**DSA – DAY 9 CODING PROBLEMS**

**Name:** Dhuvarrakesh K

**Date:**21/11/24

**Register Number:**22IT024

**1.Valid Palindrome**

**Code:**

class Solution {

    public boolean isPalindrome(String s) {

        StringBuilder st=new StringBuilder();

        for(char c:s.toCharArray()){

            if(Character.isLetterOrDigit(c)){

                st.append(c);

            }

        }

        String res=st.toString().toLowerCase();

        String reverse=new StringBuilder(res).reverse().toString();

        if(res.equals(reverse)){

            return true;

        }

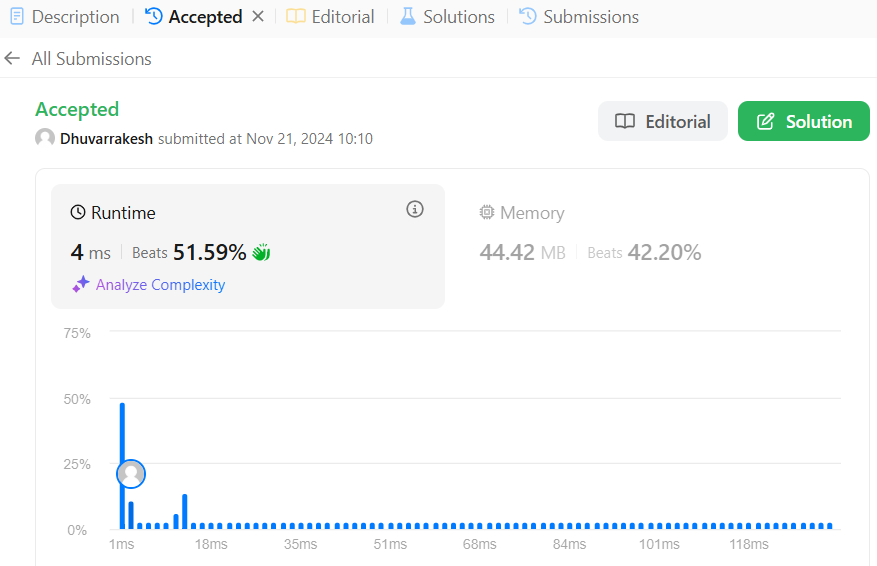
        else{

            return false;

        }

    }

}

**Output:** ****

**2.Is Subsequence**

**Code:**

class Solution {

    public boolean isSubsequence(String s, String t) {

        int a=0;

        int b=0;

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

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

                a++;

            }

            b++;

        }

        if(a==s.length()){

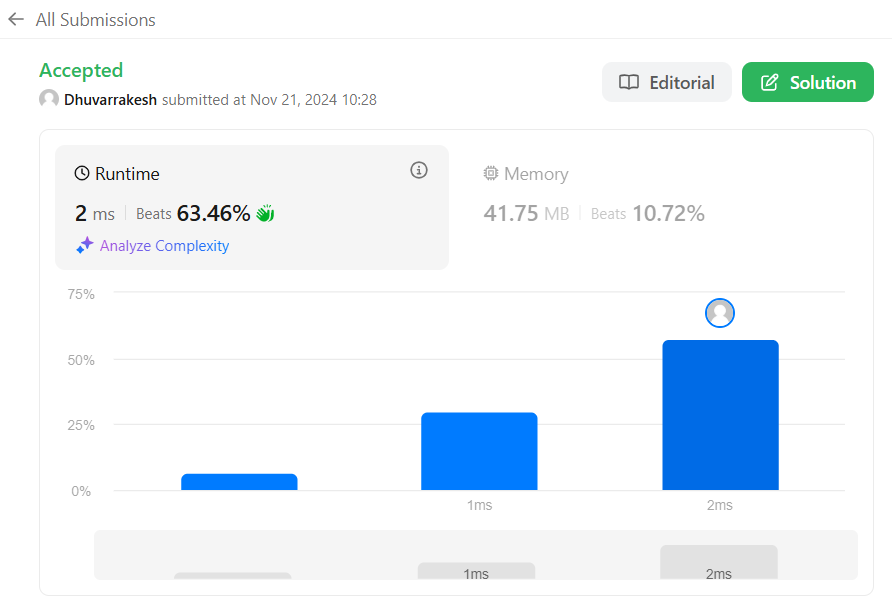
            return true;

        }

        return false;

    }

}

**Output:** ****

**3.Two Sum II-Input Array is Sorted**

**Code:**

class Solution {

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

        int[] result=new int[2];

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

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

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

                    result[0]=i+1;

                    result[1]=j+1;

                    return result;

                }

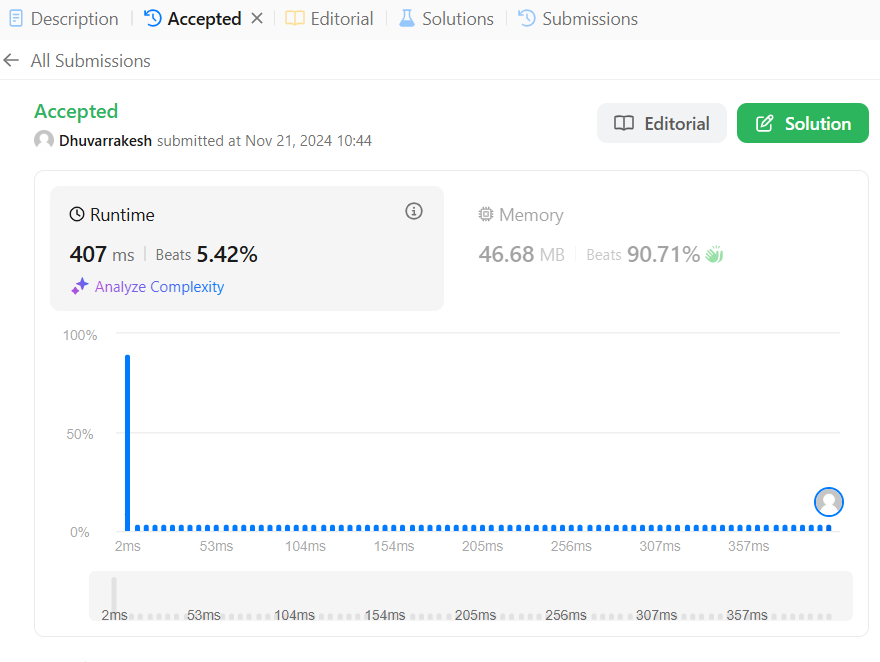
            }

        }

        return new int[0];

    }

}

**Output:** ****

**4.Container with most water**

**Code:**

class Solution {

    public int maxArea(int[] height) {

        int l=0;

        int r=height.length-1;

        int a=0;

        while(l<r){

            int area=Math.min(height[l],height[r])\*(r-l);

            if(height[l]<height[r]){

                l++;

            }

            else{

                r--;

            }

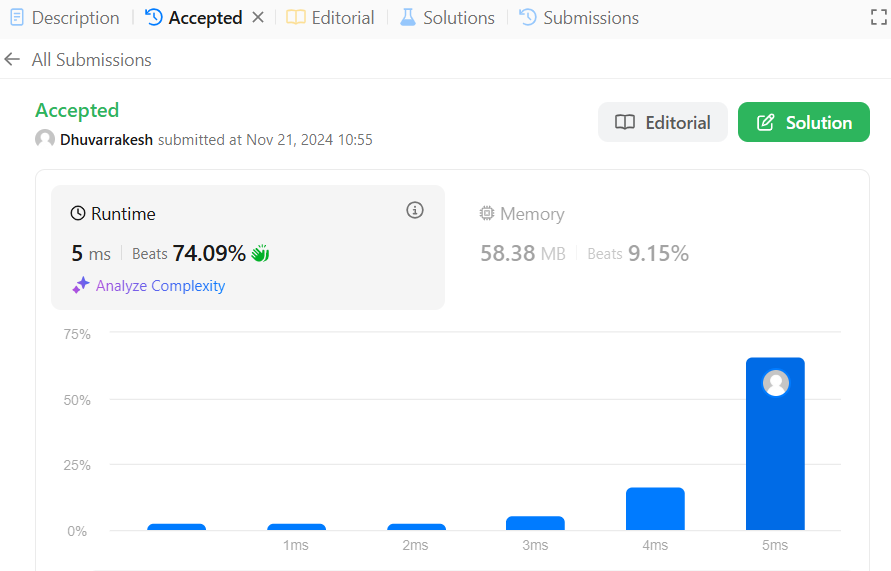
            a=Math.max(area,a);

        }

        return a;

    }

}

**Output:** ****

**5.3Sum**

**Code:**

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++;

                    }

                }

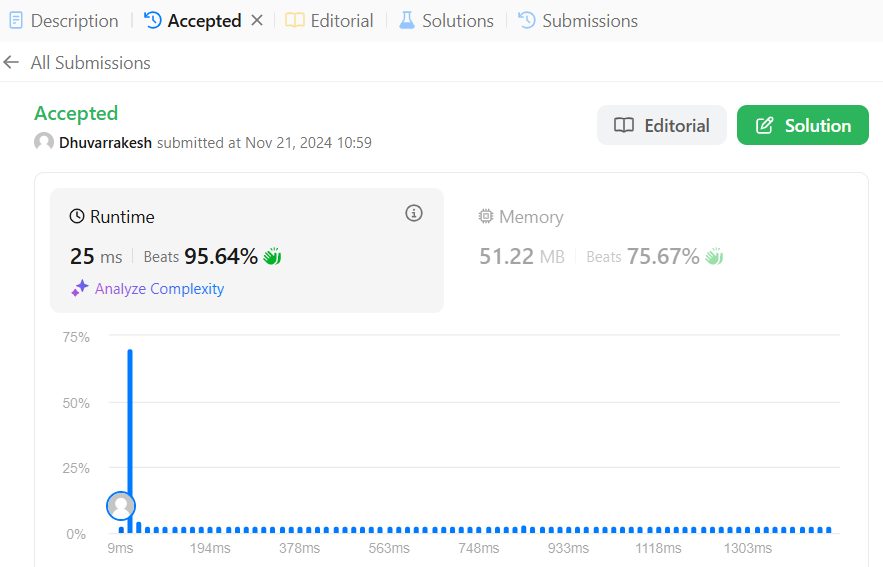
            }

        }

        return res;

    }

}

**Output:** ****

**6.Minimum size SubArray**

**Code:**

class Solution {

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

        int l=0;

        int s=0;

        int minlen=Integer.MAX\_VALUE;

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

            s+=nums[r];

            while(s>=target){

                minlen=Math.min(minlen,((r-l)+1));

                s-=nums[l];

                l++;

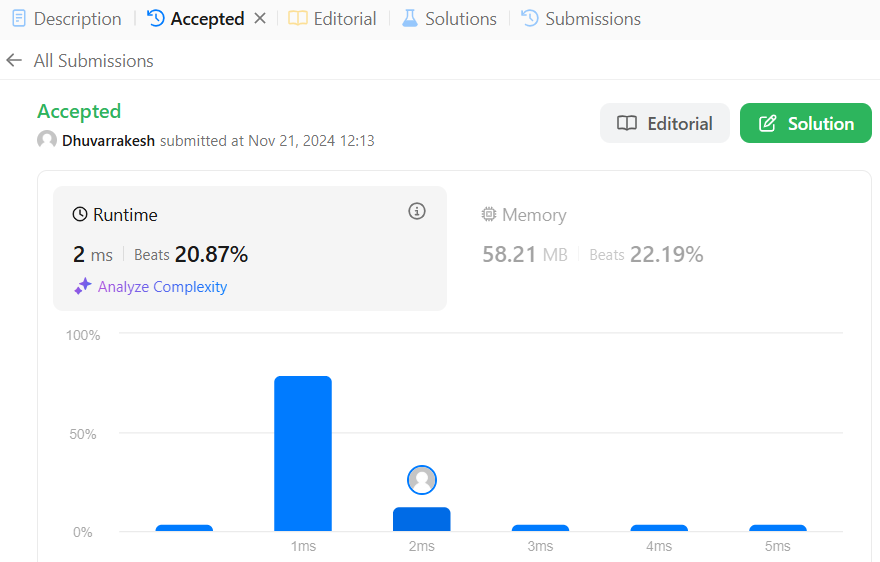
            }

        }

        return minlen==Integer.MAX\_VALUE?0:minlen;

    }

}

**Output:** ****

**7.Longest Substring without repeatition**

**Code:**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int[] count = new int[128];

        int max =0;

        int j=0;

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

            count[s.charAt(i)]++;

            while(count[s.charAt(i)]>1){

                count[s.charAt(j)]--;

                j++;

            }

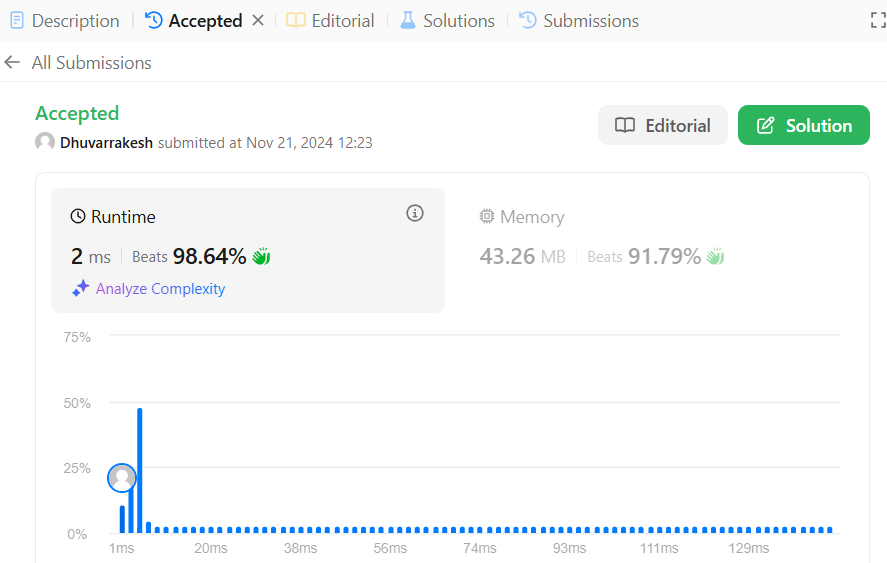
            max = Math.max(max,(i-j)+1);

        }

        return max;

    }

}

**Output:** ****

**8.Valid Parentheses**

**Code:**

class Solution {

public boolean isValid(String s) {

Stack<Character> stack = new Stack<Character>();

for (char c:s.toCharArray()){

if (c=='('||c=='[' ||c=='{'){

stack.push(c);

}else{

if(stack.isEmpty()) {

return false;

}

char top = stack.peek();

if ((c==')'&&top=='(')||(c==']'&&top=='[')||(c=='}'&&top=='{')){

stack.pop();

}else{

return false;

}

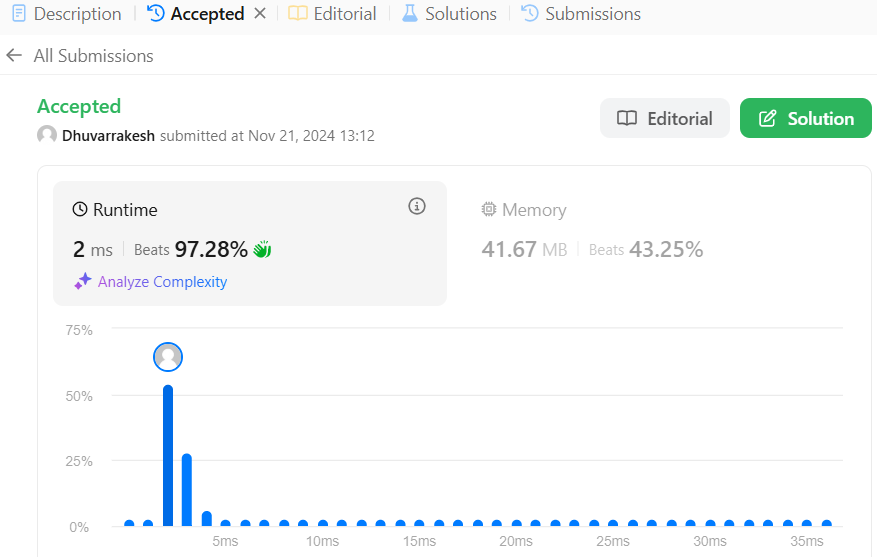
}

}

return stack.isEmpty();

}

}

**Output:** ****

**9.Simplify Path**

**Code:**

class Solution {

public String simplifyPath(String path) {

String[] arr = path.split("/");

Stack<String> stack = new Stack();

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

if (arr[i].trim() != "") {

stack.push(arr[i]);

}

}

StringBuilder sb = new StringBuilder();

int count = 0;

while (!stack.isEmpty()) {

String s = stack.pop();

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

count++;

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

continue;

} else {

if (count == 0) {

sb.insert(0, "/" + s);

} else {

count--;

}

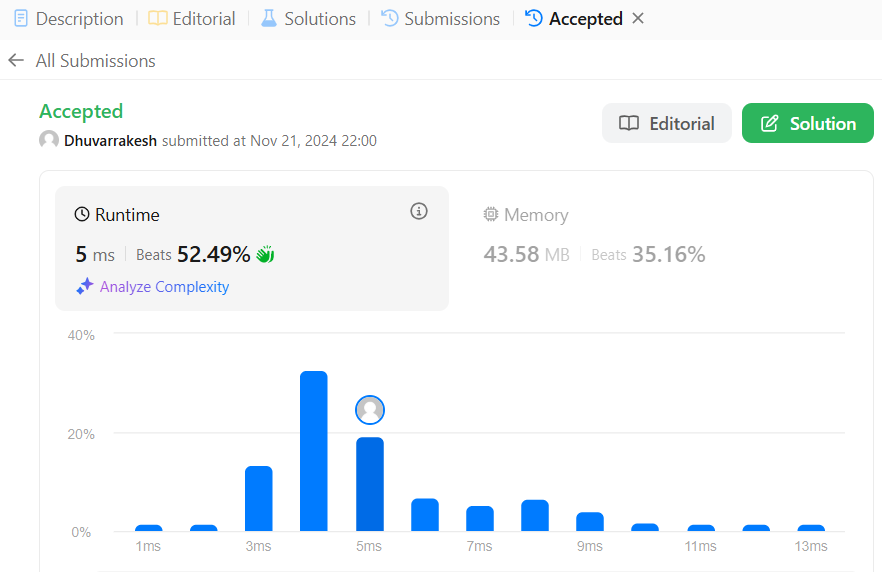
}

}

return sb.toString().length() == 0 ? "/" : sb.toString();

}

}

**Output:** ****

**10.Min Stack**

**Code:**

class MinStack {

    Stack<Integer> stack;

    Stack<Integer> minStack;

    public MinStack() {

        stack = new Stack<>();

        minStack = new Stack<>();

    }

    public void push(int val) {

        stack.push(val);

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

            minStack.push(val);

        }

    }

    public void pop() {

        if (stack.pop().equals(minStack.peek())) {

            minStack.pop();

        }

    }

    public int top() {

        return stack.peek();

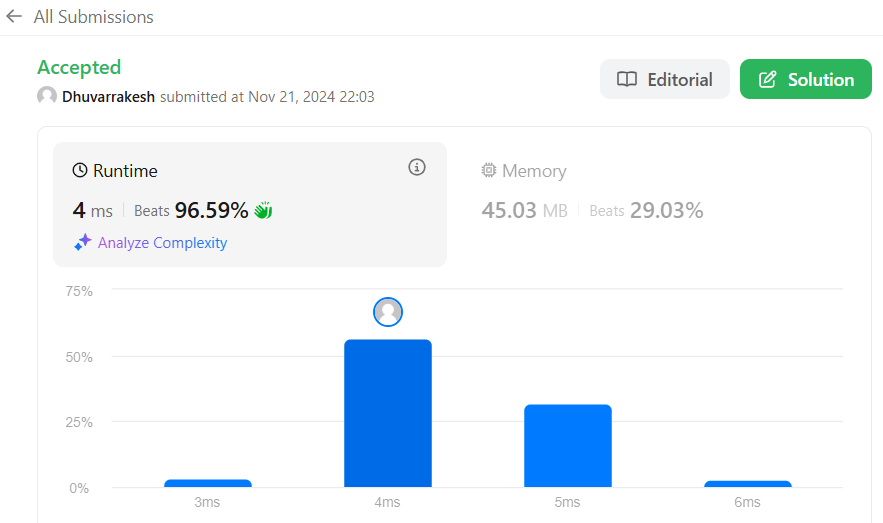
    }

    public int getMin() {

        return minStack.peek();

    }

}

**Output:** ****

**11.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));

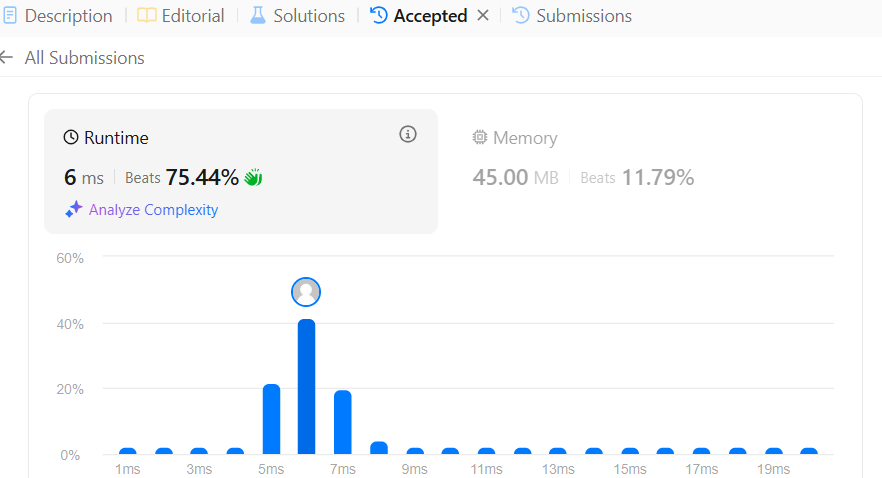
}

}

return stack.peek();

}

}

**Output:** ****

**12.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;

signValue = 1;

} else if (c == ')') {

result += signValue \* number;

result \*= operationsStack.pop();

result += operationsStack.pop();

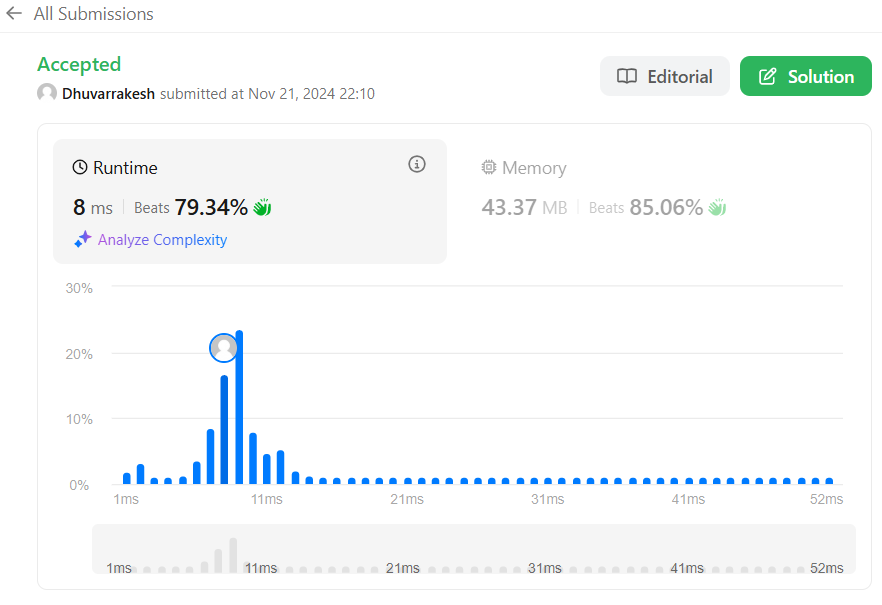
number = 0;

}

}

return result + number \* signValue;

}

} **Output:** ****

**13.Substring with concatenation of all words**

**Code:**

class Solution {

    public List<Integer> findSubstring(String s, String[] words) {

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

        int n = s.length();

        int m = words.length;

        int w = words[0].length();

        HashMap<String,Integer> map = new HashMap<>();

        for(String x : words)

        map.put(x, map.getOrDefault(x,0)+1);

        for(int i=0; i<w; i++){

            HashMap<String,Integer> temp = new HashMap<>();

            int count = 0;

            for(int j=i,k=i; j+w <= n; j=j+w){

                String word = s.substring(j,j+w);

                temp.put(word,temp.getOrDefault(word,0)+1);

                count++;

                if(count==m){

                    if(map.equals(temp)){

                        ans.add(k);

                    }

                    String remove = s.substring(k,k+w);

                    temp.computeIfPresent(remove, (a, b) -> (b > 1) ? b - 1 : null);

                    count--;

                    k=k+w;

                }

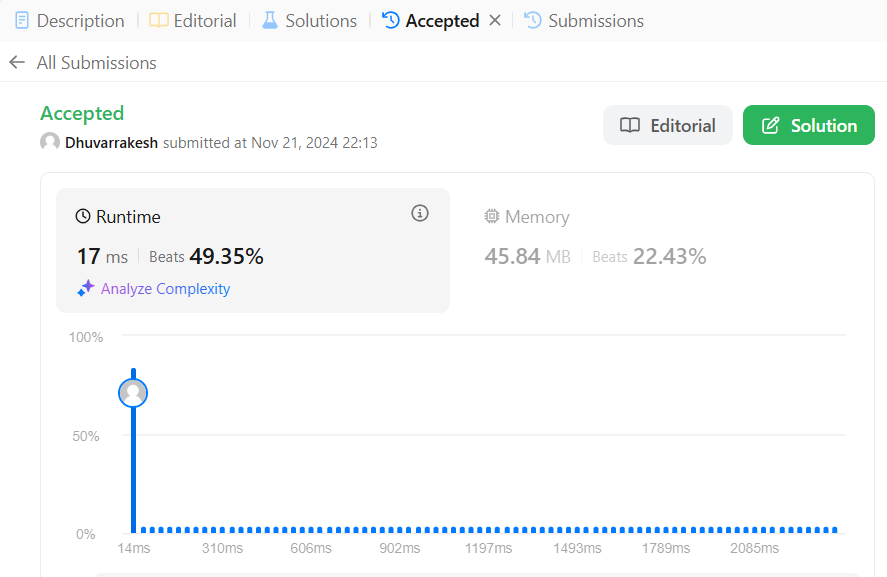
            }

        }

        return ans;

    }

}

**Output:** ****

**14.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++;

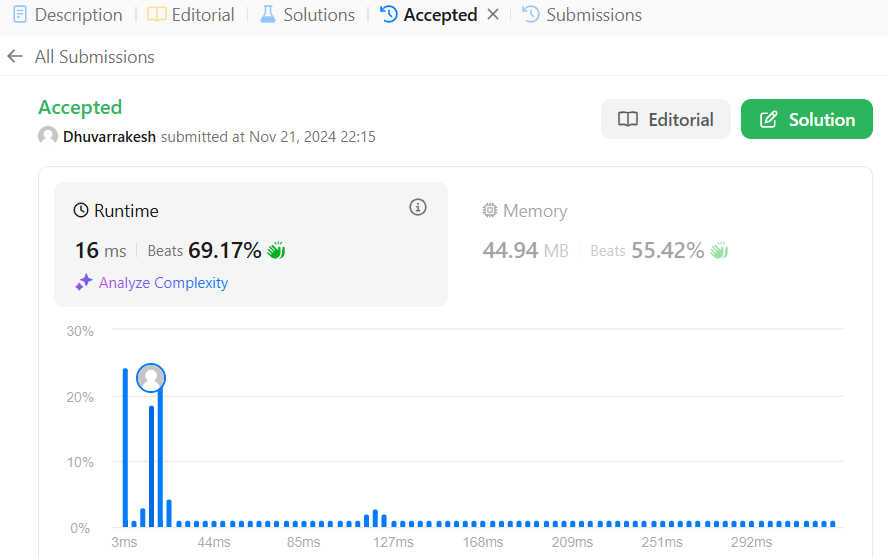
            }

        }

        return minWindow[1] >= s.length() ? "" : s.substring(minWindow[0], minWindow[1] + 1);

    }

}

**Output:** ****

**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)/2;

if(nums[mid]==target){

return mid;

}

else if(nums[mid]>target){

right=mid-1;

}

else{

left=mid+1;

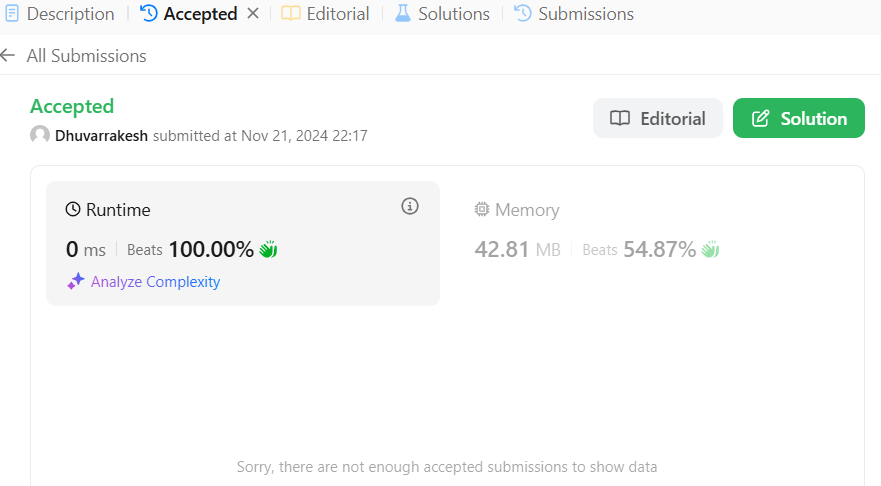
}

}

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 left = 0, right = m \* n - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            int mid\_val = matrix[mid / n][mid % n];

            if (mid\_val == target)

                return true;

            else if (mid\_val < target)

                left = mid + 1;

            else

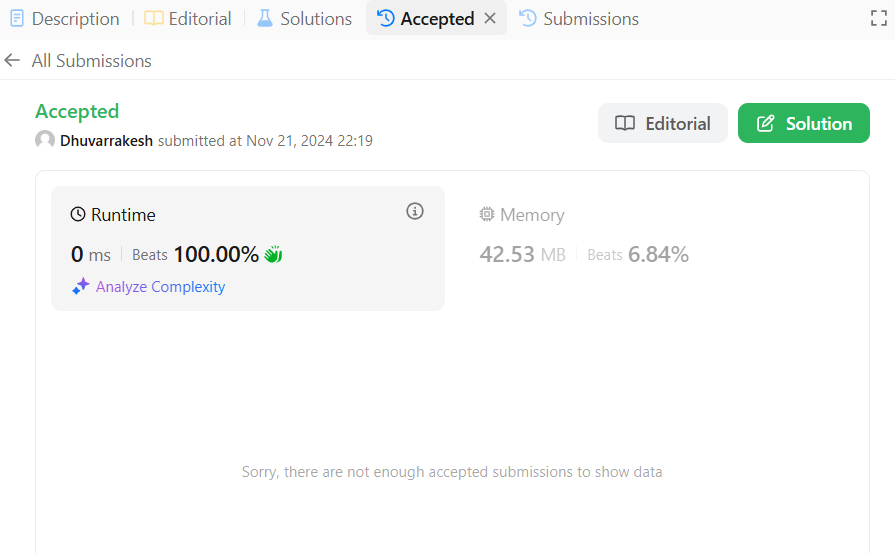
                right = mid - 1;

        }

        return false;

    }

}

**Output:** ****

**17.Find Peak Element**

**Code:**

class Solution {

public int findPeakElement(int[] arr) {

int start = 0;

int end = arr.length - 1;

while(start < end){

int mid = start + (end - start)/2;

if(arr[mid] > arr[mid+1]){

end = mid;

}

else{

start = mid +1;

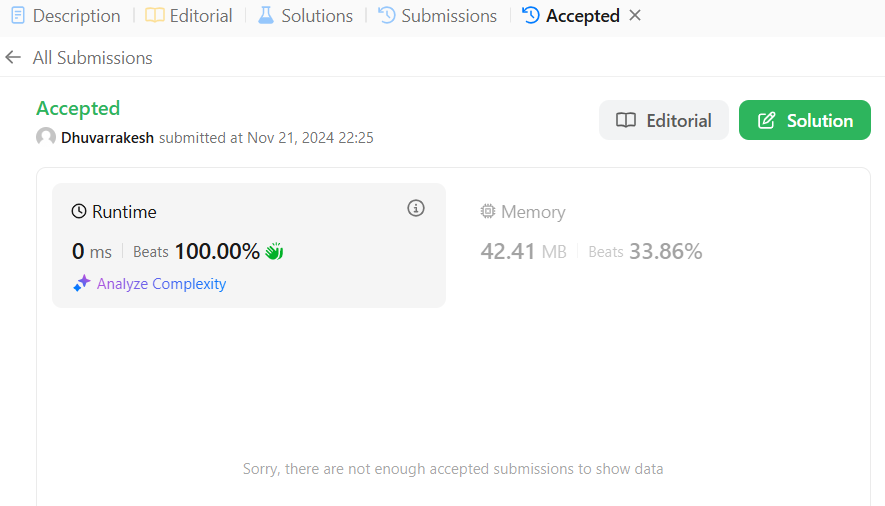
}

}

return start;

}

}

**Output:** ****

**18.Search in rotated sorted array**

**Code:**

class Solution {

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

        int low = 0, high = nums.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (nums[mid] == target) {

                return mid;

            }

            if (nums[low] <= nums[mid]) {

                if (nums[low] <= target && target < nums[mid]) {

                    high = mid - 1;

                } else {

                    low = mid + 1;

                }

            } else {

                if (nums[mid] < target && target <= nums[high]) {

                    low = mid + 1;

                } else {

                    high = mid - 1;

                }

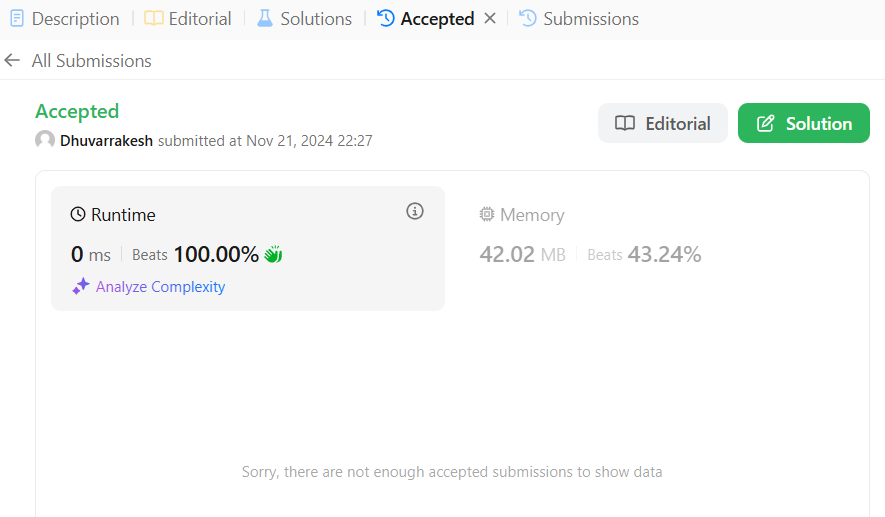
            }

        }

        return -1;

    }

}

**Output:** ****

**19.Find first and last position of element**

**Code:**

class Solution {

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

int arr[]= {-1,-1};

arr[0]=search(nums, target, true);

arr[1]=search(nums, target, false);

return arr;

}

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

int ans=-1;

int start=0;

int end=nums.length-1;

while(start<=end){

int mid= start + (end-start)/2;

if(nums[mid]<target){

start=mid+1;

}else if(nums[mid]>target){

end=mid-1;

}else{

ans=mid;

if(isFirstIdx){

end=mid-1;

}else{

start=mid+1;

}

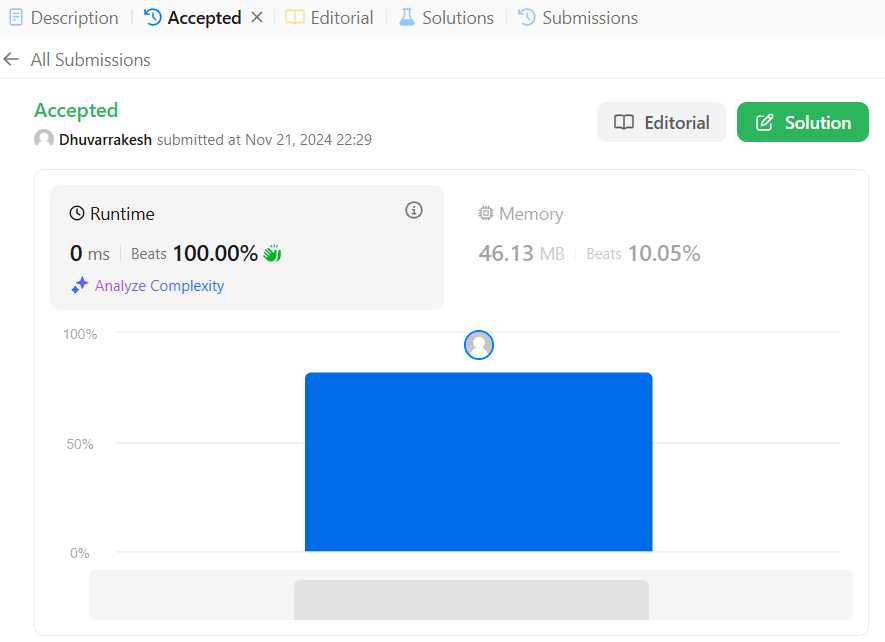
}

}

return ans;

}

}

**Output:** ****

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

**Code:**

class Solution {

    public int findMin(int[] nums) {

        int left = 0, right = nums.length - 1;

        while(left < right) {

            int mid = left + (right - left) / 2;

            if(nums[mid] < nums[right]) {

                right = mid;

            } else {

                left = mid + 1;

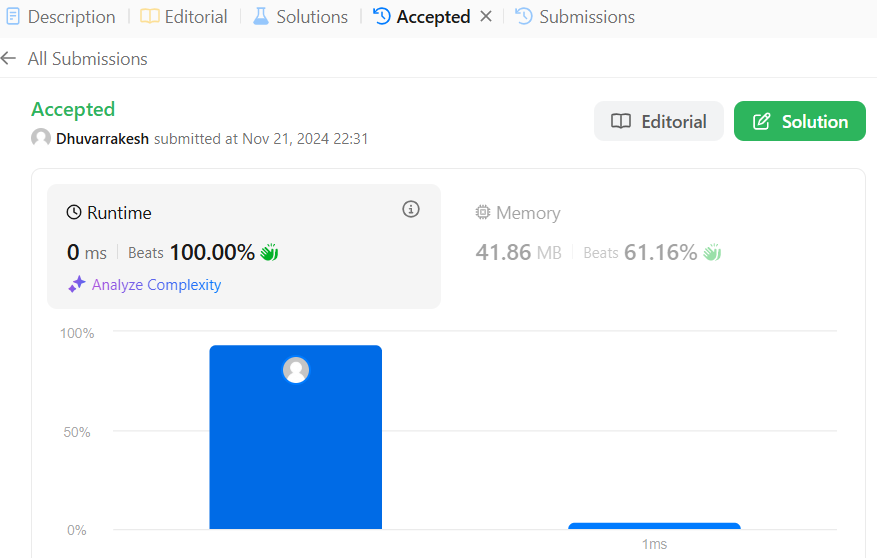
            }

        }

        return nums[left];

    }

}

**Output:** ****