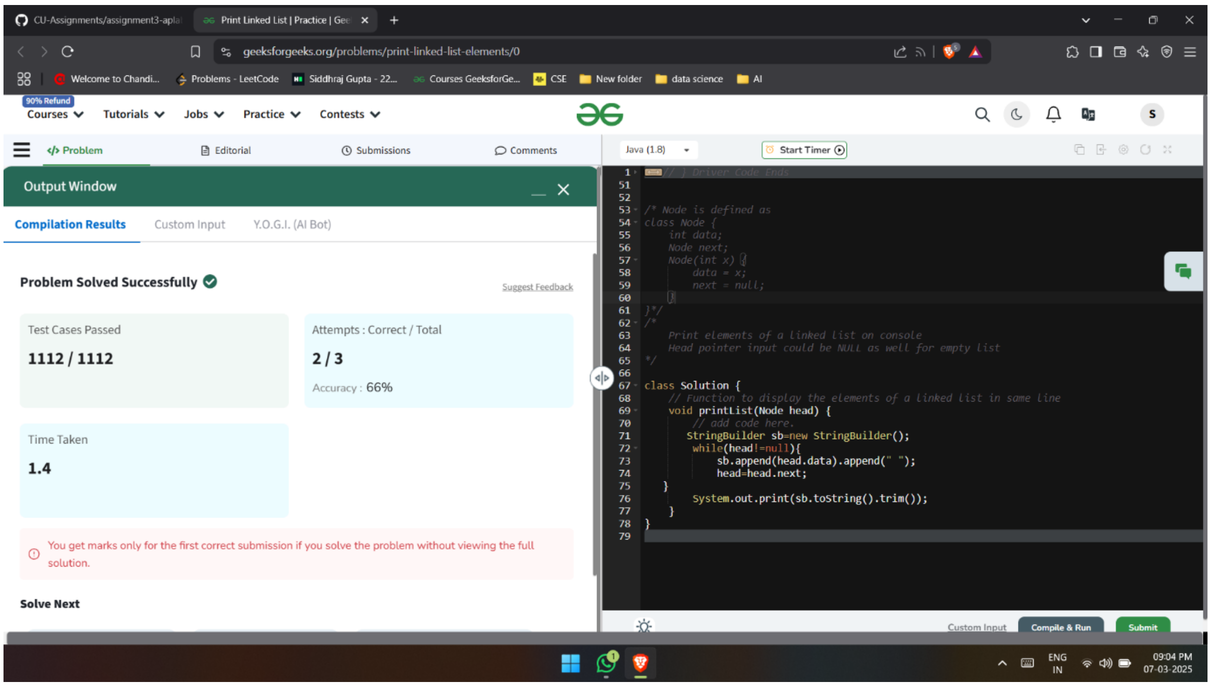
Name – Chakshu Jain

UID – 22BCS15224

Section – 608/B

AP LAB ASSIGNMENT 2

1. Print Linked List



Output Window — Compilation Results

Problem Solved Successfully

Test Cases Passed: 1112 / 1112

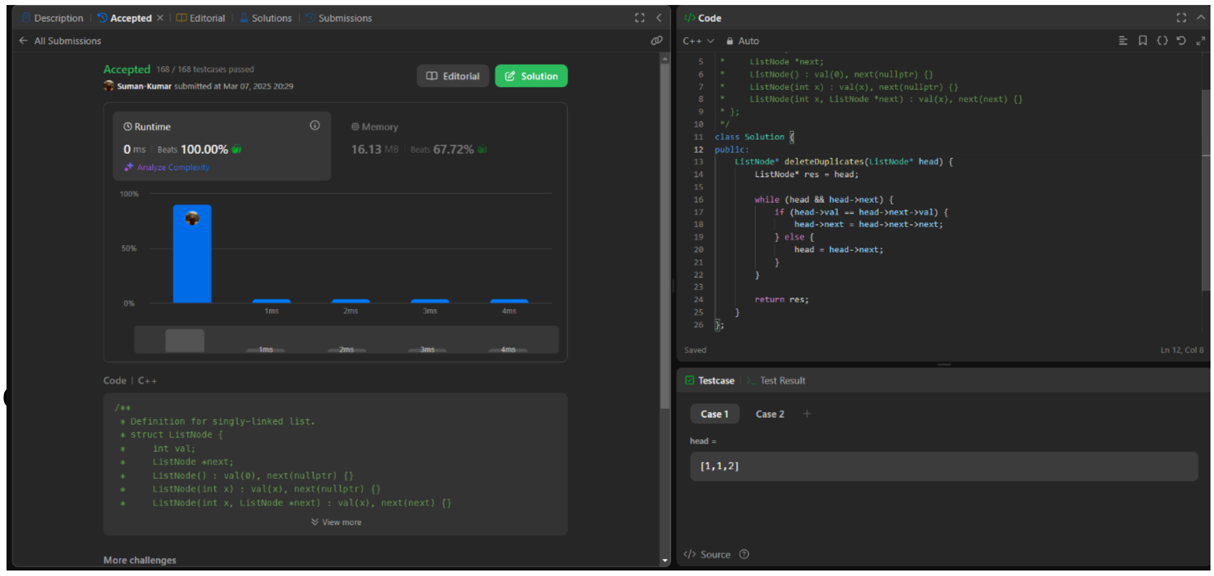
Attempts : Correct / Total: 2 / 3 — Accuracy : 66%

Time Taken: 1.4

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

```java
/* Node is defined as
class Node {
    int data;
    Node next;
    Node(int x) {
        data = x;
        next = null;
    }
}*/
/*
    Print elements of a linked list on console
    Head pointer input could be NULL as well for empty list
*/

class Solution {
    // Function to display the elements of a linked list in same line
    void printList(Node head) {
        // add code here.
       StringBuilder sb=new StringBuilder();
        while(head!=null){
            sb.append(head.data).append(" ");
            head=head.next;
    }
        System.out.print(sb.toString().trim());
    }
}
```

2. Remove duplicates from a sorted list



Accepted 168 / 168 testcases passed

Suman-Kumar submitted at Mar 07, 2025 20:29

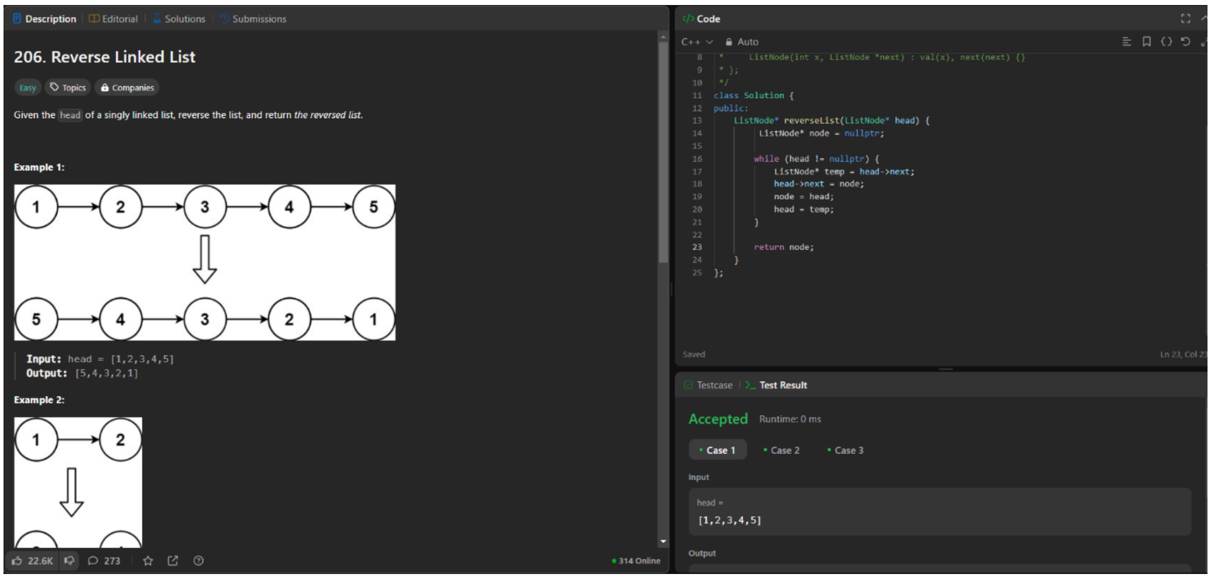
Runtime: 0 ms | Beats 100.00%

Memory: 16.13 MB | Beats 67.72%

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     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) {
        ListNode* res = head;

        while (head && head->next) {
            if (head->val == head->next->val) {
                head->next = head->next->next;
            } else {
                head = head->next;
            }
        }

        return res;
    }
};
```

Testcase — Case 1: head = [1,1,2]

3. R



206. Reverse Linked List

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

Example 1:

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

Example 2:

```
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* node = nullptr;

        while (head != nullptr) {
            ListNode* temp = head->next;
            head->next = node;
            node = head;
            head = temp;
        }

        return node;
    }
};
```

Accepted Runtime: 0 ms

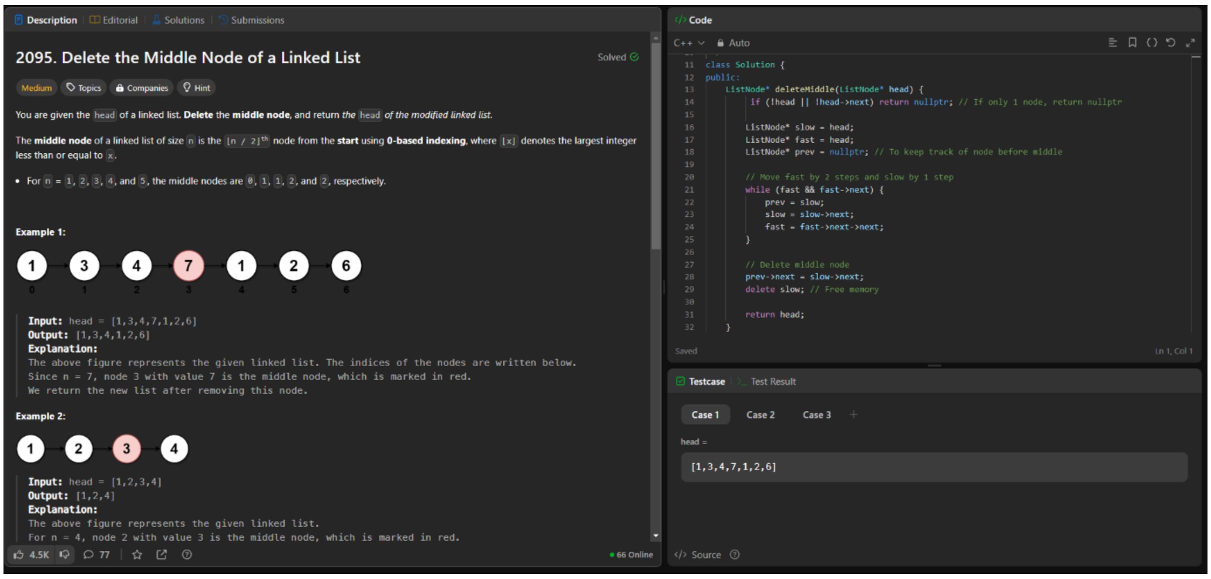
Input

head =

[1,2,3,4,5]

Output

4. Delete middle node of a list



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:

```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes are written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.
```

Example 2:

```
Input: head = [1,2,3,4]
Output: [1,2,4]
Explanation:
The above figure represents the given linked list.
For n = 4, node 2 with value 3 is the middle node, which is marked in red.
```

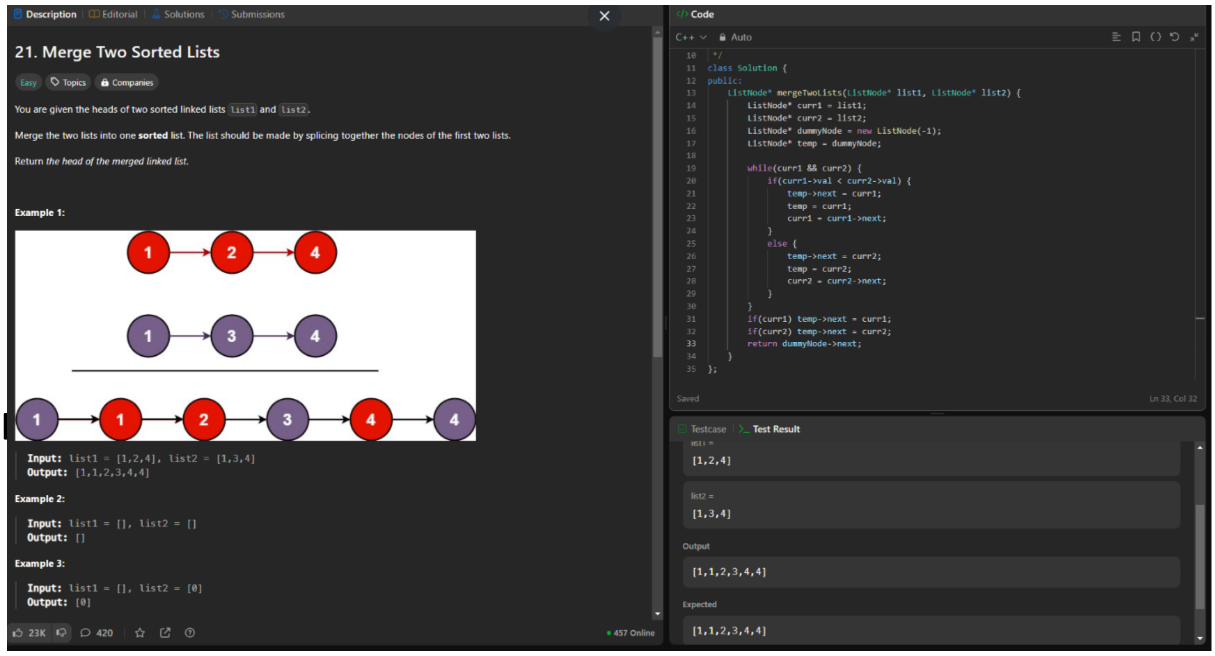
```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr; // If only 1 node, return nullptr

        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr; // To keep track of node before middle

        // Move fast by 2 steps and slow by 1 step
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        // Delete middle node
        prev->next = slow->next;
        delete slow; // Free memory

        return head;
    }
```

head =

[1,3,4,7,1,2,6]

5. Merge two sorted linked lists

6. I



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:

```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

```
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* curr1 = list1;
        ListNode* curr2 = list2;
        ListNode* dummyNode = new ListNode(-1);
        ListNode* temp = dummyNode;

        while(curr1 && curr2) {
            if(curr1->val < curr2->val) {
                temp->next = curr1;
                temp = curr1;
                curr1 = curr1->next;
            }
            else {
                temp->next = curr2;
                temp = curr2;
                curr2 = curr2->next;
            }
        }
        if(curr1) temp->next = curr1;
        if(curr2) temp->next = curr2;
        return dummyNode->next;
    }
};
```

list1 =

[1,2,4]

list2 =

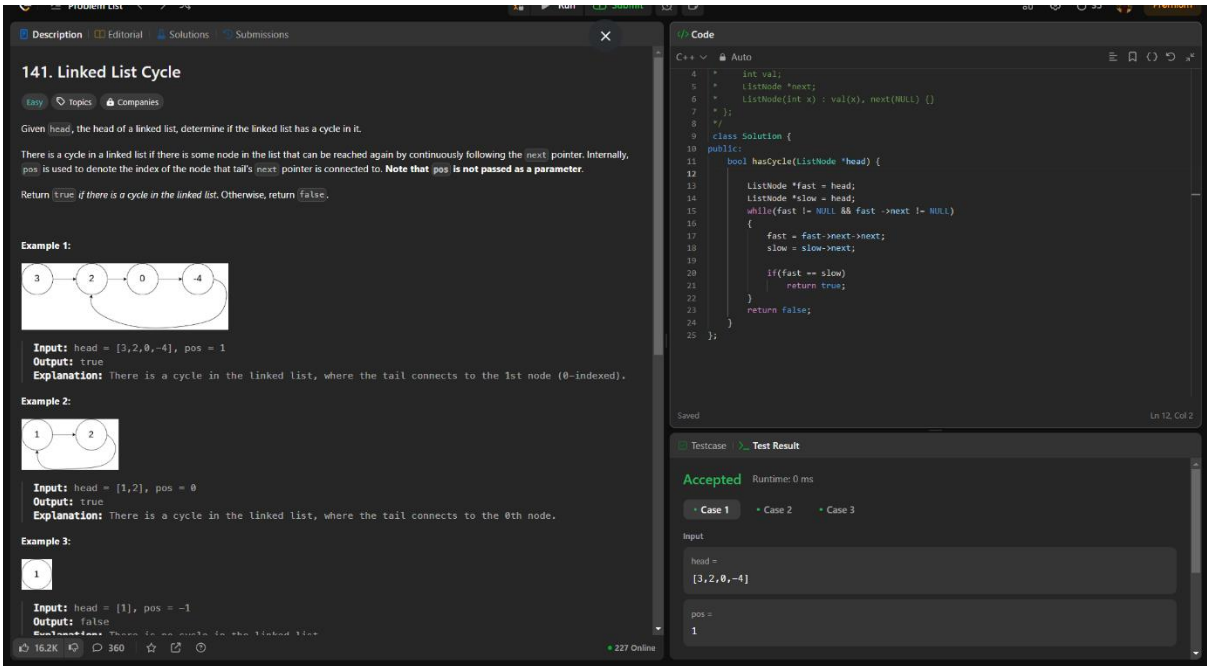
[1,3,4]

Output

[1,1,2,3,4,4]

Expected

[1,1,2,3,4,4]



Description | Editorial | Solutions | Submissions

141. Linked List Cycle

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:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
```

Example 2:

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
```

Example 3:

```
Input: head = [1], pos = -1
Output: false
```

16.2K | 360 | 227 Online

Code — C++ — Auto

```cpp
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {

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

            if(fast == slow)
                return true;
        }
        return false;
    }
};
```

Saved | Ln 12, Col 2

Testcase | Test Result

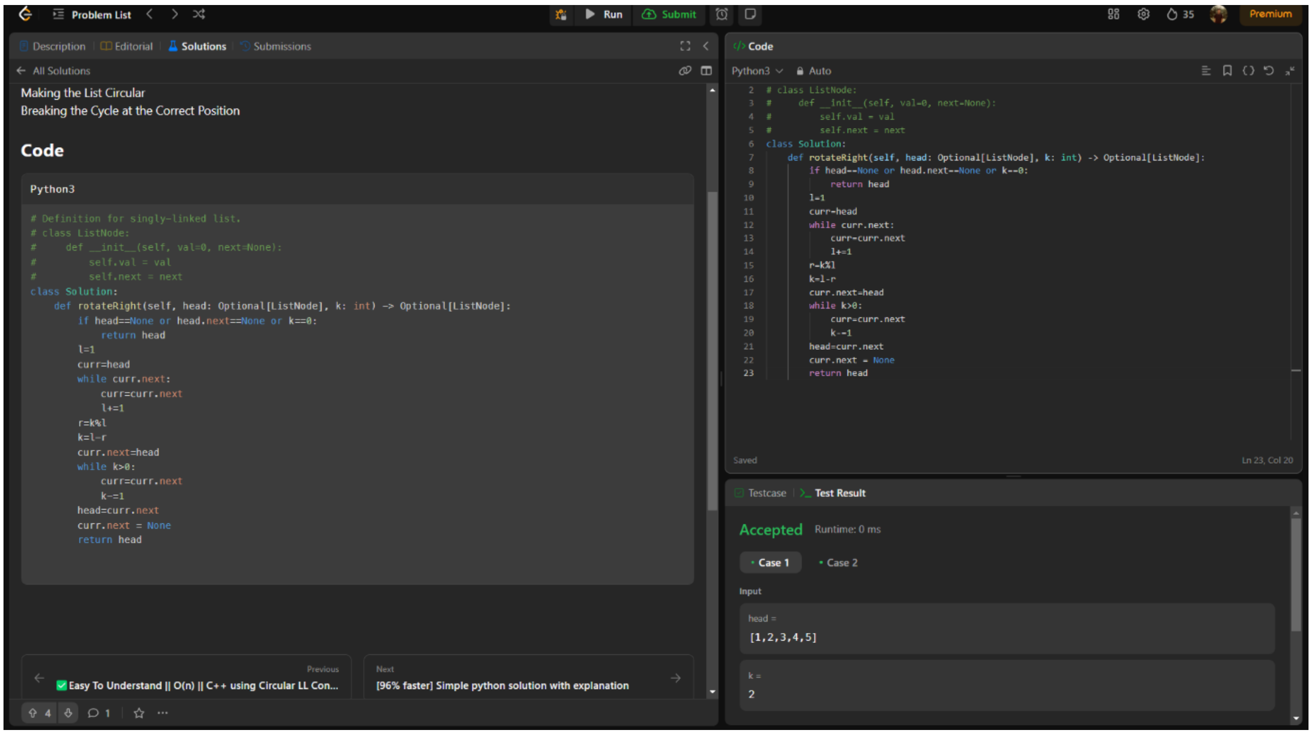
Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

```
head =
[3,2,0,-4]

pos =
1
```

7. Rotate a list



Problem List | Run | Submit | 35 | Premium

Description | Editorial | Solutions | Submissions

All Solutions

Making the List Circular
Breaking the Cycle at the Correct Position

Code

Python3

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if head==None or head.next==None or k==0:
            return head
        l=1
        curr=head
        while curr.next:
            curr=curr.next
            l+=1
        r=k%l
        k=l-r
        curr.next=head
        while k>0:
            curr=curr.next
            k-=1
        head=curr.next
        curr.next = None
        return head
```

Previous: Easy To Understand || O(n) || C++ using Circular LL Con...

Next: [96% faster] Simple python solution with explanation

4 | 1

Code — Python3 — Auto

```python
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if head==None or head.next==None or k==0:
            return head
        l=1
        curr=head
        while curr.next:
            curr=curr.next
            l+=1
        r=k%l
        k=l-r
        curr.next=head
        while k>0:
            curr=curr.next
            k-=1
        head=curr.next
        curr.next = None
        return head
```

Saved | Ln 23, Col 20

Testcase | Test Result

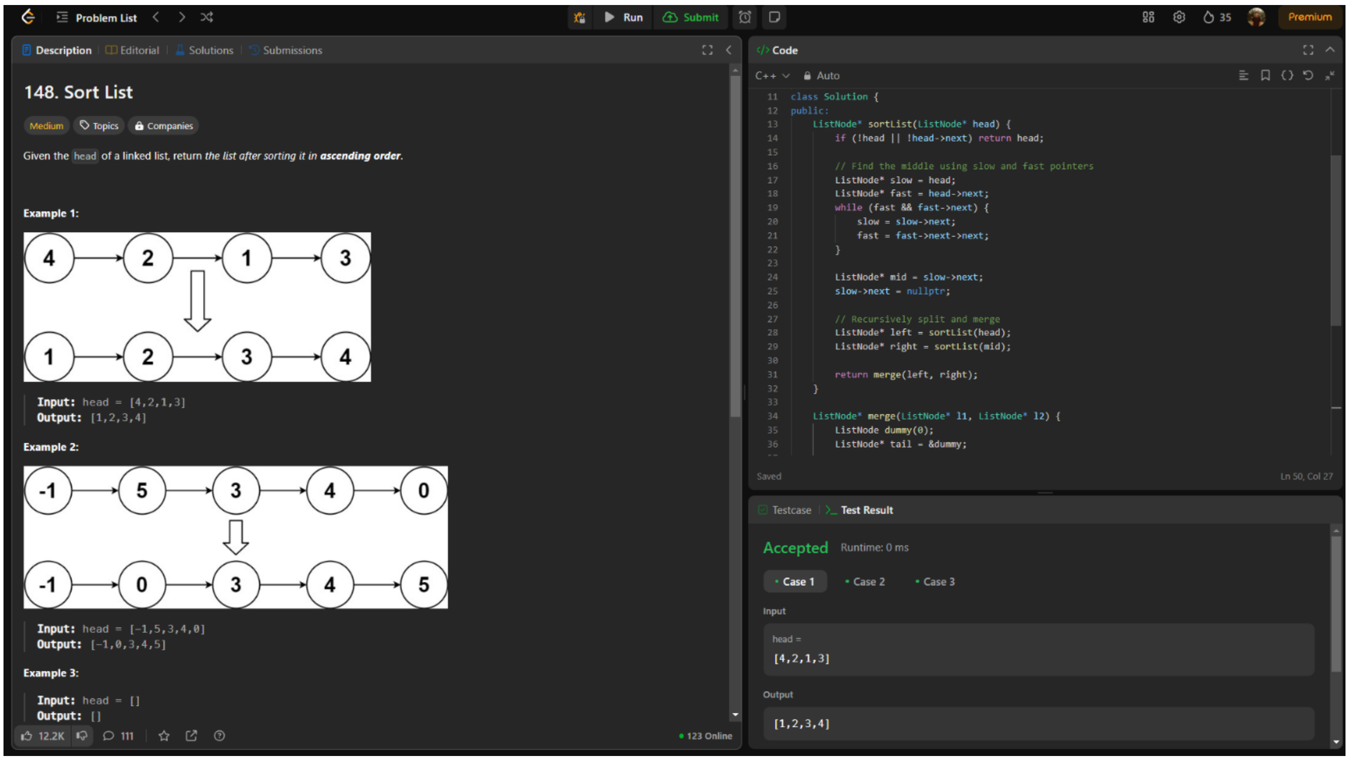
Accepted Runtime: 0 ms

Case 1 | Case 2

Input

```
head =
[1,2,3,4,5]

k =
2
```

8. Sort List



148. Sort List

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

Example 1:

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

Example 2:

```
Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]
```

Example 3:

```
Input: head = []
Output: []
```

```cpp
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        // Find the middle using slow and fast pointers
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        ListNode* mid = slow->next;
        slow->next = nullptr;

        // Recursively split and merge
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);

        return merge(left, right);
    }

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
```

Accepted Runtime: 0 ms

Input

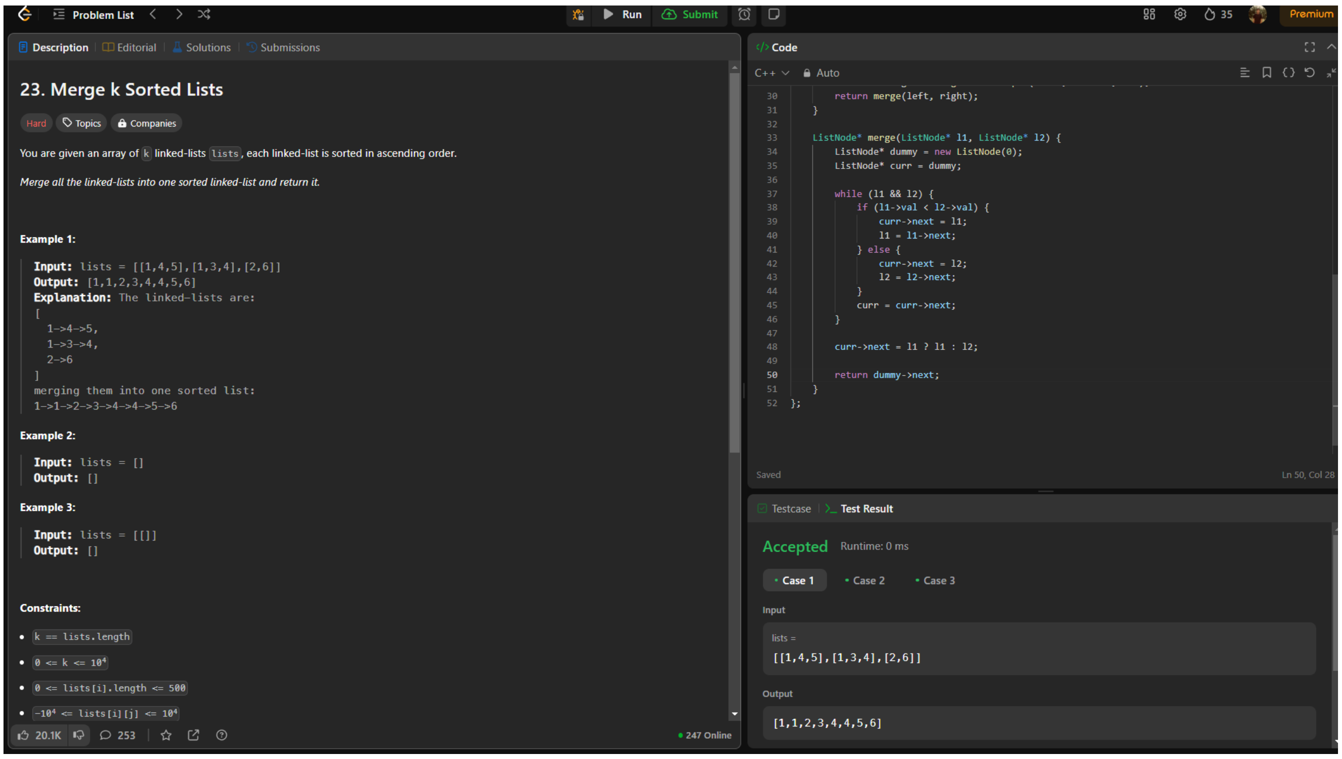
head =

```
[4,2,1,3]
```

Output

```
[1,2,3,4]
```

9. Merge k sorted lists



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:
1->1->2->3->4->4->5->6
```

Example 2:

```
Input: lists = []
Output: []
```

Example 3:

```
Input: lists = [[]]
Output: []
```

Constraints:

- `k == lists.length`
- `0 <= k <= 10^4`
- `0 <= lists[i].length <= 500`
- `-10^4 <= lists[i][j] <= 10^4`

```cpp
        return merge(left, right);
    }

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode(0);
        ListNode* curr = dummy;

        while (l1 && l2) {
            if (l1->val < l2->val) {
                curr->next = l1;
                l1 = l1->next;
            } else {
                curr->next = l2;
                l2 = l2->next;
            }
            curr = curr->next;
        }

        curr->next = l1 ? l1 : l2;

        return dummy->next;
    }
};
```

Accepted Runtime: 0 ms

Input

lists =

```
[[1,4,5],[1,3,4],[2,6]]
```

Output

```
[1,1,2,3,4,4,5,6]
```