ASSIGNMENT-1(AP)

Name: Saharsh Kumar

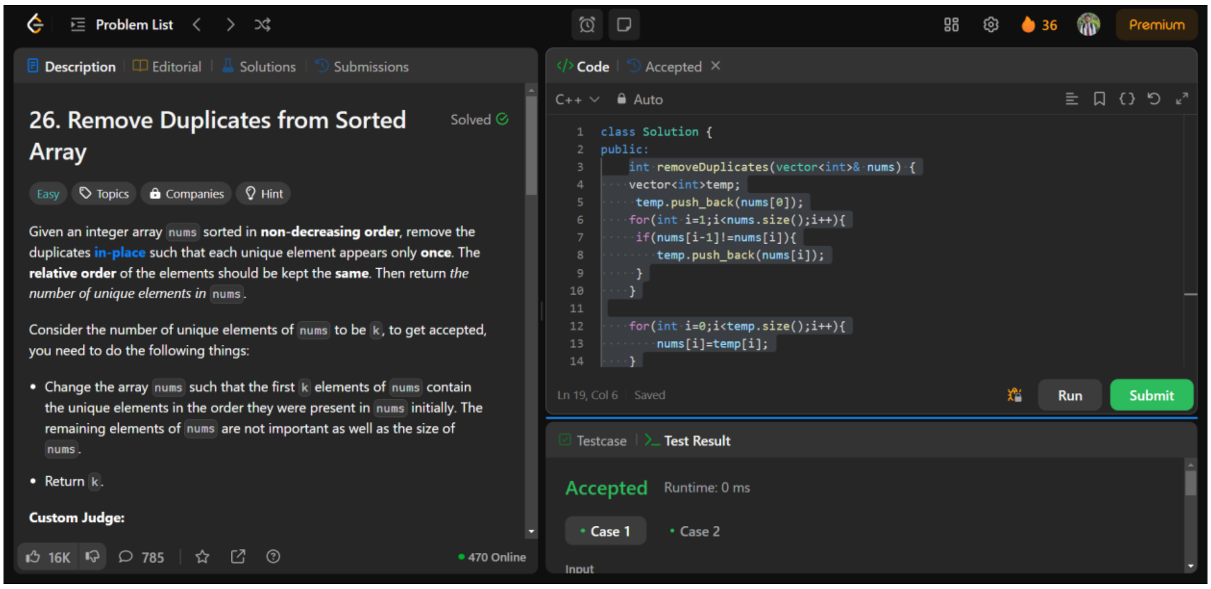
UID: 22BCS14059

Section: 22BCS_FL_IOT-604

Group: A

1) Remove duplicates from a sorted array

```
int removeDuplicates(vector<int>& nums) {
    vector<int>temp;
     temp.push_back(nums[0]);
    for(int i=1;i<nums.size();i++){
     if(nums[i-1]!=nums[i]){
        temp.push_back(nums[i]);
     }
    }

    for(int i=0;i<temp.size();i++){
        nums[i]=temp[i];
    }
    for(int i=temp.size();i<nums.size();i++){
        nums[i]=0;
    }
    return temp.size();
    }
```



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`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
    vector<int>temp;
     temp.push_back(nums[0]);
    for(int i=1;i<nums.size();i++){
     if(nums[i-1]!=nums[i]){
        temp.push_back(nums[i]);
     }
    }

    for(int i=0;i<temp.size();i++){
        nums[i]=temp[i];
    }
```

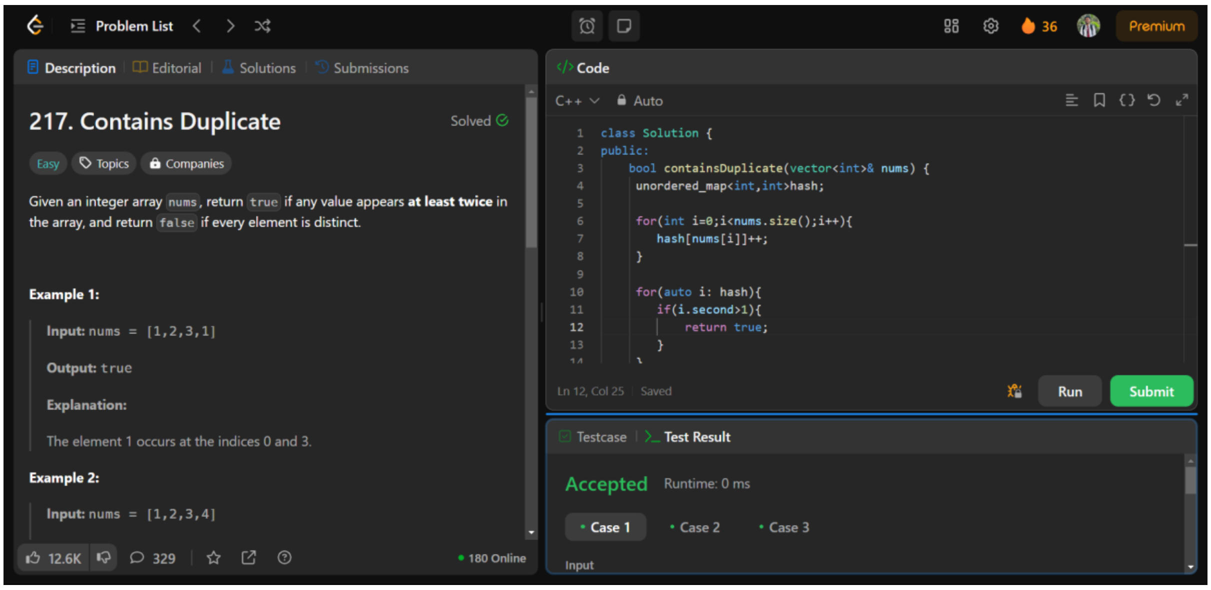
Accepted Runtime: 0 ms

Case 1 Case 2

Input

2) Contains duplicate

```
bool containsDuplicate(vector<int>& nums) {
    unordered_map<int,int>hash;
    for(int i=0;i<nums.size();i++){
      hash[nums[i]]++;
    }
    for(auto i: hash){
      if(i.second>1){
          return true;
      }
    }
    return false;
  }
```



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.

Example 1:

Input: `nums = [1,2,3,1]`

Output: `true`

Explanation:

The element 1 occurs at the indices 0 and 3.

Example 2:

Input: `nums = [1,2,3,4]`

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
     unordered_map<int,int>hash;

     for(int i=0;i<nums.size();i++){
        hash[nums[i]]++;
     }

     for(auto i: hash){
        if(i.second>1){
            return true;
        }
```

Ln 12, Col 25 Saved

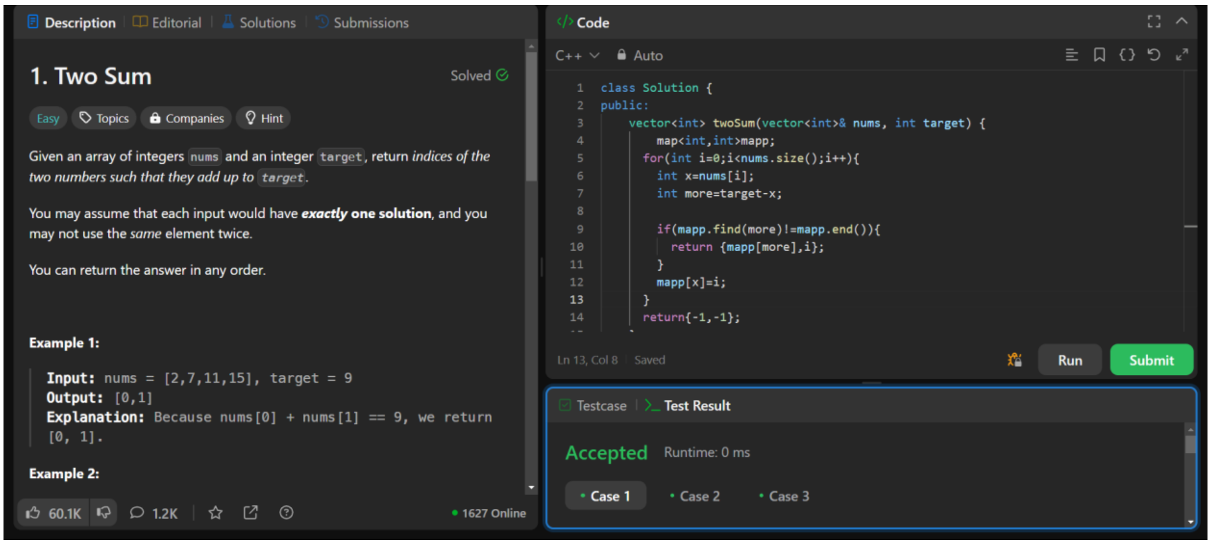
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

3)Two Sum

```
vector<int> twoSum(vector<int>& nums, int target) {
      map<int,int>mapp;
    for(int i=0;i<nums.size();i++){
      int x=nums[i];
      int more=target-x;
      if(mapp.find(more)!=mapp.end()){
        return {mapp[more],i};
      }
      mapp[x]=i;
    }
    return{-1,-1};
  }
```



1. Two Sum

Solved

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.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return
[0, 1].
```

Example 2:

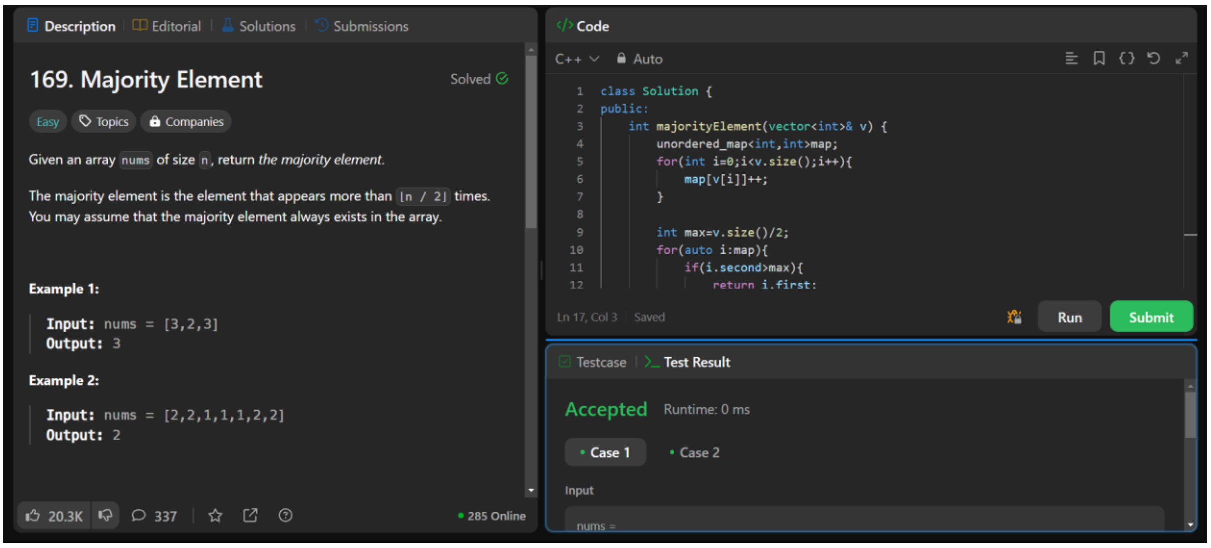
```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        map<int,int>mapp;
      for(int i=0;i<nums.size();i++){
        int x=nums[i];
        int more=target-x;

        if(mapp.find(more)!=mapp.end()){
          return {mapp[more],i};
        }
        mapp[x]=i;
      }
      return{-1,-1};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

4) Majority Element

```
int majorityElement(vector<int>& v) {
        unordered_map<int,int>map;
        for(int i=0;i<v.size();i++){
            map[v[i]]++;
        }
        int max=v.size()/2;
        for(auto i:map){
            if(i.second>max){
                return i.first;
            }
        }
        return -1;
    }
```



169. Majority Element

Solved

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.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

```
class Solution {
public:
    int majorityElement(vector<int>& v) {
        unordered_map<int,int>map;
        for(int i=0;i<v.size();i++){
            map[v[i]]++;
        }

        int max=v.size()/2;
        for(auto i:map){
            if(i.second>max){
                return i.first:
```

Ln 17, Col 3 Saved

Accepted Runtime: 0 ms

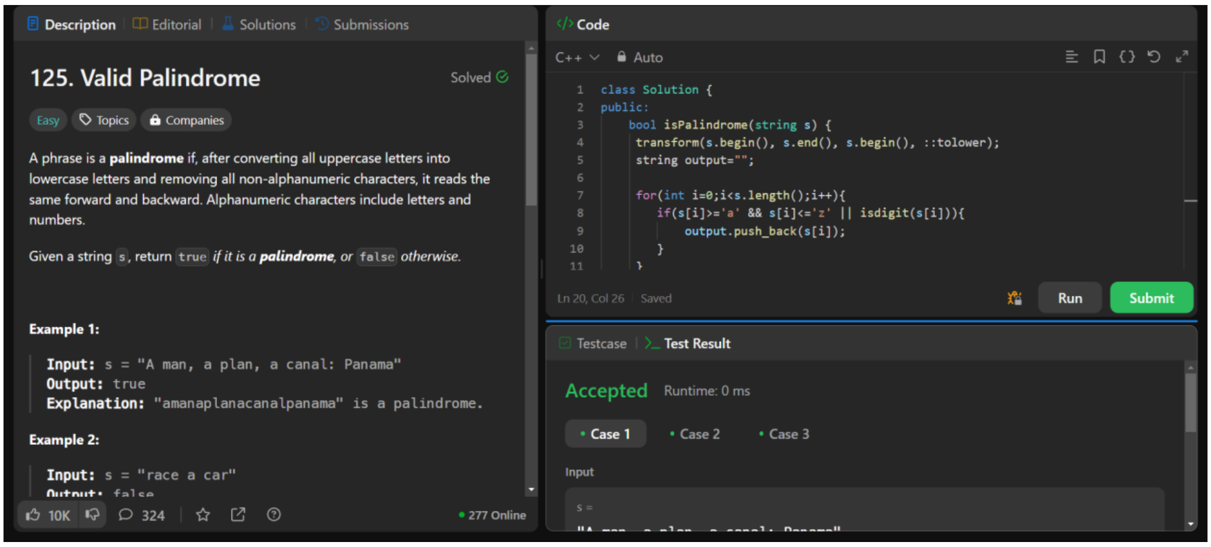
Input

nums =

5) Valid Palindrome

```
bool isPalindrome(string s) {
    transform(s.begin(), s.end(), s.begin(), ::tolower);
    string output="";

    for(int i=0;i<s.length();i++){
      if(s[i]>='a' && s[i]<='z' || isdigit(s[i])){
          output.push_back(s[i]);
      }
    }
    int i=0;
    int j=output.length()-1;
    while(i<=j){
      if(output[i]==output[j]){
          i++;
          j--;
      }
      else{
          return false;
      }
    }
```

```
  return true;
 }
```



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.*

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

```
Input: s = "race a car"
```

```
class Solution {
public:
    bool isPalindrome(string s) {
     transform(s.begin(), s.end(), s.begin(), ::tolower);
     string output="";

     for(int i=0;i<s.length();i++){
        if(s[i]>='a' && s[i]<='z' || isdigit(s[i])){
            output.push_back(s[i]);
        }
     }
```

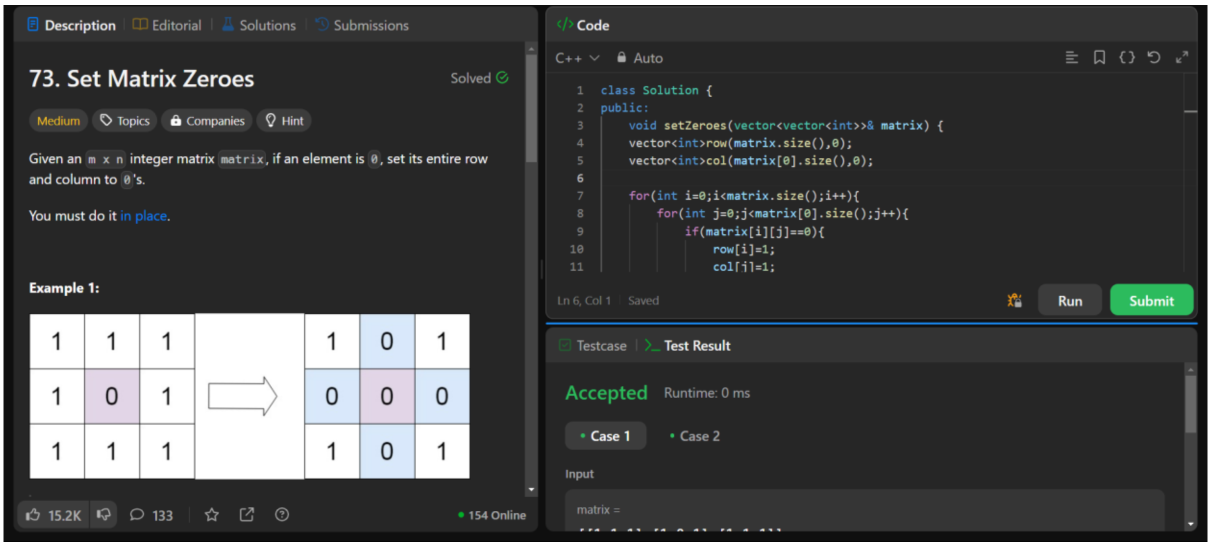
Accepted Runtime: 0 ms

s =

6) Set Matrix Zeroes

```
void setZeroes(vector<vector<int>>& matrix) {
   vector<int>row(matrix.size(),0);
   vector<int>col(matrix[0].size(),0);


   for(int i=0;i<matrix.size();i++){
      for(int j=0;j<matrix[0].size();j++){
         if(matrix[i][j]==0){
            row[i]=1;
            col[j]=1;
         }
      }
   }
   for(int i=0;i<matrix.size();i++){
      for(int j=0;j<matrix[0].size();j++){
         if(row[i]==1 || col[j]==1){
            matrix[i][j]=0;
         }
```

```
        }
    }
    }
```



73. Set Matrix Zeroes

Solved

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.

Example 1:

1	1	1
1	0	1
1	1	1

1	0	1
0	0	0
1	0	1

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
    vector<int>row(matrix.size(),0);
    vector<int>col(matrix[0].size(),0);

    for(int i=0;i<matrix.size();i++){
        for(int j=0;j<matrix[0].size();j++){
            if(matrix[i][j]==0){
                row[i]=1;
                col[j]=1;
```

Ln 6, Col 1 Saved

Accepted Runtime: 0 ms

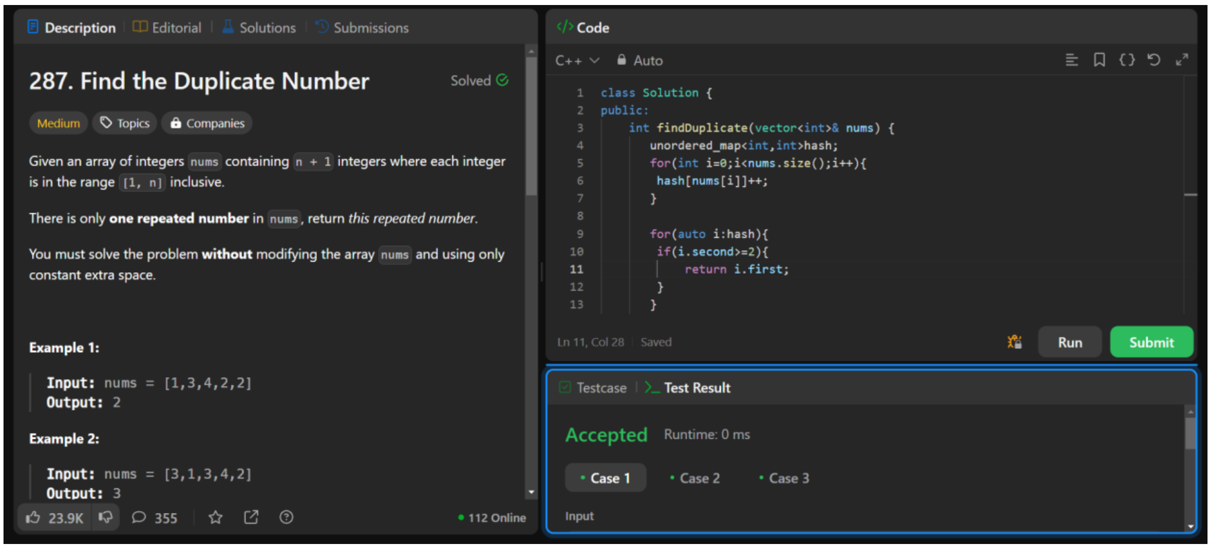
Case 1 Case 2

Input

matrix =

7) Finding duplicate number

```
int findDuplicate(vector<int>& nums) {
      unordered_map<int,int>hash;
      for(int i=0;i<nums.size();i++){
       hash[nums[i]]++;
      }
      for(auto i:hash){
       if(i.second>=2){
           return i.first;
       }
      }
      return -1;
  }
```



287. Find the Duplicate Number

Solved

Medium · Topics · Companies

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.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: nums = [3,1,3,4,2]
Output: 3
```

```cpp
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
       unordered_map<int,int>hash;
       for(int i=0;i<nums.size();i++){
        hash[nums[i]]++;
       }

       for(auto i:hash){
        if(i.second>=2){
            return i.first;
        }
       }
```

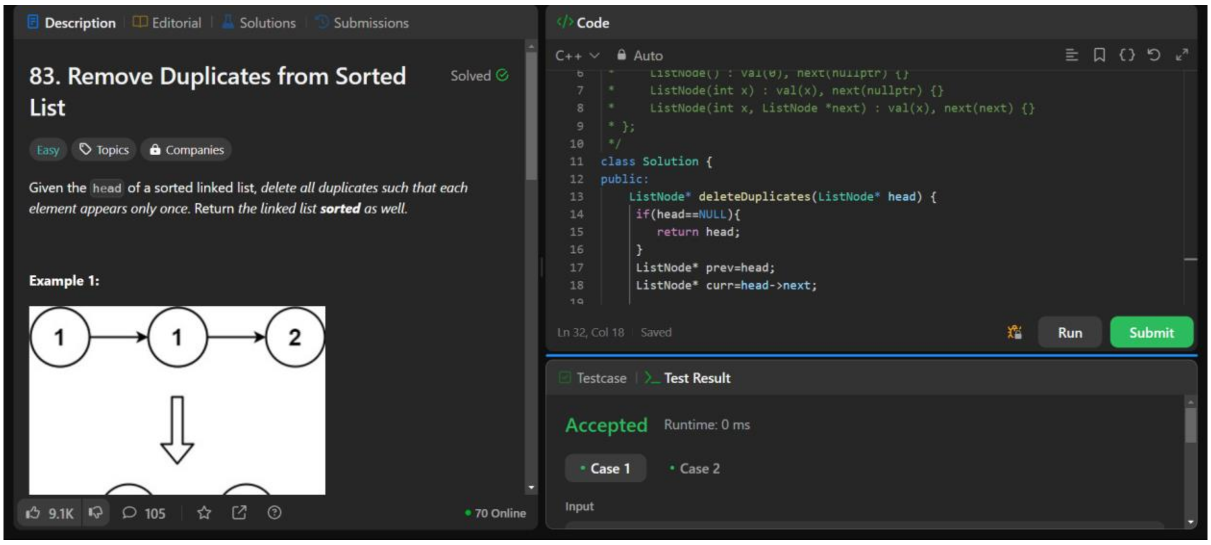
Ln 11, Col 28 Saved

Accepted Runtime: 0 ms

8) Remove duplicates from a sorted list

```cpp
ListNode* deleteDuplicates(ListNode* head) {
    if(head==NULL){
        return head;
    }
    ListNode* prev=head;
    ListNode* curr=head->next;
    while(curr!=NULL){
    ListNode* temp=curr;
    if(prev->val==temp->val){
        curr=curr->next;
        prev->next=curr;
        delete temp;
    }
    else{
        prev=curr;
        curr=curr->next;
    }
    }
    return head;
```

}


Description | Editorial | Solutions | Submissions

83. Remove Duplicates from Sorted List

Solved

Easy · Topics · Companies

Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list **sorted** as well*.

Example 1:

9.1K · 105 · 70 Online

Code

C++ · Auto

```
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
     if(head==NULL){
        return head;
     }
     ListNode* prev=head;
     ListNode* curr=head->next;
```

Ln 32, Col 18 · Saved · Run · Submit

Testcase | Test Result

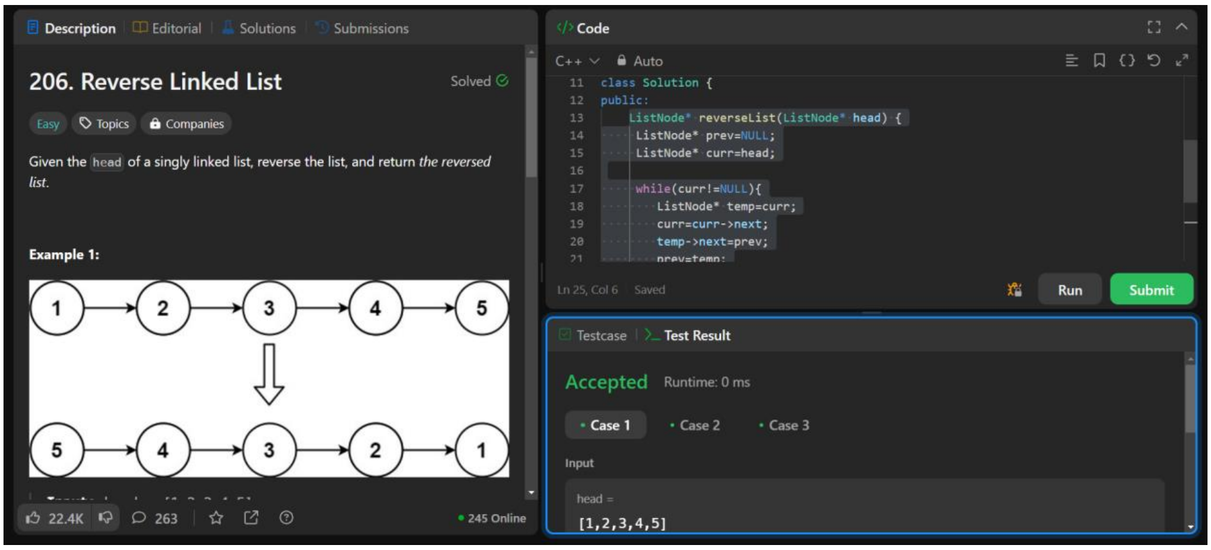
Accepted Runtime: 0 ms

Case 1 · Case 2

Input

9) Reverse a linked list

```
ListNode* reverseList(ListNode* head) {
    ListNode* prev=NULL;
    ListNode* curr=head;

    while(curr!=NULL){
      ListNode* temp=curr;
      curr=curr->next;
      temp->next=prev;
      prev=temp;
    }

    return prev;
  }
```



206. Reverse Linked List

Solved

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
     ListNode* prev=NULL;
     ListNode* curr=head;

     while(curr!=NULL){
         ListNode* temp=curr;
         curr=curr->next;
         temp->next=prev;
         prev=temp;
```

Ln 25, Col 6   Saved

Accepted   Runtime: 0 ms

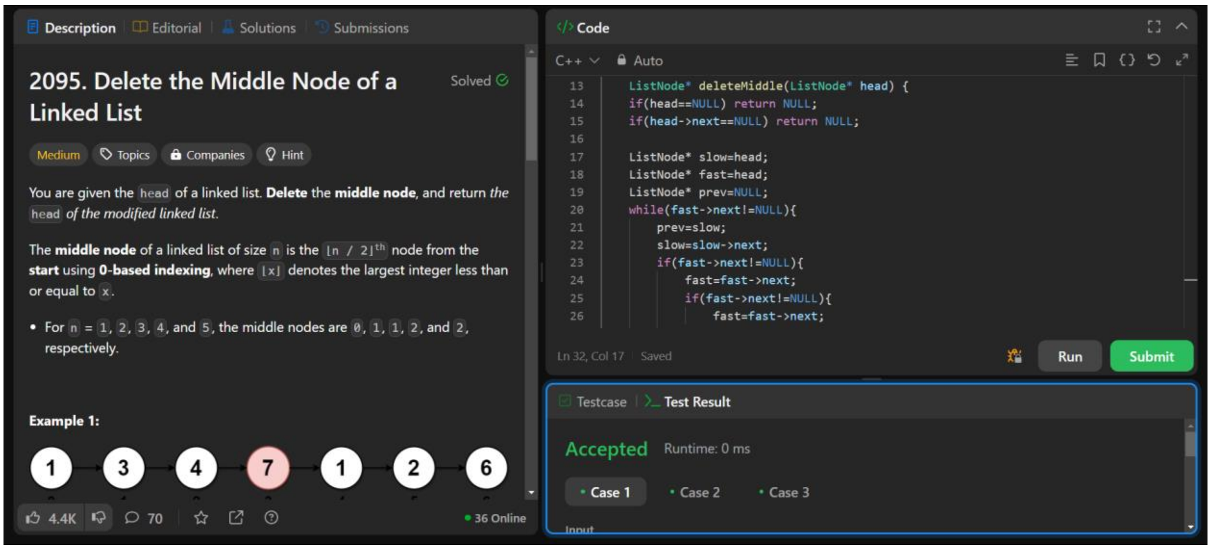
Input

head =

[1,2,3,4,5]

10) Delete middle node of a listCW

```
ListNode* deleteMiddle(ListNode* head) {
    if(head==NULL) return NULL;
    if(head->next==NULL) return NULL;

    ListNode* slow=head;
    ListNode* fast=head;
    ListNode* prev=NULL;
    while(fast->next!=NULL){
        prev=slow;
        slow=slow->next;
        if(fast->next!=NULL){
            fast=fast->next;
            if(fast->next!=NULL){
                fast=fast->next;
            }
        }
    }
    prev->next=slow->next;
    slow->next=NULL;
    return head;
    }
```



2095. Delete the Middle Node of a Linked List

You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.*

The **middle node** of a linked list of size `n` is the `⌊n / 2⌋th` node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

Example 1:

1 → 3 → 4 → 7 → 1 → 2 → 6

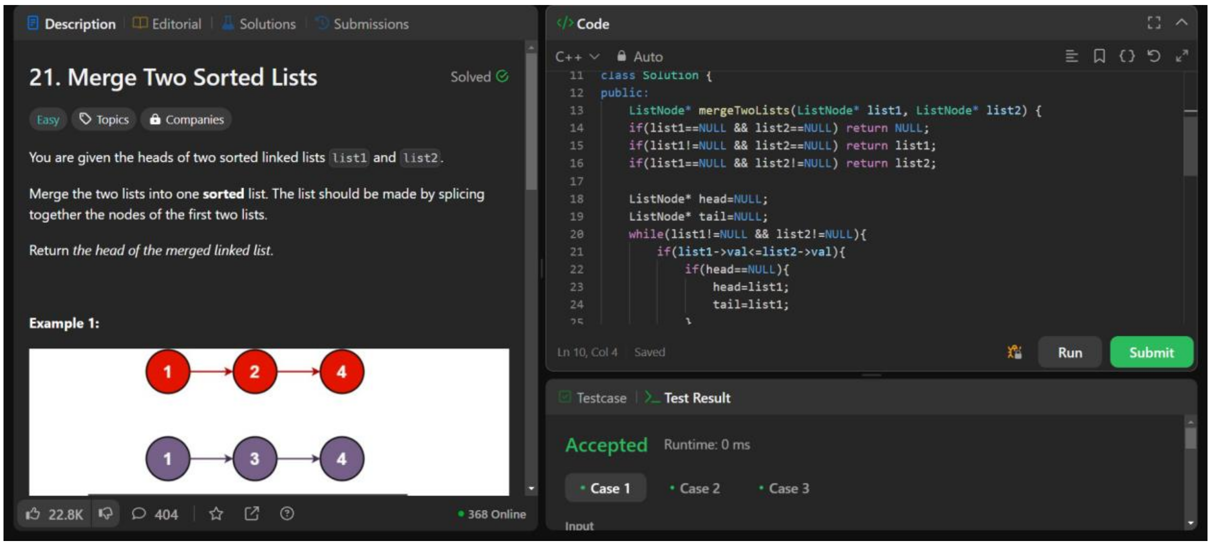
```
ListNode* deleteMiddle(ListNode* head) {
if(head==NULL) return NULL;
if(head->next==NULL) return NULL;

ListNode* slow=head;
ListNode* fast=head;
ListNode* prev=NULL;
while(fast->next!=NULL){
    prev=slow;
    slow=slow->next;
    if(fast->next!=NULL){
        fast=fast->next;
        if(fast->next!=NULL){
            fast=fast->next;
```

Ln 32, Col 17 Saved

Accepted Runtime: 0 ms

11) Merge two sorted linked lists

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if(list1==NULL && list2==NULL) return NULL;
    if(list1!=NULL && list2==NULL) return list1;
    if(list1==NULL && list2!=NULL) return list2;

    ListNode* head=NULL;
    ListNode* tail=NULL;
    while(list1!=NULL && list2!=NULL){
        if(list1->val<=list2->val){
            if(head==NULL){
                head=list1;
                tail=list1;
            }
            else{
                tail->next=list1;
                tail=list1;
            }
            list1=list1->next;
        }
        else{
            if(head==NULL){
                head=list2;
                tail=list2;
            }
            else{
                tail->next=list2;
                tail=list2;
            }
            list2=list2->next;
        }
    }
    while(list1!=NULL){
        tail->next=list1;
        tail=list1;
        list1=list1->next;
    }
    while(list2!=NULL){
        tail->next=list2;
        tail=list2;
        list2=list2->next;
    }
    return head;
    }
```



21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list.*

Example 1:

C++ Auto

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if(list1==NULL && list2==NULL) return NULL;
    if(list1!=NULL && list2==NULL) return list1;
    if(list1==NULL && list2!=NULL) return list2;

    ListNode* head=NULL;
    ListNode* tail=NULL;
    while(list1!=NULL && list2!=NULL){
        if(list1->val<=list2->val){
            if(head==NULL){
                head=list1;
                tail=list1;
```

Ln 10, Col 4 Saved

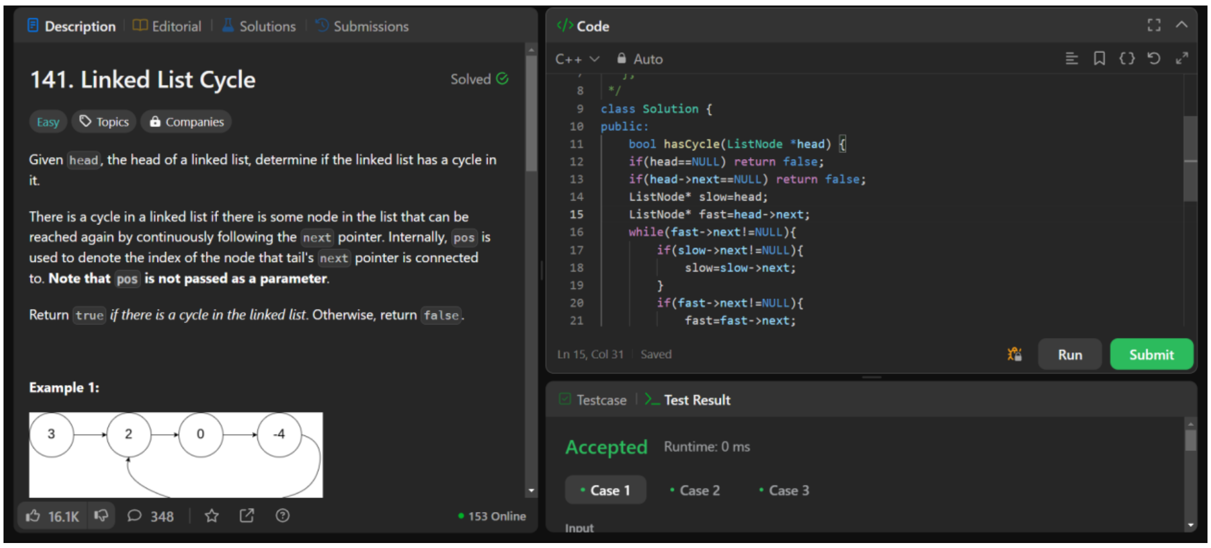
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

12) Detect a cycle in a linked list

```
bool hasCycle(ListNode *head) {
    if(head==NULL) return false;
    if(head->next==NULL) return false;
    ListNode* slow=head;
    ListNode* fast=head->next;
    while(fast->next!=NULL){
        if(slow->next!=NULL){
            slow=slow->next;
        }
        if(fast->next!=NULL){
            fast=fast->next;
            if(slow==fast){
                return true;
            }
            if(fast->next!=NULL){
                fast=fast->next;
                if(slow==fast){
                    return true;
                }
            }
        }
    }
    return false;
    }
```



Description | Editorial | Solutions | Submissions

141. Linked List Cycle

Solved

Easy | Topics | Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

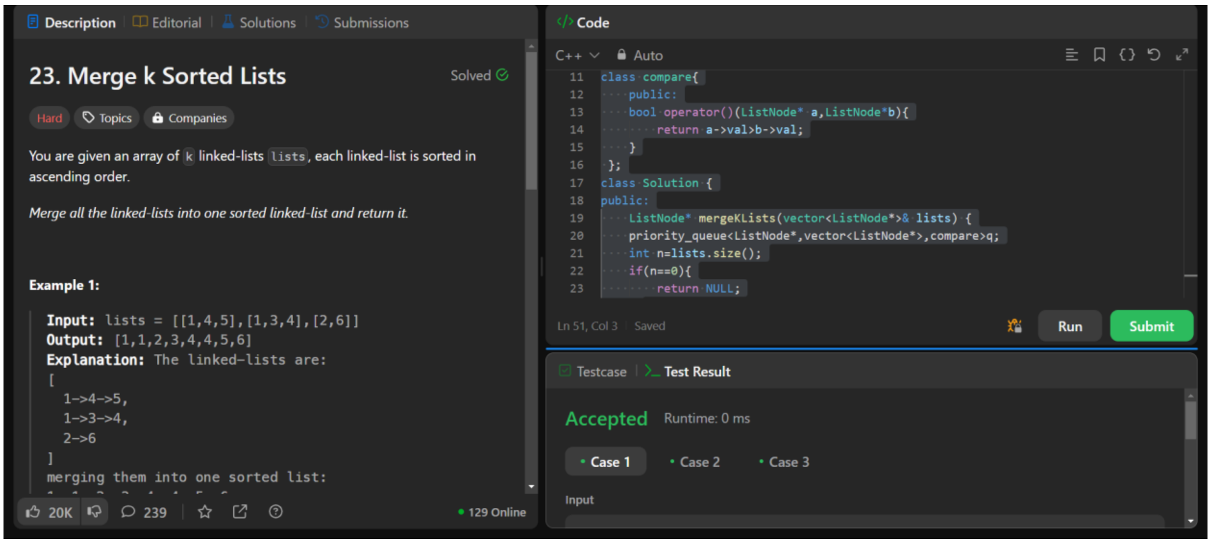
```
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
    if(head==NULL) return false;
    if(head->next==NULL) return false;
    ListNode* slow=head;
    ListNode* fast=head->next;
    while(fast->next!=NULL){
        if(slow->next!=NULL){
            slow=slow->next;
        }
        if(fast->next!=NULL){
            fast=fast->next;
```

Ln 15, Col 31 | Saved

Accepted Runtime: 0 ms

13)

```
class compare{
    public:
    bool operator()(ListNode* a,ListNode*b){
        return a->val>b->val;
    }
 };
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*,vector<ListNode*>,compare>q;
    int n=lists.size();
    if(n==0){
        return NULL;
    }
    for(int i=0;i<n;i++){
        ListNode* temp=lists[i];
        if(temp!=NULL){
            q.push(temp);
        }
    }
    ListNode* head=NULL;
    ListNode* tail=NULL;

    while(!q.empty()){
        ListNode* temp=q.top();
        q.pop();
        if(head==NULL){
            head=temp;
            tail=temp;
        }
        else{
            tail->next=temp;
            tail=temp;
        }
        if(temp->next!=NULL){
            q.push(temp->next);
        }
    }
    return head;
    }
};
```



23. Merge k Sorted Lists

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
```

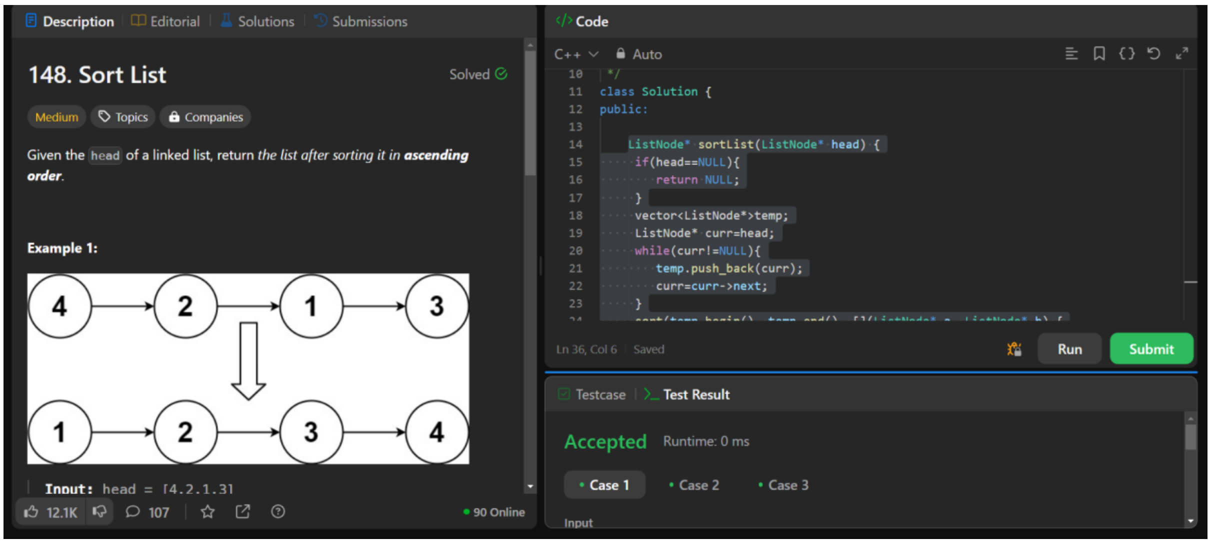
```
class compare{
    public:
    bool operator()(ListNode* a,ListNode*b){
        return a->val>b->val;
    }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*,vector<ListNode*>,compare>q;
    int n=lists.size();
    if(n==0){
        return NULL;
```

Ln 51, Col 3 Saved

Accepted Runtime: 0 ms

14) Sort List

```
ListNode* sortList(ListNode* head) {
    if(head==NULL){
      return NULL;
    }
    vector<ListNode*>temp;
    ListNode* curr=head;
    while(curr!=NULL){
      temp.push_back(curr);
      curr=curr->next;
    }
    sort(temp.begin(), temp.end(), [](ListNode* a, ListNode* b) {
   return a->val < b->val;
});
    ListNode* newhead=temp[0];
    ListNode* cur=temp[0];
    for(int i=1;i<temp.size();i++){
      cur->next=temp[i];
      cur=temp[i];
    }
    cur->next=NULL;

    return newhead;


  }
```



Description | Editorial | Solutions | Submissions

148. Sort List Solved

Medium | Topics | Companies

Given the `head` of a linked list, return *the list after sorting it in **ascending** order*.

Example 1:

`Input: head = [4,2,1,3]`

12.1K | 107 | 90 Online

Code

C++ | Auto

```
*/
class Solution {
public:

    ListNode* sortList(ListNode* head) {
    if(head==NULL){
        return NULL;
    }
    vector<ListNode*>temp;
    ListNode* curr=head;
    while(curr!=NULL){
        temp.push_back(curr);
        curr=curr->next;
    }
```

Ln 36, Col 6 | Saved | Run | Submit

Testcase | Test Result

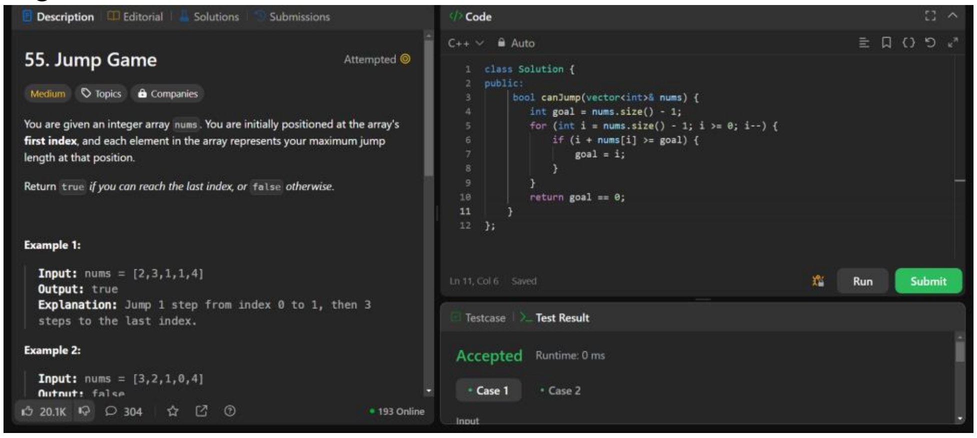
Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

15)Jump Game

```
bool canJump(vector<int>& nums) {
      int goal = nums.size() - 1;
      for (int i = nums.size() - 1; i >= 0; i--) {
         if (i + nums[i] >= goal) {
             goal = i;
         }
      }
      return goal == 0;
  }
```



Description | Editorial | Solutions | Submissions

55. Jump Game Attempted

Medium | Topics | Companies

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.*

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3
steps to the last index.
```

Example 2:

```
Input: nums = [3,2,1,0,4]
Output: false
```

20.1K | 304 | 193 Online

Code

C++ | Auto

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int goal = nums.size() - 1;
        for (int i = nums.size() - 1; i >= 0; i--) {
            if (i + nums[i] >= goal) {
                goal = i;
            }
        }
        return goal == 0;
    }
};
```

Ln 11, Col 6 | Saved | Run | Submit

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input