Date: 19.11.2024

DSA PRACTICE - DAY 7

1. Next Permutation

class Solution {

    public void nextPermutation(int[] arr) {

        int n=arr.length;

        int pivot=-1;

        for(int i=n-2;i>=0;i--){

            if(arr[i]<arr[i+1]){

                pivot=i;

                break;

            }

        }

        if(pivot==-1){

            reverse(arr,0,n-1);

            return ;

        }

        for(int i=n-1;i>pivot;i--){

            if(arr[i]>arr[pivot]){

                swap(arr,i,pivot);

                break;

            }

        }

        reverse(arr,pivot+1,n-1);

    }

    static void reverse(int[] arr,int start,int end){

        while(start<end){

            swap(arr,start++,end--);

        }

    }

    static void swap(int[] arr, int i,int j){

        int temp=arr[i];

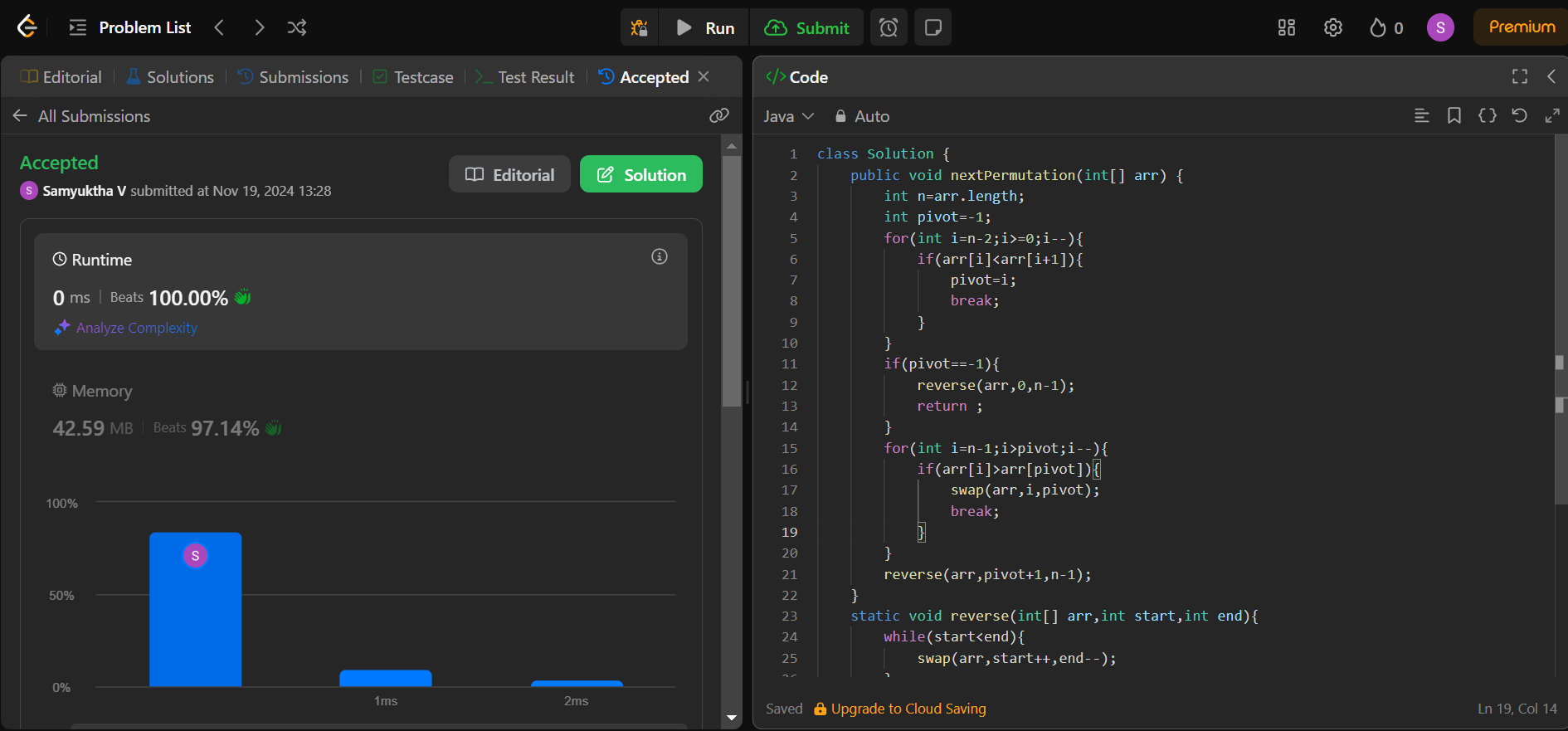
        arr[i]=arr[j];

        arr[j]=temp;

    }

}

Output:



Time Complexity: O(N)

Space Complexity: O(1)

1. Spiral Matrix

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

**Example 1:**



**Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]

**Output:** [1,2,3,6,9,8,7,4,5]

Solution:

class Solution {

    public List<Integer> spiralOrder(int[][] mat) {

        int m=mat.length;

        int n=mat[0].length;

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

        int t=0,b=m-1,l=0,r=n-1;

        while(t<=b && l<=r){

            for(int i=l;i<r+1;i++){

                ans.add(mat[t][i]);

            }

            t++;

            for(int i=t;i<=b;i++){

                ans.add(mat[i][r]);

            }

            r--;

            if(t<=b){

                for(int i=r;i>=l;i--){

                    ans.add(mat[b][i]);

                }

                b--;

            }

            if(l<=r){

                for(int i=b;i>=t;i--){

                    ans.add(mat[i][l]);

                }

                l++;

            }

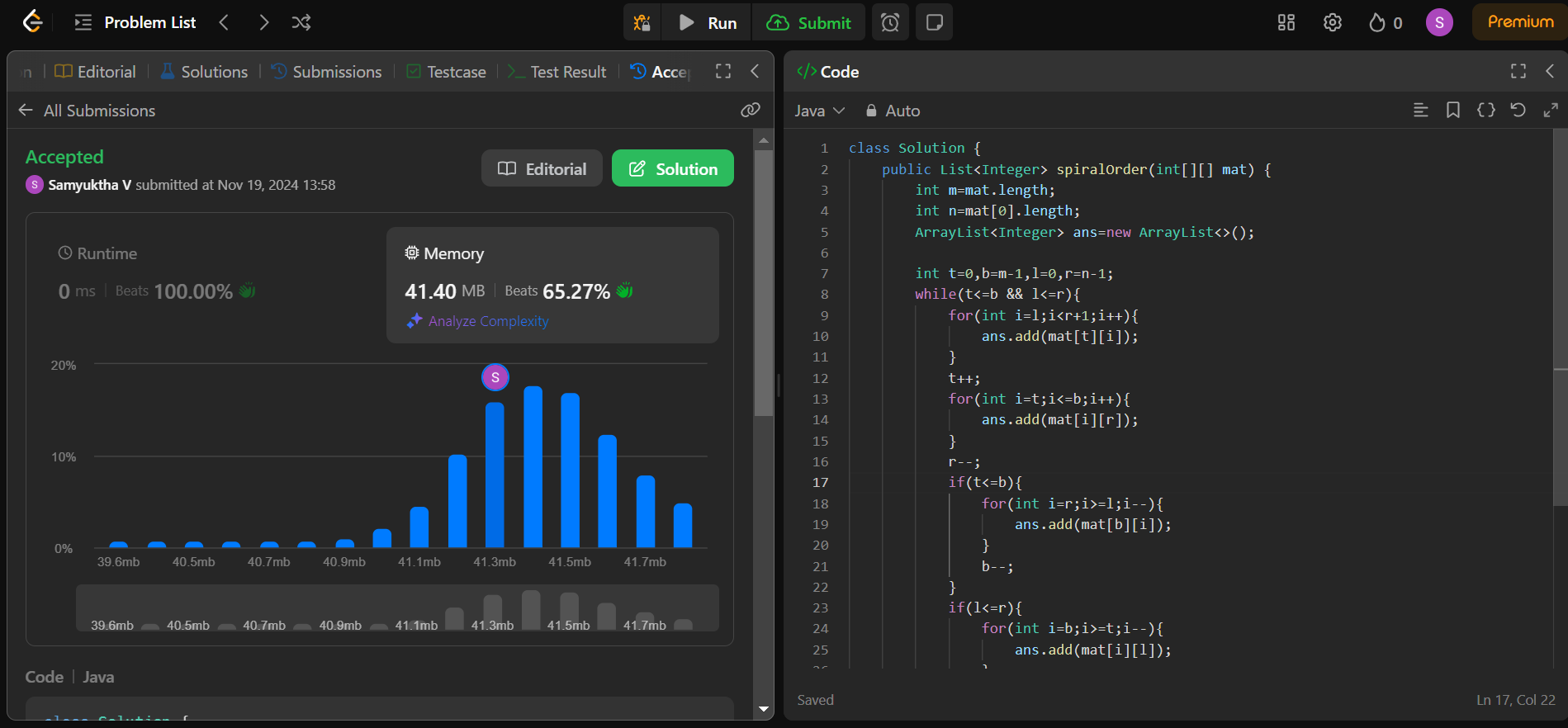
        }

        return ans;

    }

}

Output:



Time Complexity: O(M\*N)

Space Complexity: O(N)

1. Longest Substring without Repeating Character

Given a string s, find the length of the **longest** **substring** without repeating characters.

**Example 1:**

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

Solution:

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int n=s.length();

        if(n==0 || n==1){

            return n;

        }

        int max=-1;

        boolean[] visited=new boolean[256];

        int left=0,right=0;

        while(right<n){

            while(visited[s.charAt(right)]){

                visited[s.charAt(left)]=false;

                left++;

            }

            visited[s.charAt(right)]=true;

            max=Math.max(max,right-left+1);

            right++;

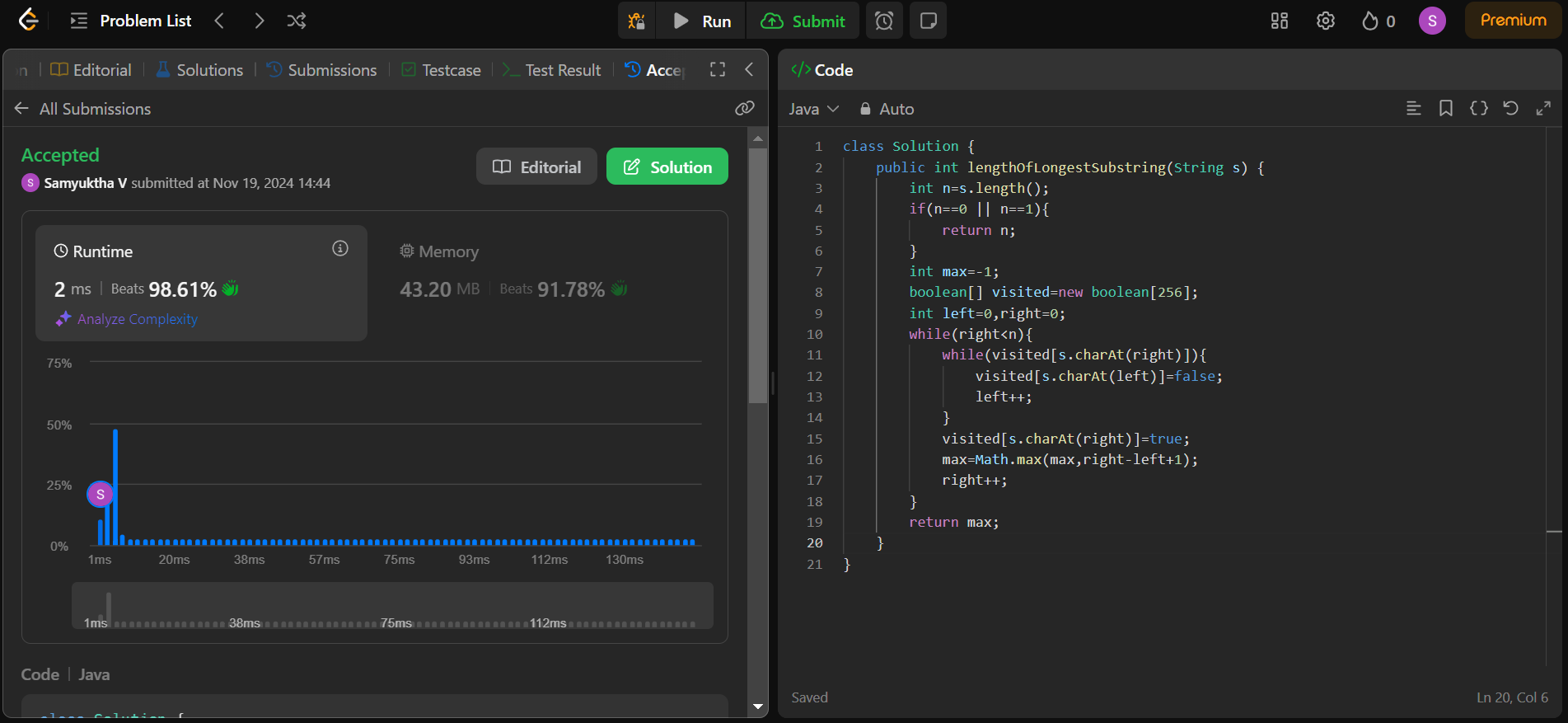
        }

        return max;

    }

}

Output:



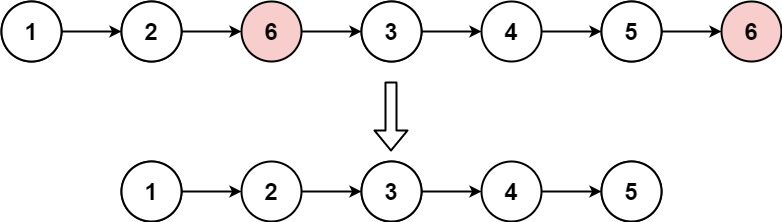
Time Complexity: O(N)

Space Complexity: O(1)

1. Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

**Example 1:**



**Input:** head = [1,2,6,3,4,5,6], val = 6

**Output:** [1,2,3,4,5]

Solution:

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode removeElements(ListNode head, int val) {

        if(head==null) return head;

        ListNode dummy=new ListNode(0);

        dummy.next=head;

        ListNode current=dummy;

        while(current.next!=null){

            if(current.next.val==val){

                current.next=current.next.next;

            }

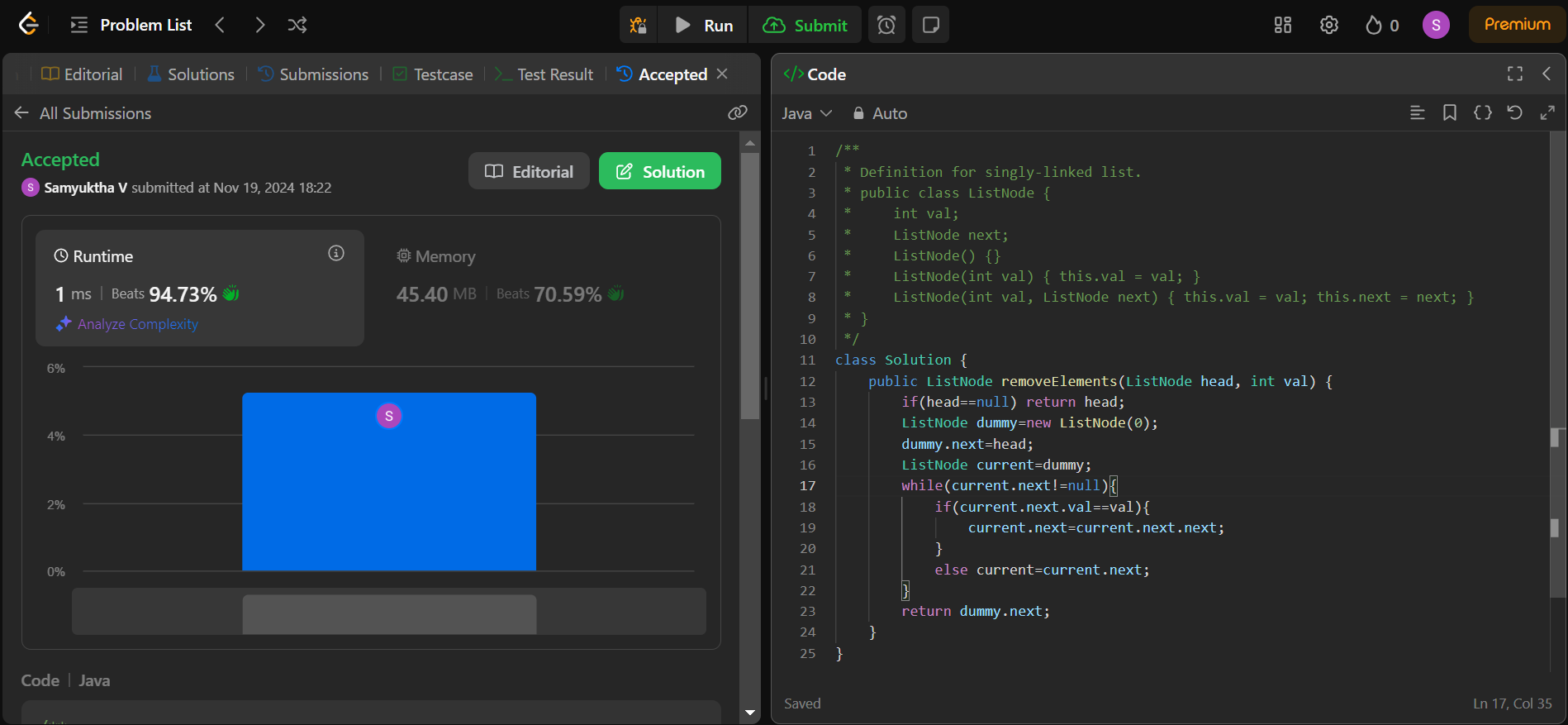
            else current=current.next;

        }

        return dummy.next;

    }

Output:



Time Complexity: O(N)

Space Complexity: O(1)

1. Palindrome Linked List

Given the head of a singly linked list, return true*if it is a palindrome or*false*otherwise*.

**Example 1:**



**Input:** head = [1,2,2,1]

**Output:** true

Solution:

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public boolean isPalindrome(ListNode head) {

        ListNode slow=head,fast=head;

        while(fast.next!=null && fast.next.next!=null){

            slow=slow.next;

            fast=fast.next.next;

        }

        ListNode head2=reverse(slow.next);

        slow.next=null;

        while(head!=null && head2!=null){

            if(head.val!=head2.val){

                return false;

            }

            head=head.next;

            head2=head2.next;

        }

        return true;

    }

    static ListNode reverse(ListNode head){

        ListNode prev,next,curr;

        prev=null;

        curr=head;

        while(curr!=null){

            next=curr.next;

            curr.next=prev;

            prev=curr;

            curr=next;

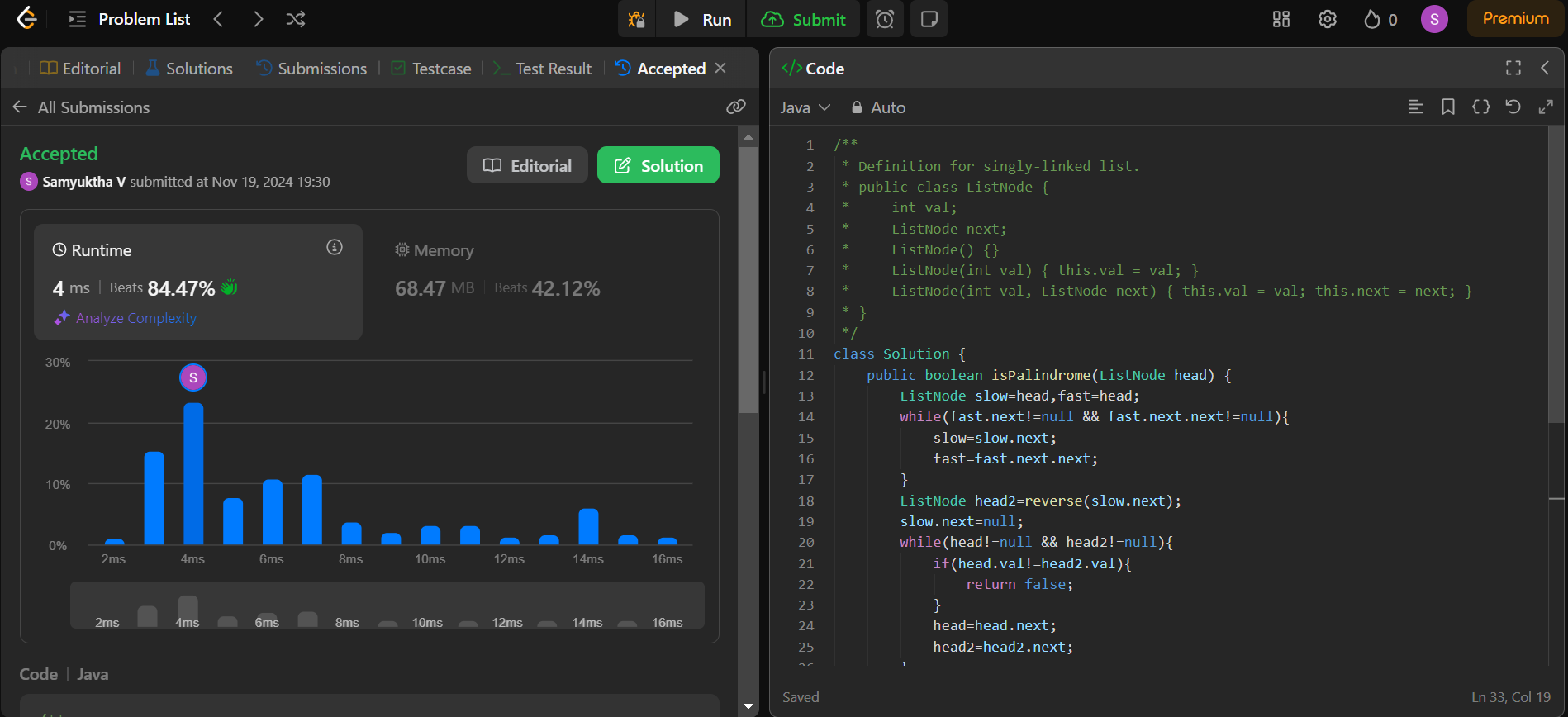
        }

        return prev;

    }

}

Output:



Time Complexity: O(N)

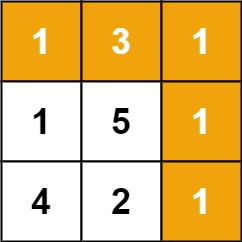
Space Complexity: O(1)

1. Minimum Path Sum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

**Example 1:**



**Input:** grid = [[1,3,1],[1,5,1],[4,2,1]]

**Output:** 7

**Explanation:** Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Solution:

class Solution {

    public int minPathSum(int[][] grid) {

        int m=grid.length,n=grid[0].length;

        for(int i=1;i<n;i++){

            grid[0][i]+=grid[0][i-1];

        }

        for(int i=1;i<m;i++){

            grid[i][0]+=grid[i-1][0];

        }

        for(int i=1;i<m;i++){

            for(int j=1;j<n;j++){

                grid[i][j]+=Math.min(grid[i-1][j],grid[i][j-1]);

            }

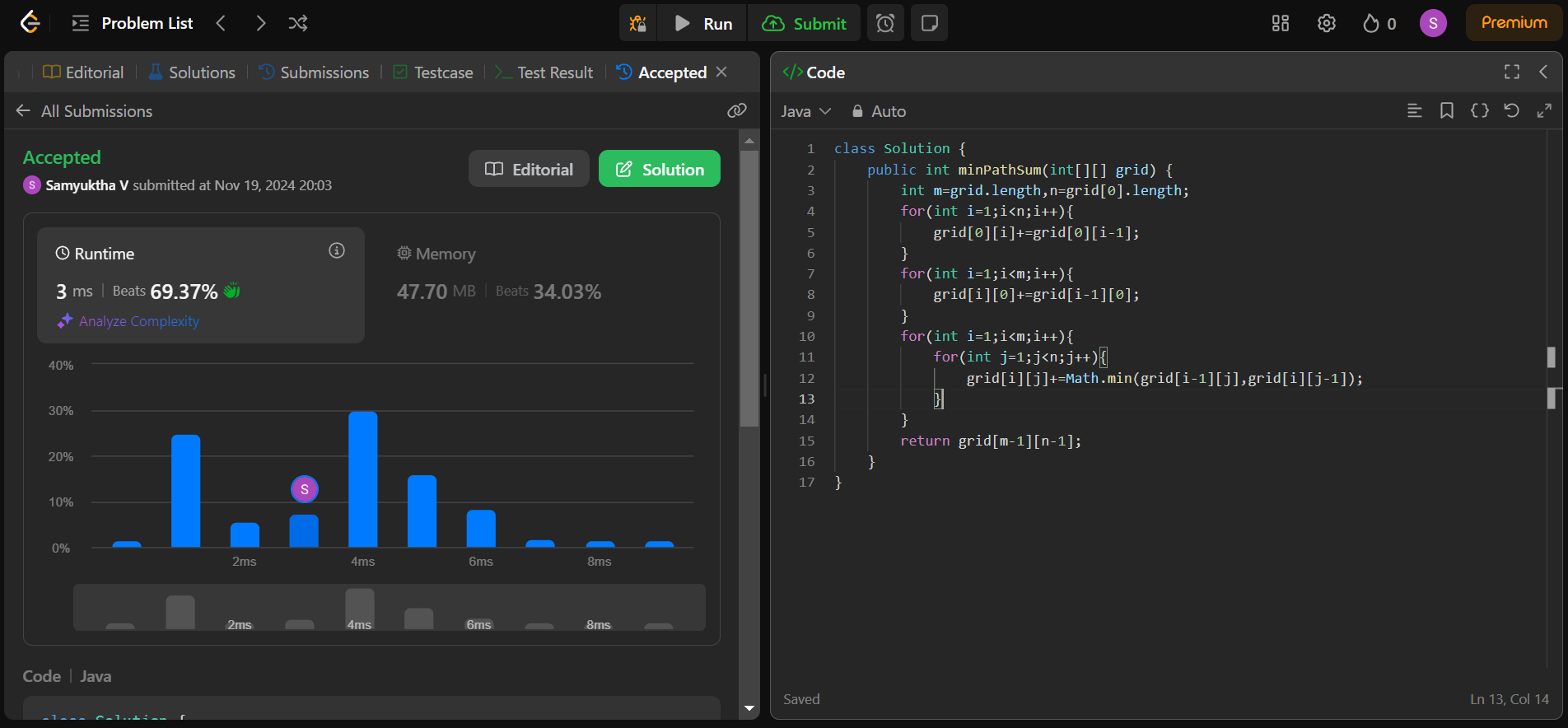
        }

        return grid[m-1][n-1];

    }

}

Output:



Time Complexity: O(N)

Space Complexity: O(1)