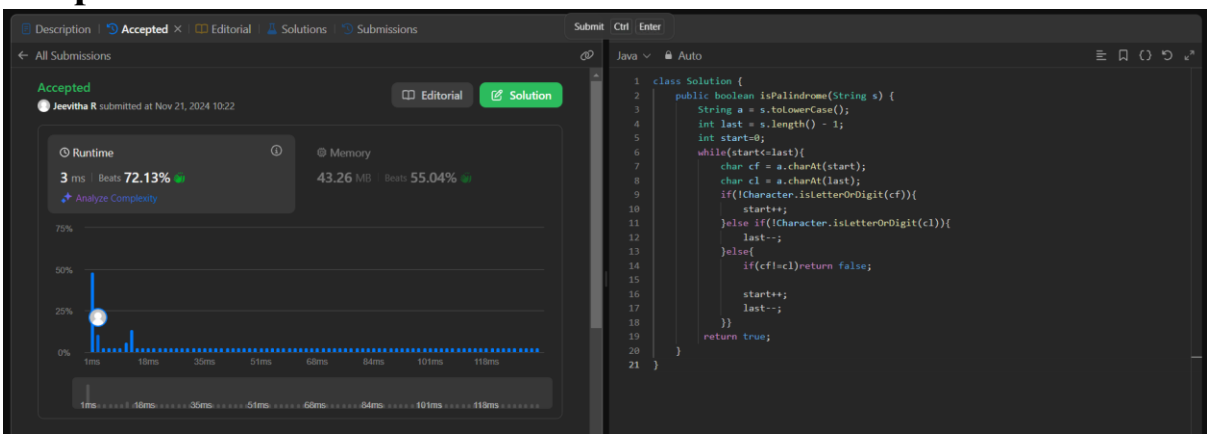
DSA PRACTICE QUESTIONS- DAY 9

NAME: Jeevitha R
REG NO: 22IT040
DATE: 21/11/2024

1. Valid Palindrome

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

                start++;
                last--;
            }}
         return true;
    }
}
```

Output



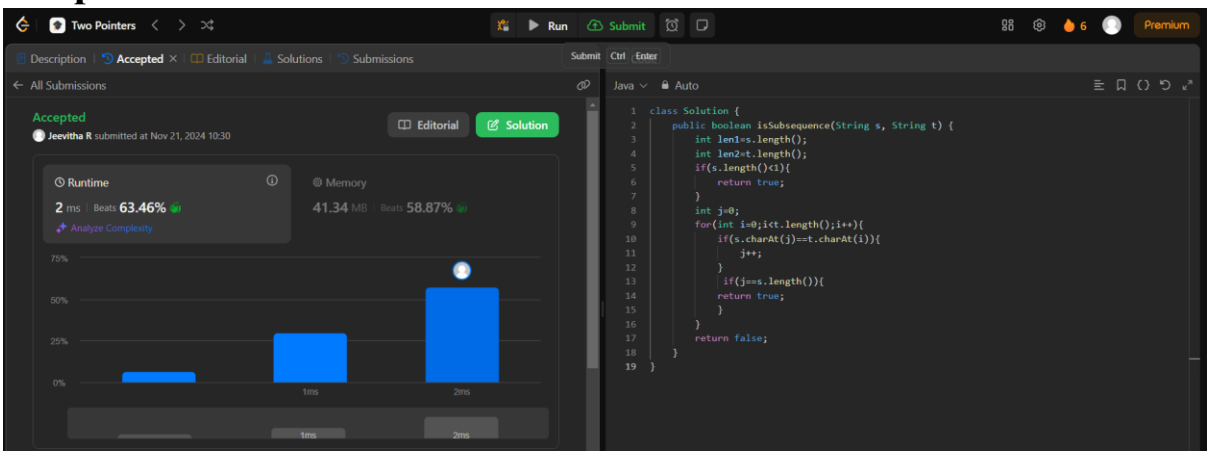
Time complexity: O(n)
Space complexity: O(1)

2. Is Subsequence

Code Solution

```
class Solution {
    public boolean isSubsequence(String s, String t) {
        int len1=s.length();
        int len2=t.length();
        if(s.length()<1){
            return true;
        }
        int j=0;
        for(int i=0;i<t.length();i++){
            if(s.charAt(j)==t.charAt(i)){
                j++;
            }
             if(j==s.length()){
            return true;
            }
        }
        return false;
    }
}
```

Output



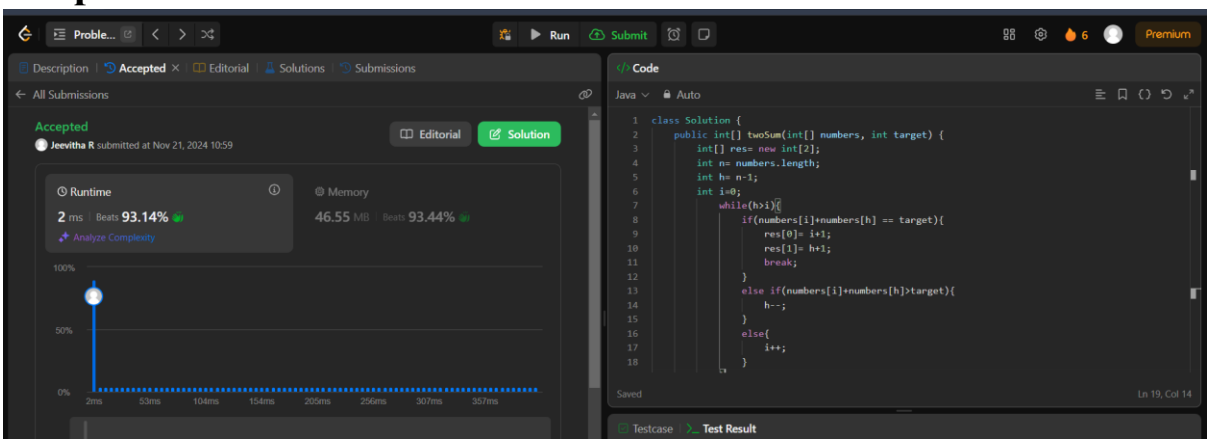
Time complexity: O(n)
Space complexity: O(1)

3. Two Sum II- Input Array is Sorted

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int[] res= new int[2];
        int n= numbers.length;
        int h= n-1;
        int i=0;
            while(h>i){
```

```
            if(numbers[i]+numbers[h] == target){
                res[0]= i+1;
                res[1]= h+1;
                break;
            }
            else if(numbers[i]+numbers[h]>target){
                h--;
            }
            else{
                i++;
            }
        }
        return res;
    }
}
```

Output



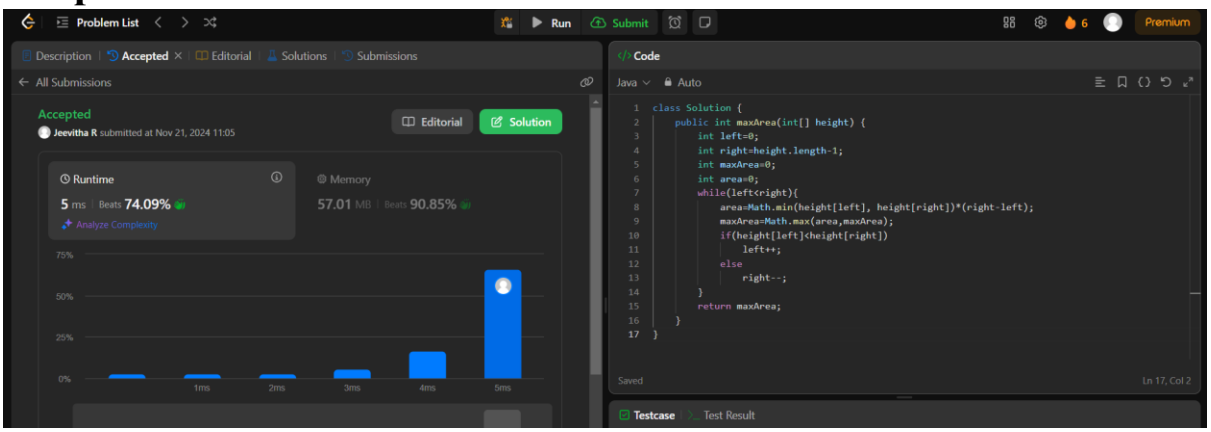
Time complexity: O(n)
Space complexity: O(1)

4. Container with most water

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

```
            else
                right--;
        }
        return maxArea;
    }
}
```

Output



Time complexity: O(n)

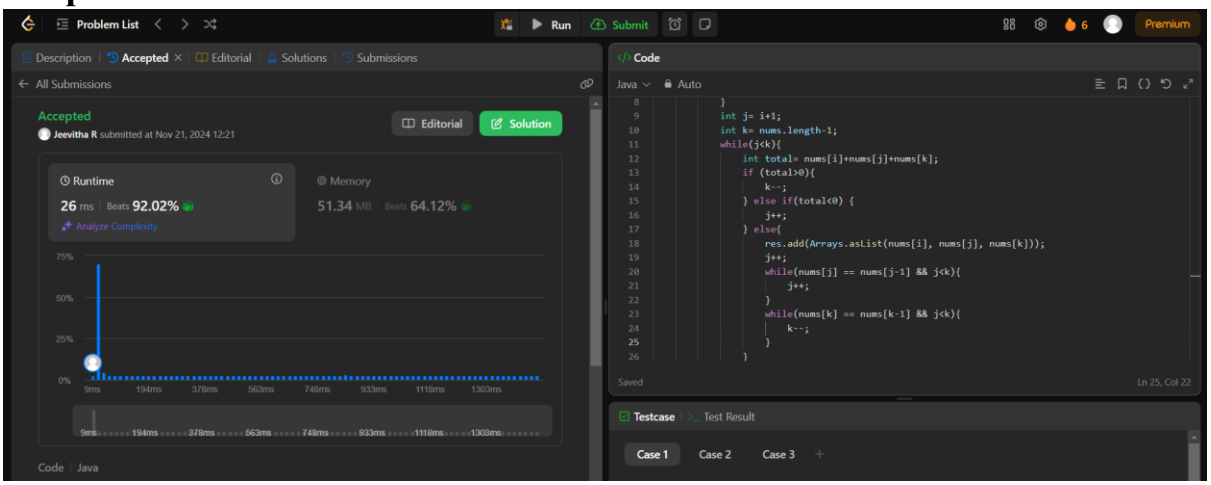
Space complexity: O(1)

5. 3Sum

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        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){
                int total= nums[i]+nums[j]+nums[k];
                if (total>0){
                    k--;
                } else if(total<0) {
                    j++;
                } else{
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    j++;
                    while(nums[j] == nums[j-1] && j<k){
                        j++;
                    }
                    while(nums[k] == nums[k-1] && j<k){
```

```
                k--;
            }
        }
    }
    return res;
  }
}
```

Output



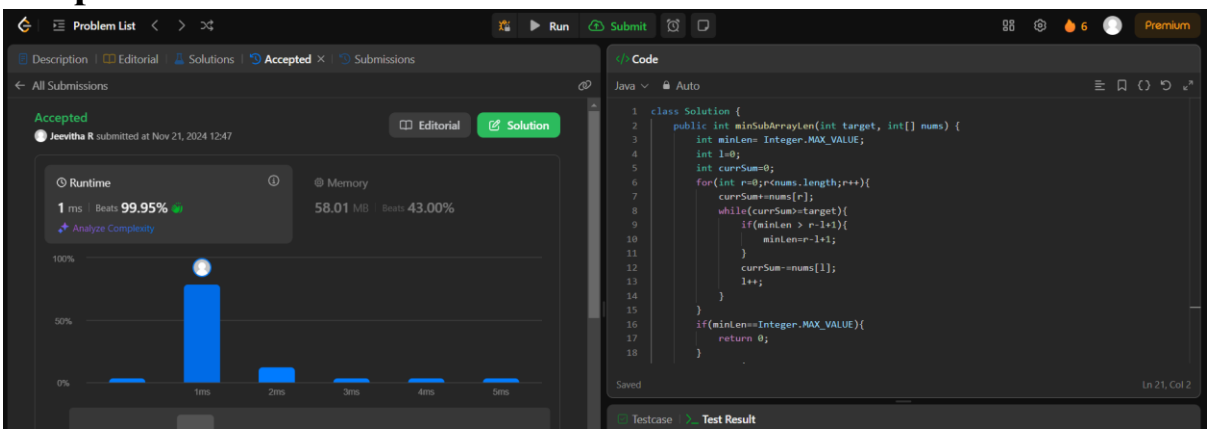
Time complexity: O(n^2)
Space complexity: O(n)

6. Minimum Size Subarray Sum

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

```
        return minLen;
    }
}
```

Output

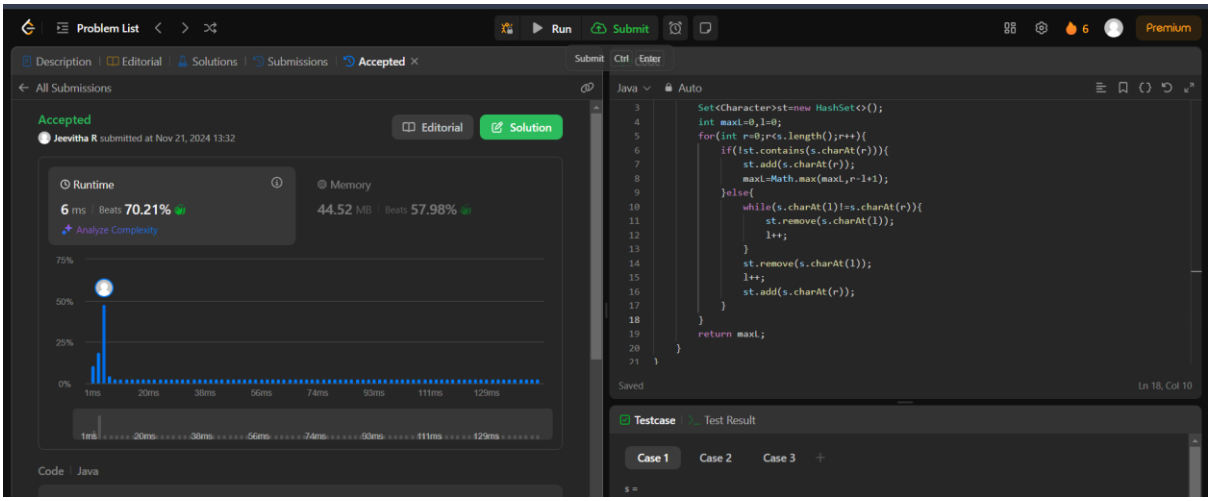


Time complexity: O(n)
Space complexity: O(1)

7. Longest Substring without repeating characters

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

Output

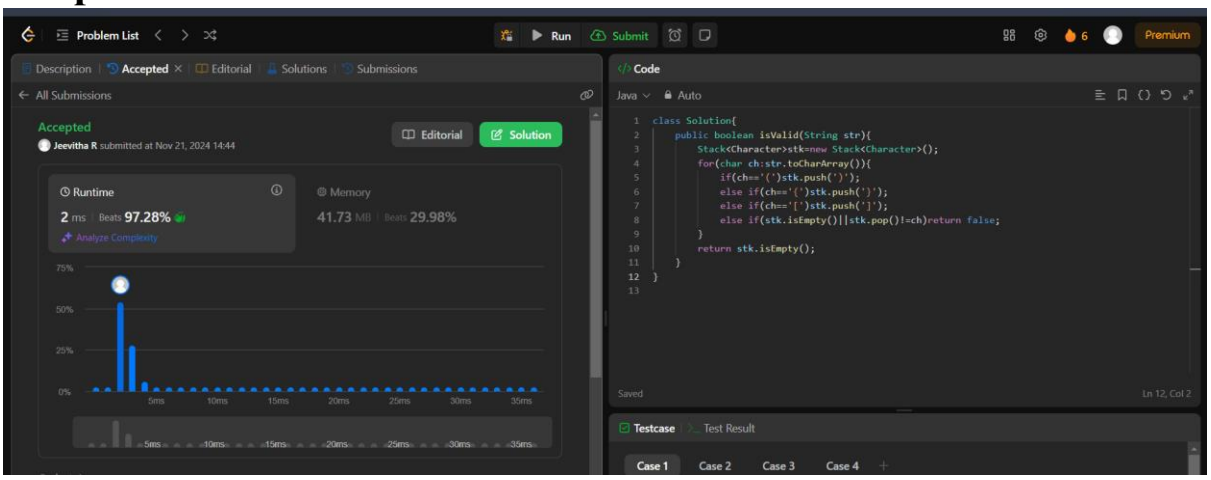


Time complexity: O(n)
Space complexity: O(n)

8. Valid Parantheses

```
class Solution{
    public boolean isValid(String str){
        Stack<Character>stk=new Stack<Character>();
        for(char ch:str.toCharArray()){
            if(ch=='(')stk.push(')');
            else if(ch=='{')stk.push('}');
            else if(ch=='[')stk.push(']');
            else if(stk.isEmpty()||stk.pop()!=ch)return false;
        }
        return stk.isEmpty();
    }
}
```

Output

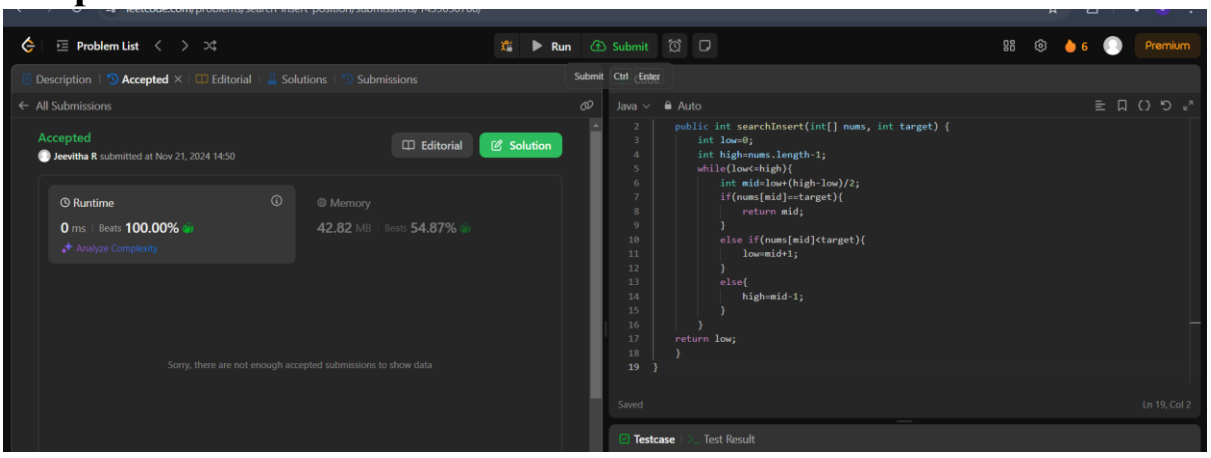


Time complexity: O(n)
Space complexity: O(n)

9. Search Insert Position

```
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



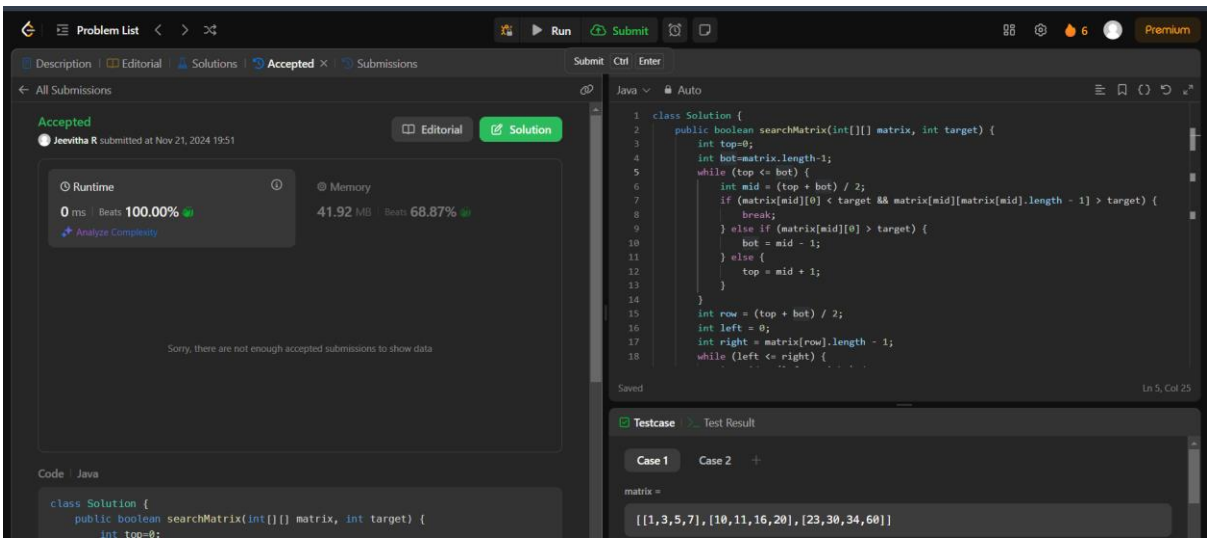
Time complexity: O(log n)
Space complexity: O(1)

10. Search in 2D matrix

```
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;
    }
}
```

Output

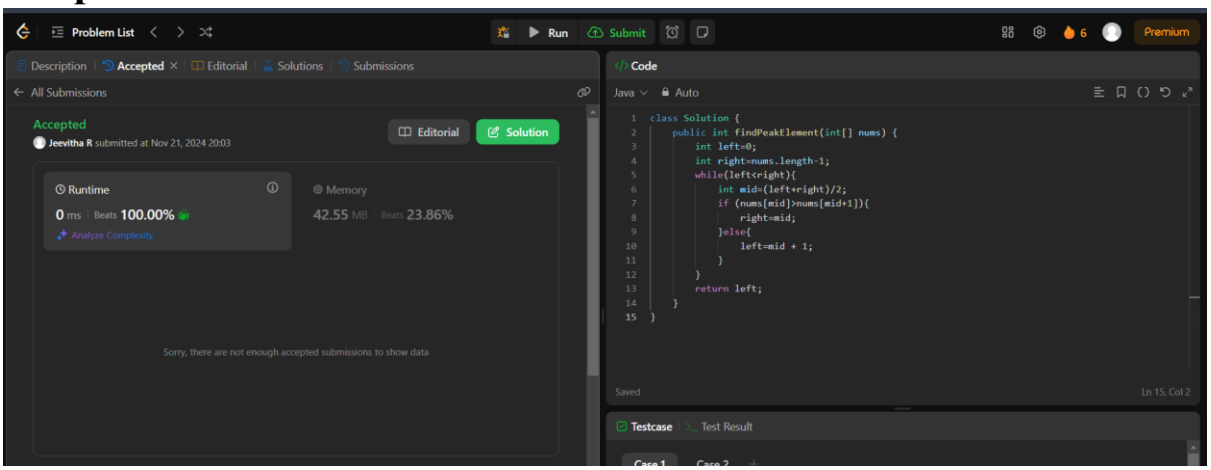


Time complexity: O(log(m*n))
Space complexity: O(1)

11. Find Peak Element

```
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



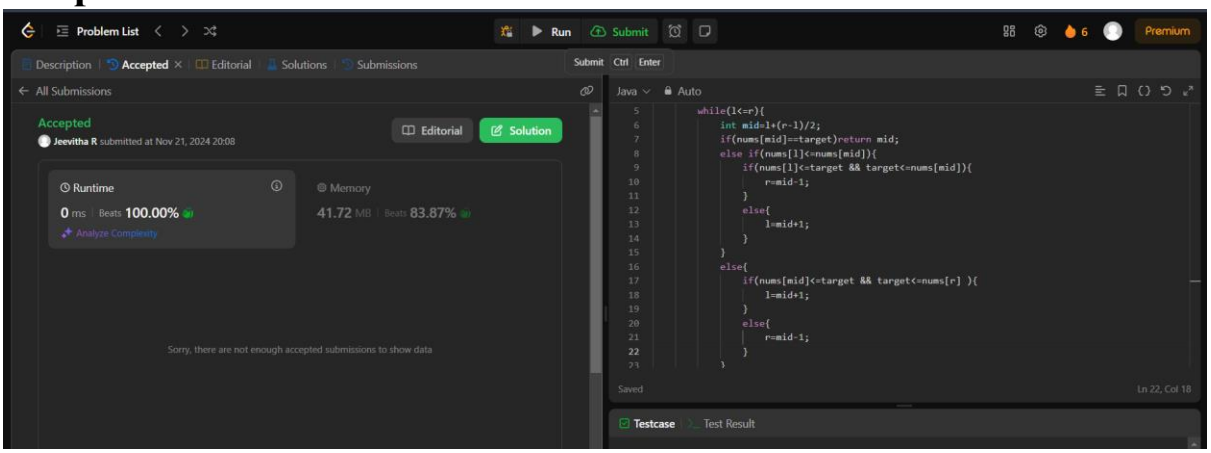
Time complexity: O(log N)
Space complexity: O(1)

12. Search in rotated sorted array

```
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



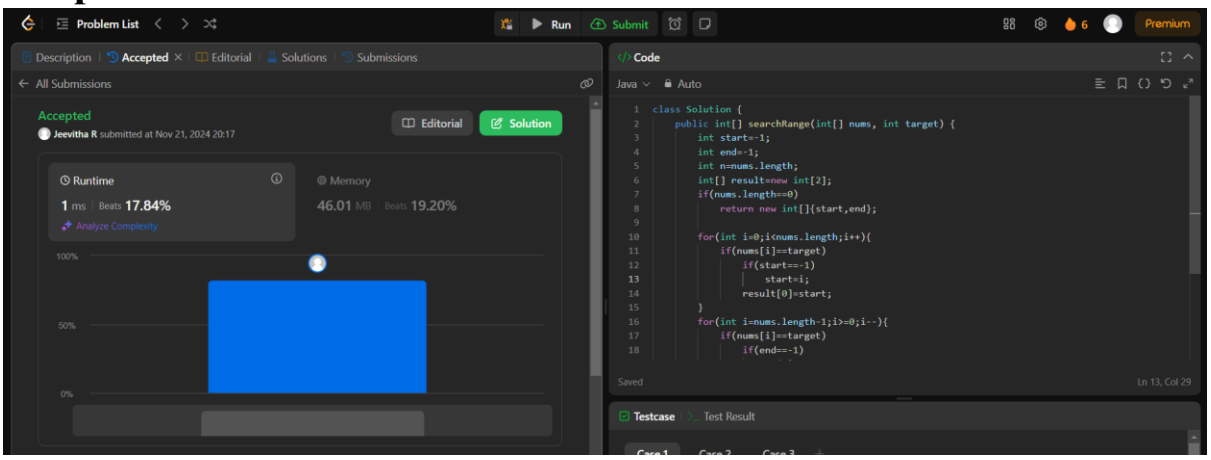
Time complexity: O(log n)
Space complexity: O(1)

13. Find the first and last position of an element in the sorted array

```
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



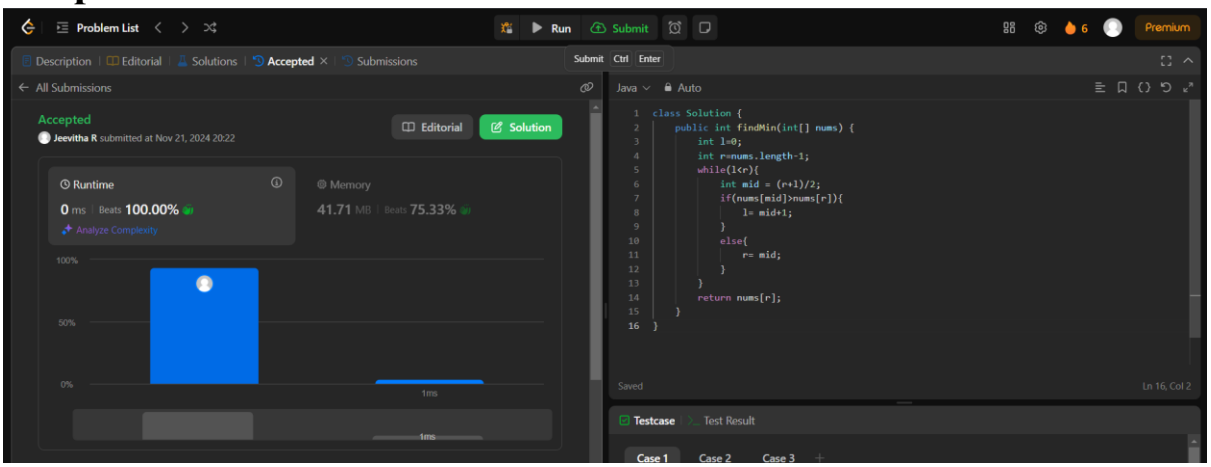
Time complexity: O(n)
Space complexity: O(1)

14. Find minimum in rotated sorted array

```
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

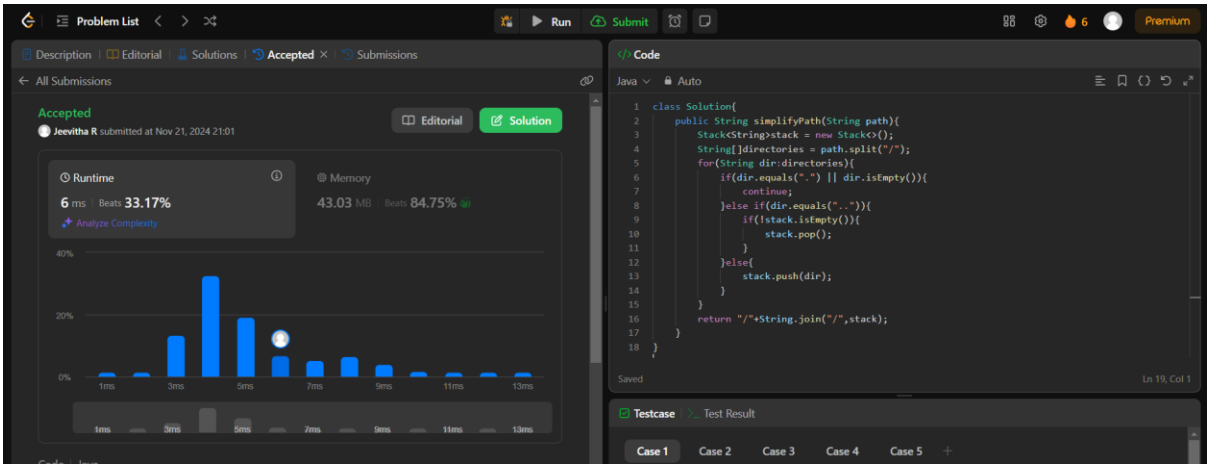


Time complexity: O(log n)
Space complexity: O(1)

15. Simplify Path

```
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



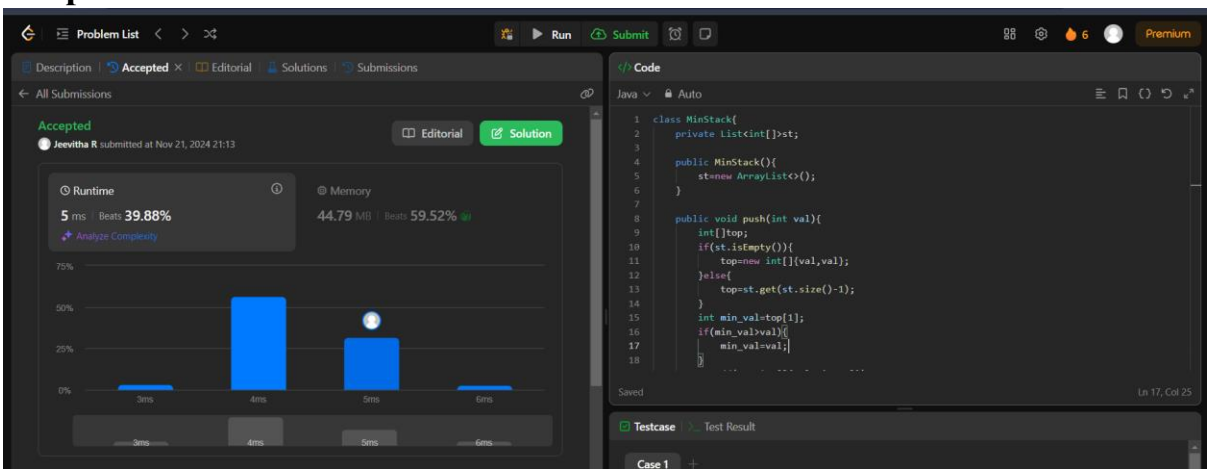
Time complexity: O(n)
Space complexity: O(n)

16. Min stack

```
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



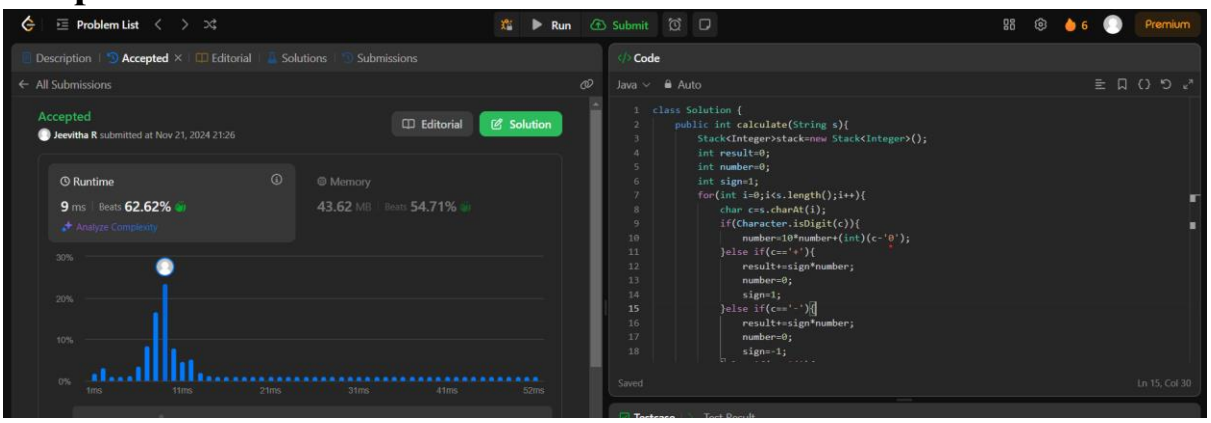
Time complexity: O(1)
Space complexity: O(n)

17. Basic Calculator

```
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)