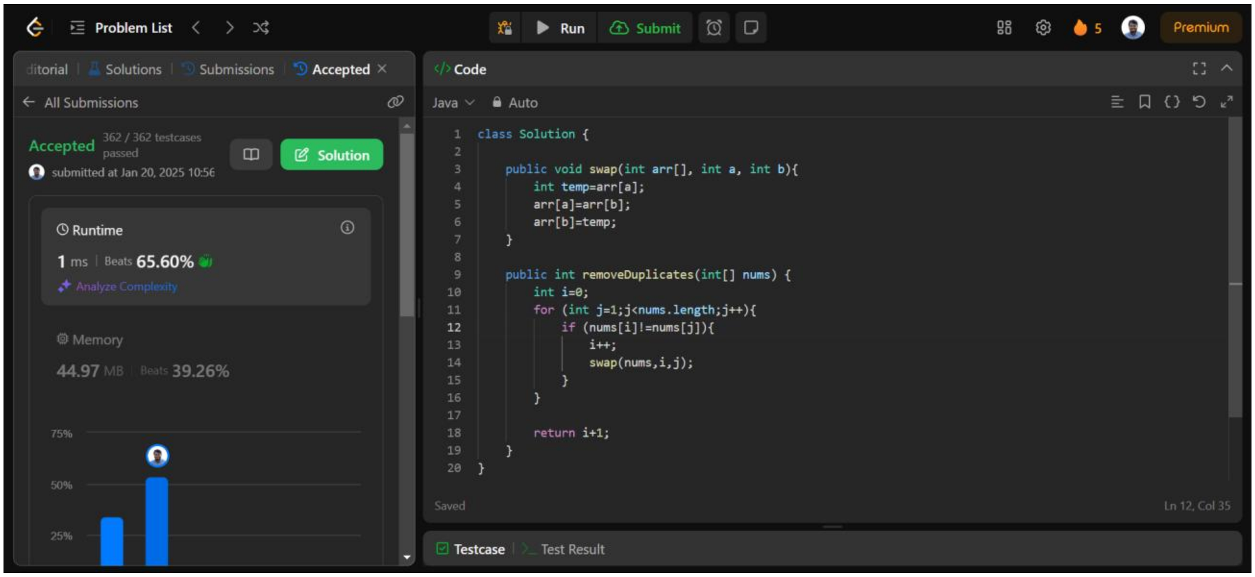
SNEHASIS MEDDA 22BCS16163 DAY 1

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.



Accepted 362 / 362 testcases passed

submitted at Jan 20, 2025 10:56

Runtime

1 ms | Beats 65.60%

Memory

44.97 MB | Beats 39.26%

```java
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;
    }
}
```

Saved

Ln 12, Col 35

```java
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;}}
```

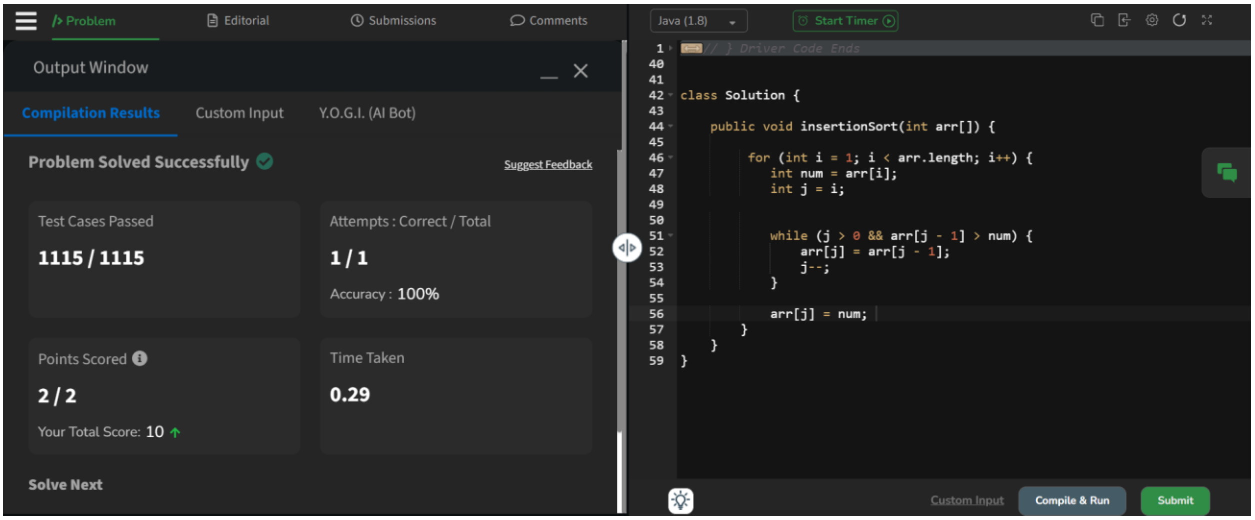
Insertion Sort

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].



Problem Solved Successfully

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
2 / 2
Your Total Score: 10

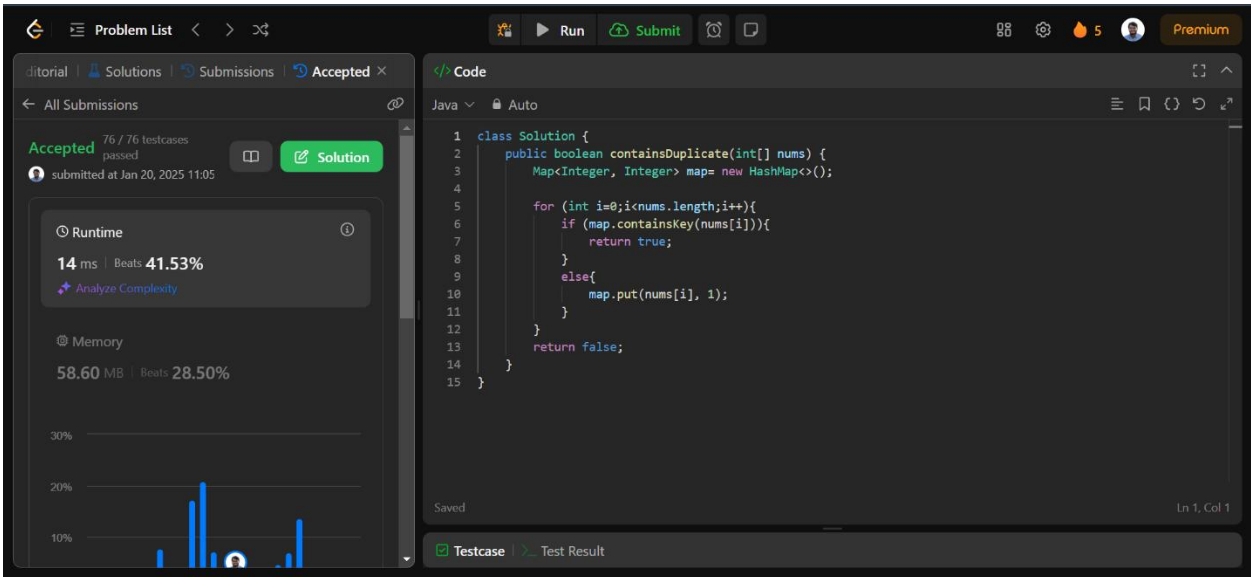
Time Taken
0.29

```
// } Driver Code Ends


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;
        }
    }
}
```

```
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.



Accepted 76 / 76 testcases passed

submitted at Jan 20, 2025 11:05

Runtime

14 ms | Beats 41.53%

Memory

58.60 MB | Beats 28.50%

```java
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;
    }
}
```

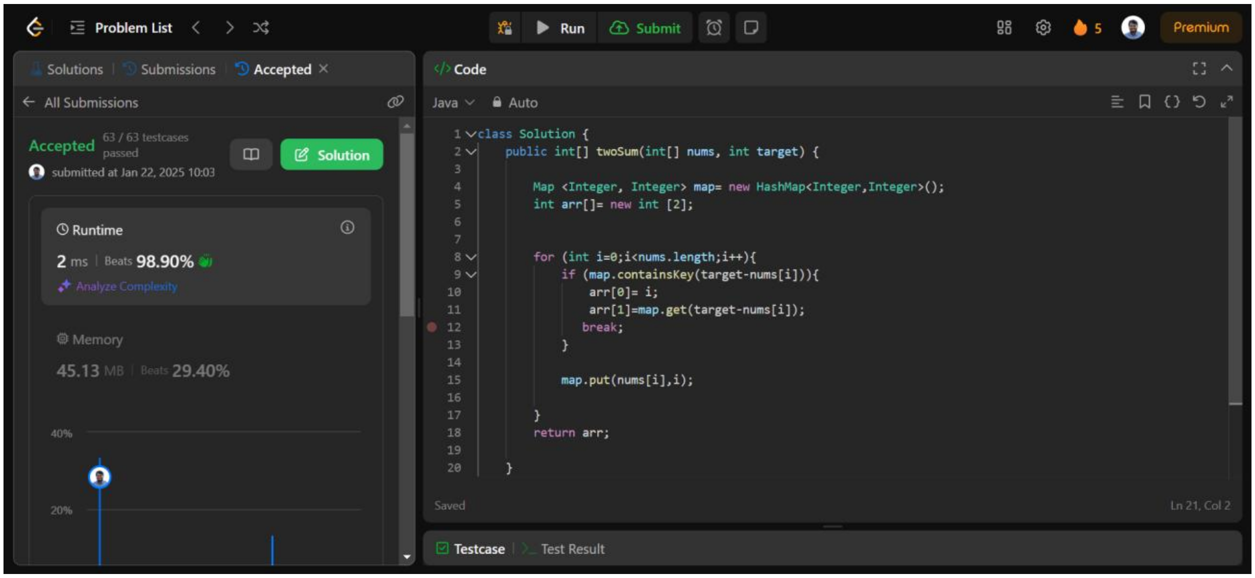
```java
class Solution {
   public boolean containsDuplicate(int[] nums) {
      Map<Integer, Integer> map= new HashMap<>();


      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.



Accepted 63 / 63 testcases passed

submitted at Jan 22, 2025 10:03

Runtime

2 ms | Beats 98.90%

Memory

45.13 MB | Beats 29.40%

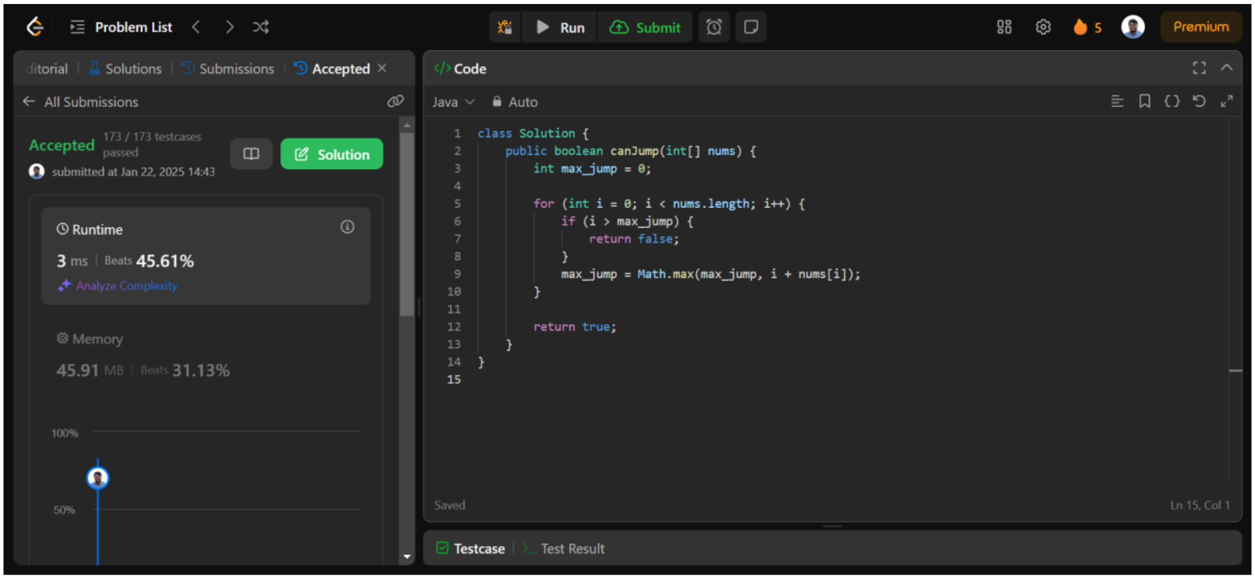
```
class Solution {
    public int[] twoSum(int[] nums, int target) {

        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;

    }
```

```
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*.



Accepted 173 / 173 testcases passed

submitted at Jan 22, 2025 14:43

Runtime

3 ms | Beats 45.61%

Memory

45.91 MB | Beats 31.13%

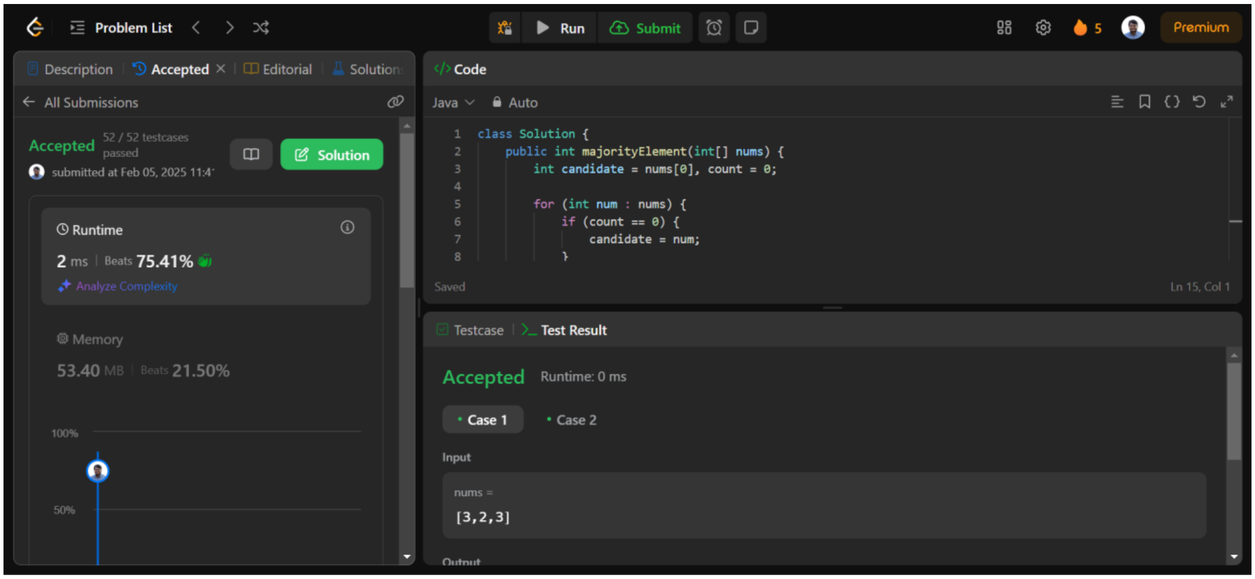
```java
class Solution {
    public boolean canJump(int[] nums) {
        int max_jump = 0;

        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;
    }
}
```

```java
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 ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.



Accepted 52 / 52 testcases passed
submitted at Feb 05, 2025 11:4

2 ms | Beats 75.41%

53.40 MB | Beats 21.50%

```
class Solution {
    public int majorityElement(int[] nums) {
        int candidate = nums[0], count = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
```

Accepted Runtime: 0 ms

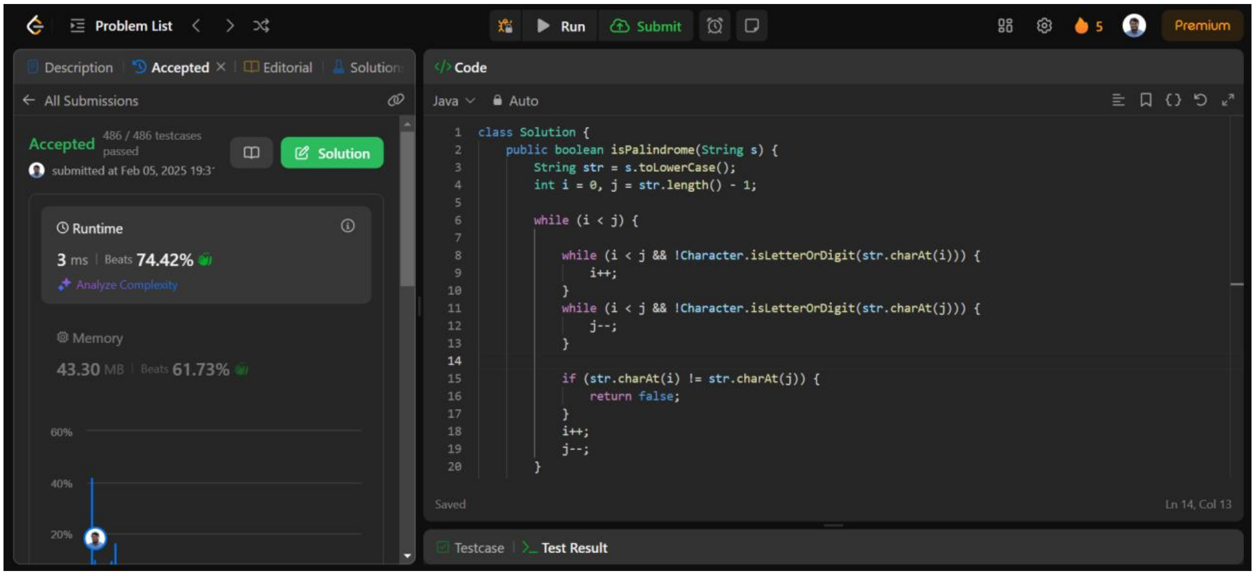
nums =
[3,2,3]

```
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*.



Accepted 486 / 486 testcases passed

submitted at Feb 05, 2025 19:3

Runtime

3 ms | Beats 74.42%

Analyze Complexity

Memory

43.30 MB | Beats 61.73%

```
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--;
        }
```

Saved

Ln 14, Col 13

```
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;
    }
}
```

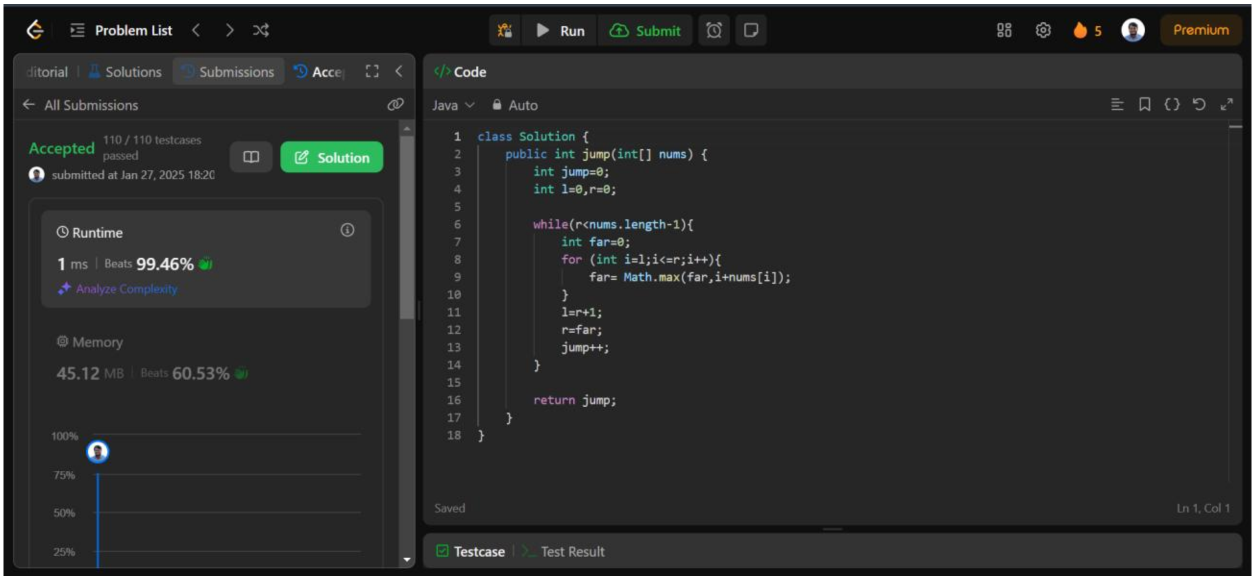
45. Jump Game II

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 <= j <= 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].



Accepted 110 / 110 testcases passed

submitted at Jan 27, 2025 18:20

Runtime

1 ms | Beats 99.46%

Memory

45.12 MB | Beats 60.53%

```
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;
    }
}
```

Saved

Ln 1, Col 1

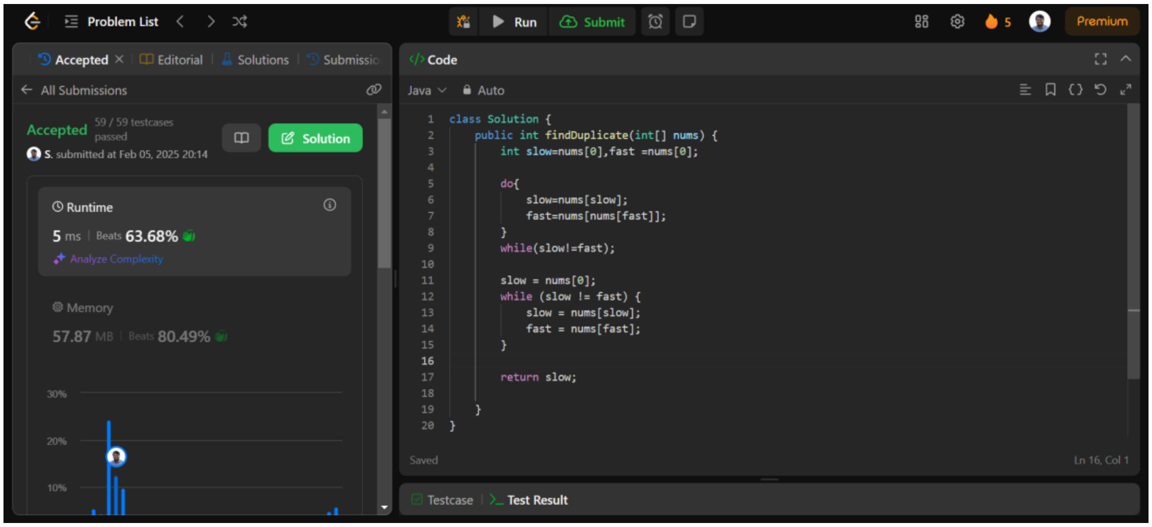
```
class Solution {
  public int jump(int[] nums) {
    int jump=0;
    int l=0,r=0;
    while(r<nums.length-1){
      int far=0;
      for (int i=l;i<=r;i++){
        far= Math.max(far, i+nums[i]);
      }
      l=r+1;
      r=far;
      jump++;
    }
    return jump;
  }
}
```

[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.



Accepted 59 / 59 testcases passed

S. submitted at Feb 05, 2025 20:14

Runtime

5 ms | Beats 63.68%

Analyze Complexity

Memory

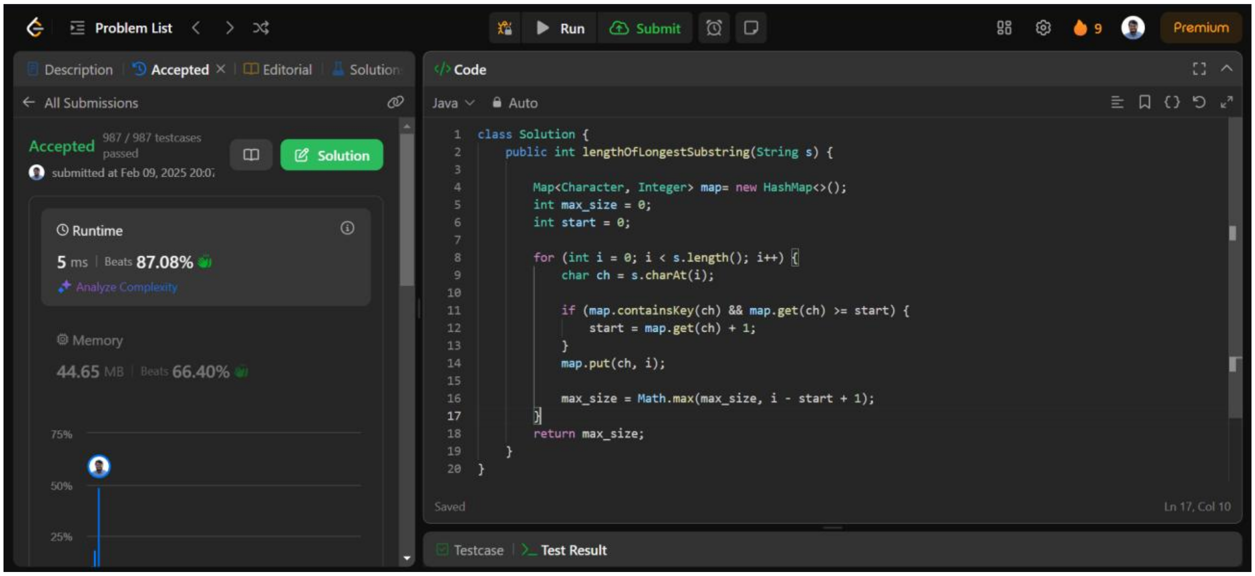
57.87 MB | Beats 80.49%

```
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;

    }
}
```

```
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.



Accepted 987 / 987 testcases passed

submitted at Feb 09, 2025 20:07

Runtime

5 ms | Beats 87.08%

Memory

44.65 MB | Beats 66.40%

```
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;
    }
}
```

Saved

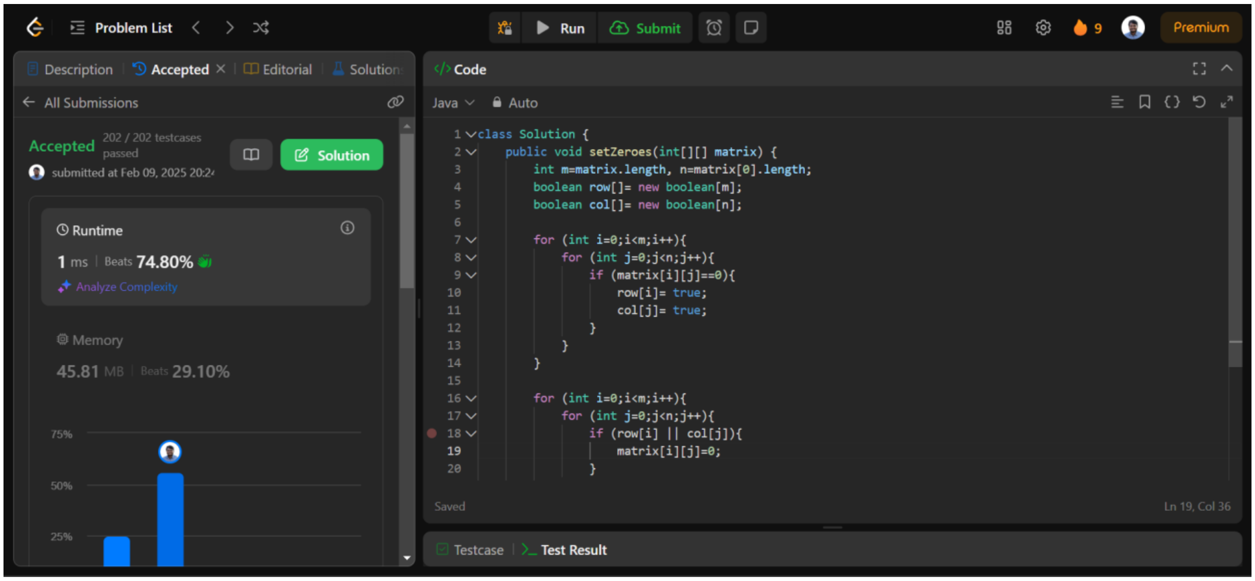
Ln 17, Col 10

```
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 x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

You must do it in place.



Accepted 202 / 202 testcases passed

submitted at Feb 09, 2025 20:2

Runtime

1 ms | Beats 74.80%

Analyze Complexity

Memory

45.81 MB | Beats 29.10%

```
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;
                }
```

Saved

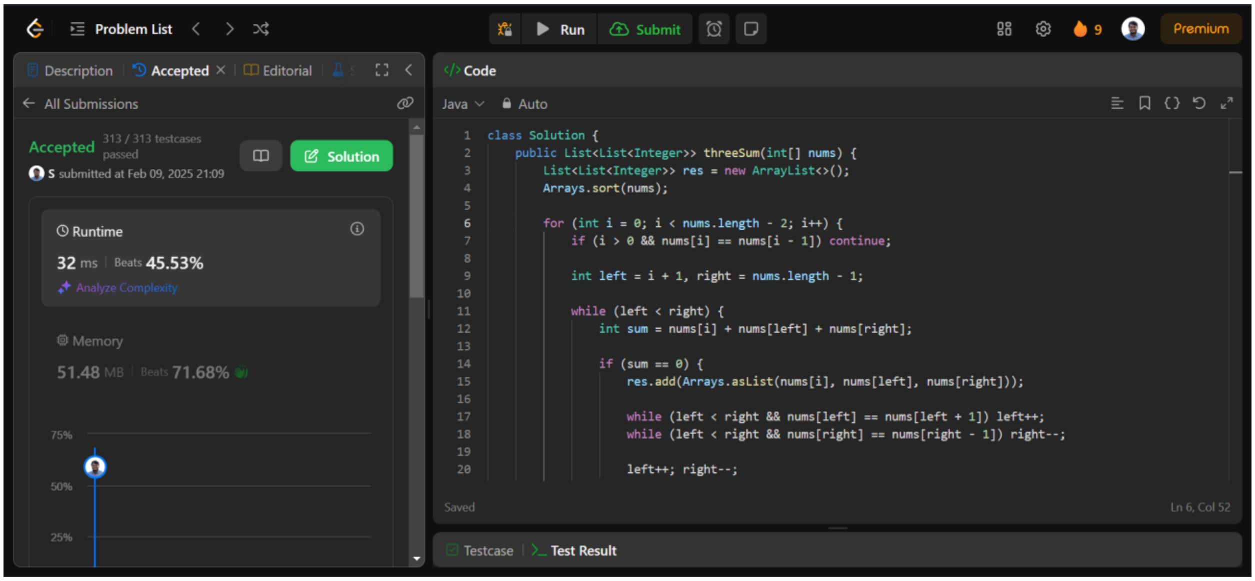
Ln 19, Col 36

```
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.



Accepted 313 / 313 testcases passed

S submitted at Feb 09, 2025 21:09

Runtime

32 ms Beats 45.53%

Memory

51.48 MB Beats 71.68%

```
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--;
```

Saved

Ln 6, Col 52

```
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;
    }
}
```