Name : Ramanarayanan k 19/11/2024

22IT083 (IT)

1)next permutation

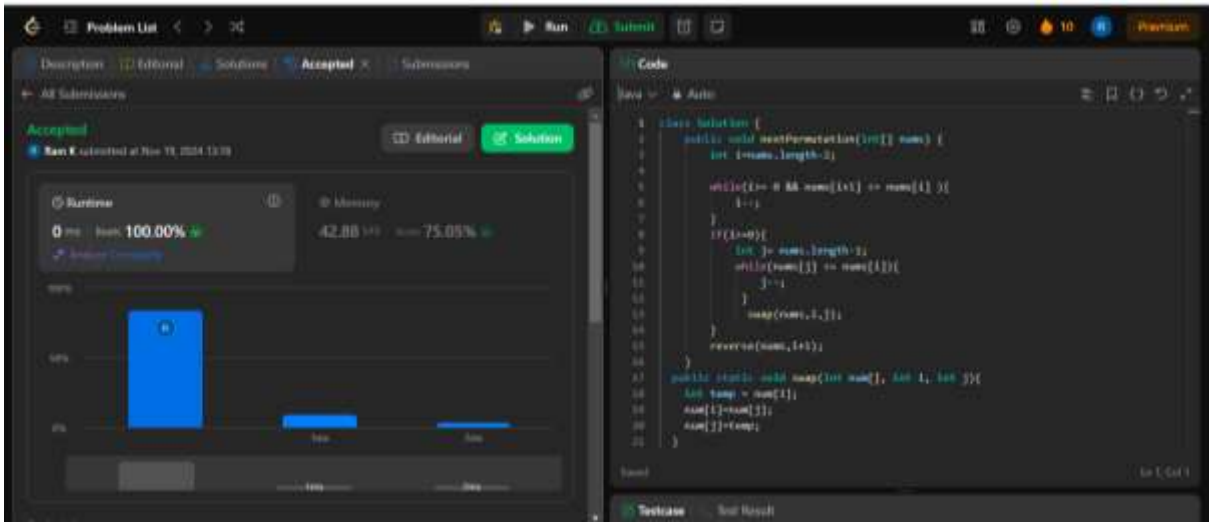
Code :

```
class Solution {
    public void nextPermutation(int[] nums) {
        int i=nums.length-2;

        while(i>= 0 && nums[i+1] <= nums[i] ){
            i--;
        }
        if(i>=0){
            int j= nums.length-1;
            while(nums[j] <= nums[i]){
                j--;
             }
              swap(nums,i,j);
        }
        reverse(nums,i+1);
    }
  public static void swap(int num[], int i, int j){
    int temp = num[i];
    num[i]=num[j];
    num[j]=temp;
  }
  public static void reverse(int num[], int p){
```

```
    int n = p;
    int l = num.length-1;
    while(n<l){
        swap(num,n,l);
        n++;
        l--;
    }
     }
}
```

Output:

TC:o(n)



2)Spiral Matrix

CODE :

```
import java.util.*;
```

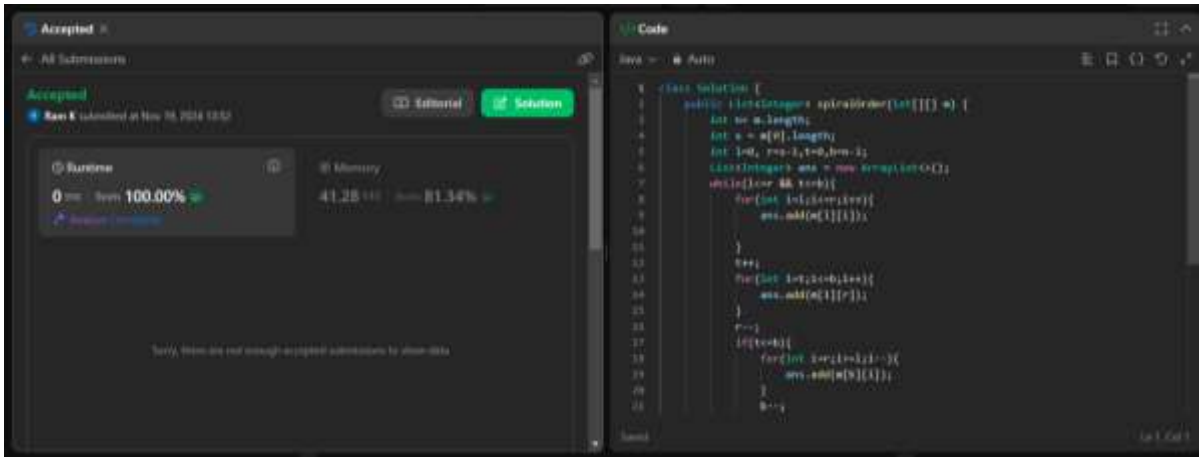
```
public class SpiralOrderMatrix {

    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix == null || matrix.length == 0) {
            return result;
        }

        int m = matrix.length;
        int n = matrix[0].length;

        int top = 0, bottom = m - 1, left = 0, right = n - 1;

        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                result.add(matrix[top][i]);
            }
            top++;

            for (int i = top; i <= bottom; i++) {
                result.add(matrix[i][right]);
            }
            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
```

```
            result.add(matrix[bottom][i]);
        }
        bottom--;
    }

    if (left <= right) {
        for (int i = bottom; i >= top; i--) {
            result.add(matrix[i][left]);
        }
        left++;
    }
  }

  return result;
}

public static void main(String[] args) {
  int[][] matrix = {
        { 1, 2, 3, 48 },
        { 5, 6, 7, 8 },
        { 9, 10, 11, 12 },
        { 13, 14, 15, 16 }
  };

  List<Integer> result = spiralOrder(matrix);
```

```
        for (int num : result) {
            System.out.print(num + " ");
        }
    }
}
```

Output:

TC:O(n*m)



3)Longest substring without repeating characters

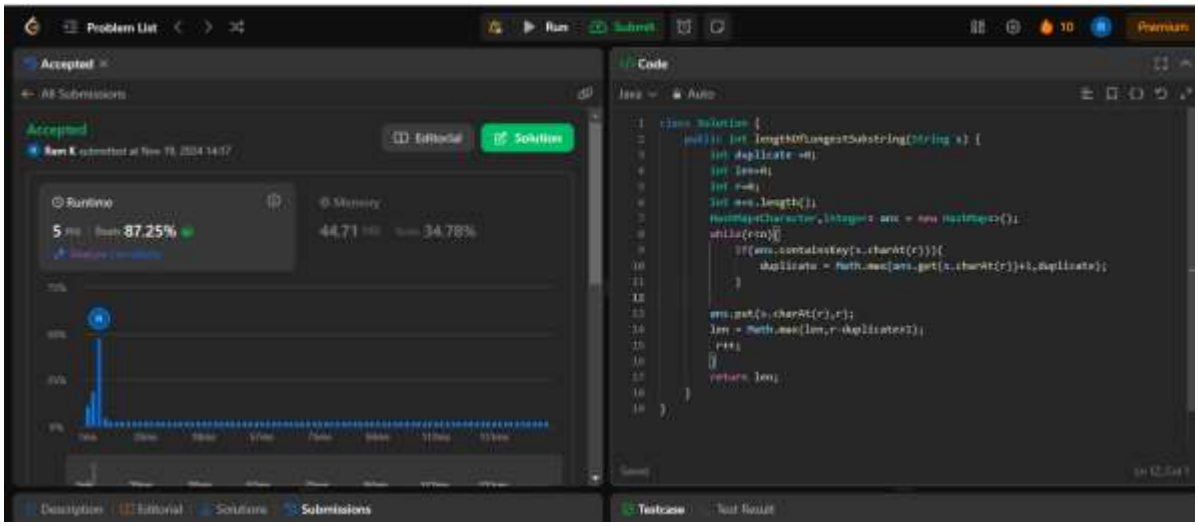
Code :

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int duplicate =0;
        int len=0;
        int r=0;
        int n=s.length();
        HashMap<Character,Integer> ans = new HashMap<>();
        while(r<n){
```

```
            if(ans.containsKey(s.charAt(r))){
                duplicate = Math.max(ans.get(s.charAt(r))+1,duplicate);
            }

        ans.put(s.charAt(r),r);
        len = Math.max(len,r-duplicate+1);
         r++;
        }
        return len;
    }
}
```

Output:

TC:O(n)



4)Remove Linked list Elements

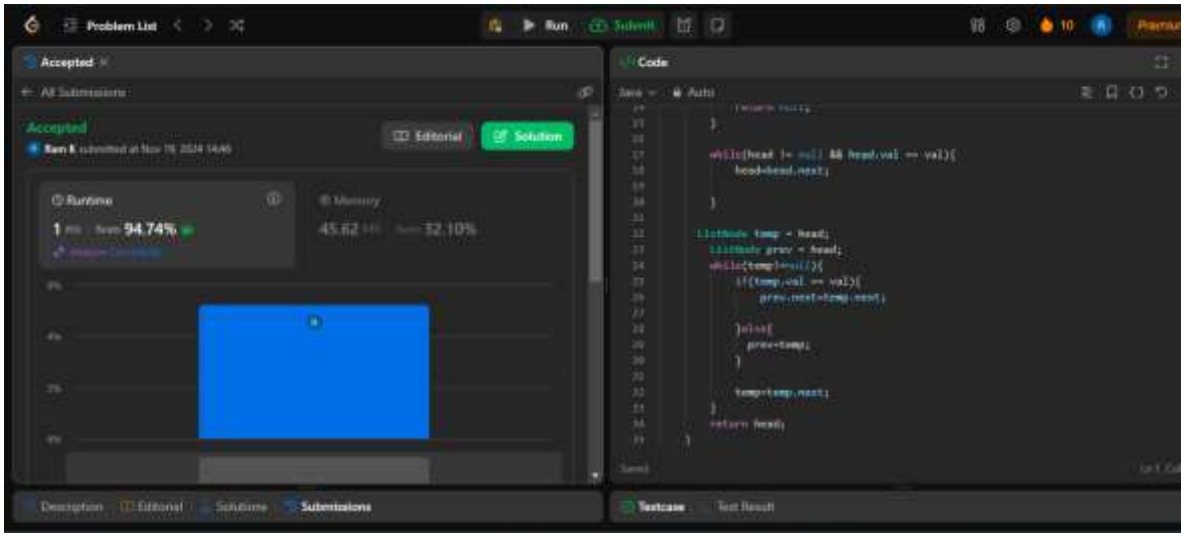
Code:

```
class Solution {
```

```
    public ListNode removeElements(ListNode head, int val) {
        if(head == null){
            return null;
        }

        while(head != null && head.val == val){
            head=head.next;


        }

       ListNode temp = head;
        ListNode prev = head;
        while(temp!=null){
            if(temp.val == val){
                prev.next=temp.next;

            }else{
              prev=temp;
            }

            temp=temp.next;
        }
        return head;
    }
}
```

Output:

TC:o(n)



5) Palindrome Linked list

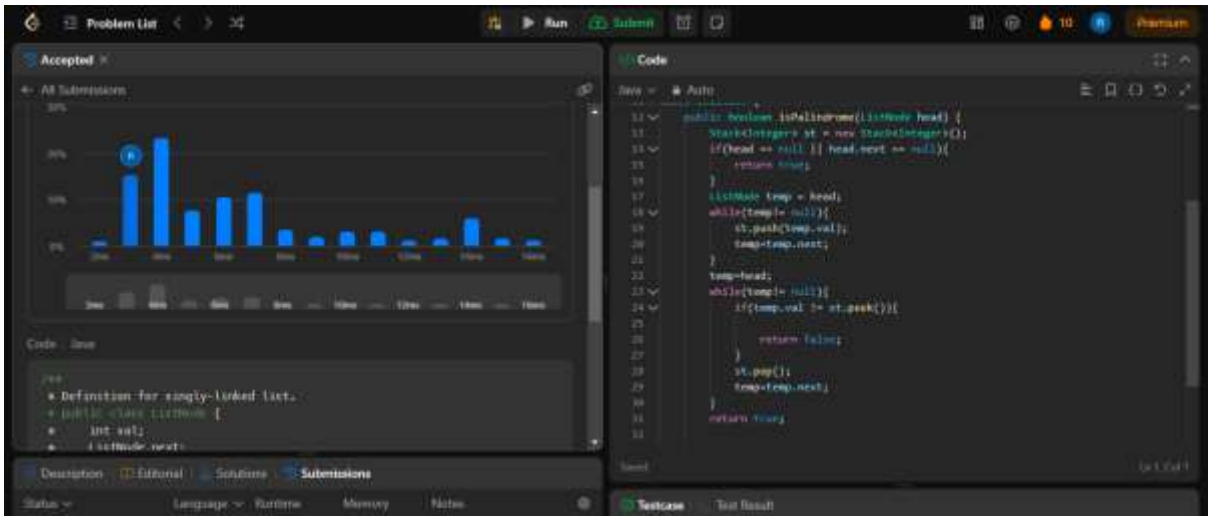
Code:

```
import java.util.*;

public class Palindromic{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        boolean flag=true;

        for (int i = 0; i < s.length(); i++) {
            for (int j = 0; j <= i; j++) {
                StringBuilder x = new StringBuilder();
                String a = s.substring(j, s.length() - i + j);
                x.append(a);
                String b = x.reverse().toString();
```

```
            if (a.equals(b) && flag==true) {
                System.out.print(a);
                flag=false;
            }
        }
    }
}
```

Output:

TC:o(n)



6)Validate binary serach tree

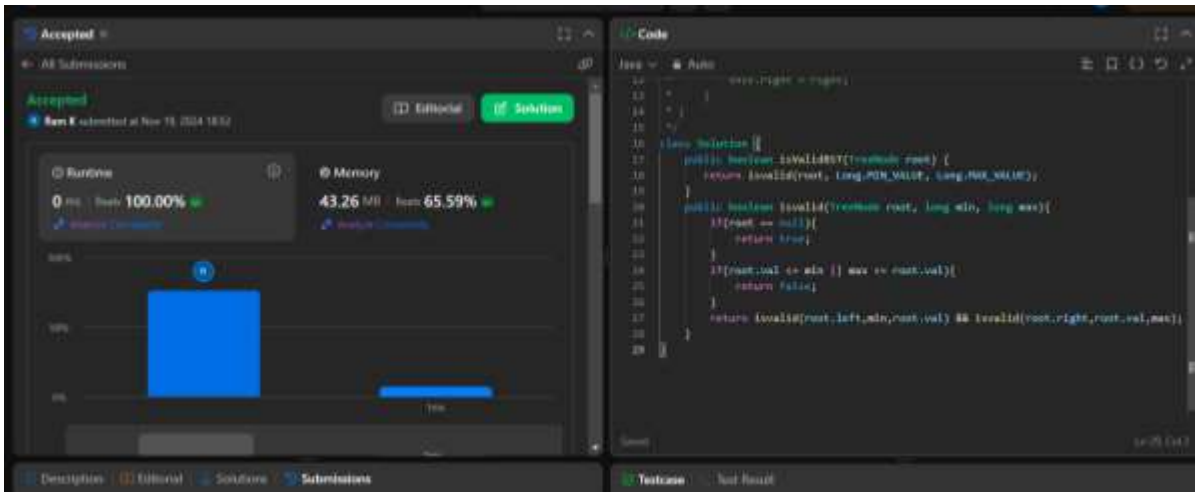
Code:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isvalid(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
    public boolean isvalid(TreeNode root, long min, long max){
```

```
        if(root == null){
            return true;
        }
        if(root.val <= min || max <= root.val){
            return false;
        }
        return isvalid(root.left,min,root.val) && isvalid(root.right,root.val,max);
    }
}
```

Output:

TC:O(1)



7)Word Ladder

Code :

```
import java.util.*;

class Pair {
```

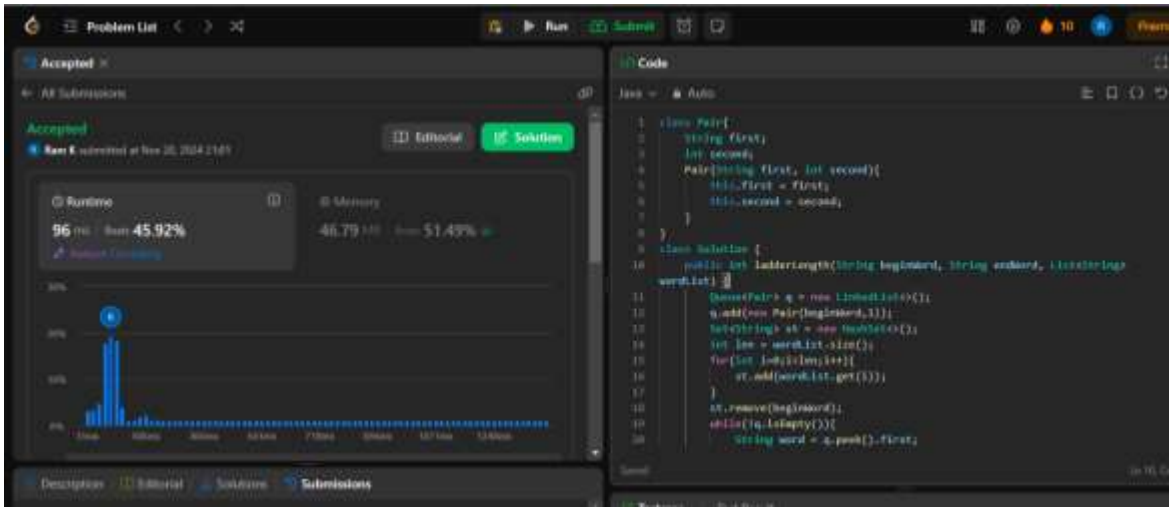
```
    String first;
    int second;

    Pair(String first, int second) {
        this.first = first;
        this.second = second;
    }
}

class wordladder1 {
    public int ladderLength(String beginWord, String endWord, List<String>
wordList) {

        Set<String> st = new HashSet<>(wordList);



        if (!st.contains(endWord)) {
            return 0;
        }



        Queue<Pair> q = new LinkedList<>();
        q.add(new Pair(beginWord, 1));
        st.remove(beginWord);

        while (!q.isEmpty()) {
            String word = q.peek().first;
```

```
        int step = q.peek().second;
        q.remove();


        if (word.equals(endWord)) {
            return step;
        }


        for (int i = 0; i < word.length(); i++) {
            for (char ch = 'a'; ch <= 'z'; ch++) {
                char[] replaceChar = word.toCharArray();
                replaceChar[i] = ch;
                String replaceWord = new String(replaceChar);


                if (st.contains(replaceWord)) {
                    st.remove(replaceWord);
                    q.add(new Pair(replaceWord, step + 1));
                }
            }
        }
    }
    return 0;
}
```

```
    public static void main(String[] args) {
        wordladder1 solution = new wordladder1();
        String beginWord = "hit";
        String endWord = "cog";
        List<String> wordList = Arrays.asList("hot", "dot", "dog", "lot", "log",
"cog");

        int result = solution.ladderLength(beginWord, endWord, wordList);
        System.out.println("The length of the shortest transformation sequence: "
+ result);
    }
}
```

Output:

TC:O(n*n)



8)Word ladder 11

Code:

```
import java.util.*;
```

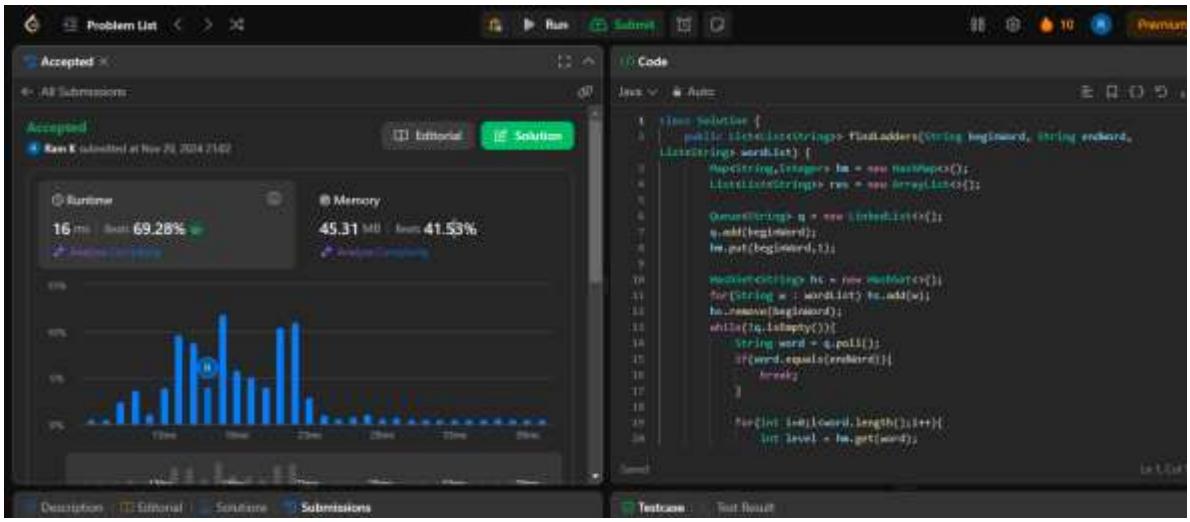
```
class Solution {
    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {
        Set<String> st = new HashSet<>(wordList);
        Queue<ArrayList<String>> q = new LinkedList<>();
        ArrayList<String> initialPath = new ArrayList<>();
        initialPath.add(beginWord);
        q.add(initialPath);

        ArrayList<String> useOnLevel = new ArrayList<>();
        useOnLevel.add(beginWord);

        int level = 0;
        List<List<String>> ans = new ArrayList<>();

        while (!q.isEmpty()) {
            ArrayList<String> currentPath = q.peek();
            q.remove();

            if (currentPath.size() > level) {
                level++;
                for (String usedWord : useOnLevel) {
                    st.remove(usedWord);
                }
                useOnLevel.clear();
            }
```

```
String word = currentPath.get(currentPath.size() - 1);

if (word.equals(endWord)) {
    if (ans.isEmpty()) {
        ans.add(currentPath);
    } else if (ans.get(0).size() == currentPath.size()) {
        ans.add(currentPath);
    }
}

for (int i = 0; i < word.length(); i++) {
    for (char ch = 'a'; ch <= 'z'; ch++) {
        char[] replaceChar = word.toCharArray();
        replaceChar[i] = ch;
        String replaceWord = new String(replaceChar);

        if (st.contains(replaceWord)) {
            currentPath.add(replaceWord);
            ArrayList<String> tempPath = new ArrayList<>(currentPath);
            q.add(tempPath);
            useOnLevel.add(replaceWord);
            currentPath.remove(currentPath.size() - 1);
        }
    }
}
```

```
        }
        return ans;
    }
}
```

Output:

TC:O(n*m)



9)Minimum Path sum problem

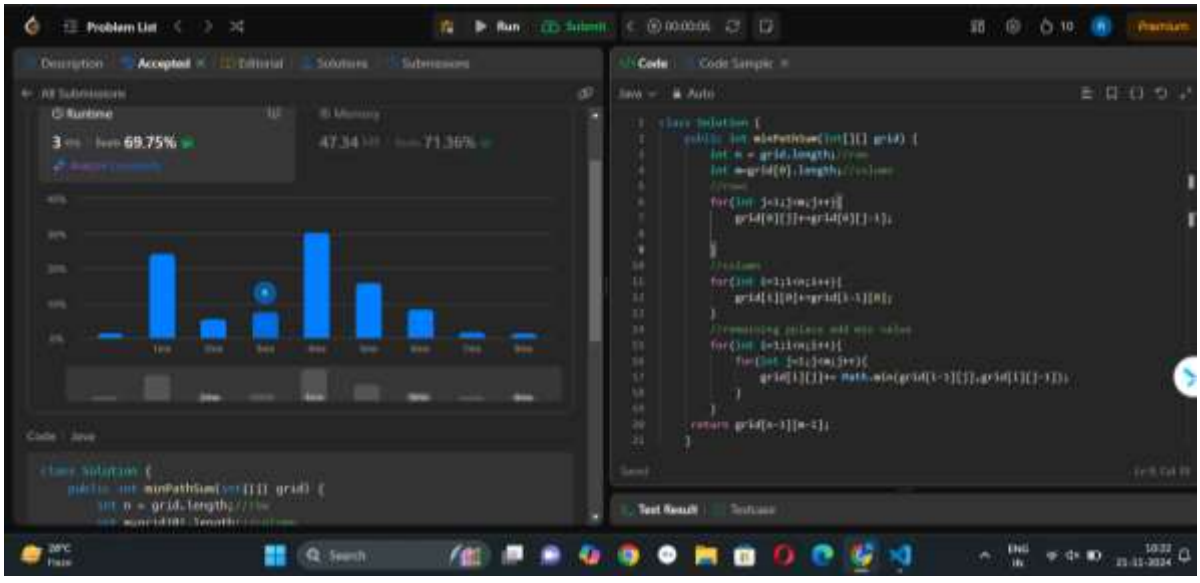
Code :

```
class Solution {
    public int minPathSum(int[][] grid) {
        int n = grid.length;//row
        int m=grid[0].length;//column
        //rows
        for(int j=1;j<m;j++){
            grid[0][j]+=grid[0][j-1];
```

```
        }
        //column
        for(int i=1;i<n;i++){
            grid[i][0]+=grid[i-1][0];
        }
        //remaining pplace add min value
        for(int i=1;i<n;i++){
            for(int j=1;j<m;j++){
                grid[i][j]+= Math.min(grid[i-1][j],grid[i][j-1]);
            }
        }
    return grid[n-1][m-1];
   }
}
```

Output:

TC: O(n)



10: