**1.Valid Palindrome:**

**CODE:**

class Solution {

    public boolean isPalindrome(String s) {

        StringBuilder st = new StringBuilder();

        for(int i = 0;i<s.length();i++){

            if(Character.isLetter(s.charAt(i))){

                st.append(Character.toLowerCase(s.charAt(i)));

            }

            else if(Character.isDigit(s.charAt(i))){

                st.append(s.charAt(i));

            }

        }

        String c = st.toString();

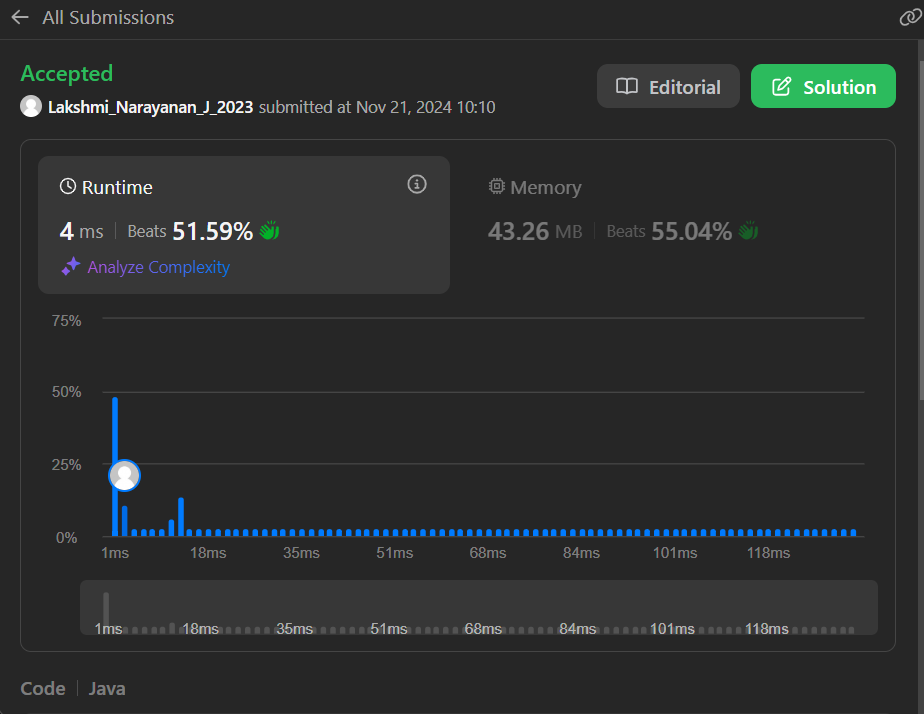
        String ch = st.reverse().toString();

        return c.equals(ch);

    }

}

**OUTPUT:**



**2.Is Subsequence**

**CODE:**

class Solution {

    public boolean isSubsequence(String s, String t) {

        int a = 0;

        int i = 0;

        while(i<t.length() && a<s.length()){

            if (t.charAt(i)==s.charAt(a)){

                a++;

            }

            i++;

        }

        if(a==s.length()){

            return true;

        }else{

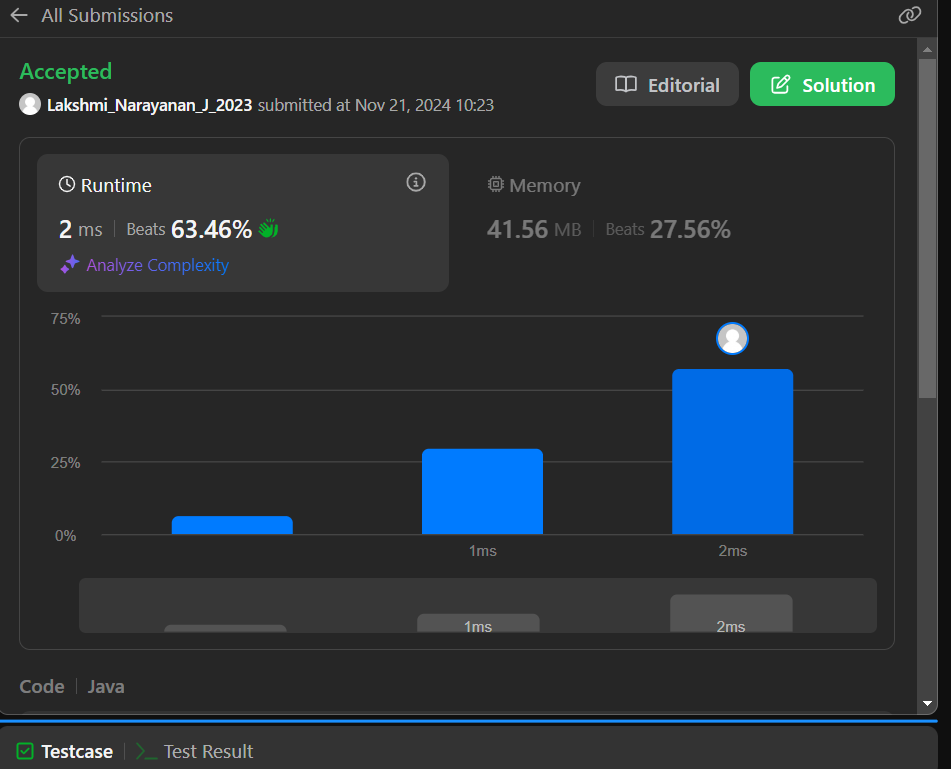
            return false;

        }

    }

}

**OUTPUT:**

****

**3.** [**Two Sum II - Input Array Is Sorted**](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/)**:**

**CODE:**

class Solution {

    public int[] twoSum(int[] ans, int target) {

        int i = 0;

        int j = ans.length-1;

        int[] arr = new int[2];

        while(i<j){

            if(ans[i]+ans[j]==target){

                arr[0]=i+1;

                arr[1]=j+1;

                return arr;

            }

            else if(ans[i]+ans[j]<target){

                i++;

            }

            else{

                j--;

            }

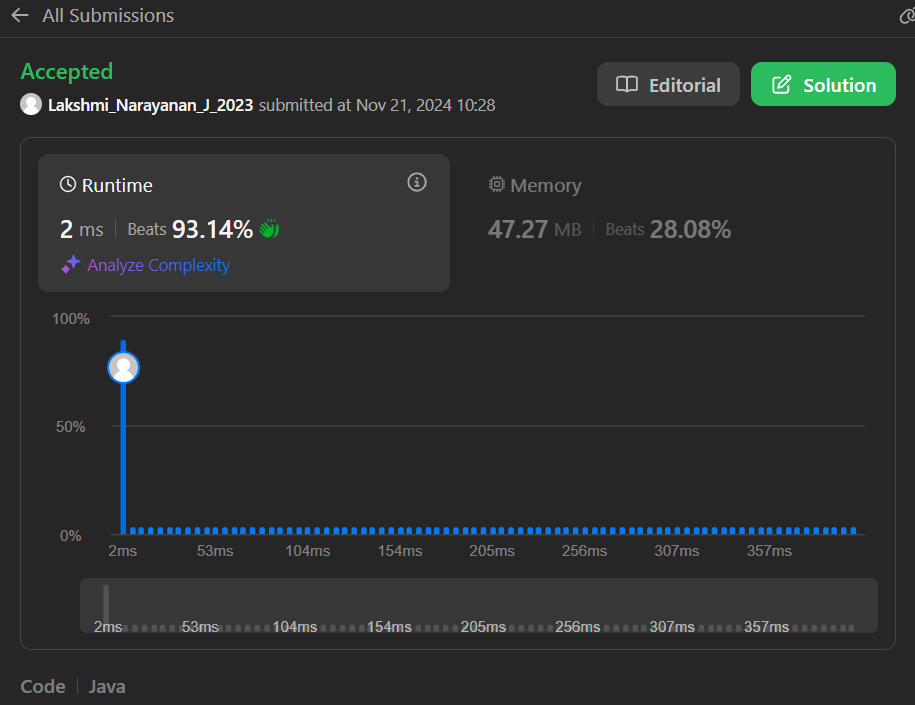
        }

        return arr;

    }

}

**OUTPUT:**

****

**4.Container with most water**

**CODE:**

class Solution {

    public int maxArea(int[] height) {

        int left = 0;

        int right = height.length-1;

        int current = 0;

        int a=0;

        while(left!=right){

            current = Math.min(height[left],height[right])\*(right-left);

            if(height[left]<height[right]){

                left++;

            }

            else{

                right--;

            }

            a=Math.max(a,current);

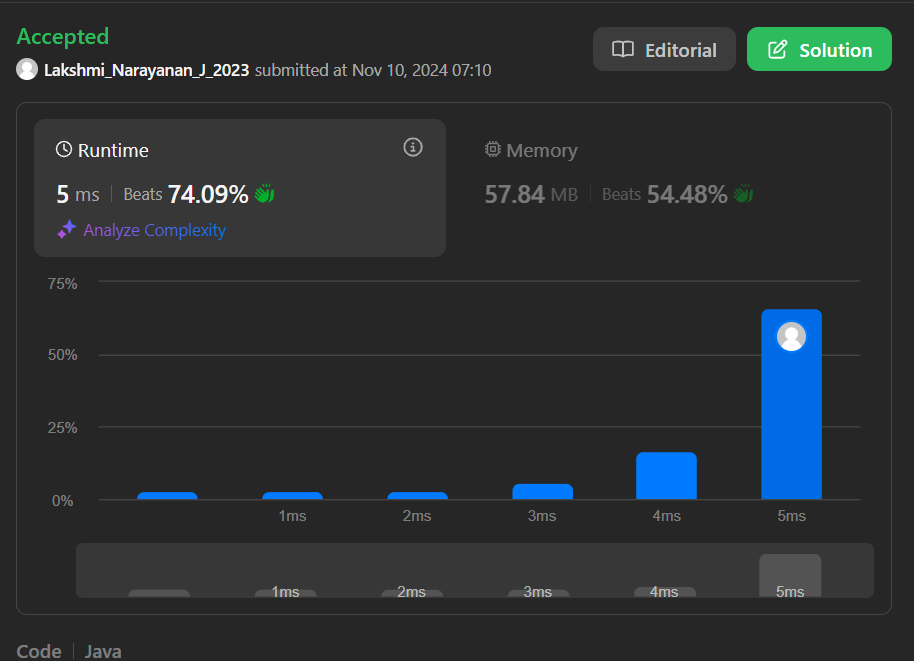
        }

        return a;

    }

}

**OUTPUT:**

****

**5.3sum**

**CODE:**

class Solution {

    public List<List<Integer>> threeSum(int[] nums) {

        HashSet<List<Integer>> hs = new HashSet<>();

        Arrays.sort(nums);

        for(int i = 0; i < nums.length-2;i++){

            int j = i+1;

            int k=nums.length-1;

            if(i>0 && nums[i-1]==nums[i]){

                continue;

            }

            while(j<k){

                int s = nums[i]+nums[j]+nums[k];

                if(s==0){

                    List <Integer> l = new ArrayList<>();

                    l.add(nums[i]);

                    l.add(nums[j]);

                    l.add(nums[k]);

                    hs.add(l);

                }

                if(s<0){

                    j++;

                }else{

                    k--;

                }

            }

        }

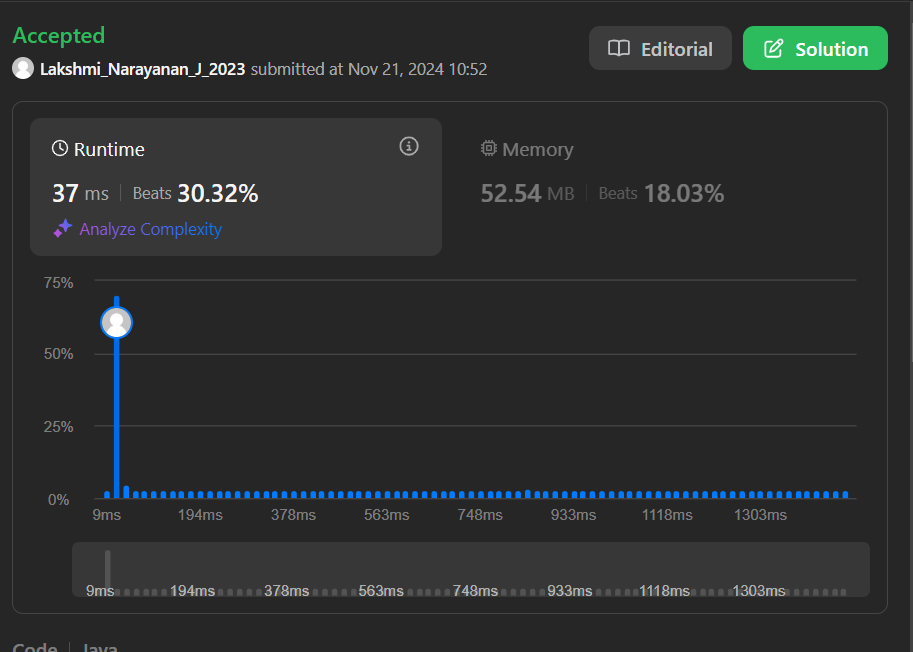
        List<List<Integer>> ans = new ArrayList<>(hs);

        return ans;

    }

}

**OUTPUT:**

****

**6.Minimum size subarray sum**

**CODE:**

class Solution {

    public int minSubArrayLen(int target, int[] nums) {

        int sum=0;

        int start=0;

        int ans = nums.length+1;

        for(int i=0;i<nums.length;i++){

            sum+=nums[i];

            while(sum>=target){

                ans=Math.min(ans,i-start+1);

                sum-=nums[start];

                start++;

            }

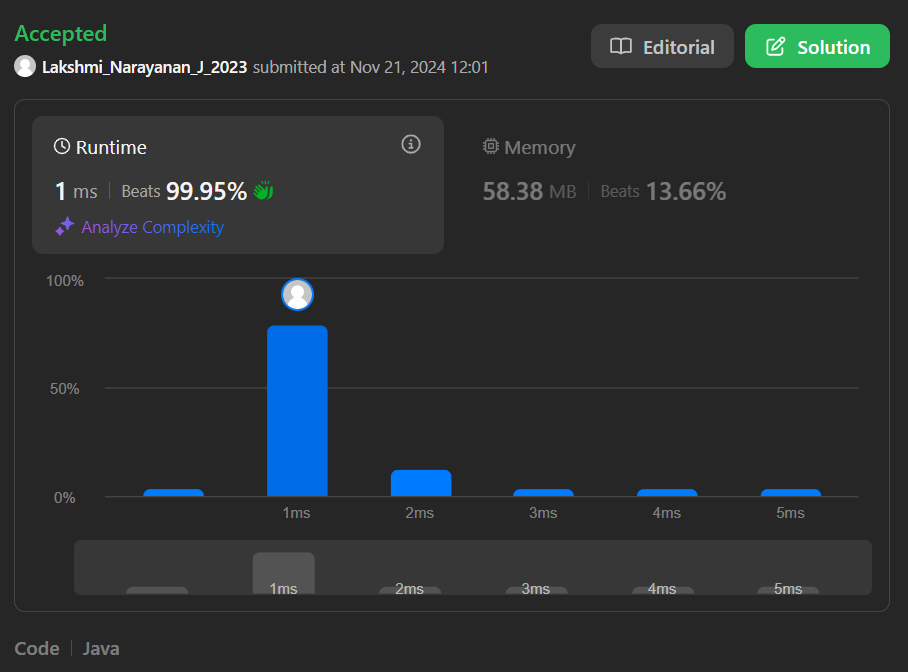
        }

        return ans==nums.length+1?0:ans;

    }

}

**OUTPUT:**

****

**7.Longest substring without repeating characters**

**CODE:**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int ans = 0;

        Set <Character> l = new HashSet<>();

        int j = 0;

        for(int i = 0; i<s.length(); i++){

            if(!l.contains(s.charAt(i))){

                l.add(s.charAt(i));

                ans = Math.max(ans,i-j+1);

            }

            else{

                while(l.contains(s.charAt(i))){

                    l.remove(s.charAt(j));

                    j++;

                }

                l.add(s.charAt(i));

            }

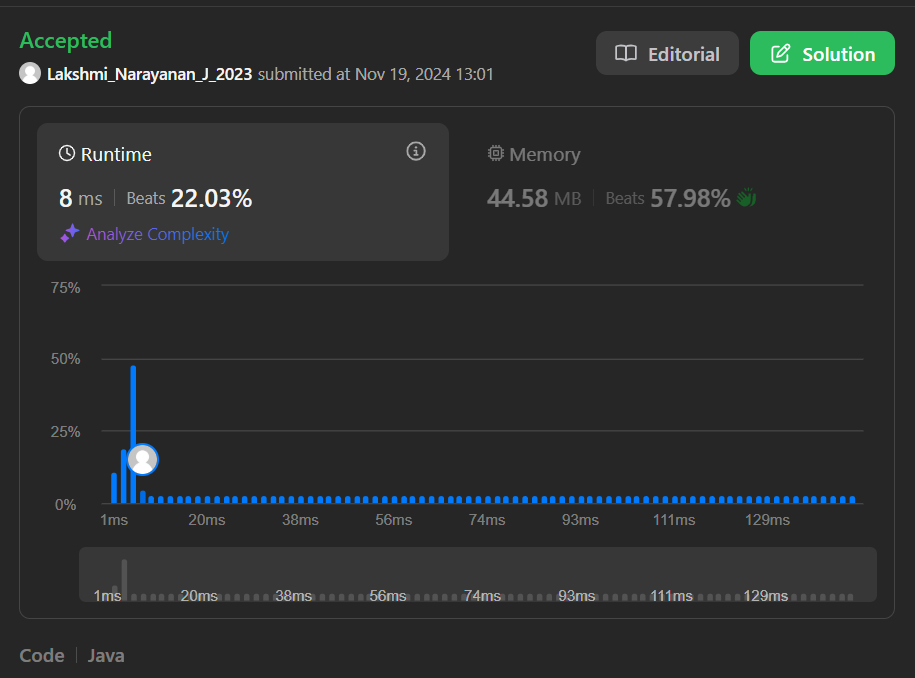
        }

        return ans;

    }

}

**OUTPUT:**

****

**8.Valid Parantheses**

**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.containsValue(c)) {

stack.push(c);

} else if (mapping.containsKey(c)) {

if (stack.isEmpty() || mapping.get(c) != stack.pop()) {

return false;

}

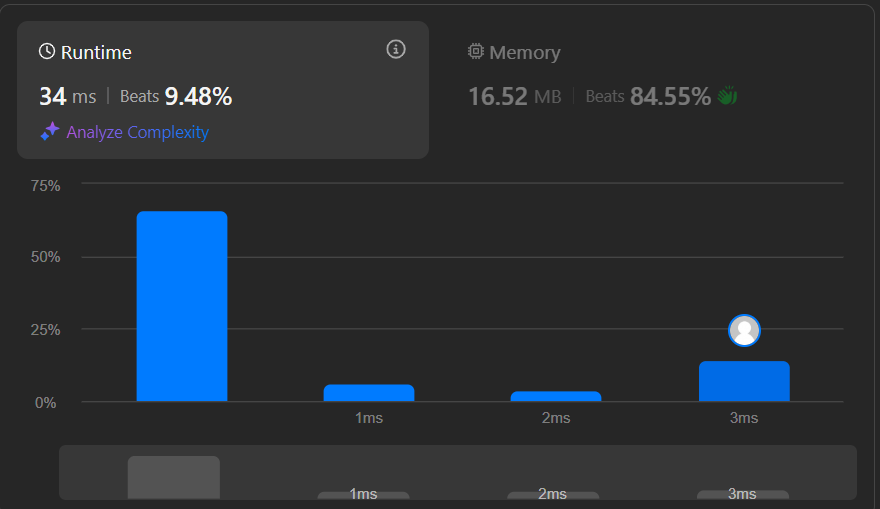
}

}

return stack.isEmpty();

}

**OUTPUT:**

****

**9.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);

            }

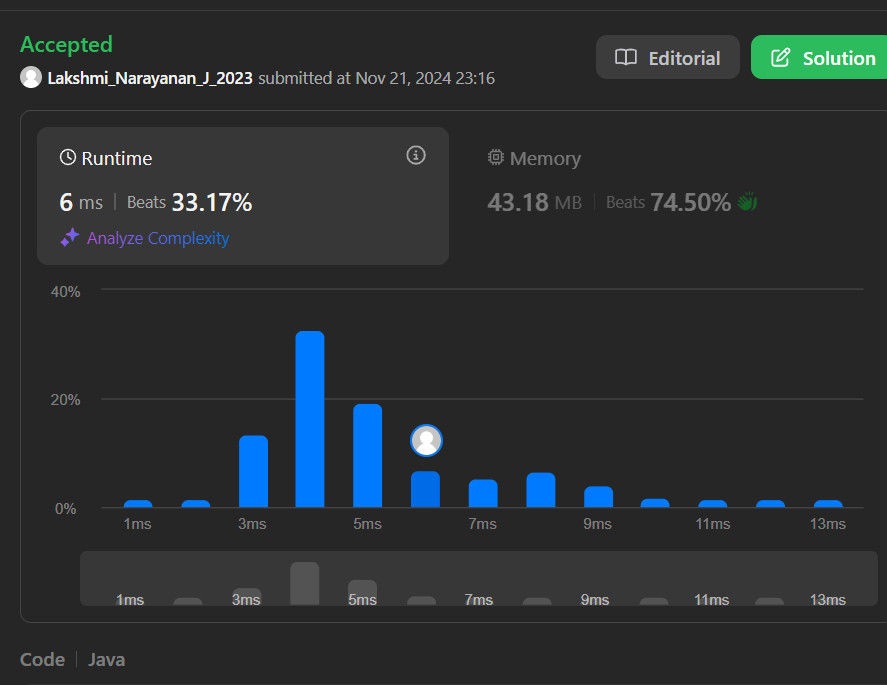
        }

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

    }

}

**OUTPUT:**

****

**10.Min Stack**

**CODE:**

class MinStack {

    Stack<Integer> s;

    Stack<Integer> min;

    public MinStack() {

        s=new Stack<>();

        min=new Stack<>();

    }

    public void push(int val) {

        s.push(val);

        if(min.isEmpty()||val<=min.peek()){

            min.push(val);

        }

    }

    public void pop() {

        int p = s.pop();

        if(p==min.peek()){

            min.pop();

        }

    }

    public int top() {

        return s.peek();

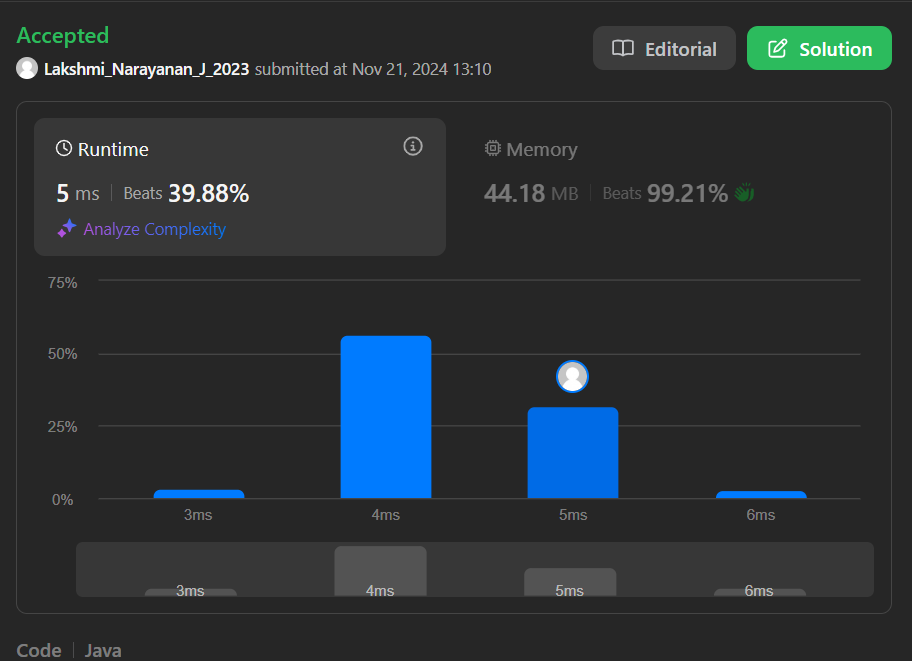
    }

    public int getMin() {

        return min.peek();

    }

}

**OUTPUT:  
**

**11.Evaluate Reverse Polish Notation**

**CODE:**

**class Solution {**

public int evalRPN(String[] tokens) {

        Stack<Integer> st = new Stack<>();

        for(String s:tokens){

            if(Character.isDigit(s.charAt(0)) || (s.length()>1 && s.charAt(0)=='-')){

                st.push(Integer.valueOf(s));

            }

            else{

                int b = st.pop();

                int a = st.pop();

                if (s.equals("\*")){

                    st.push(a\*b);

                }

                else if(s.equals("+")){

                    st.push(a+b);

                }

                else if(s.equals("/")){

                    st.push(a/b);

                }

                else{

                    st.push(a-b);

                }

            }

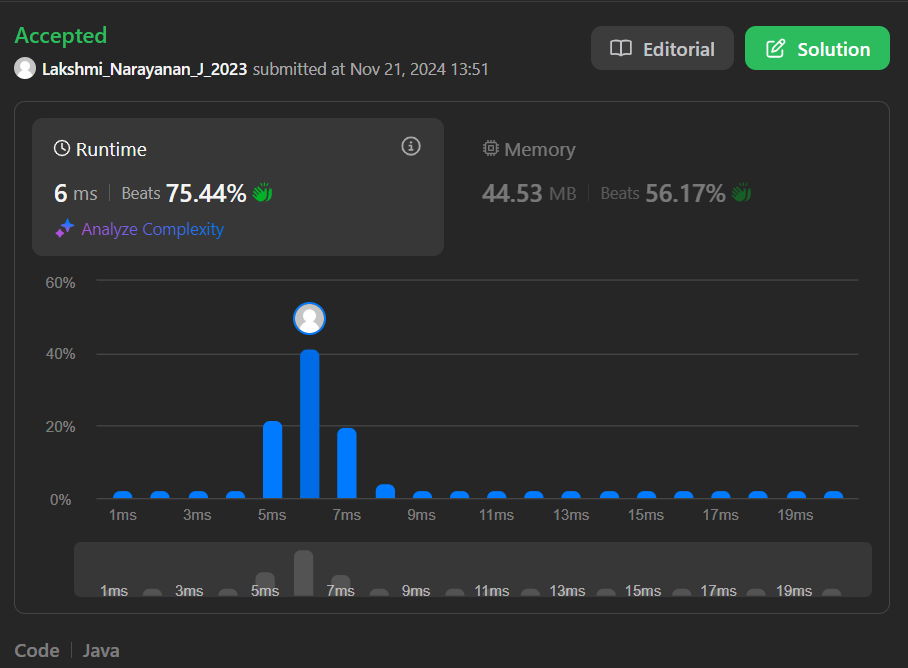
        }

        return st.pop();

    }

}

**OUTPUT:**

****

**12.Search Insert Position**

**CODE:**

class Solution {

    public int searchInsert(int[] nums, int target) {

        for(int i = 0;i<nums.length;i++){

            if(nums[i]==target){

                return i;

            }

            else if(i<nums.length-1 && nums[i]<target && nums[i+1]>target){

                return i+1;

            }

        }

        if(target<nums[0]){

            return 0;

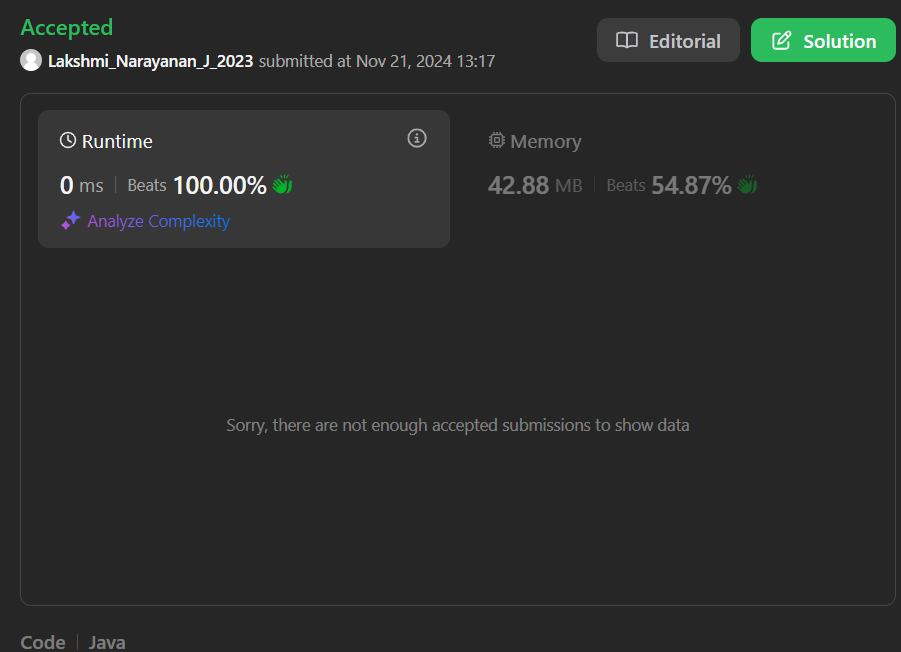
        }else{

            return nums.length;

        }

    }

}

**OUTPUT:  
**

**13.Search a 2d matrix**

**CODE:**

class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

        int a = 0;

        for(int i = 0;i<matrix.length;i++){

            for(int j = 0;j<matrix[0].length;j++){

                if(matrix[i][j]==target){

                    a=1;

                }

            }

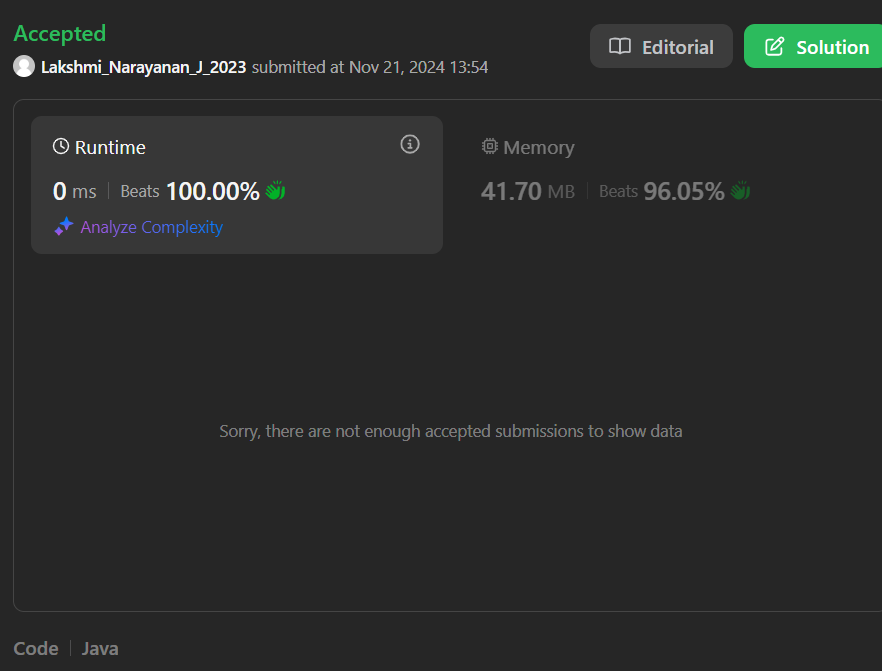
        }

        return a==1;

    }

}

**OUTPUT:**

****

**14.Find Peek Element**

**CODE:**

class Solution {

    public int findPeakElement(int[] nums) {

        int a = 0;

        if (nums.length==1) return 0;

        for(int i = 0;i<nums.length;i++){

            if(i==0 && nums[i]>nums[i+1]){

                a=Math.max(a,i);

            }

            else if(i==nums.length-1 && nums[i]>nums[i-1]){

                a=Math.max(a,i);

            }

            else if(i!=nums.length-1 && nums[i]>nums[i+1] && nums[i]>nums[i-1]){

                a=Math.max(a,i);

            }

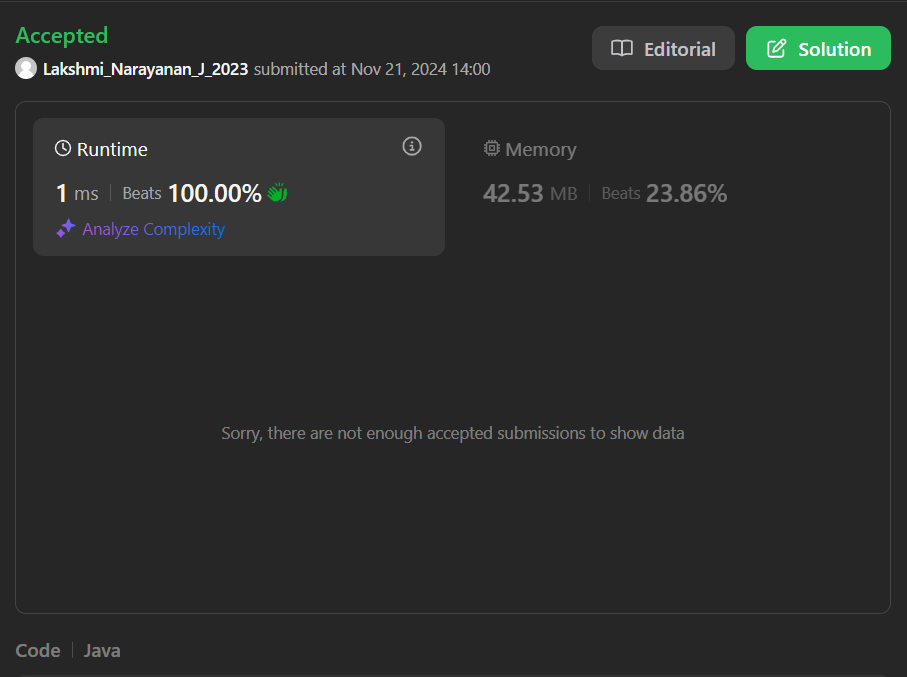
        }

        return a;

    }

}

**OUTPUT:**

****

**15.Search in rotated sorted array**

**CODE:**

class Solution {

    public int search(int[] nums, int target) {

        for(int i=0; i< nums.length; i++){

            if (nums[i]==target){

                return i;

            }

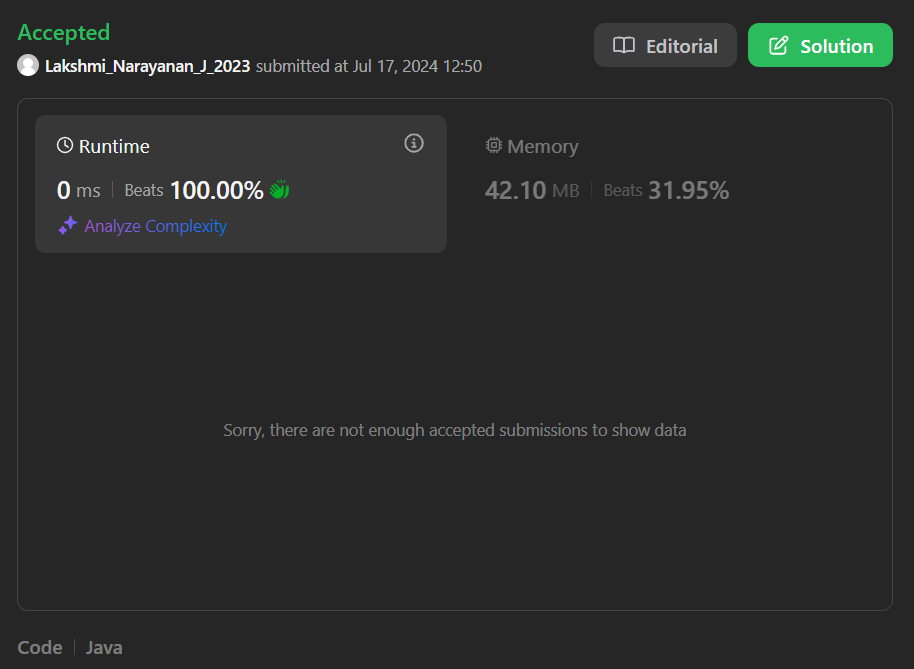
        }

        return -1;

    }

}

**OUTPUT:**

****

**16.First and last position of element in sorted array**

**CODE:**

class Solution {

    public int[] searchRange(int[] nums, int target) {

        int[] result = new int[2];

        result[0]=-1;

        result[1]=-1;

        int a = 0;

        for(int i = 0;i<nums.length;i++){

            if (a==0 && nums[i]==target){

                result[0]=i;

                a=1;

            }

            else if(a==1 && nums[i]==target){

                result[1]=i;

            }

        }

        if(result[0]!=-1 && result[1]==-1){

            result[1]=result[0];

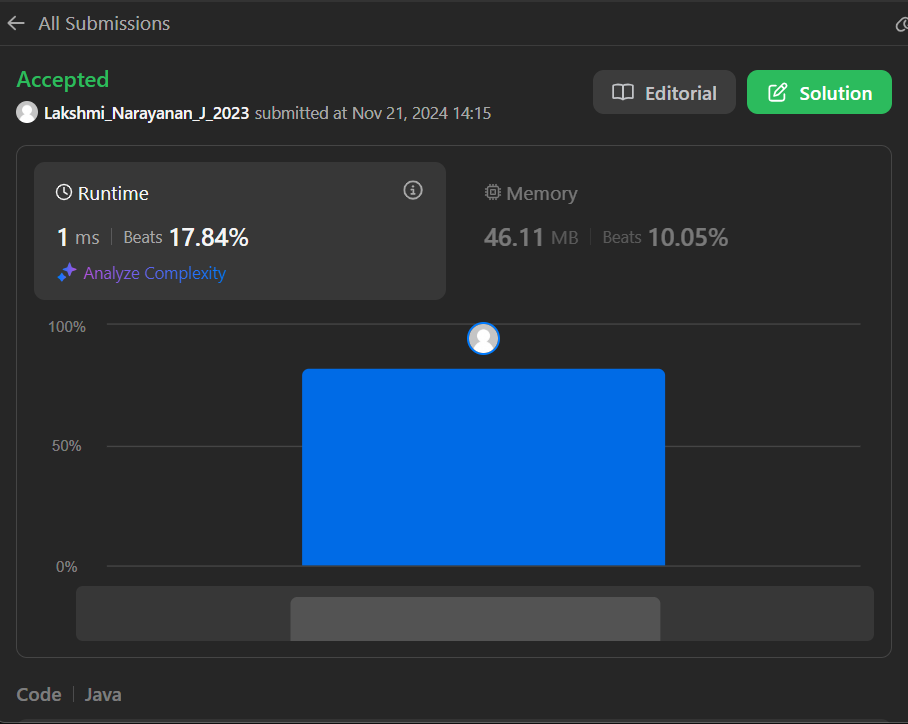
        }

        return result;

    }

}

**OUTPUT:**

****

**17.Find minimum in rotated sorted array**

**CODE:**

class Solution {

    public int findMin(int[] nums) {

        for(int i = 0;i<nums.length-1;i++){

            if (nums[i]>nums[i+1]){

                return nums[i+1];

            }

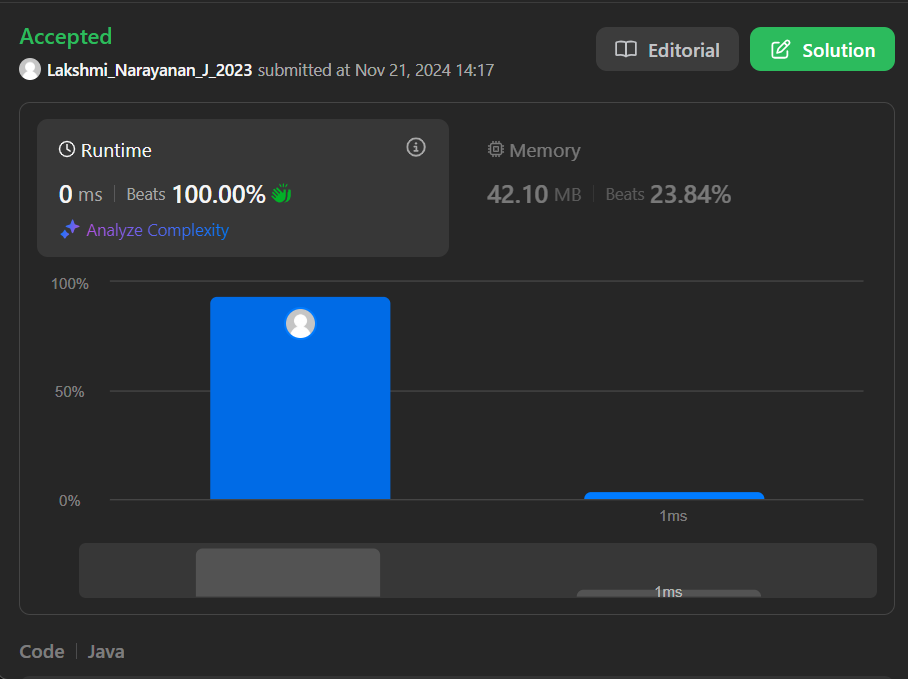
        }

        return nums[0];

    }

}

**OUTPUT:**

****