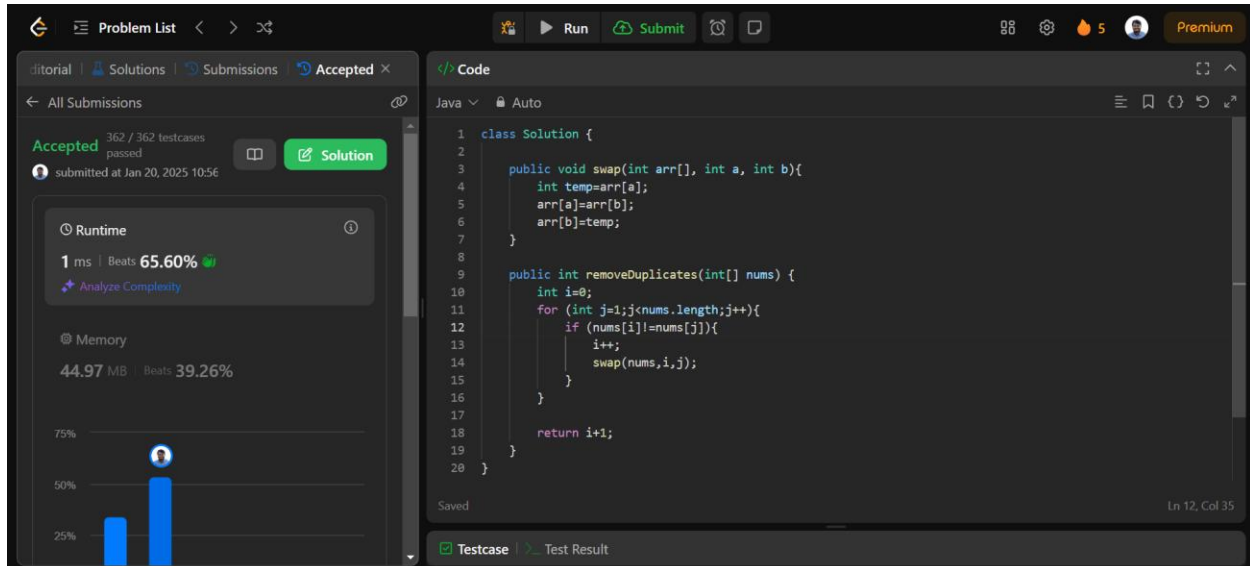


26. Remove Duplicates from Sorted Array

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in `nums`*.

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with tabs for 'Problem List', 'Solutions', 'Submissions', and 'Accepted'. The 'Accepted' tab is active, showing a submission status of 'Accepted' with 362/362 testcases passed, submitted on Jan 20, 2025 at 10:56. Below this, there's a 'Runtime' section showing '1 ms' and 'Beats 65.60%', and a 'Memory' section showing '44.97 MB' and 'Beats 39.26%'. A bar chart at the bottom of the sidebar shows performance comparison. The main editor area displays a Java code file named 'Solution.java'. The code defines a 'Solution' class with two methods: 'swap' and 'removeDuplicates'. The 'removeDuplicates' method uses a two-pointer approach to shift unique elements to the front of the array and return the count of unique elements. The code is as follows:

```
1 class Solution {
2
3     public void swap(int arr[], int a, int b){
4         int temp=arr[a];
5         arr[a]=arr[b];
6         arr[b]=temp;
7     }
8
9     public int removeDuplicates(int[] nums) {
10        int i=0;
11        for (int j=1;j<nums.length;j++){
12            if (nums[i]!=nums[j]){
13                i++;
14                swap(nums,i,j);
15            }
16        }
17
18        return i+1;
19    }
20 }
```

```
class Solution {

    public void swap(int arr[], int a, int b){

        int temp=arr[a];

        arr[a]=arr[b];

        arr[b]=temp;

    }

    public int removeDuplicates(int[] nums) {

        int i=0;

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

            if (nums[i]!=nums[j]){

                i++;

                swap(nums,i,j);

            }

        }

        return i+1;}}
```

Insertion Sort

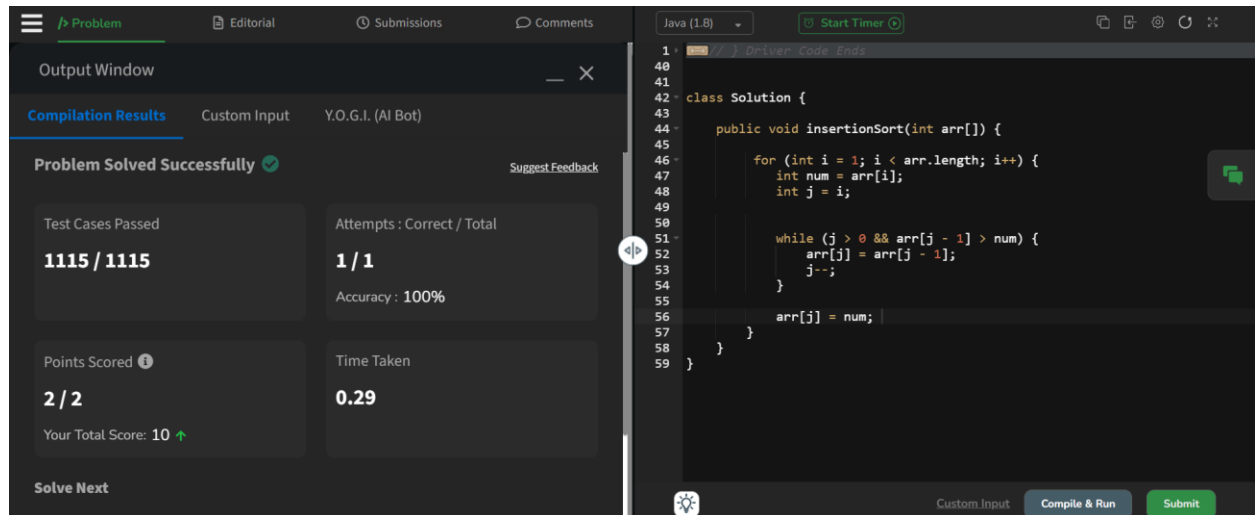
The task is to complete the **insertsort()** function which is used to implement Insertion Sort.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Explanation: The sorted array will be [1, 3, 4, 7, 9].

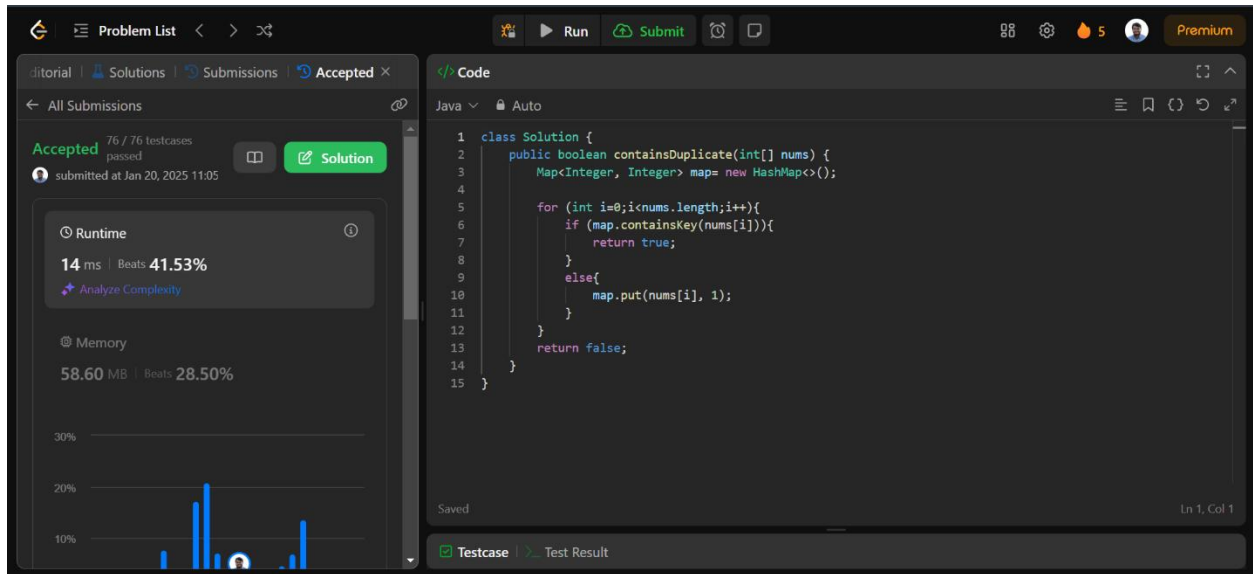


```
1
40
41
42 class Solution {
43
44     public void insertionSort(int arr[]) {
45
46         for (int i = 1; i < arr.length; i++) {
47             int num = arr[i];
48             int j = i;
49
50             while (j > 0 && arr[j - 1] > num) {
51                 arr[j] = arr[j - 1];
52                 j--;
53             }
54
55             arr[j] = num;
56         }
57     }
58 }
59
```

```
class Solution {
    public void insertionSort(int arr[]) {
        for (int i = 1; i < arr.length; i++) {
            int num = arr[i];
            int j = i;
            while (j > 0 && arr[j - 1] > num) {
                arr[j] = arr[j - 1];
                j--;
            }
            arr[j] = num;
        }
    }
}
```

217. Contains Duplicate

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.



The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with tabs for 'editorial', 'Solutions', 'Submissions', and 'Accepted'. The 'Accepted' tab is active, showing a submission status of 'Accepted' for 76/76 testcases, passed, submitted at Jan 20, 2025 11:05. Below this, there's a 'Runtime' section showing 14 ms and 41.53% beats, and a 'Memory' section showing 58.60 MB and 28.50% beats. A bar chart is visible at the bottom of the sidebar. The main editor area shows a Java code file named 'Auto'. The code is as follows:

```
1 class Solution {
2     public boolean containsDuplicate(int[] nums) {
3         Map<Integer, Integer> map= new HashMap<>();
4
5         for (int i=0;i<nums.length;i++){
6             if (map.containsKey(nums[i])){
7                 return true;
8             }
9             else{
10                map.put(nums[i], 1);
11            }
12        }
13        return false;
14    }
15 }
```

```
class Solution {
    public boolean containsDuplicate(int[] nums) {
        Map<Integer, Integer> map= new HashMap<>();

        for (int i=0;i<nums.length;i++){
            if (map.containsKey(nums[i])){
                return true;
            }
            else{
                map.put(nums[i], 1);
            }
        }
        return false;
    }
}
```

1. Two Sum

Given an array of integers *nums* and an integer *target*, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

You can return the answer in any order.

```
1 class Solution {
2     public int[] twoSum(int[] nums, int target) {
3
4         Map <Integer, Integer> map= new HashMap<Integer,Integer>();
5         int arr[]= new int [2];
6
7
8         for (int i=0;i<nums.length;i++){
9             if (map.containsKey(target-nums[i])){
10                 arr[0]= i;
11                 arr[1]=map.get(target-nums[i]);
12                 break;
13             }
14
15             map.put(nums[i],i);
16
17         }
18         return arr;
19     }
20 }
```

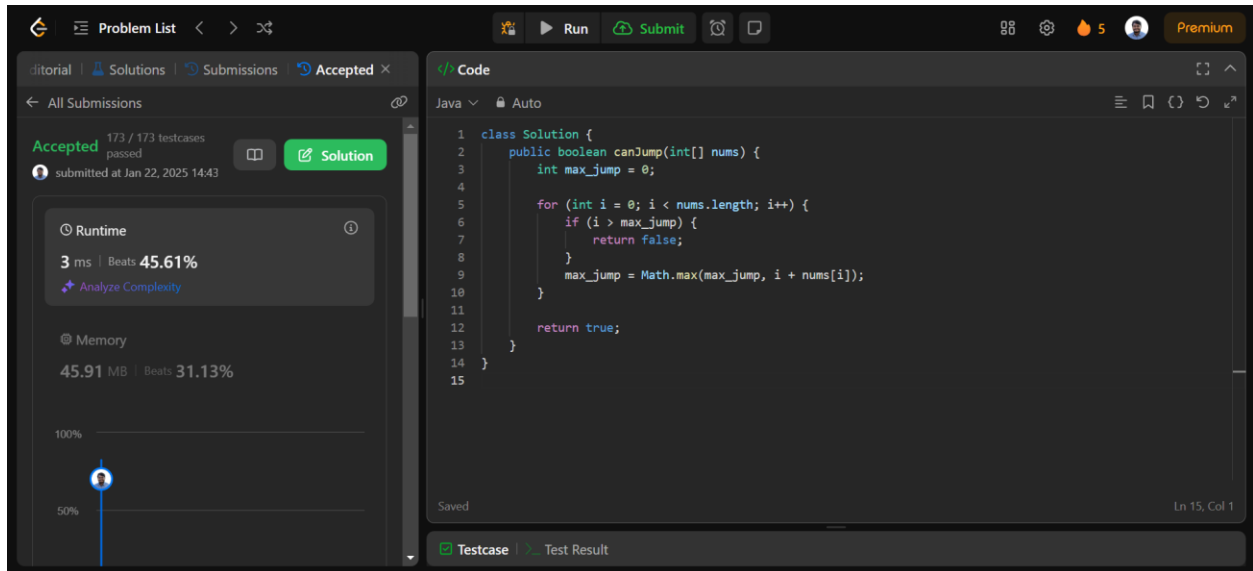
```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map <Integer, Integer> map= new HashMap<Integer,Integer>();
        int arr[]= new int [2];
        for (int i=0;i<nums.length;i++){
            if (map.containsKey(target-nums[i])){
                arr[0]= i;
                arr[1]=map.get(target-nums[i]);
                break;
            }
            map.put(nums[i],i);
        }
        return arr; }}

```

55. Jump Game

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` if you can reach the last index, or `false` otherwise.

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with tabs for 'Problem List', 'Solutions', 'Submissions', and 'Accepted'. The 'Accepted' tab is active, showing a submission status of 'Accepted' with '173 / 173 testcases passed' and a submission time of 'submitted at Jan 22, 2025 14:43'. Below this, there's a 'Runtime' section showing '3 ms | Beats 45.61%' and a 'Memory' section showing '45.91 MB | Beats 31.13%'. A progress bar is visible at the bottom of the sidebar. The main editor area shows a Java code file named 'Solution.java'. The code is as follows:

```
1 class Solution {
2     public boolean canJump(int[] nums) {
3         int max_jump = 0;
4
5         for (int i = 0; i < nums.length; i++) {
6             if (i > max_jump) {
7                 return false;
8             }
9             max_jump = Math.max(max_jump, i + nums[i]);
10        }
11
12        return true;
13    }
14 }
15
```

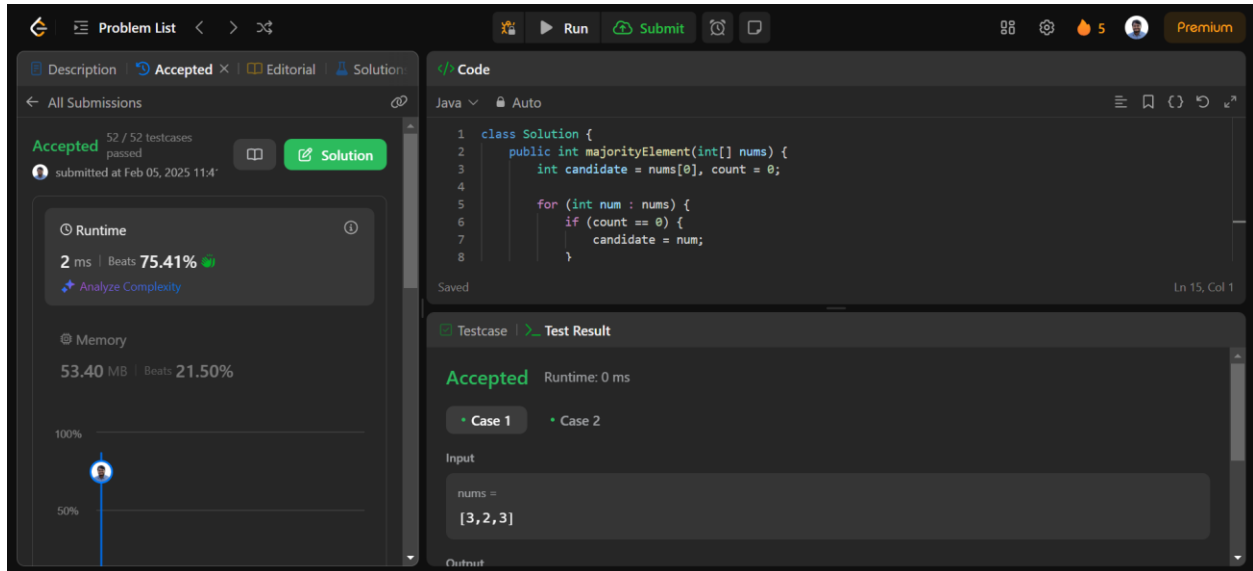
The status bar at the bottom indicates 'Saved' and 'Ln 15, Col 1'.

```
class Solution {  
    public boolean canJump(int[] nums) {  
        int max_jump = 0;  
  
        for (int i = 0; i < nums.length; i++) {  
            if (i > max_jump) {  
                return false;  
            }  
            max_jump = Math.max(max_jump, i + nums[i]);  
        }  
  
        return true;  
    }  
}
```

169. Majority Element

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

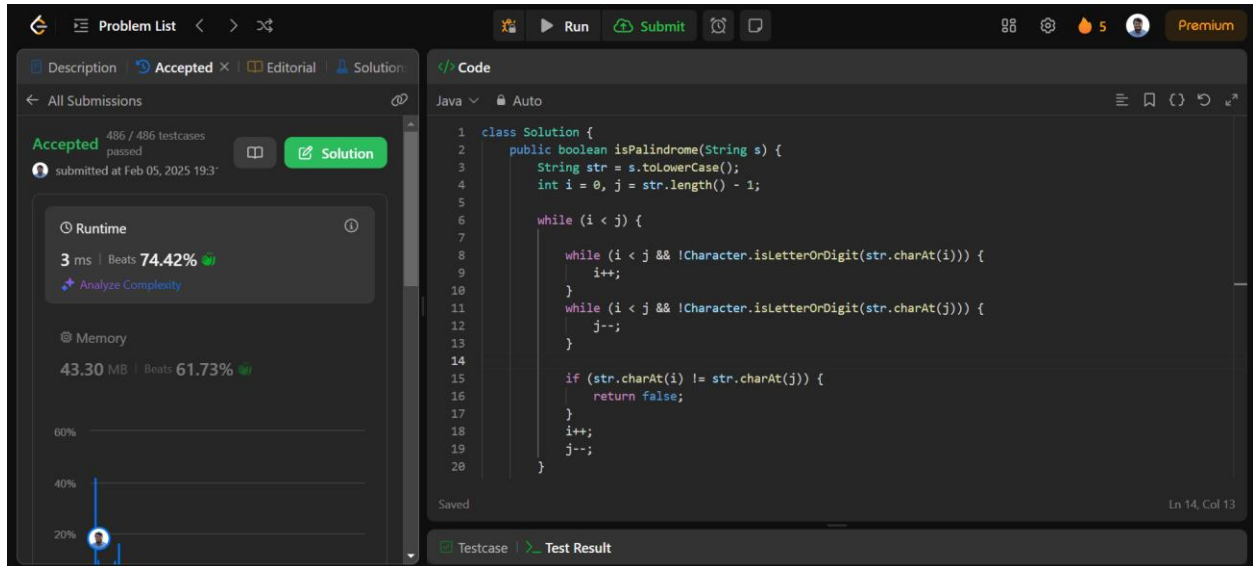


```
class Solution {  
    public int majorityElement(int[] nums) {  
        int candidate = nums[0], count = 0;  
  
        for (int num : nums) {  
            if (count == 0) {  
                candidate = num;  
            }  
            count += (num == candidate) ? 1 : -1;  
        }  
        return candidate;  
    }  
}
```

125. Valid Palindrome

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string *s*, return true *if it is a palindrome*, or false *otherwise*.



```
1 class Solution {
2     public boolean isPalindrome(String s) {
3         String str = s.toLowerCase();
4         int i = 0, j = str.length() - 1;
5
6         while (i < j) {
7             while (i < j && !Character.isLetterOrDigit(str.charAt(i))) {
8                 i++;
9             }
10            while (i < j && !Character.isLetterOrDigit(str.charAt(j))) {
11                j--;
12            }
13
14            if (str.charAt(i) != str.charAt(j)) {
15                return false;
16            }
17            i++;
18            j--;
19        }
20    }
21 }
```

```
class Solution {
    public boolean isPalindrome(String s) {
        String str = s.toLowerCase();
        int i = 0, j = str.length() - 1;
        while (i < j) {
            while (i < j && !Character.isLetterOrDigit(str.charAt(i))) i++;
            while (i < j && !Character.isLetterOrDigit(str.charAt(j))) j--;
            if (str.charAt(i) != str.charAt(j)) return false;
            i++; j--;
        }
        return true;
    }
}
```

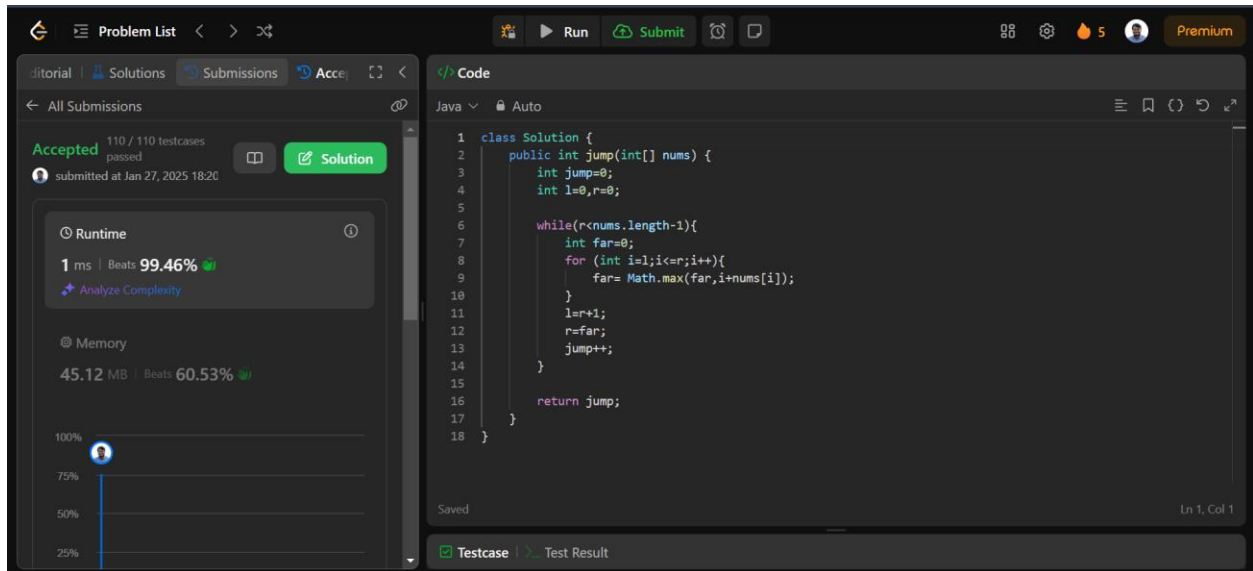
45. Jump Game II

You are given a **0-indexed** array of integers `nums` of length `n`. You are initially positioned at `nums[0]`.

Each element `nums[i]` represents the maximum length of a forward jump from index `i`. In other words, if you are at `nums[i]`, you can jump to any `nums[i + j]` where:

- $0 \leq j \leq \text{nums}[i]$ and
- $i + j < n$

Return *the minimum number of jumps to reach* `nums[n - 1]`. The test cases are generated such that you can reach `nums[n - 1]`.



```
1 class Solution {
2     public int jump(int[] nums) {
3         int jump=0;
4         int l=0,r=0;
5
6         while(r<nums.length-1){
7             int far=0;
8             for (int i=l;i<=r;i++){
9                 far= Math.max(far,i+nums[i]);
10            }
11            l=r+1;
12            r=far;
13            jump++;
14        }
15
16        return jump;
17    }
18 }
```

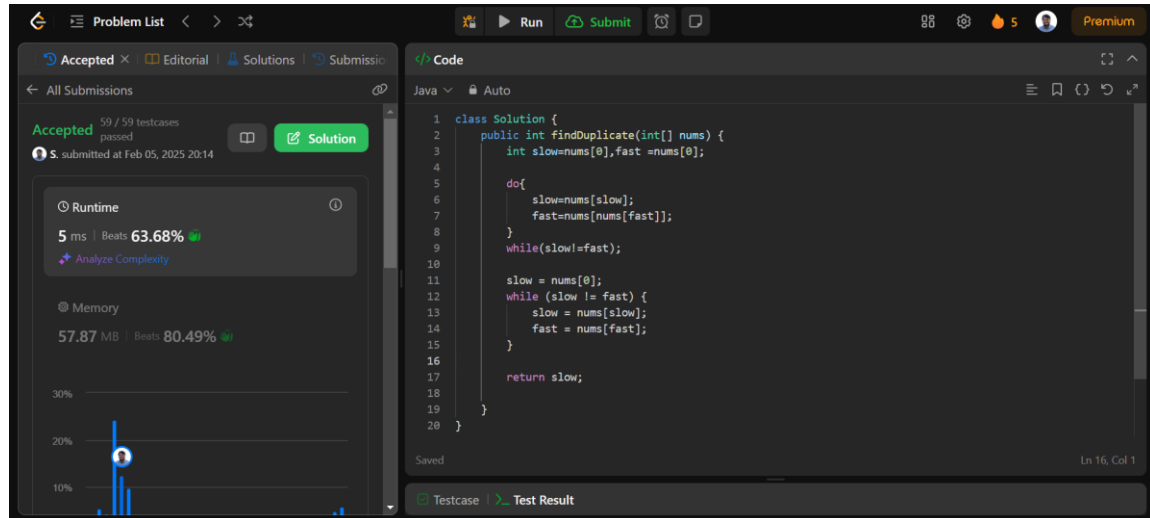
```
class Solution {
    public int jump(int[] nums) {
        int jump=0;
        int l=0,r=0;
        while(r<nums.length-1){
            int far=0;
            for (int i=l;i<=r;i++){
                far= Math.max(far, i+nums[i]);
            }
            l=r+1;
            r=far;
            jump++;
        }
        return jump;
    }
}
```


287. Find the Duplicate Number

Given an array of integers `nums` containing $n + 1$ integers where each integer is in the range $[1, n]$ inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.



```
1 class Solution {
2     public int findDuplicate(int[] nums) {
3         int slow=nums[0], fast =nums[0];
4
5         do{
6             slow=nums[slow];
7             fast=nums[nums[fast]];
8         }
9         while(slow!=fast);
10
11        slow = nums[0];
12        while (slow != fast) {
13            slow = nums[slow];
14            fast = nums[fast];
15        }
16
17        return slow;
18    }
19 }
20 }
```

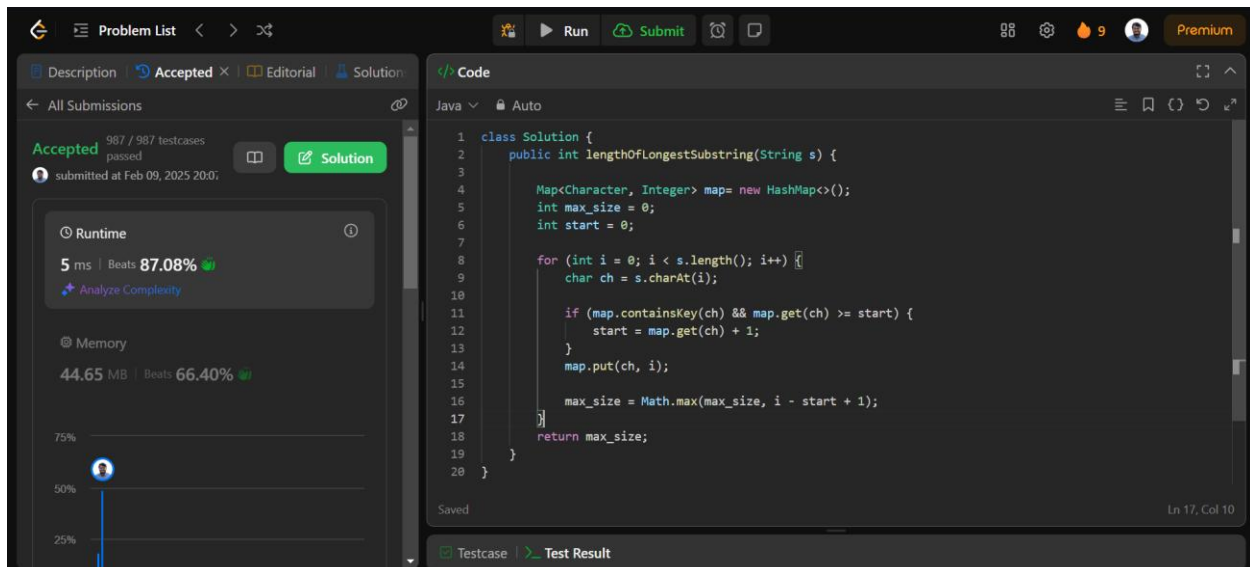
```
class Solution {
    public int findDuplicate(int[] nums) {
        int slow=nums[0],fast =nums[0];

        do{
            slow=nums[slow];
            fast=nums[nums[fast]];
        }
        while(slow!=fast);

        slow = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
}
```

3. Longest Substring Without Repeating Characters

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



The screenshot shows the LeetCode submission page for the problem 'Longest Substring Without Repeating Characters'. The submission is marked as 'Accepted' with 987/987 testcases passed, submitted on Feb 09, 2025 at 20:07. The runtime is 5 ms, beating 87.08% of submissions, and the memory usage is 44.65 MB, beating 66.40%. The Java code is displayed in the editor, showing a class 'Solution' with a method 'lengthOfLongestSubstring' that uses a HashMap to track characters and their indices to find the longest substring without repeating characters.

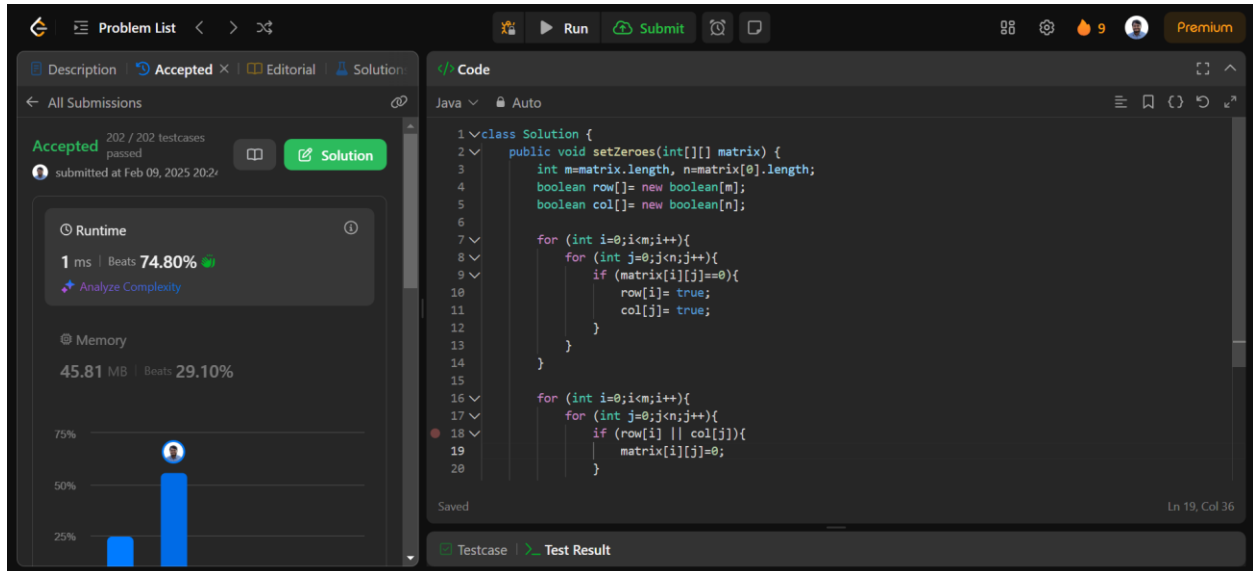
```
1 class Solution {
2     public int lengthOfLongestSubstring(String s) {
3
4         Map<Character, Integer> map= new HashMap<>();
5         int max_size = 0;
6         int start = 0;
7
8         for (int i = 0; i < s.length(); i++) {
9             char ch = s.charAt(i);
10
11             if (map.containsKey(ch) && map.get(ch) >= start) {
12                 start = map.get(ch) + 1;
13             }
14             map.put(ch, i);
15
16             max_size = Math.max(max_size, i - start + 1);
17         }
18         return max_size;
19     }
20 }
```

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Map<Character, Integer> map= new HashMap<>();
        int max_size = 0;
        int start = 0;
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (map.containsKey(ch) && map.get(ch) >= start) {
                start = map.get(ch) + 1;
            }
            map.put(ch, i);
            max_size = Math.max(max_size, i - start + 1);
        }
        return max_size;
    }
}
```

73. Set Matrix Zeroes

Given an $m \times n$ integer matrix `matrix`, if an element is 0, set its entire row and column to 0's.

You must do it [in place](#).



The screenshot shows a LeetCode interface with the problem 'Set Matrix Zeroes' solved in Java. The left sidebar shows the solution is 'Accepted' with a runtime of 1 ms and memory of 45.81 MB. The main editor displays the following Java code:

```
1 class Solution {
2     public void setZeroes(int[][] matrix) {
3         int m=matrix.length, n=matrix[0].length;
4         boolean row[] = new boolean[m];
5         boolean col[] = new boolean[n];
6
7         for (int i=0;i<m;i++){
8             for (int j=0;j<n;j++){
9                 if (matrix[i][j]==0){
10                    row[i]= true;
11                    col[j]= true;
12                }
13            }
14        }
15
16        for (int i=0;i<m;i++){
17            for (int j=0;j<n;j++){
18                if (row[i] || col[j]){
19                    matrix[i][j]=0;
20                }
21            }
22        }
23    }
24 }
```

```
class Solution {
    public void setZeroes(int[][] matrix) {
        int m=matrix.length, n=matrix[0].length;
        boolean row[] = new boolean[m];
        boolean col[] = new boolean[n];

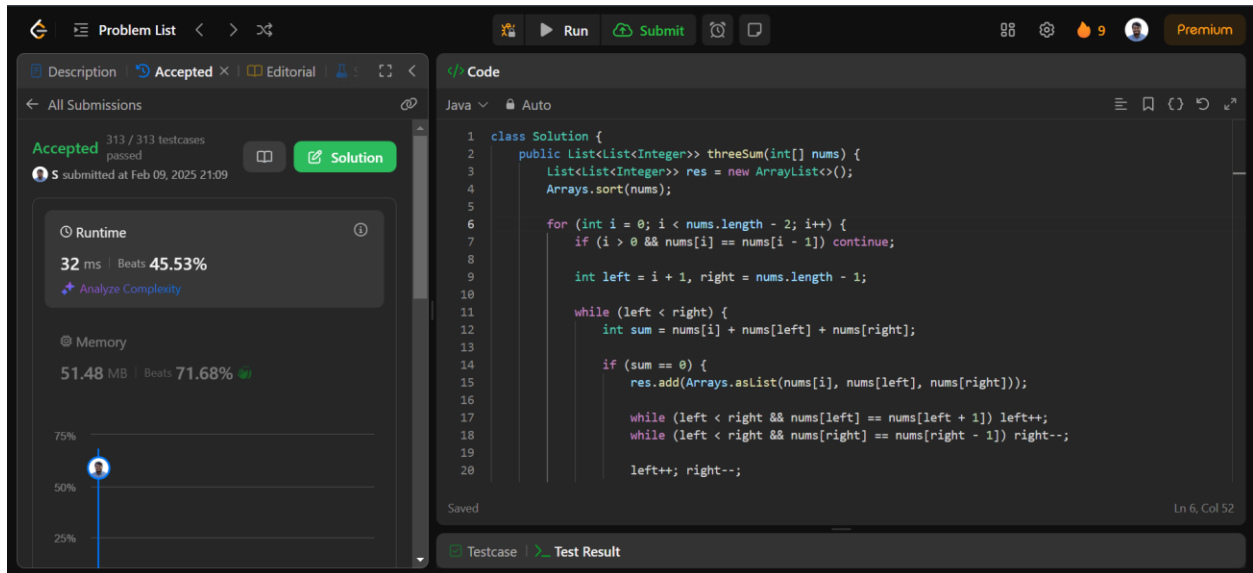
        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (matrix[i][j]==0) { row[i]= true; col[j]= true; }
            }
        }

        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (row[i] || col[j]) matrix[i][j]=0;
            }
        }
    }
}
```

15. 3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.



```
1 class Solution {
2     public List<List<Integer>> threeSum(int[] nums) {
3         List<List<Integer>> res = new ArrayList<>();
4         Arrays.sort(nums);
5
6         for (int i = 0; i < nums.length - 2; i++) {
7             if (i > 0 && nums[i] == nums[i - 1]) continue;
8
9             int left = i + 1, right = nums.length - 1;
10
11             while (left < right) {
12                 int sum = nums[i] + nums[left] + nums[right];
13
14                 if (sum == 0) {
15                     res.add(Arrays.asList(nums[i], nums[left], nums[right]));
16
17                     while (left < right && nums[left] == nums[left + 1]) left++;
18                     while (left < right && nums[right] == nums[right - 1]) right--;
19
20                     left++; right--;
21                 }
22             }
23         }
24         return res;
25     }
26 }
```

```
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 - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    res.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right--;
                    left++; right--;
                }
            }
        }
    }
}
```

```
        else if (sum < 0) left++;  
        else right--;  
    }  
}  
return res;  
}  
}
```