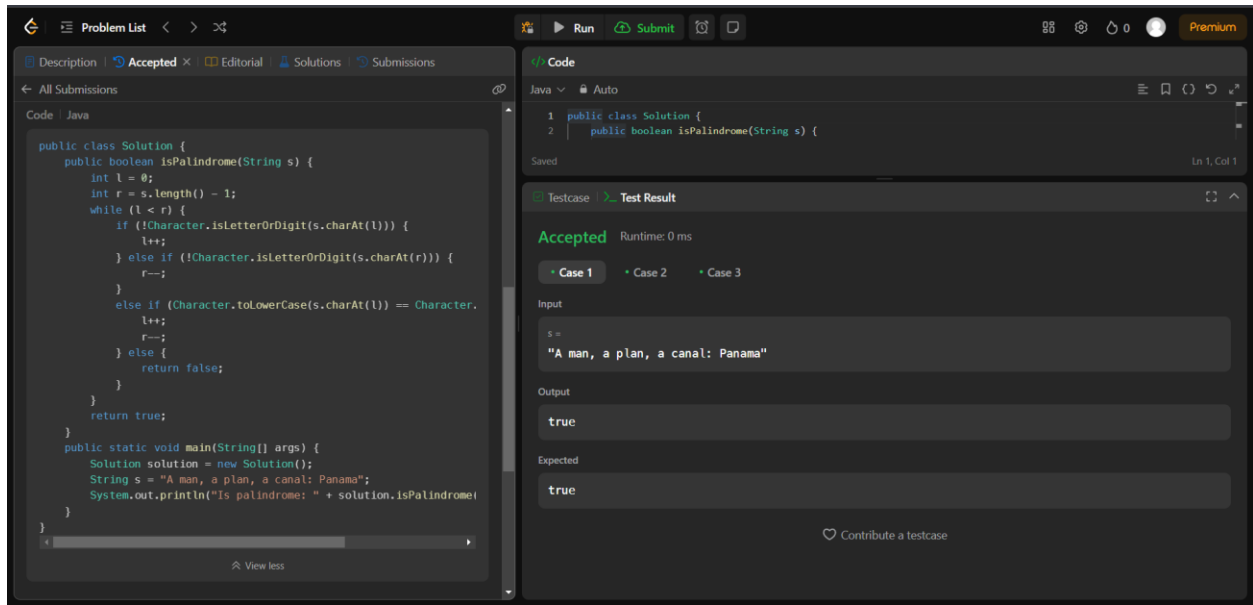


**1.VALID PALINDROME:****Code:**

```
public class Solution {
    public boolean isPalindrome(String s) {
        int l = 0;
        int r = s.length() - 1;
        while (l < r) {
            if (!Character.isLetterOrDigit(s.charAt(l))) {
                l++;
            } else if (!Character.isLetterOrDigit(s.charAt(r))) {
                r--;
            }
            else if (Character.toLowerCase(s.charAt(l)) ==
Character.toLowerCase(s.charAt(r))) {
                l++;
                r--;
            } else {
                return false;
            }
        }
        return true;
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        String s = "A man, a plan, a canal: Panama";
        System.out.println("Is palindrome: " + solution.isPalindrome(s));
    }
}
```

**OUTPUT:**



## 2.IS SUBSEQUENCE:

### Code:

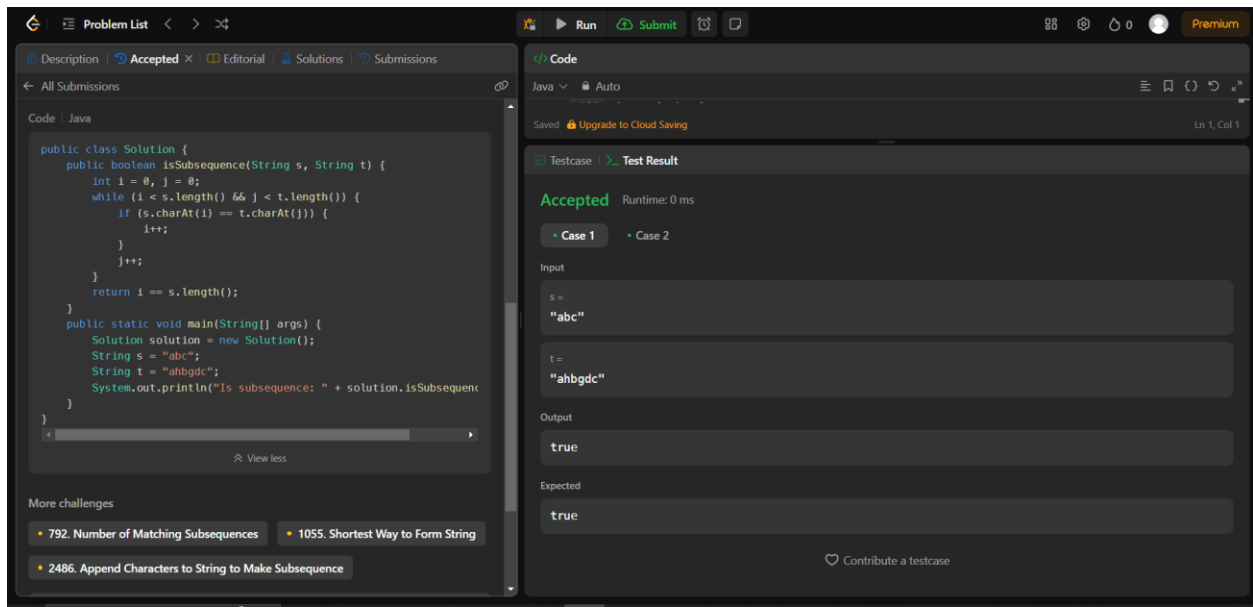
```

public 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();
    }
}

public static void main(String[] args) {
    Solution solution = new Solution();
    String s = "abc";
    String t = "ahbgdc";
    System.out.println("Is subsequence: " + solution.isSubsequence(s, t));
}

```

### OUTPUT:

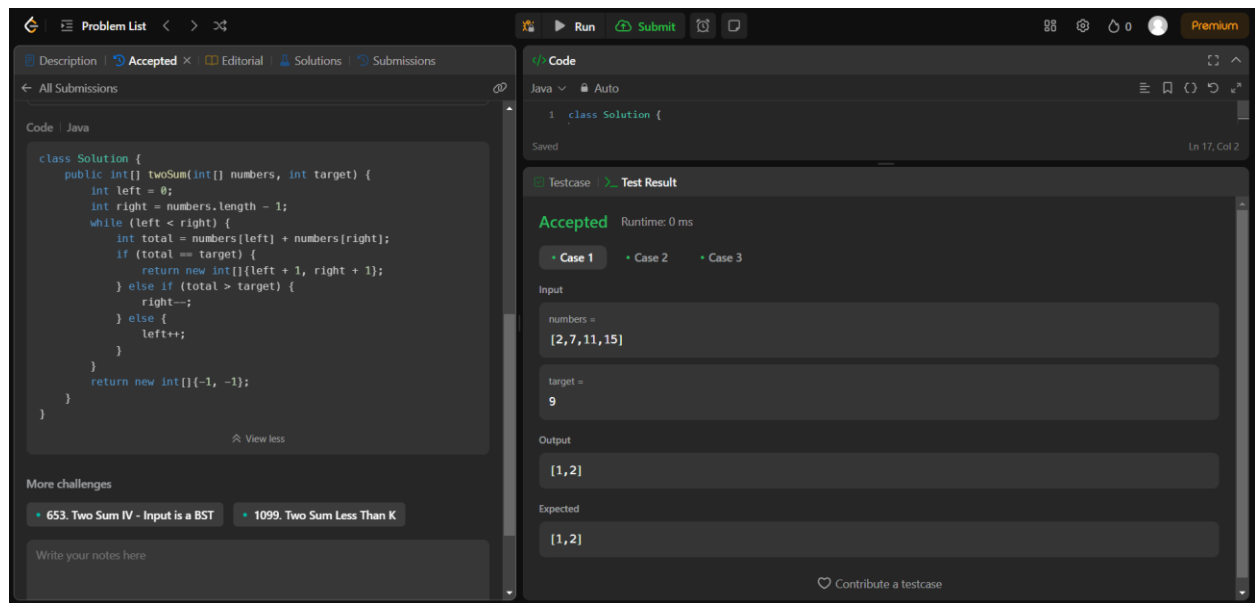


### 3.TWO SUM II -INPUT ARRAY IS SORTED

#### CODE:

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int left = 0;
        int right = numbers.length - 1;
        while (left < right) {
            int total = numbers[left] + numbers[right];
            if (total == target) {
                return new int[]{left + 1, right + 1};
            } else if (total > target) {
                right--;
            } else {
                left++;
            }
        }
        return new int[]{-1, -1};
    }
}
```

#### OUTPUT:

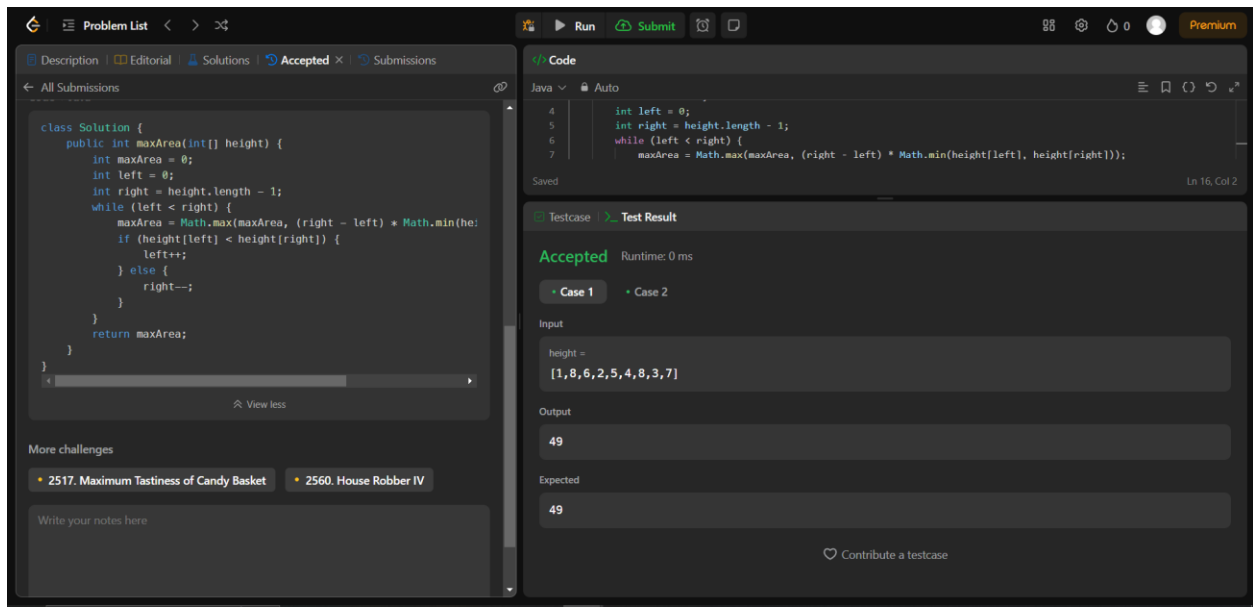


#### 4.CONTAINER WITH MOST WATER:

##### CODE:

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

##### OUTPUT:



### 5.3SUM:

#### CODE:

```

import java.util.*;
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> l = new ArrayList<>();
        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;
            int res = 0 - nums[i];
            int start = i + 1;
            int end = nums.length - 1;
            while (start < end) {
                int sum = nums[start] + nums[end];
                if (sum == res) {
                    l.add(Arrays.asList(nums[i], nums[start], nums[end]));
                    while (start < end && nums[start] == nums[start + 1]) start++;
                    while (start < end && nums[end] == nums[end - 1]) end--;
                    start++;
                    end--;
                } else if (sum > res) {
                    end--;
                } else {
                    start++;
                }
            }
        }
        return l;
    }
}

```

```

        start++;
    }
}
}
return l;
}
}

```

## OUTPUT:

```

import java.util.*;
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> l = new ArrayList<>();
        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;
            int res = 0 - nums[i];
            int start = i + 1;
            int end = nums.length - 1;
            while (start < end) {
                int sum = nums[start] + nums[end];
                if (sum == res) {
                    l.add(Arrays.asList(nums[i], nums[start], nums[end]));
                    while (start < end && nums[start] == nums[start + 1]) start++;
                    while (start < end && nums[end] == nums[end - 1]) end--;
                } else if (sum > res) {
                    end--;
                } else {
                    start++;
                }
            }
        }
        return l;
    }
}

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

Output

[[[-1, -1, 2], [-1, 0, 1]]]

Expected

[[[-1, -1, 2], [-1, 0, 1]]]

Contribute a testcase

## 6.MINIMUM SIZE SUBARRAY SUM:

### CODE:

```

class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int minLen = Integer.MAX_VALUE;
        int left = 0;
        int curSum = 0;
        for (int right = 0; right < nums.length; right++) {
            curSum += nums[right];
            while (curSum >= target) {
                if (right - left + 1 < minLen) {
                    minLen = right - left + 1;
                }
                curSum -= nums[left];
                left++;
            }
        }
        return minLen == Integer.MAX_VALUE ? 0 : minLen;
    }
}

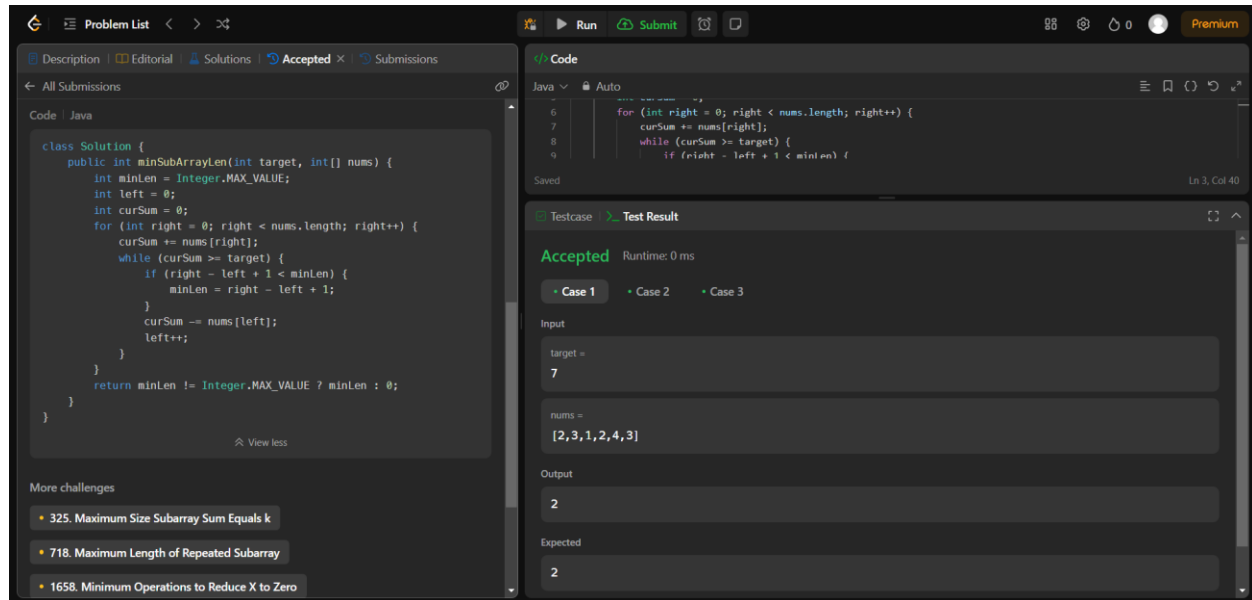
```

```

        left++;
    }
}
return minLen != Integer.MAX_VALUE ? minLen : 0;
}
}

```

## OUTPUT:



## 7.LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS:

### CODE:

```

class Solution {
    public int lengthOfLongestSubstring(String s) {
        int left = 0;
        int maxLength = 0;
        HashSet<Character> charSet = new HashSet<>();
        for (int right = 0; right < s.length(); right++) {
            while (charSet.contains(s.charAt(right))) {
                charSet.remove(s.charAt(left));
                left++;
            }
            charSet.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }
    }
}

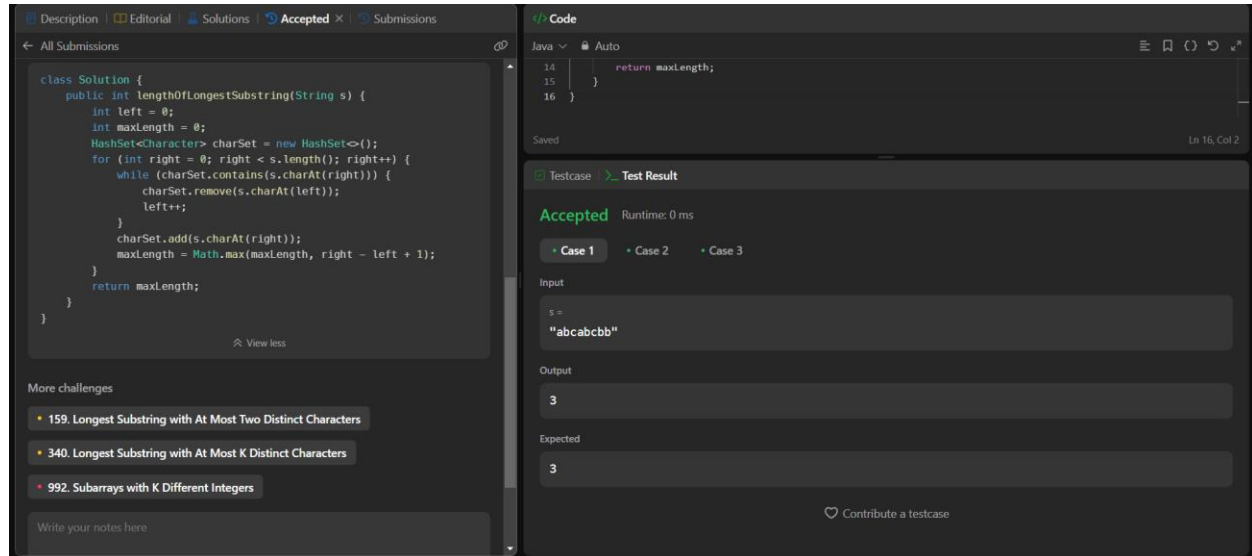
```

```

        return maxLength;
    }
}

```

**OUTPUT:**



## 8.SUBSTRING WITH CONCATENATION OF ALL WORDS:

**CODE:**

```

class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        final Map<String, Integer> counts = new HashMap<>();
        for (final String word : words) {
            counts.put(word, counts.getDefault(word, 0) + 1);
        }
        final List<Integer> indexes = new ArrayList<>();
        final int n = s.length(), num = words.length, len = words[0].length();
        for (int i = 0; i < n - num * len + 1; i++) {
            final Map<String, Integer> seen = new HashMap<>();
            int j = 0;
            while (j < num) {
                final String word = s.substring(i + j * len, i + (j + 1) * len);
                if (counts.containsKey(word)) {
                    seen.put(word, seen.getDefault(word, 0) + 1);
                    if (seen.get(word) > counts.getDefault(word, 0)) {
                        break;
                    }
                }
                j++;
            }
            if (j == num) {
                indexes.add(i);
            }
        }
    }
}

```



```

        break;
    }
    j++;
}
if (j == num) {
    indexes.add(i);
}
}
return indexes;
}
}

```

## OUTPUT:

The screenshot displays a LeetCode submission page. On the left, the 'Accepted' tab is selected, showing a Java solution for the problem 'Longest Substring Without Repeating Characters'. The code uses a sliding window approach with a HashSet to track unique characters. On the right, the 'Testcase' tab is active, showing the test result for 'Case 1'. The input string is 'abcabcbb', and the output is '3', which matches the expected result. The runtime is 0 ms.

## 9.MINIMUM WINDOW SUBSTRING:

### CODE:

```

class Solution {
    public String minWindow(String s, String t) {
        if (s.length() < t.length()) {
            return "";
        }
        Map<Character, Integer> charCount = new HashMap<>();
        for (char ch : t.toCharArray()) {
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
        }
        int targetCharsRemaining = t.length();
        int[] minWindow = {0, Integer.MAX_VALUE};
    }
}

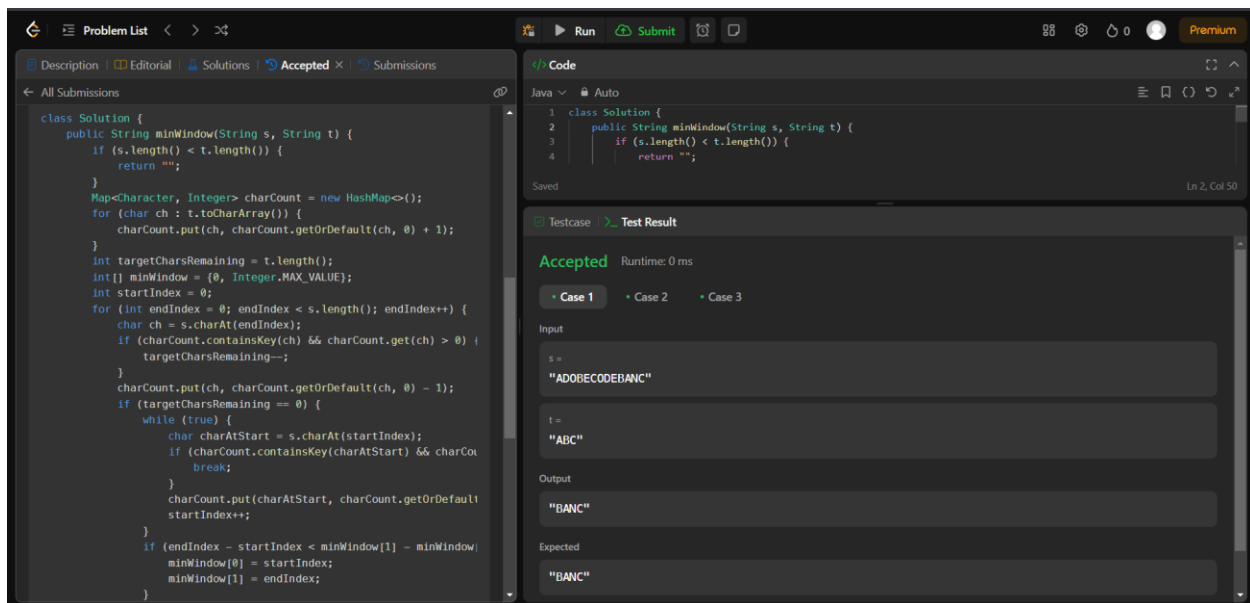
```

```

int startIndex = 0;
for (int endIndex = 0; endIndex < s.length(); endIndex++) {
    char ch = s.charAt(endIndex);
    if (charCount.containsKey(ch) && charCount.get(ch) > 0) {
        targetCharsRemaining--;
    }
    charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);
    if (targetCharsRemaining == 0) {
        while (true) {
            char charAtStart = s.charAt(startIndex);
            if (charCount.containsKey(charAtStart) && charCount.get(charAtStart) ==
0) {
                break;
            }
            charCount.put(charAtStart, charCount.getOrDefault(charAtStart, 0) + 1);
            startIndex++;
        }
        if (endIndex - startIndex < minWindow[1] - minWindow[0]) {
            minWindow[0] = startIndex;
            minWindow[1] = endIndex;
        }
        charCount.put(s.charAt(startIndex),
charCount.getOrDefault(s.charAt(startIndex), 0) + 1);
        targetCharsRemaining++;
        startIndex++;
    }
}
return minWindow[1] >= s.length() ? "" : s.substring(minWindow[0],
minWindow[1] + 1);
}
}

```

**OUTPUT:**

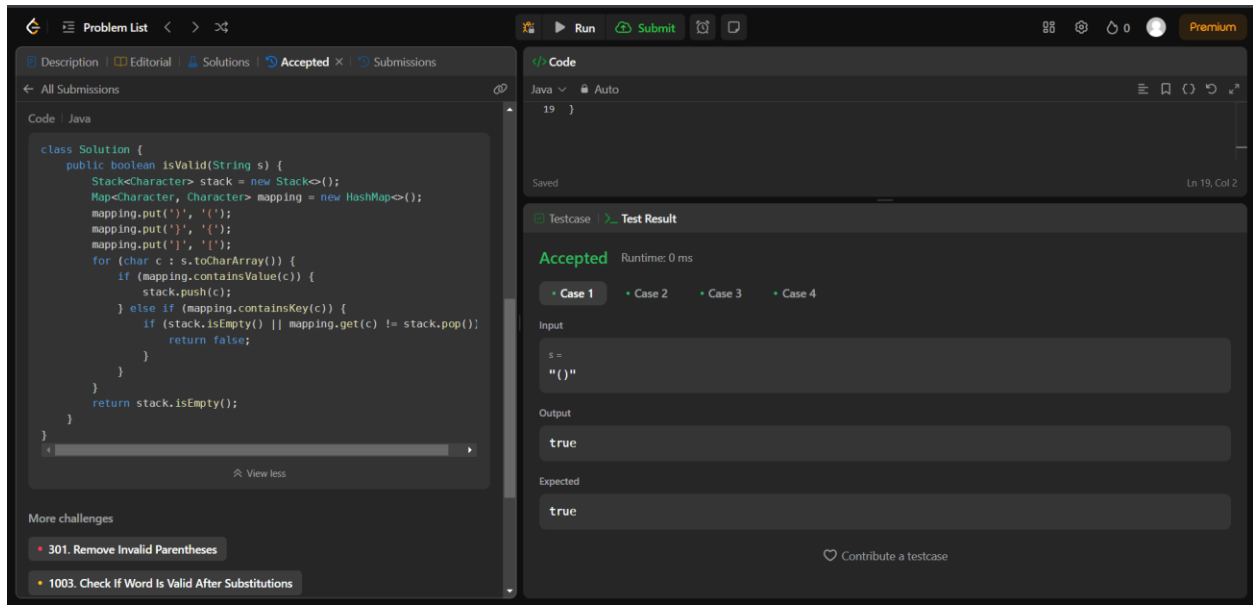


## 10.VALID PARENTHESES:

### CODE:

```
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 c : s.toCharArray()) {
            if (mapping.containsKey(c)) {
                stack.push(c);
            } else if (mapping.containsValue(c)) {
                if (stack.isEmpty() || mapping.get(c) != stack.pop()) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
}
```

### OUTPUT:

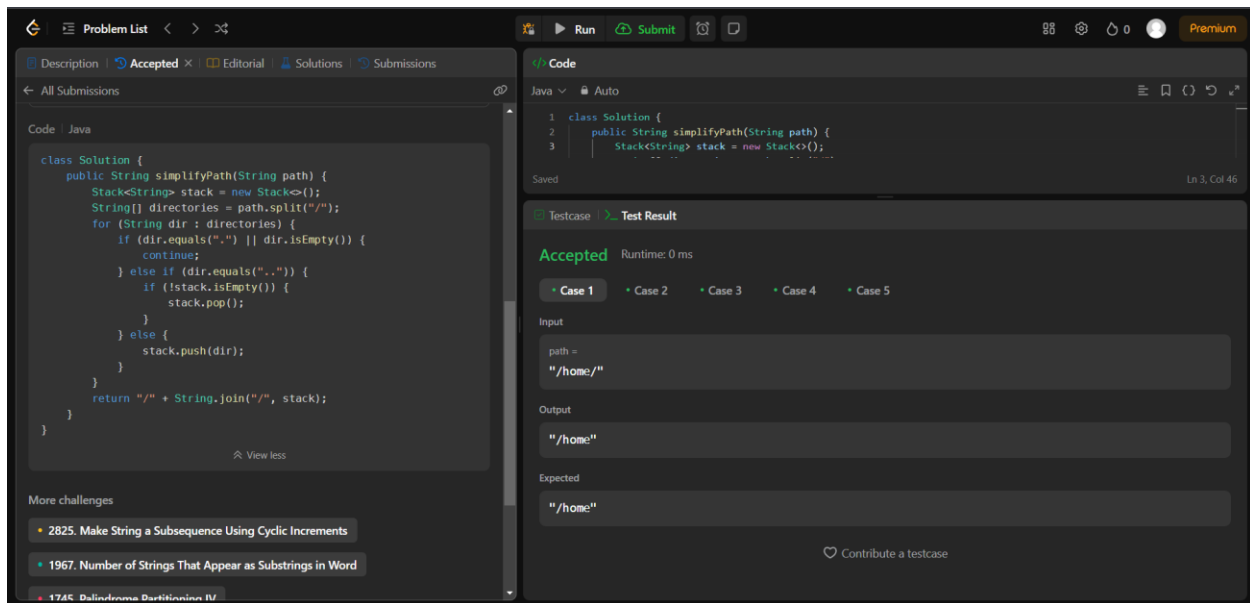


## 11.SIMPLIFY PATH:

### CODE:

```
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:



## 12.MIN STACK:

### CODE:

```
class MinStack {
    private List<int[]> st;
    public MinStack() {
        st = new ArrayList<>();
    }
    public void push(int val) {
        int[] top = st.isEmpty() ? new int[]{ val, val } : 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() {
        return st.isEmpty() ? -1 : st.get(st.size() - 1)[0];
    }
    public int getMin() {
        return st.isEmpty() ? -1 : st.get(st.size() - 1)[1];
    }
}
```

```
}
```

## OUTPUT:

```
class MinStack {
    private List<int> st;
    public MinStack() {
        st = new ArrayList<>();
    }
    public void push(int val) {
        int[] top = st.isEmpty() ? new int[]{val, val} : 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() {
        return st.isEmpty() ? -1 : st.get(st.size() - 1)[0];
    }
    public int getMin() {
        return st.isEmpty() ? -1 : st.get(st.size() - 1)[1];
    }
}
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1

Input

["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]

Output

[null, null, null, null, -3, null, 0, -2]

Expected

[null, null, null, null, -3, null, 0, -2]

## 13.EVALUATE REVERSE POLISH NOTATION:

### CODE:

```
class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();
        for (String c : tokens) {
            if (c.equals("+")) {
                stack.push(stack.pop() + stack.pop());
            } else if (c.equals("-")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first - second);
            } else if (c.equals("*")) {
                stack.push(stack.pop() * stack.pop());
            } else if (c.equals("/")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first / second);
            } else {
                stack.push(Integer.parseInt(c));
            }
        }
    }
}
```

```

        return stack.peek();
    }
}

```

**OUTPUT:**

The screenshot shows a code editor with a Java solution for a calculator problem. The code uses a stack to handle operator precedence. The test result shows 'Accepted' for Case 1 with input tokens ['2', '1', '+', '3', '\*'] and output 9.

```

class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();
        for (String c : tokens) {
            if (c.equals("+")) {
                stack.push(stack.pop() + stack.pop());
            } else if (c.equals("-")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first - second);
            } else if (c.equals("*")) {
                stack.push(stack.pop() * stack.pop());
            } else if (c.equals("/")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first / second);
            } else {
                stack.push(Integer.parseInt(c));
            }
        }
        return stack.peek();
    }
}

```

Testcase: Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: tokens = ["2", "1", "+", "3", "\*"]

Output: 9

Expected: 9

## 14.BASIC CALCULATOR:

**CODE:**

```

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

```

```

        signValue = 1;
    } else if (c == ')') {
        result += signValue * number;
        result *= operationsStack.pop();
        result += operationsStack.pop();
        number = 0;
    }
}
return result + number * signValue;
}
}

```

## OUTPUT:

The screenshot displays a code editor with a Java solution for a calculator problem. The code uses a stack to handle parentheses and operators. The test result shows 'Accepted' with a runtime of 0 ms for the input '1 + 1'.

```

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

```

**Testcase** **Test Result**

**Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

Input

s = "1 + 1"

Output

2

Expected

2

Contribute a testcase

## 15.SEARCH INSERT POSITION:

### CODE:

```

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

```

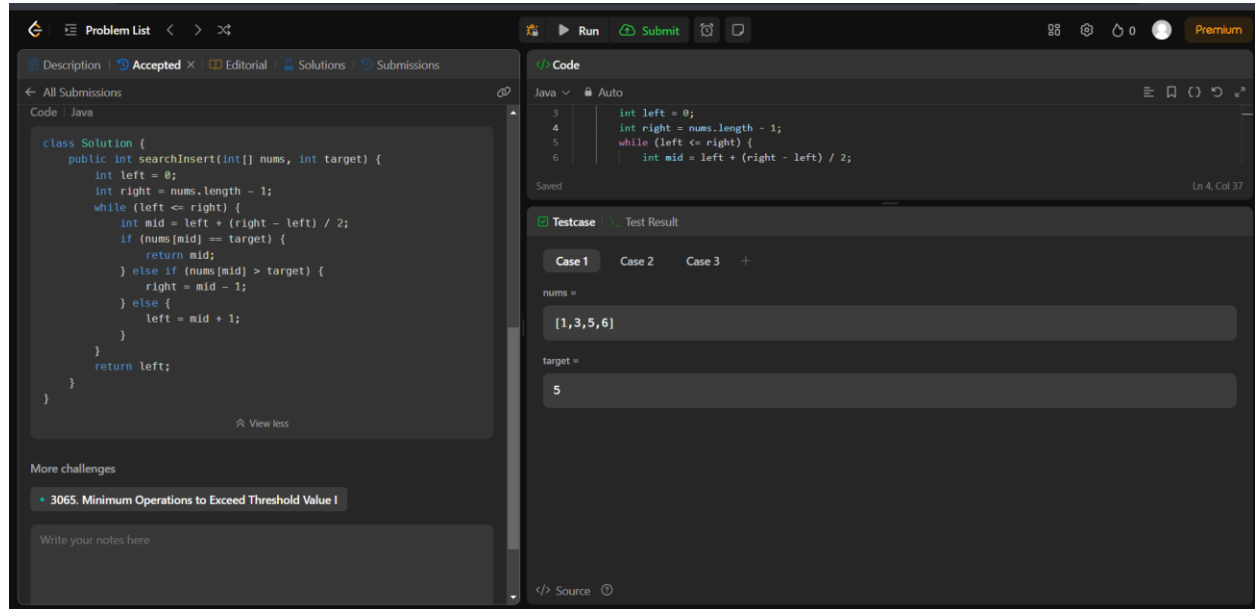


```

    } else {
        left = mid + 1;
    }
}
return left;
}
}

```

## OUTPUT:



## 16.SEARCH A 2D MATRIX:

### CODE:

```

class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length;
        int n = matrix[0].length;
        int i=0;
        int j=n-1;
        while(i<m && j>=0){
            if(matrix[i][j]==target) return true;
            if(matrix[i][j]>target){
                j--;
            }
            else{
                i++;
            }
        }
    }
}

```

```

    }
    return false;
}
}

```

**OUTPUT:**

The screenshot shows a code editor with a Java solution for a matrix search problem. The code defines a class `Solution` with a method `searchMatrix` that takes a 2D matrix and a target value. The method uses a binary search approach to find the target. The test case shows a matrix and a target value.

```

class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length;
        int n = matrix[0].length;
        int i = 0;
        int j = n - 1;
        while (i < m && j >= 0) {
            if (matrix[i][j] == target) return true;
            if (matrix[i][j] > target) {
                j--;
            } else {
                i++;
            }
        }
        return false;
    }
}

```

More challenges

- 240. Search a 2D Matrix II
- 2468. Split Message Based on Limit

Write your notes here

Testcase

Case 1 Case 2 +

matrix =

```
[[1,3,5,7],[10,11,16,20],[23,30,34,60]]
```

target =

```
3
```

**17.FIND PEAK ELEMENT:**

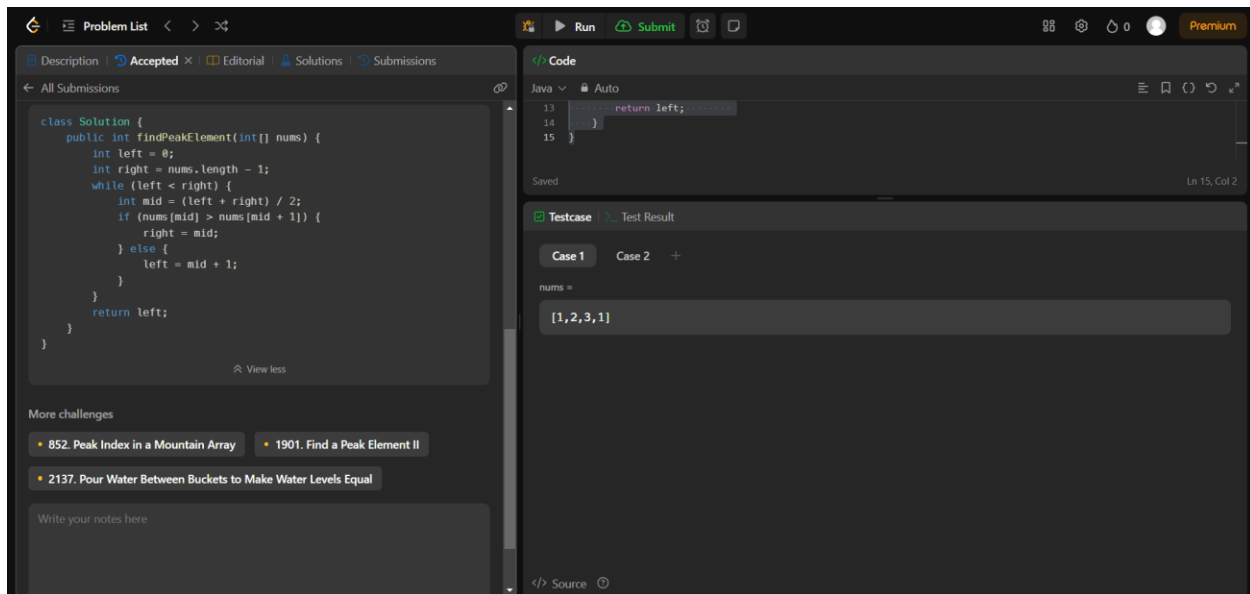
**CODE:**

```

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:**



## 18.SEARCH IN ROTATED SORTED ARRAY:

### CODE:

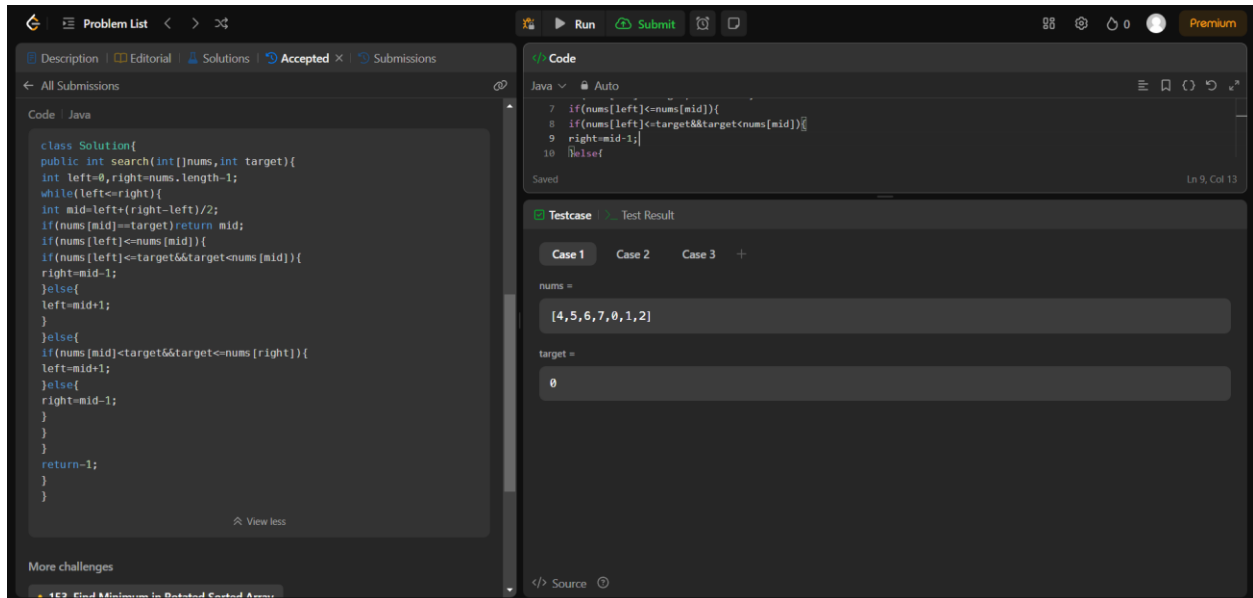
```

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

```

```
}
```

## OUTPUT:



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

Testcase

Case 1 Case 2 Case 3 +

nums =

[4, 5, 6, 7, 0, 1, 2]

target =

0

## 19.FIND FIRST AND LAST POSITION OF ELEMENT SORTED IN ARRAY:

### CODE:

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};
        int left = binarySearch(nums, target, true);
        int right = binarySearch(nums, target, false);
        result[0] = left;
        result[1] = right;
        return result;
    }
    private int binarySearch(int[] nums, int target, boolean isSearchingLeft) {
        int left = 0;
        int right = nums.length - 1;
        int idx = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > target) {
                right = mid - 1;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                idx = mid;
                if (isSearchingLeft) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
        }
        return idx;
    }
}
```

```

    } else {
        idx = mid;
        if (isSearchingLeft) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
}
return idx;
}
}

```

**OUTPUT:**

The screenshot shows a code editor with a Java solution for finding the minimum in a rotated sorted array. The code is in a class Solution with methods searchRange and binarySearch. The test case shows nums = [5, 7, 7, 8, 8, 10] and target = 8.

```

class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};
        int left = binarySearch(nums, target, true);
        int right = binarySearch(nums, target, false);
        result[0] = left;
        result[1] = right;
        return result;
    }
    private int binarySearch(int[] nums, int target, boolean isSearch) {
        int left = 0;
        int right = nums.length - 1;
        int idx = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > target) {
                right = mid - 1;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                idx = mid;
                if (isSearchingLeft) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
        }
        return idx;
    }
}

```

Testcase: Case 1 Case 2 Case 3 +

nums = [5, 7, 7, 8, 8, 10]

target = 8

**20.FIND MINIMUM IN ROTATED SORTED ARRAY:**

**CODE:**

```

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

```

```

        left = mid + 1;
    }
}
return nums[left];
}
}

```

**OUTPUT:**

The screenshot displays the LeetCode submission interface for the problem "154. Find Minimum in Rotated Sorted Array II". The interface is divided into several sections:

- Problem List:** Shows the problem title and a "Premium" badge.
- Description:** Contains the problem statement and a "View less" link.
- Accepted:** A green badge indicating the solution is accepted.
- Editorial:** A link to the editorial.
- Solutions:** A link to view other solutions.
- Submissions:** A link to view the submission history.
- Code:** A section showing the Java code used for the solution. The code is as follows:
 

```

1 class Solution {
2     public int findMin(int[] nums) {
3         int left = 0;
4         int right = nums.length - 1;
5         while (left < right) {

```
- Testcase:** A section showing the test results. The status is "Accepted" with a runtime of 0 ms. The input is "nums = [3,4,5,1,2]" and the output is "1".
- Test Result:** A section showing the test results for different cases. Case 1 is selected, and the output is "1".
- More challenges:** A section showing other problems, including "154. Find Minimum in Rotated Sorted Array II".
- Write your notes here:** A text area for the user to write notes.