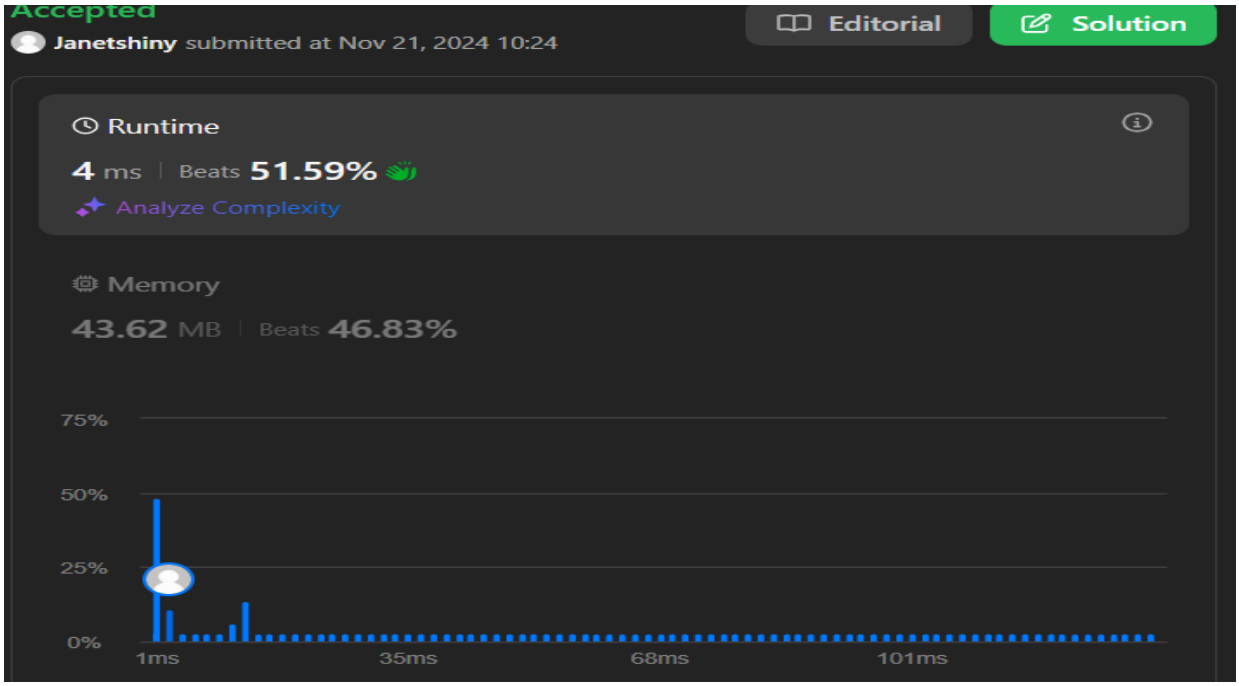
1.Valid Palindrome:

Solution:

```
class Solution {
    public boolean isPalindrome(String s) {
        StringBuilder a=new StringBuilder();
        for(char c:s.toCharArray()){
            if(Character.isLetterOrDigit(c)){
                a.append(Character.toLowerCase(c));
            }

        }
        String forward=a.toString();
        String rev=a.reverse().toString();
        return forward.equals(rev);
    }
}
```

Output:



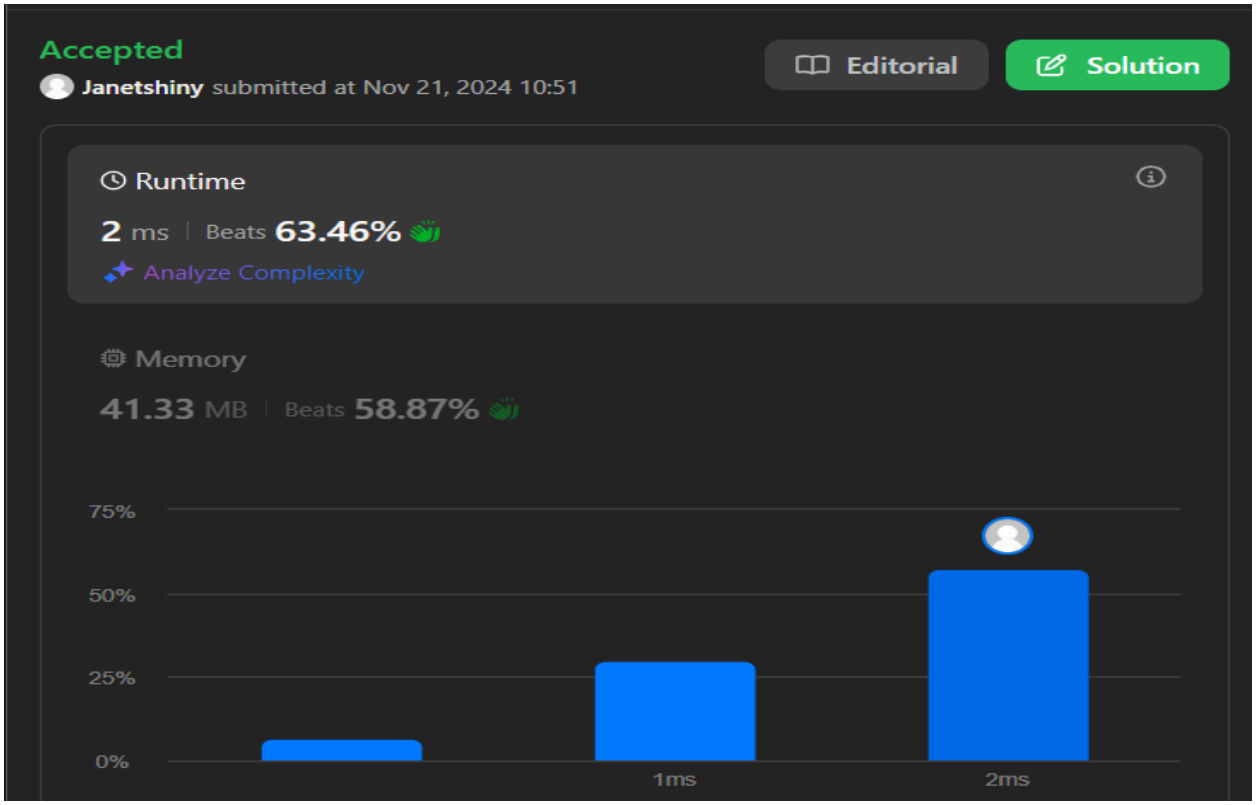
Time Complexity:O(n)
Space Complexity:O(n)

2.Is subsequence

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)
Space Complexity:O(1)

3.Two Sum II -Input Array is Sorted

Solution:

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

Output:

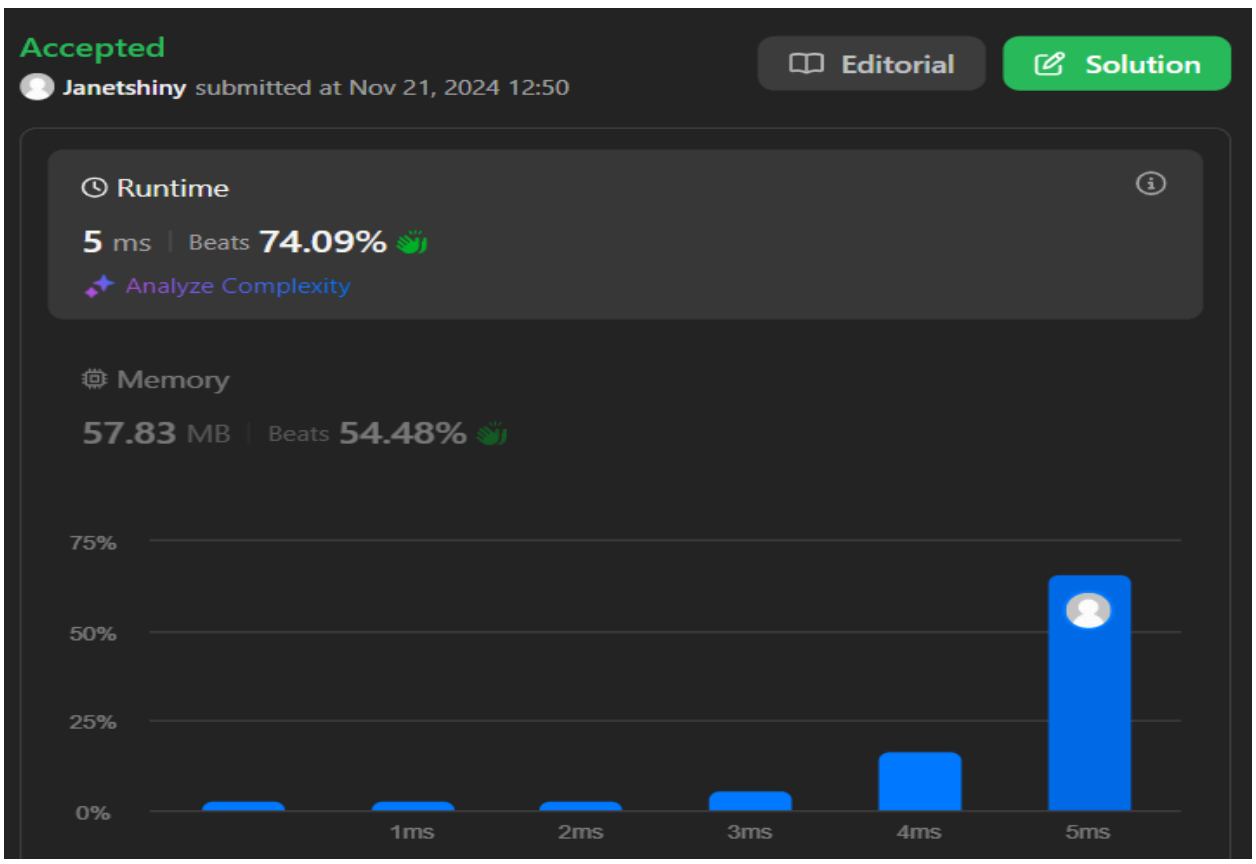
Time Complexity:O(n)
Space Complexity:O(1)

4.Container with Most water

Solution:

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

Output:



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

5.3Sum:

Solution:

```
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;
    }
}
```

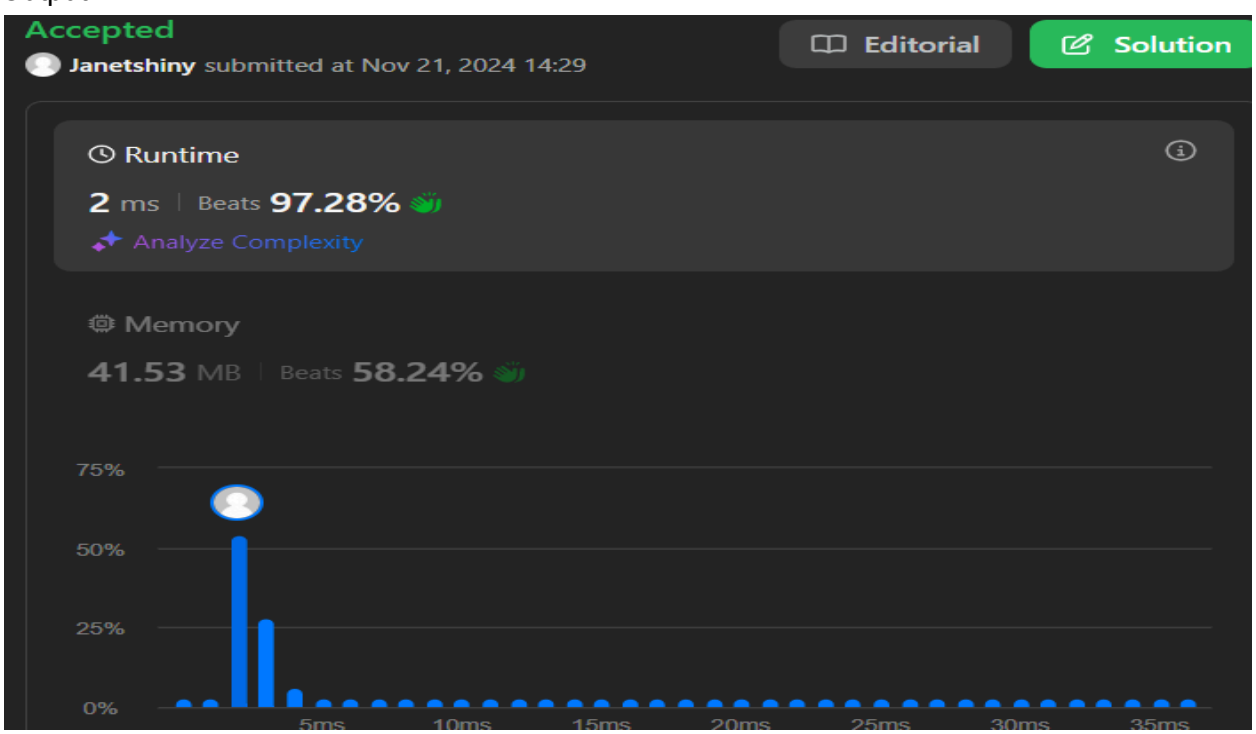
Time Complexity:O(n^2)
Space Complexity:O(n)

6.Valid parentheses

Solution:

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> a=new Stack<>();
        for(char c:s.toCharArray()){
            if(c=='(' || c=='{' ||c=='[') a.push(c);
            else{
                if(a.isEmpty()) return false;
                char top=a.peek();
                if(c==')' && top=='(' ||c=='}' && top=='{'||c==']' && top=='[')
a.pop();
                else{ return false;}
            }
        }
        return a.isEmpty();
    }
}
```

Output:



Time Complexity:O(n)

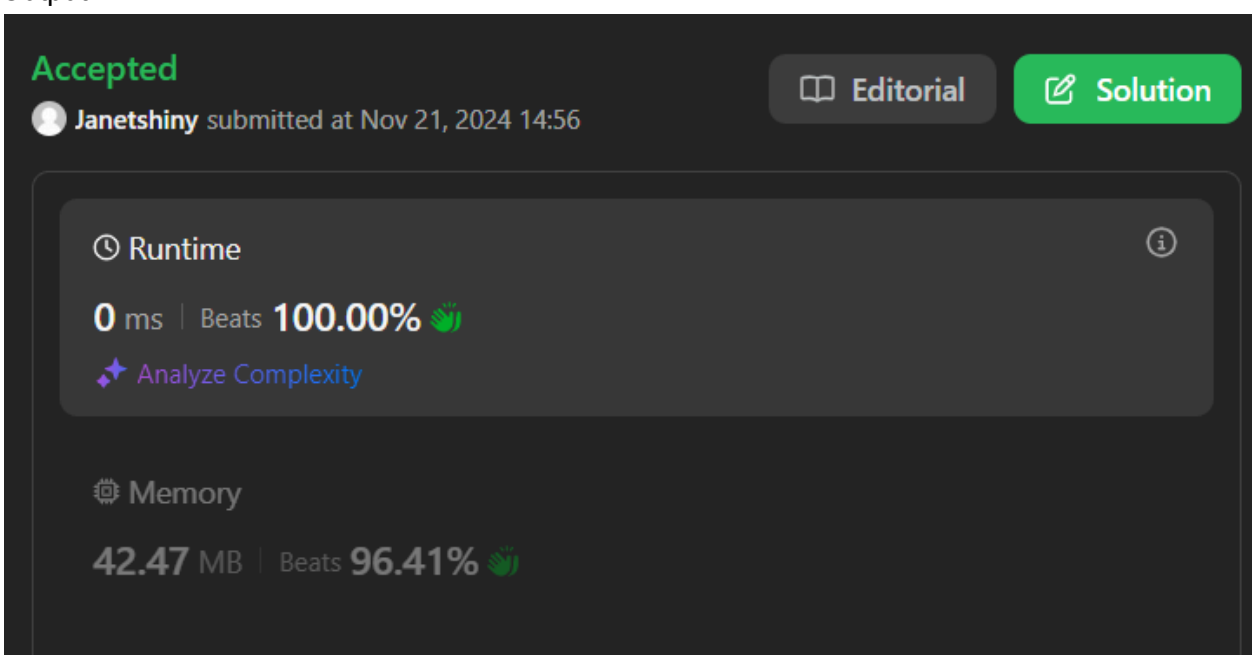
Space Complexity:O(n)

7.Search Insert Position

Solution:

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

       }
       return left;
    }
}
```

Output:



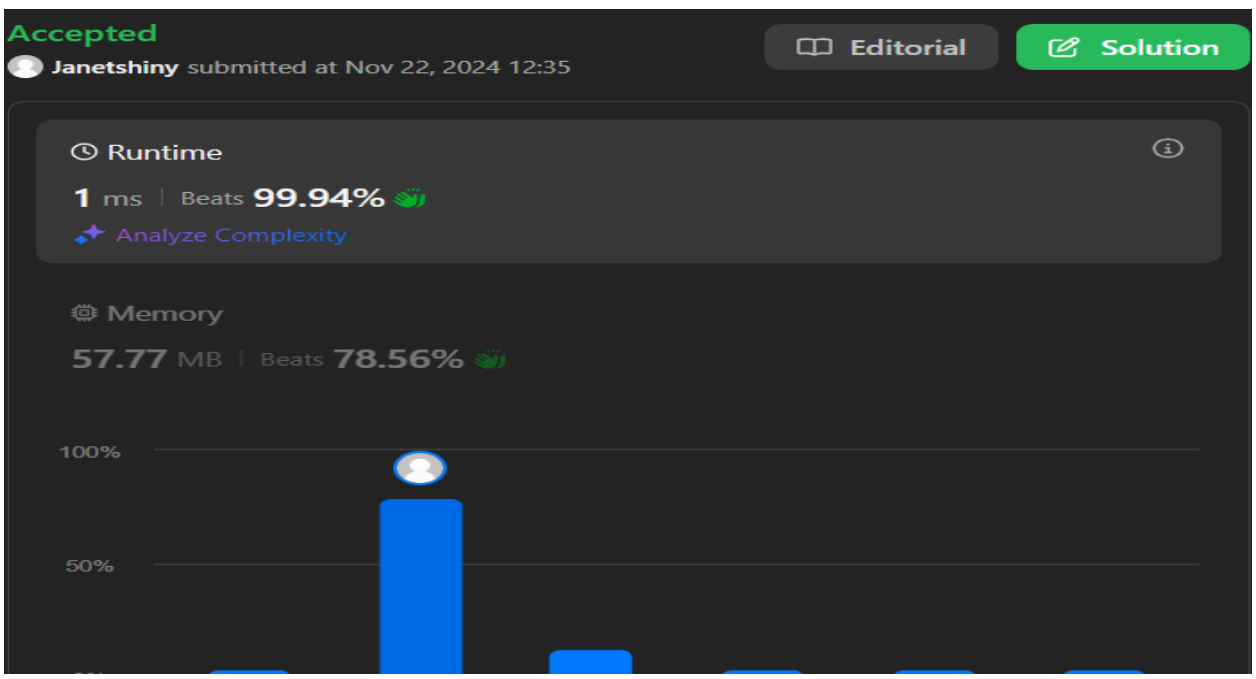
Time Complexity:O(log n)
Space Complexity:O(1)

8.Minimum Subarray Sum

Solution:

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

    }
}
```

Output:



Time Complexity:O(n)

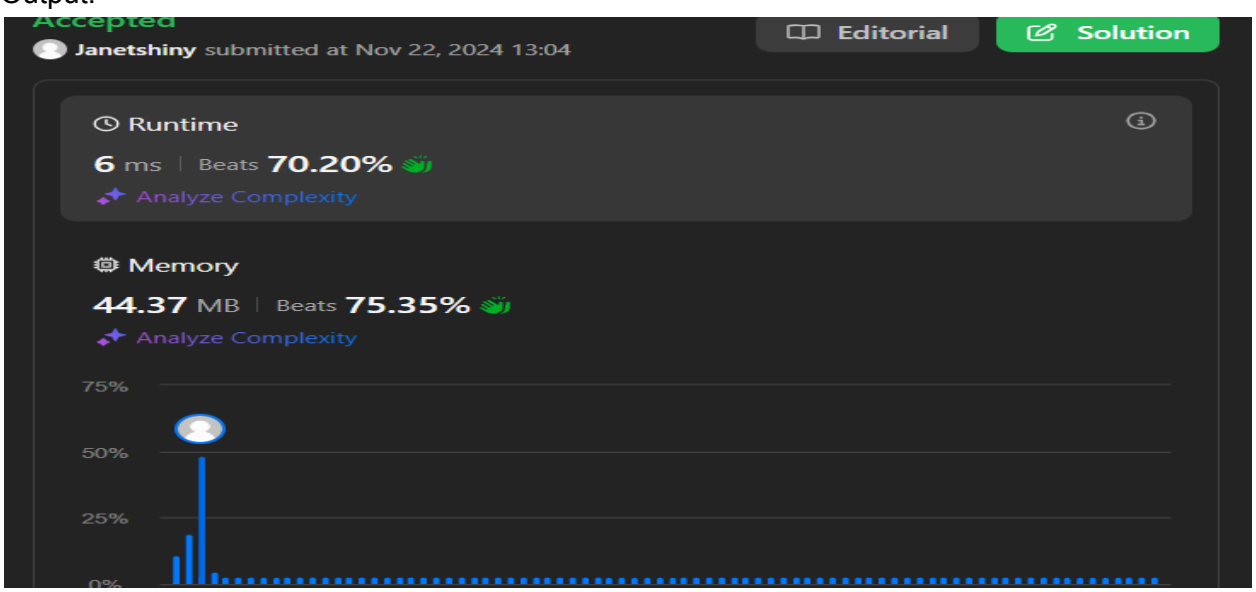
Space Complexity:O(1)

9.Longest Substring without Repeating Character

Solution:

```java
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:



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

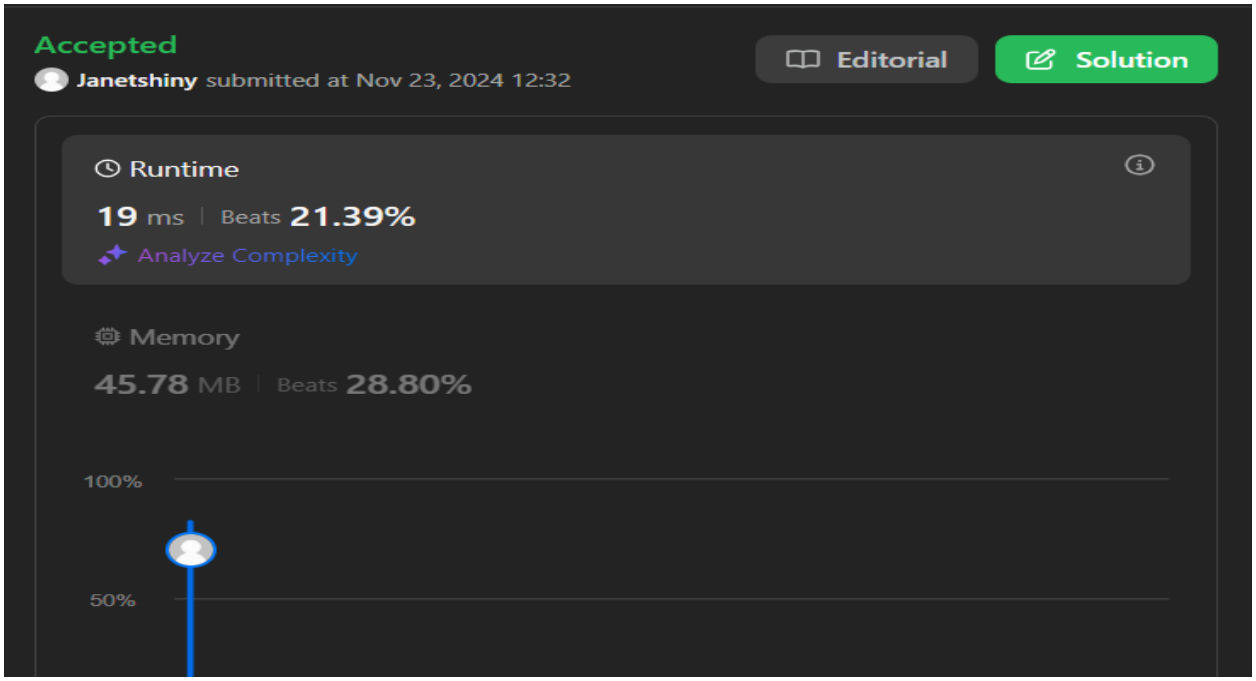
10.Substring with Concatenation of All Words

Solution:

```
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;
    }
}
```

Time Complexity:O(n*w)
Space Complexity:O(m)

Output:



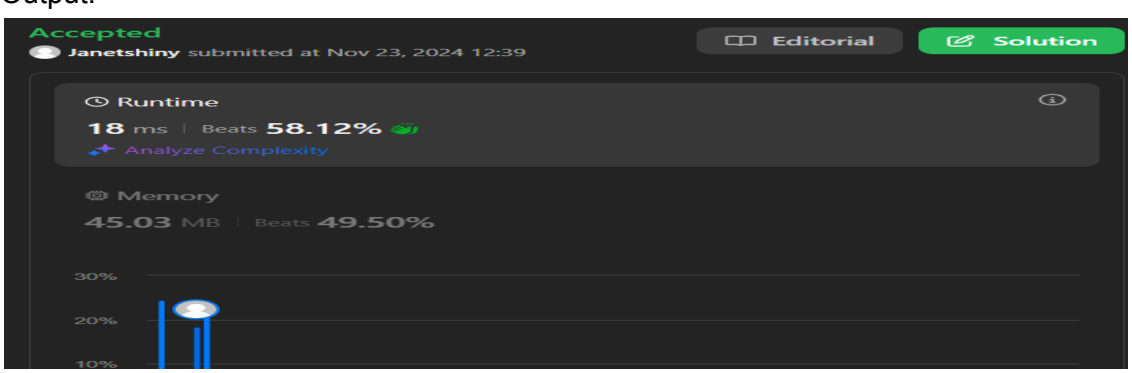
11.Minimum Window Substring

Solution:

```
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:



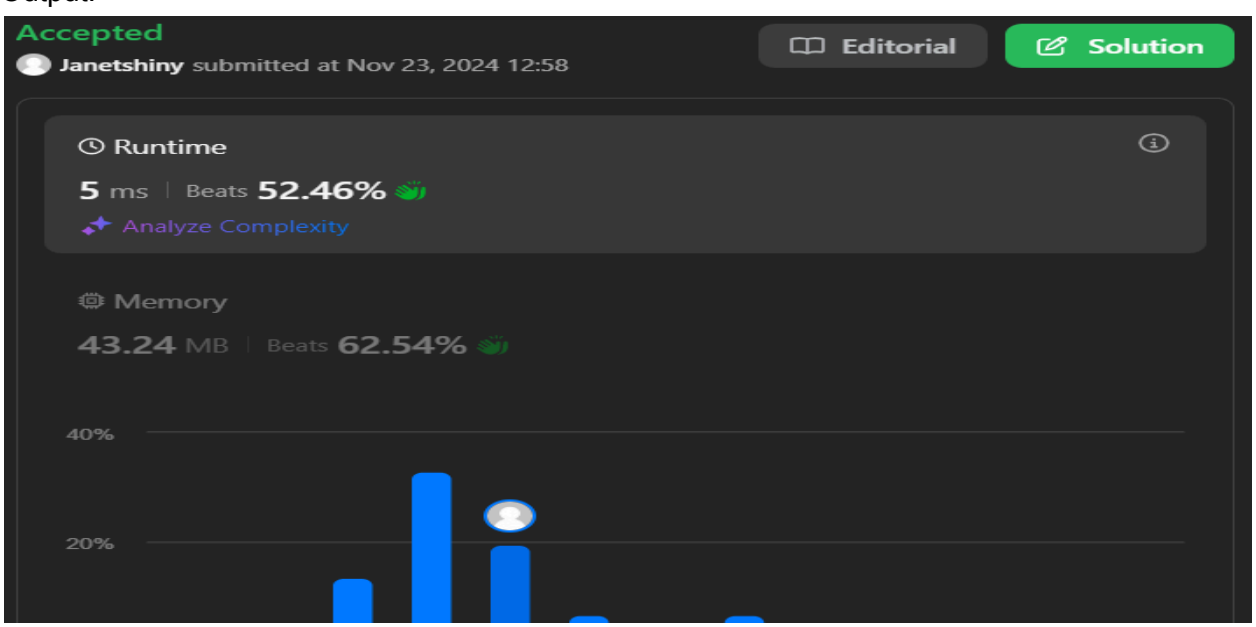
Time Complexity:O(s+t)
Space Complexity:O(s+t)

12.Simplify Path

Solution:

```
class Solution {
    public String simplifyPath(String path) {
        Stack<String> a=new Stack<>();
        String[] dir=path.split("/");
        for(String d:dir){
            if(d.equals(".") || d.isEmpty()){
                continue;
            }
            else if(d.equals("..")){
                if(!a.isEmpty()) a.pop();
            }
            else a.push(d);
        }
        return "/"+String.join("/",a);
    }
}
```

Output:



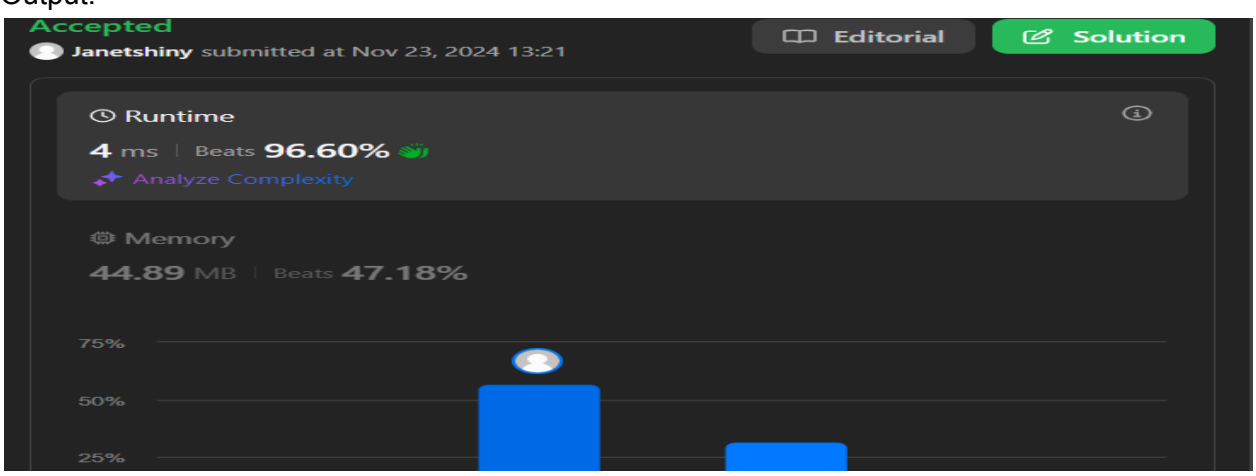
Time Complexity:O(n)
Space Complexity:O(n)

13.Min Stack

Solution:

```
class MinStack {
    private List<int[]> stack;
    public MinStack() {
         stack=new ArrayList<>();
    }

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

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

    public int top() {
          return stack.isEmpty() ? -1: stack.get(stack.size()-1)[0];
    }

    public int getMin() {
        return stack.isEmpty() ? -1: stack.get(stack.size()-1)[1];}}
```

Output:



Time Complexity:O(n)

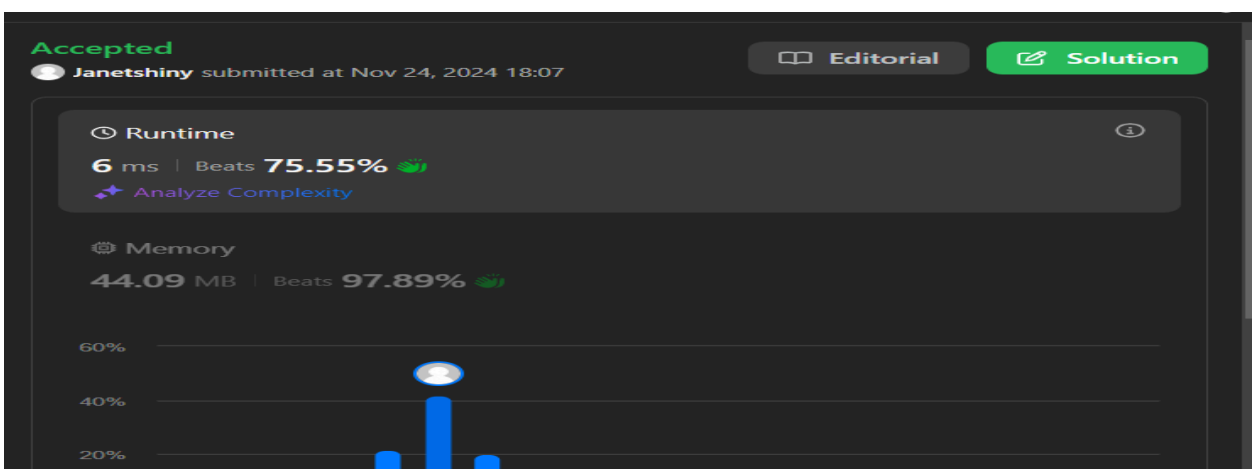
Space Complexity:O(n)

14.Evaluate Reverse Polish:

Solution:

```
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:



Time Complexity:O(n)

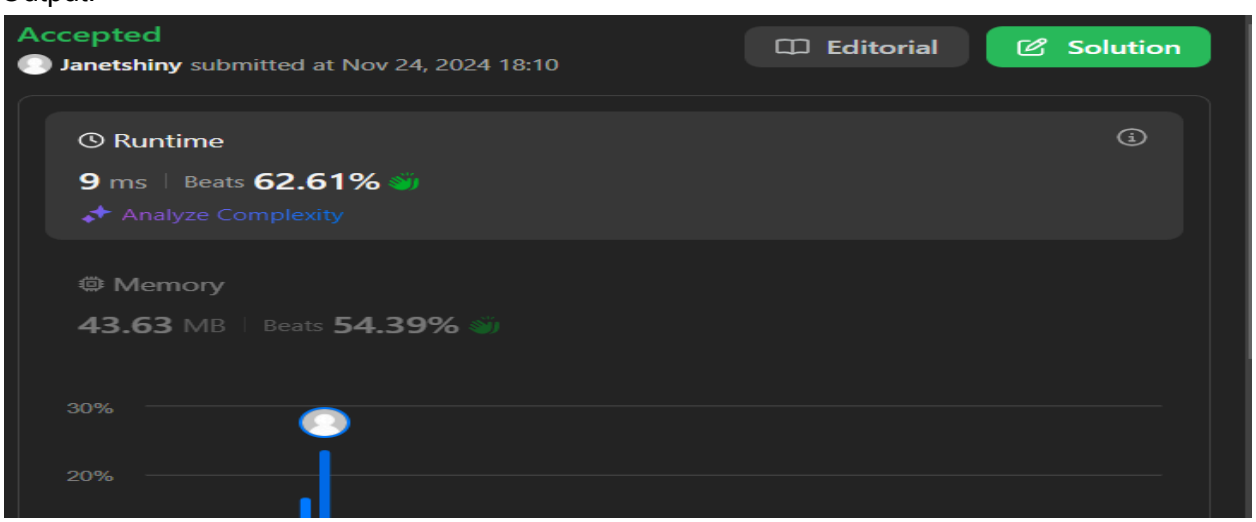
Space Complexity:O(n)

15.Basic Calculator

Solution:

```
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:



Time Complexity:O(n)

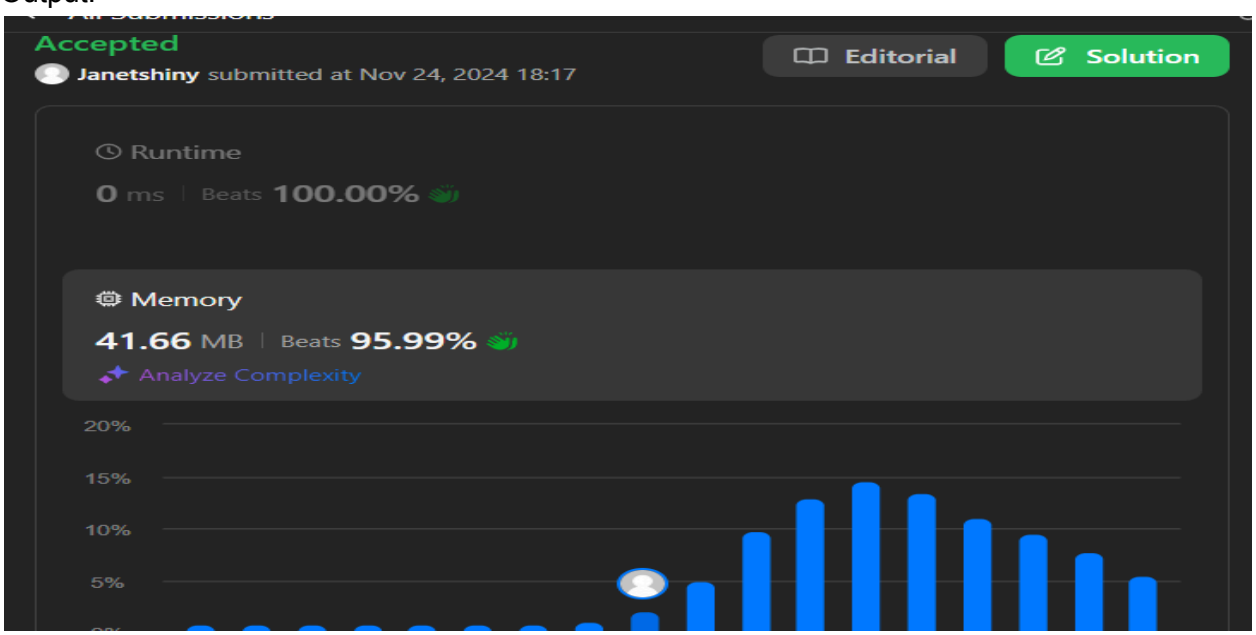
Space Complexity:O(n)

16.Search a 2D Matrix

Solution:

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int rows=matrix.length;
        int cols=matrix[0].length;
        int i=0;
        int j=rows*cols-1;
        while(i<=j){
            int mid=(i+j)/2;
            int row=mid/cols;
            int col=mid%cols;
            if(matrix[row][col]==target)
               return true;
            else if(matrix[row][col]>target)
               j=mid-1;
            else
               i=mid+1;
        }
        return false;
    }
}
```

Output:



Time Complexity:O(log(m*n))

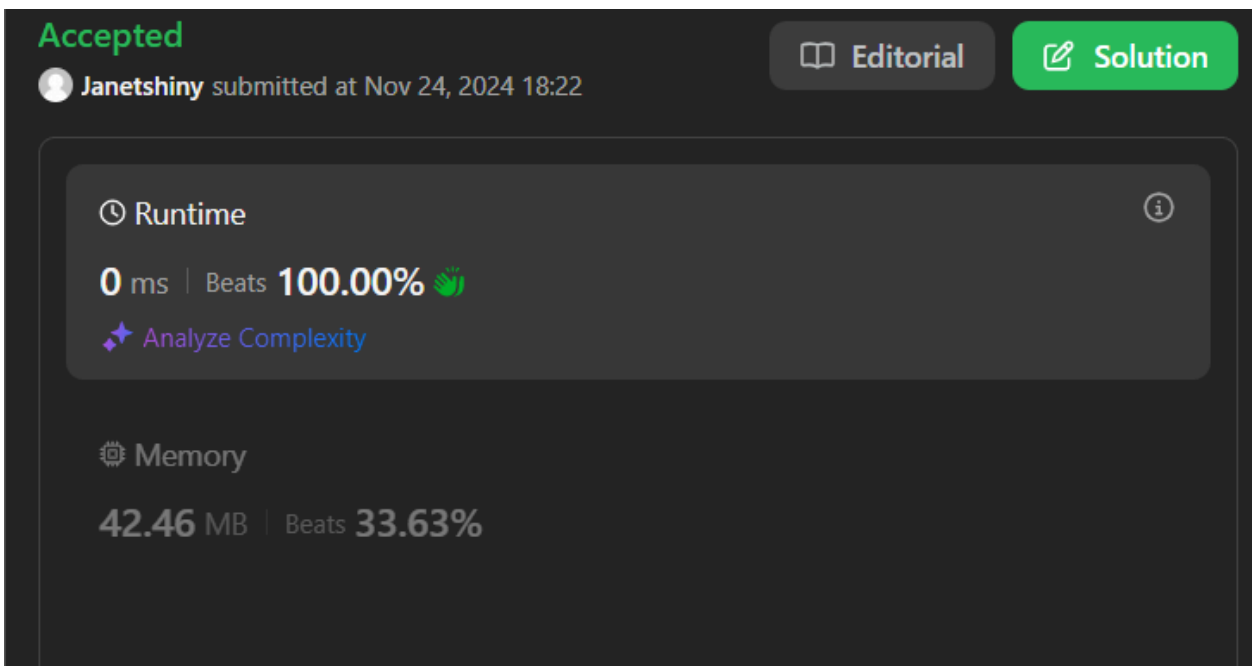
Space Complexity:O(1)

17.Find Peak element

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:



Time Complexity:O(log(n))

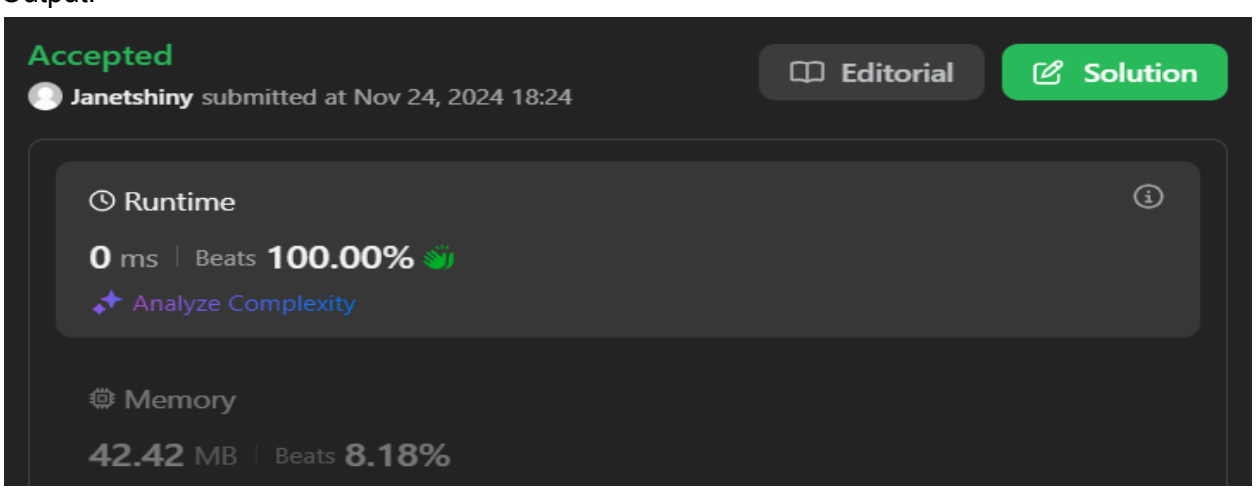
Space Complexity:O(1)

18.Search In Rotated Sorted Array

Solution:

```
class Solution {
    public int search(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] >= nums[left]) {
                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:



Time Complexity:O(log(n))

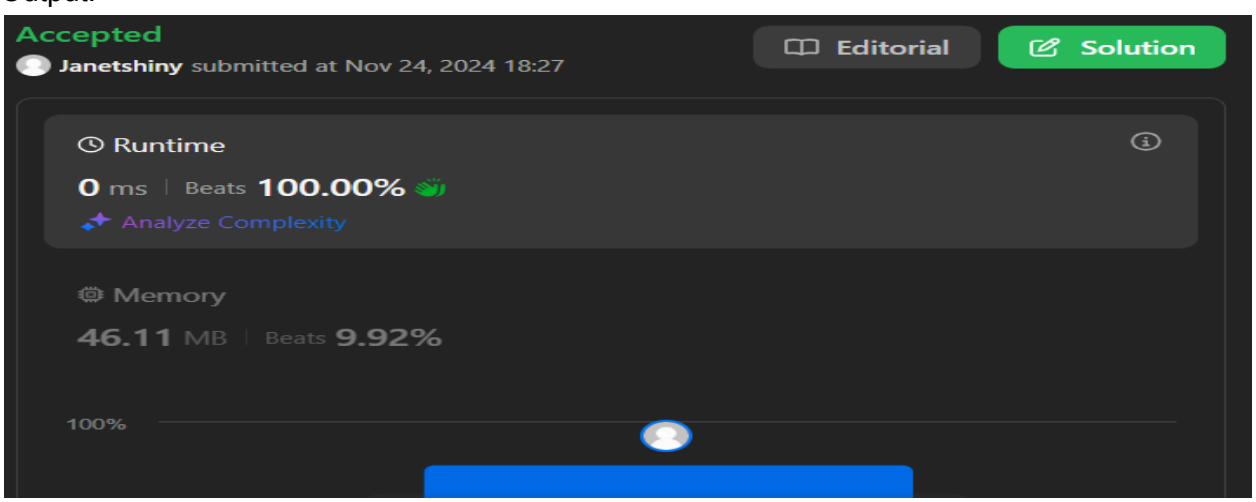
Space Complexity:O(1)

19.Find First and Last Position In a Sorted Array

Solution:

```
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;}}
```

Output:



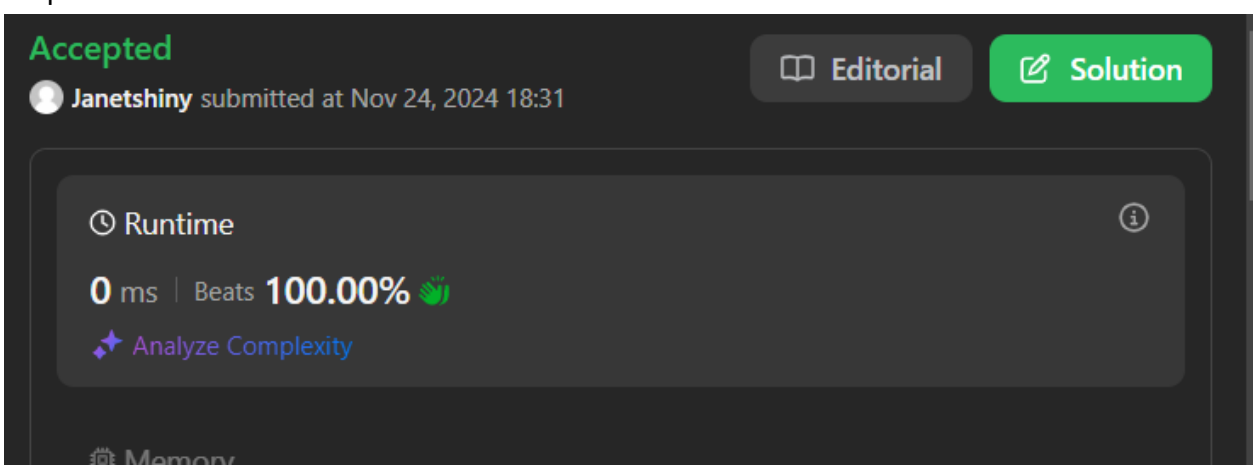
Time Complexity:O(log(n))

Space Complexity:O(1)

20.
Solution:

```
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:



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