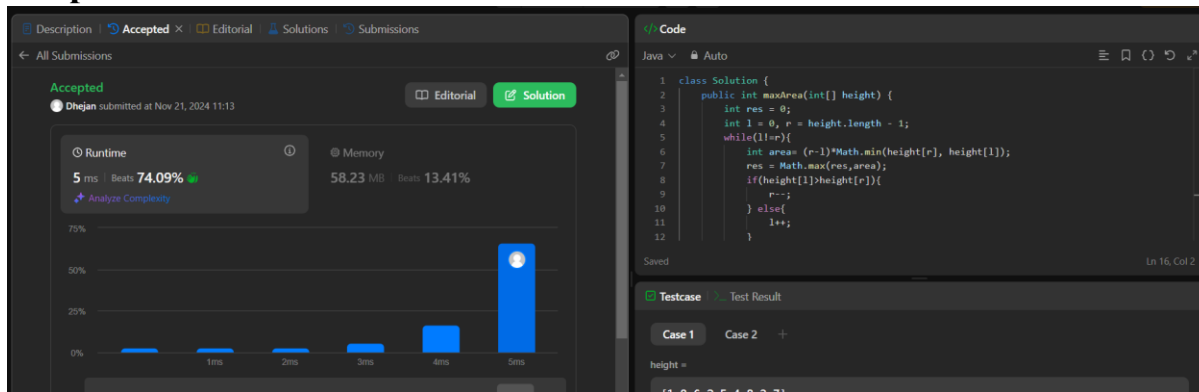
### Code Solution

```

class Solution {
    public int maxArea(int[] height) {
        int res = 0;
        int l = 0, r = height.length - 1;
        while(l!=r){
            int area= (r-l)*Math.min(height[r], height[l]);
            res = Math.max(res,area);
            if(height[l]>height[r]){
                r--;
            } else{
                l++;
            }
        }
        return res;
    }
}

```

## Output



**Time complexity:**  $O(n)$

**Space complexity:**  $O(1)$

## 5. 3Sum

### Code Solution

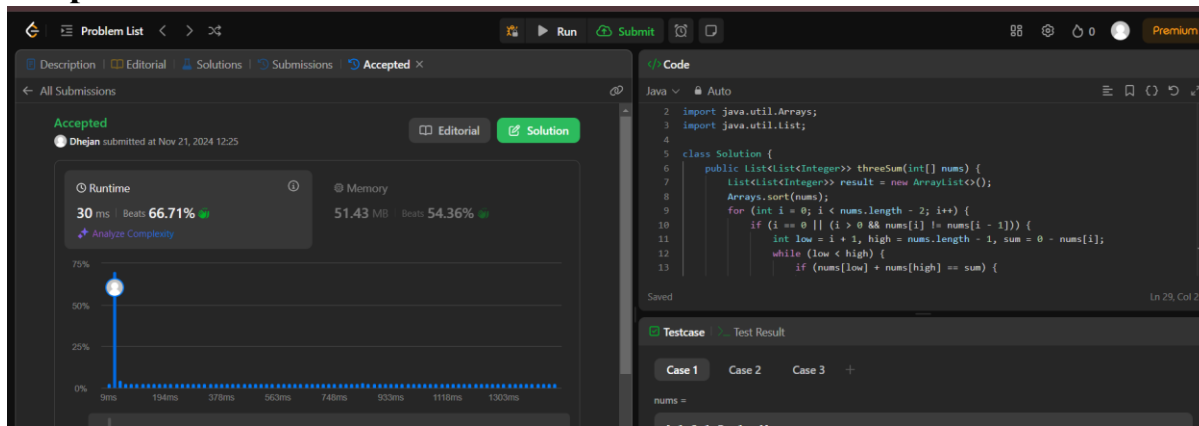
```
import java.util.ArrayList;
```

```
import java.util.Arrays;
```

```
import java.util.List;
```

```
class Solution {  
    public List<List<Integer>> threeSum(int[] nums) {  
        List<List<Integer>> result = new ArrayList<>();  
        Arrays.sort(nums);  
        for (int i = 0; i < nums.length - 2; i++) {  
            if (i == 0 || (i > 0 && nums[i] != nums[i - 1])) {  
                int low = i + 1, high = nums.length - 1, sum = 0 - nums[i];  
                while (low < high) {  
                    if (nums[low] + nums[high] == sum) {  
                        result.add(Arrays.asList(nums[i], nums[low], nums[high]));  
                        while (low < high && nums[low] == nums[low + 1]) low++;  
                        while (low < high && nums[high] == nums[high - 1]) high--;  
                        low++;  
                        high--;  
                    } else if (nums[low] + nums[high] < sum) {  
                        low++;  
                    } else {  
                        high--;  
                    }  
                }  
            }  
        }  
        return result;  
    }  
}
```

## Output



**Time complexity:**  $O(n^2)$

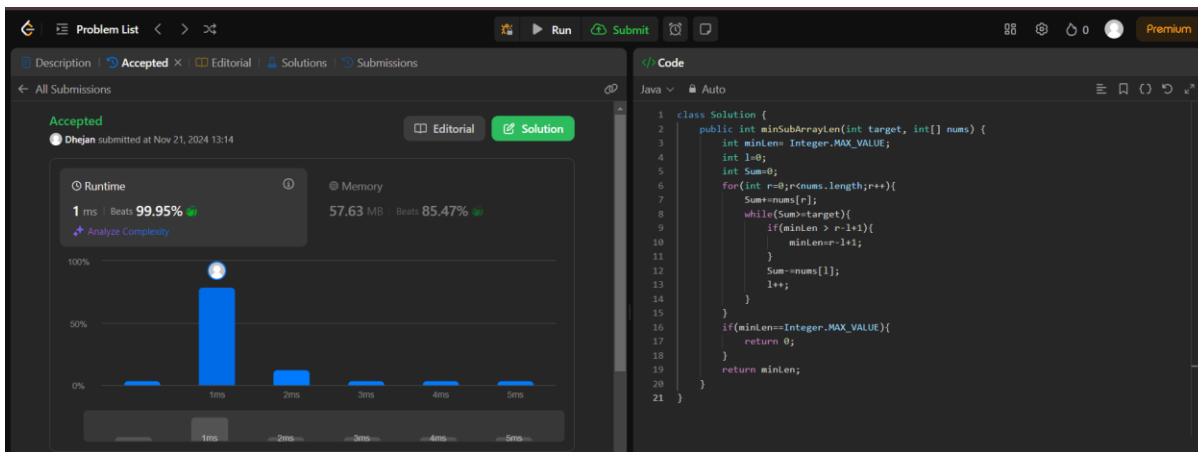
**Space complexity:**  $O(n)$

## 6. Minimum Size Subarray Sum

### Code Solution

```
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int minLen= Integer.MAX_VALUE;
        int l=0;
        int Sum=0;
        for(int r=0;r<nums.length;r++){
            Sum+=nums[r];
            while(Sum>=target){
                if(minLen > r-l+1){
                    minLen=r-l+1;
                }
                Sum-=nums[l];
                l++;
            }
        }
        if(minLen==Integer.MAX_VALUE){
            return 0;
        }
        return minLen;
    }
}
```

## Output



**Time complexity:  $O(n)$**

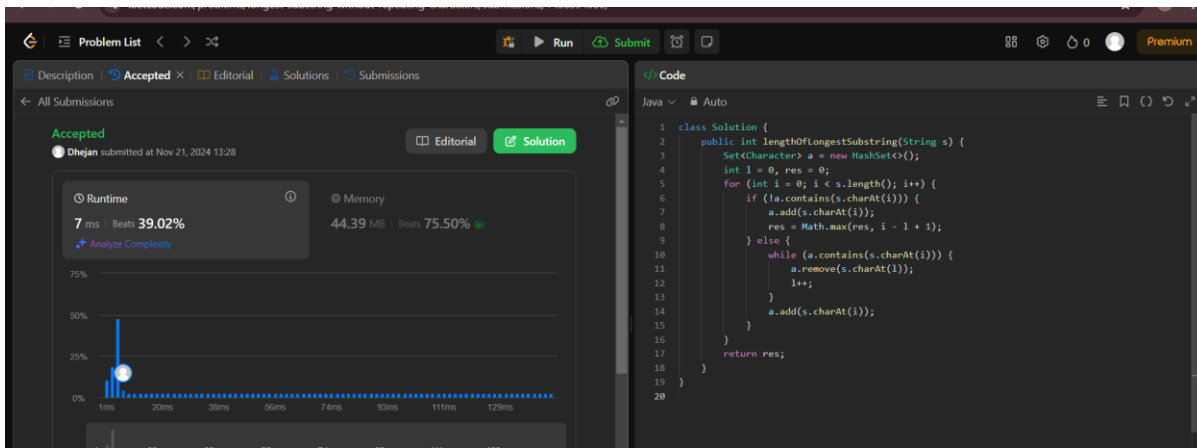
**Space complexity:  $O(1)$**

## 7. Longest Substring without repeating characters

### Code Solution

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> a = new HashSet<>();
        int l = 0, res = 0;
        for (int i = 0; i < s.length(); i++) {
            if (!a.contains(s.charAt(i))) {
                a.add(s.charAt(i));
                res = Math.max(res, i - l + 1);
            } else {
                while (a.contains(s.charAt(i))) {
                    a.remove(s.charAt(l));
                    l++;
                }
                a.add(s.charAt(i));
            }
        }
        return res;
    }
}
```

### Output



**Time complexity:**  $O(n)$

**Space complexity:**  $O(n)$

## 8. Search Insert Position

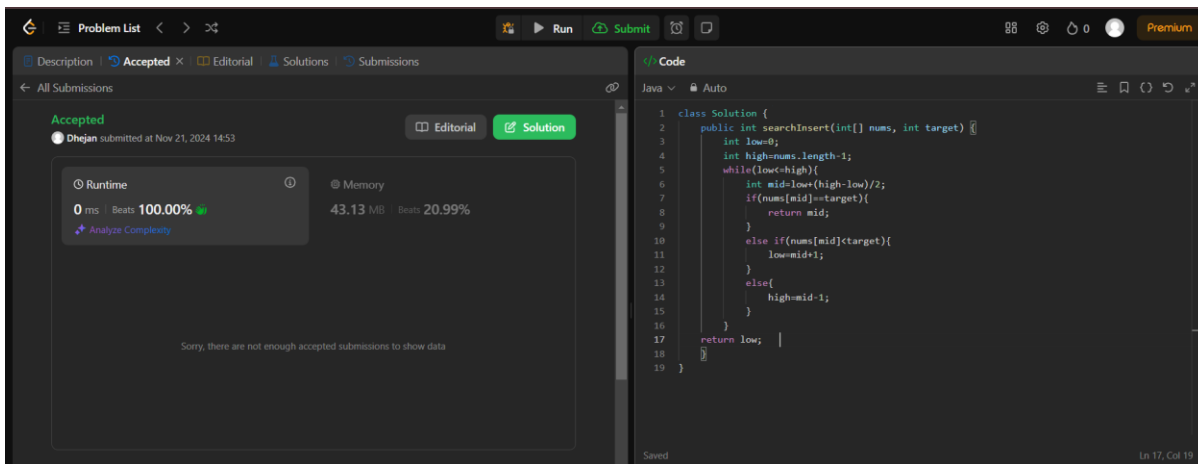
### Code Solution

```

class Solution {
    public int searchInsert(int[] nums, int target) {
        int low=0;
        int high=nums.length-1;
        while(low<=high){
            int mid=low+(high-low)/2;
            if(nums[mid]==target){
                return mid;
            }
            else if(nums[mid]<target){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
        return low;
    }
}

```

**Output**



**Time complexity:**  $O(\log n)$

**Space complexity:**  $O(1)$

## 9. Search in 2D Matrix

### Code Solution

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int top=0;
        int bot=matrix.length-1;
        while(top <= bot){
            int mid = (top + bot)/2;
            if(matrix[mid][0] < target && matrix[mid][matrix[mid].length - 1] > target){
                break;
            }else if(matrix[mid][0] > target){
                bot = mid-1;
            }else{
                top = mid + 1;
            }
        }
        int row =(top+bot)/2;
        int left = 0;
        int right = matrix[row].length-1;
        while(left<=right){
            int mid =(left+right)/2;
            if (matrix[row][mid] == target){
                return true;
            }else if(matrix[row][mid] > target){
                right = mid-1;
            }else{
                left = mid + 1;
            }
        }
        return false;
    }
}
```



}

The screenshot shows a LeetCode submission interface. On the left, the 'Accepted' status is confirmed with a runtime of 0 ms and memory of 41.90 MB. The right pane displays the Java code for the 'searchMatrix' function, which uses a binary search approach to find a target in a sorted matrix.

```
1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         int top = 0;
4         int bot = matrix.length - 1;
5         while (top <= bot) {
6             int mid = (top + bot) / 2;
7             if (matrix[mid][0] < target && matrix[mid][matrix[mid].length - 1] > target) {
8                 break;
9             } else if (matrix[mid][0] > target) {
10                bot = mid - 1;
11            } else {
12                top = mid + 1;
13            }
14        }
15        int row = (top + bot) / 2;
16        int left = 0;
17        int right = matrix[row].length - 1;
18        while (left <= right) {
19            int mid = (left + right) / 2;
20            if (matrix[row][mid] == target) {
21                return true;
22            } else if (matrix[row][mid] > target) {
23                right = mid - 1;
24            }
25        }
26        return false;
27    }
28 }
```

**Time complexity:**  $O(\log(m*n))$

**Space complexity:**  $O(1)$

## 10. Find Peak Element

### Code Solution

```
class Solution {
    public int findPeakElement(int[] nums) {
        int left=0;
        int right=nums.length-1;
        while(left<right){
            int mid=(left+right)/2;
            if (nums[mid]>nums[mid+1]){
                right=mid;
            }else{
                left=mid + 1;
            }
        }
        return left;
    }
}
```

### Output

The screenshot shows a LeetCode submission interface for the 'Find Peak Element' problem. The left pane shows the submission is 'Accepted' with a runtime of 0 ms and memory of 42.32 MB. The right pane displays the Java code for the 'findPeakElement' function, which uses a binary search to find a peak element in an array.

```
1 class Solution {
2     public int findPeakElement(int[] nums) {
3         int left=0;
4         int right=nums.length-1;
5         while(left<right){
6             int mid=(left+right)/2;
7             if (nums[mid]>nums[mid+1]){
8                 right=mid;
9             }else{
10                left=mid + 1;
11            }
12        }
13        return left;
14    }
15 }
```

**Time complexity:**  $O(\log n)$

**Space complexity:**  $O(1)$

## 11. Search in rotated sorted array

### Code Solution

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

### Output

The screenshot displays a code editor interface with two main panes. The left pane shows the submission status for the problem 'Search in rotated sorted array'. It indicates that the solution is 'Accepted' and was submitted by 'Dhejan' on Nov 21, 2024, at 20:13. The runtime is 0 ms, which beats 100.00% of other submissions, and the memory usage is 42.27 MB, which beats 21.99%. The right pane shows the Java code for the solution, which is a class named 'Solution' with a method 'search' that implements a binary search algorithm on a rotated sorted array. The code is as follows:

```
1 class Solution {
2     public int search(int[] nums, int target) {
3         int l=0;
4         int r=nums.length-1;
5         while(l<=r){
6             int mid=l+(r-l)/2;
7             if(nums[mid]==target)return mid;
8             else if(nums[l]<=nums[mid]){
9                 if(nums[l]<=target && target<=nums[mid]){
10                     r=mid-1;
11                 }
12                 else{
13                     l=mid+1;
14                 }
15             }
16             else{
17                 if(nums[mid]<=target && target<=nums[r]){
18                     l=mid+1;
19                 }
20                 else{
21                     r=mid-1;
22                 }
23             }
24         }
25         return -1;
26     }
27 }
```

**Time complexity:  $O(\log n)$**

**Space complexity:  $O(1)$**

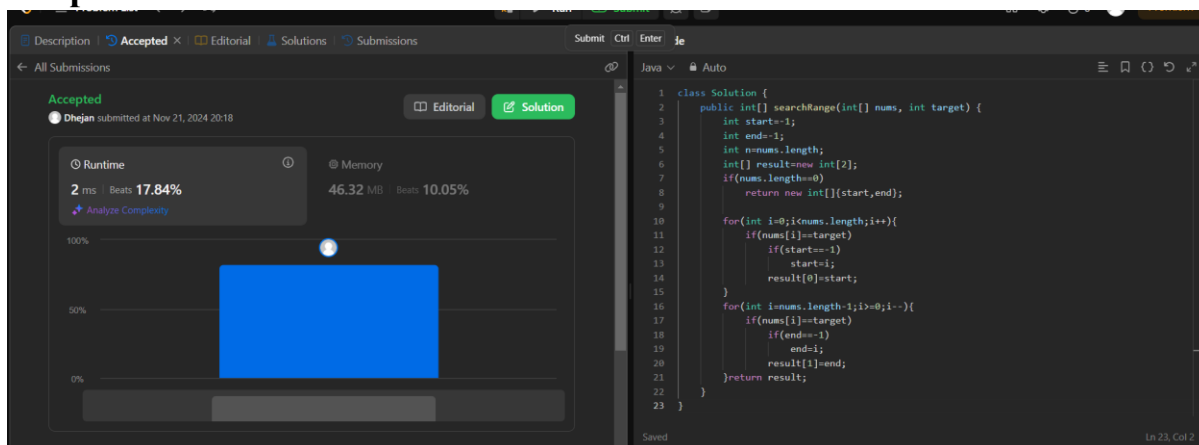
## 12. Find the first and last of element in the sorted array

### Code Solution

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int start=-1;
        int end=-1;
        int n=nums.length;
        int[] result=new int[2];
        if(nums.length==0)
            return new int[] {start,end};

        for(int i=0;i<nums.length;i++){
            if(nums[i]==target)
                if(start==-1)
                    start=i;
            result[0]=start;
        }
        for(int i=nums.length-1;i>=0;i--){
            if(nums[i]==target)
                if(end==-1)
                    end=i;
            result[1]=end;
        }return result;
    }
}
```

### Output



**Time complexity:  $O(n)$**

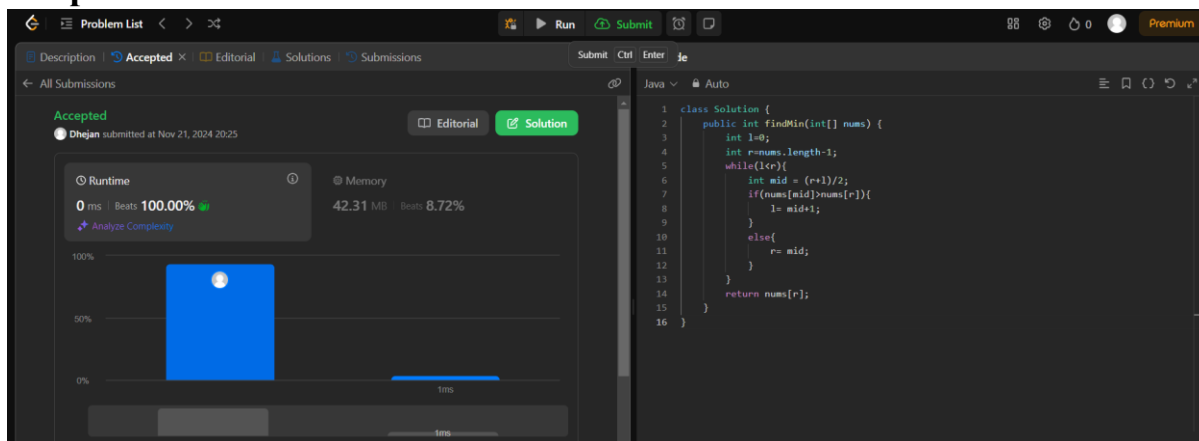
**Space complexity:  $O(1)$**

### 13. Find the minimum in the rotated sorted array

#### Code Solution

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

#### Output



**Time complexity:**  $O(\log n)$

**Space complexity:**  $O(1)$

### 14. Valid Parantheses

#### Code Solution

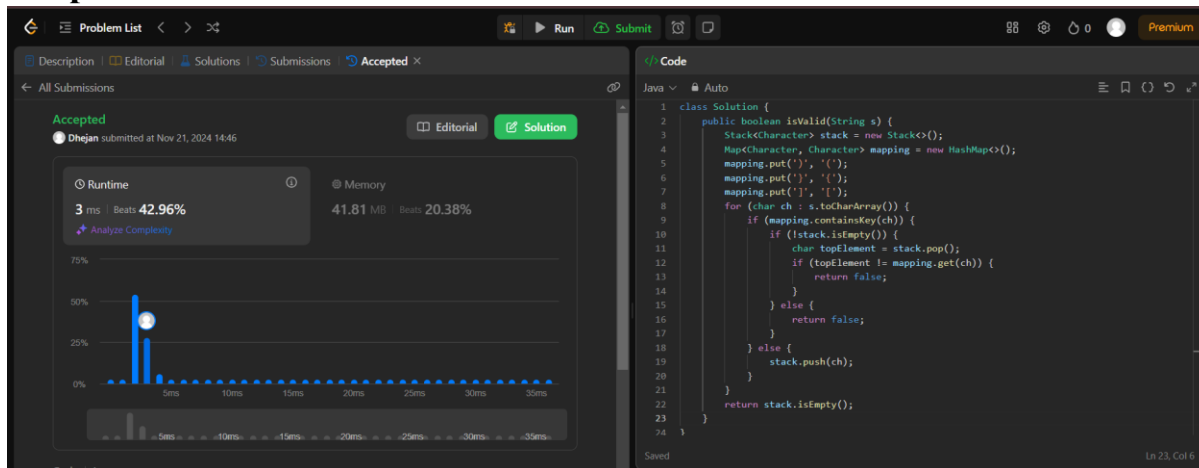
```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        Map<Character, Character> mapping = new HashMap<>();
        mapping.put('(', ')');
        mapping.put('{', '}');
        mapping.put('[', ']');
        for (char ch : s.toCharArray()) {
            if (mapping.containsKey(ch)) {
```

```

        if (!stack.isEmpty()) {
            char topElement = stack.pop();
            if (topElement != mapping.get(ch)) {
                return false;
            }
        } else {
            return false;
        }
    } else {
        stack.push(ch);
    }
}
return stack.isEmpty();
}
}

```

## Output



**Time complexity:  $O(n)$**

**Space complexity:  $O(n)$**

## 15. Simplify Path

### Code Solution

```

class Solution {
    public String simplifyPath(String path) {
        Stack<String> stack = new Stack<>();
        String[] directories = path.split("/");
        for (String dir : directories) {
            if (dir.equals(".") || dir.isEmpty()) {
                continue;
            } else if (dir.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop();
                }
            } else {

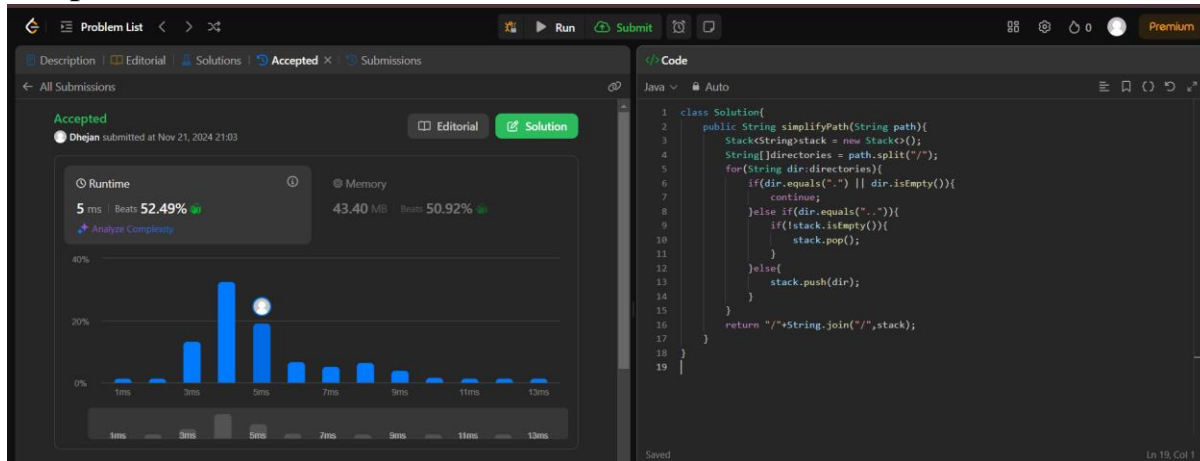
```

```

        stack.push(dir);
    }
}
return "/" + String.join("/", stack);
}
}

```

## Output



**Time complexity:**  $O(n)$

**Space complexity:**  $O(n)$

## 16. Min stack

### Code Solution

```

class MinStack {
    private List<int[]> st;

    public MinStack() {
        st = new ArrayList<>();
    }

    public void push(int val) {
        int[] top;
        if (st.isEmpty()) {
            top = new int[] { val, val };
        } else {
            top = st.get(st.size() - 1);
        }
        int min_val = top[1];
        if (min_val > val) {
            min_val = val;
        }
        st.add(new int[] { val, min_val });
    }
}

```

```

public void pop(){
    st.remove(st.size()-1);
}

public int top(){
    if(st.isEmpty()){
        return -1;
    }else{
        return st.get(st.size()-1)[0];
    }
}

public int getMin(){
    if(st.isEmpty()){
        return -1;
    }else{
        return st.get(st.size()-1)[1];
    }
}
}

```

/\*\*

\* Your MinStack object will be instantiated and called as such:

\* MinStack obj = new MinStack();

\* obj.push(val);

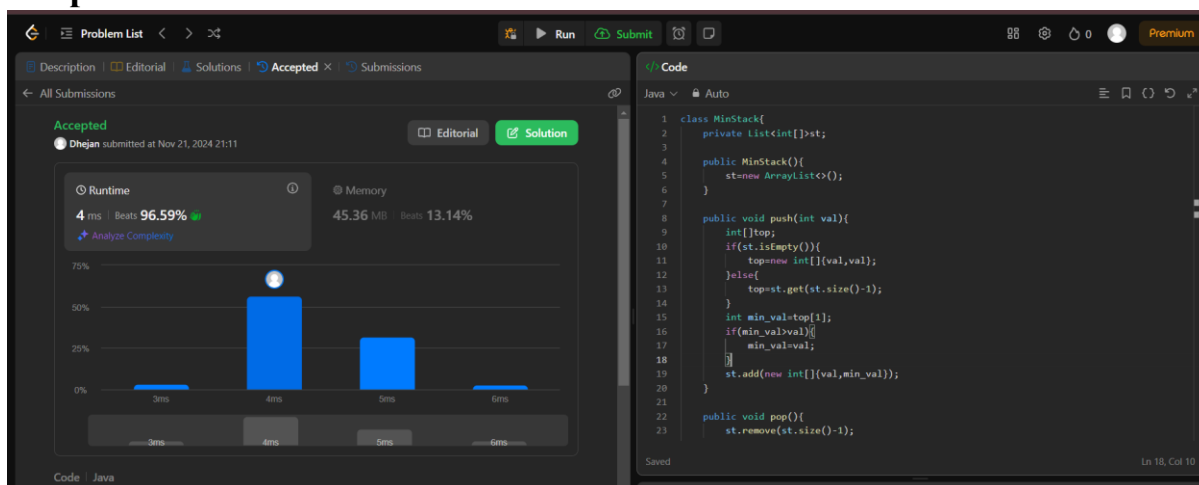
\* obj.pop();

\* int param\_3 = obj.top();

\* int param\_4 = obj.getMin();

\*/

## Output



**Time complexity:  $O(1)$**

**Space complexity:  $O(n)$**

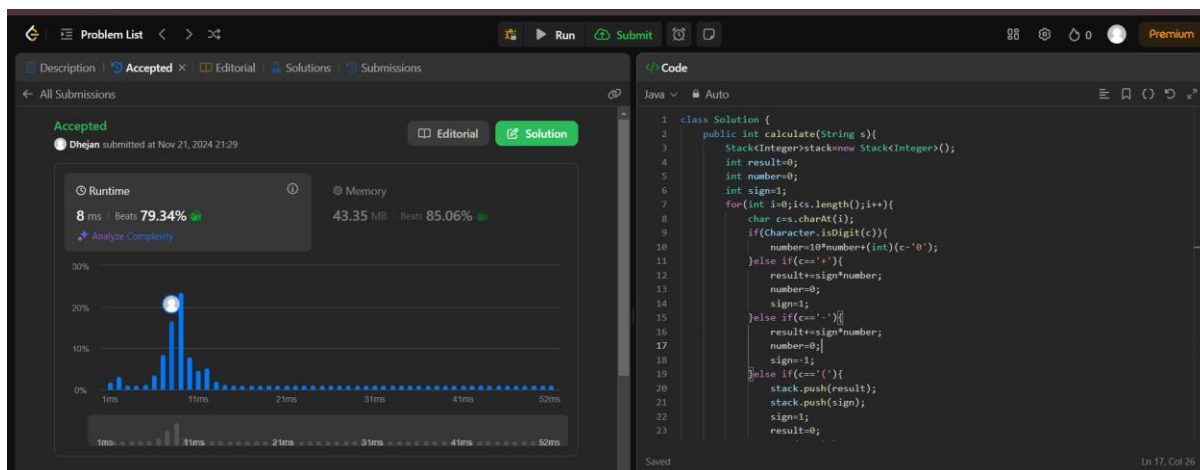
## 17. Basic Calculator

### *Code Solution*

```
class Solution {
    public int calculate(String s){
        Stack<Integer>stack=new Stack<Integer>();
        int result=0;
        int number=0;
        int sign=1;
        for(int i=0;i<s.length();i++){
            char c=s.charAt(i);
            if(Character.isDigit(c)){
                number=10*number+(int)(c-'0');
            }else if(c=='+'){
                result+=sign*number;
                number=0;
                sign=1;
            }else if(c=='-'){
                result+=sign*number;
                number=0;
                sign=-1;
            }else if(c=='('){
                stack.push(result);
                stack.push(sign);
                sign=1;
                result=0;
            }else if(c==')'){
                result+=sign*number;
                number=0;
                result*=stack.pop();
                result+=stack.pop();
            }
        }
        if(number!=0) result+=sign*number;
        return result;
    }
}
```

### **Output**





***Time complexity:  $O(n)$***

***Space complexity:  $O(n)$***