DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Assignment-2

Student Name: Armaan Jaswal | **UID:** 22BCS11756
Branch: BE- CSE-General | **Section/Group:** 22BCS-IOT-606/B
Semester: 6th | **Date of Submission:**06/03/25

Subject Name: Advanced Programming Lab-2 | **Subject Code:** 22CSP-351

Submitted to: Ms. Pratima Sonali Horo(E18304)

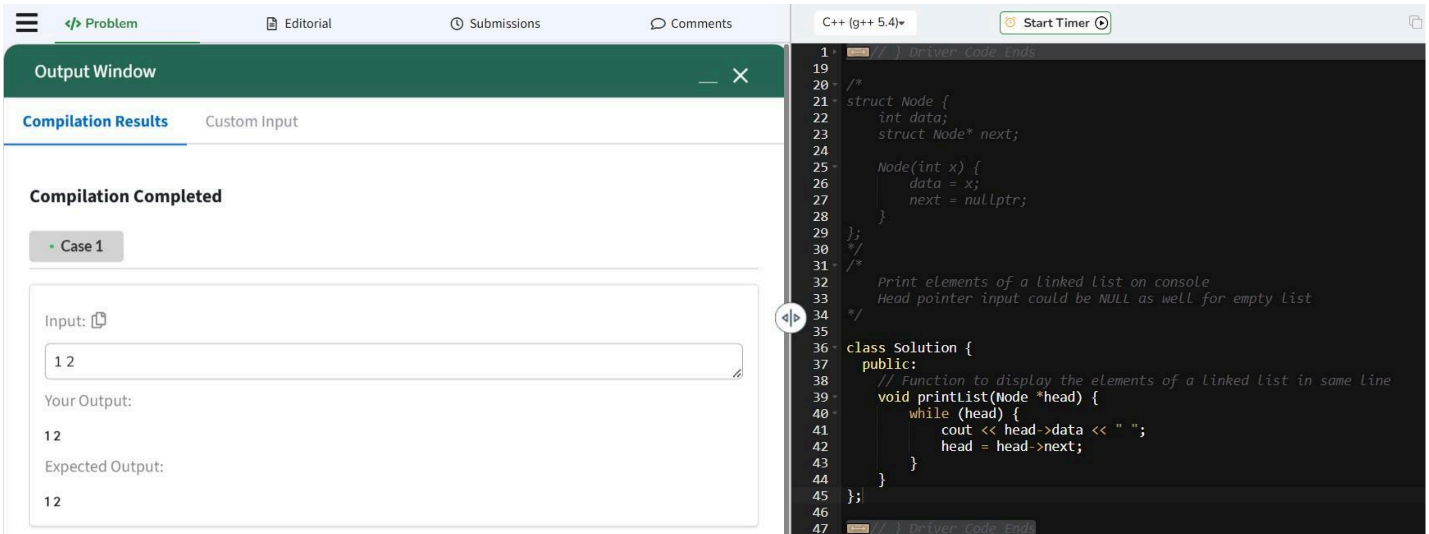
(i): Linked Lists:

Question-1: Print Linked List:

Given a linked list. Print all the elements of the linked list separated by space followed.

Answer:

```
class Solution {
public:
  // Function to display the elements of a linked list in same line
  void printList(Node *head) {
     while (head) {
        cout << head->data << " ";
        head = head->next;
     }
  }
};
```



Compilation Completed

Case 1

Input:

```
1 2
```

Your Output:

```
1 2
```

Expected Output:

```
1 2
```

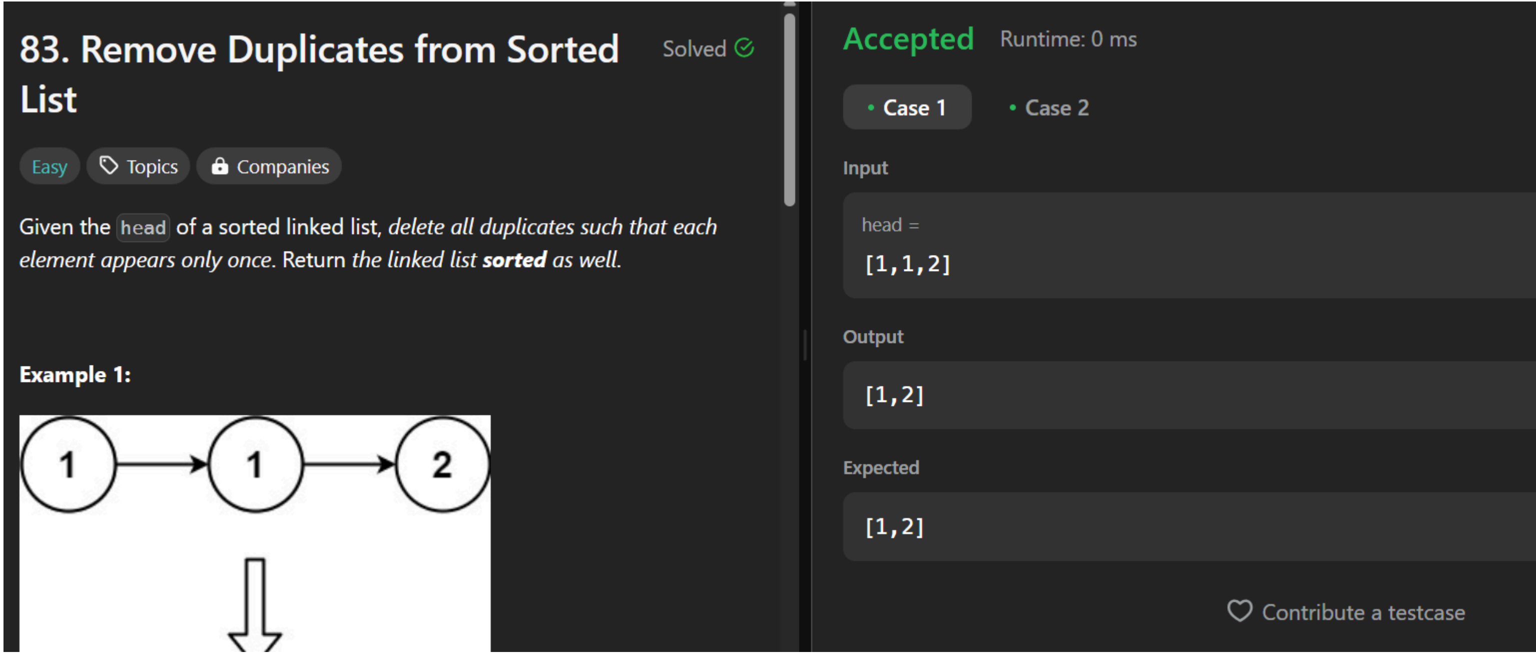
```
// } Driver Code Ends

/*
struct Node {
    int data;
    struct Node* next;

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

class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
    }
};

// } Driver Code Ends
```

Question-2: Remove Duplicates from Sorted List

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.*

Answer:

```
class Solution {

public:

    ListNode* deleteDuplicates(ListNode* head) {

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



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:

Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input

head =

```
[1,1,2]
```

Output

```
[1,2]
```

Expected

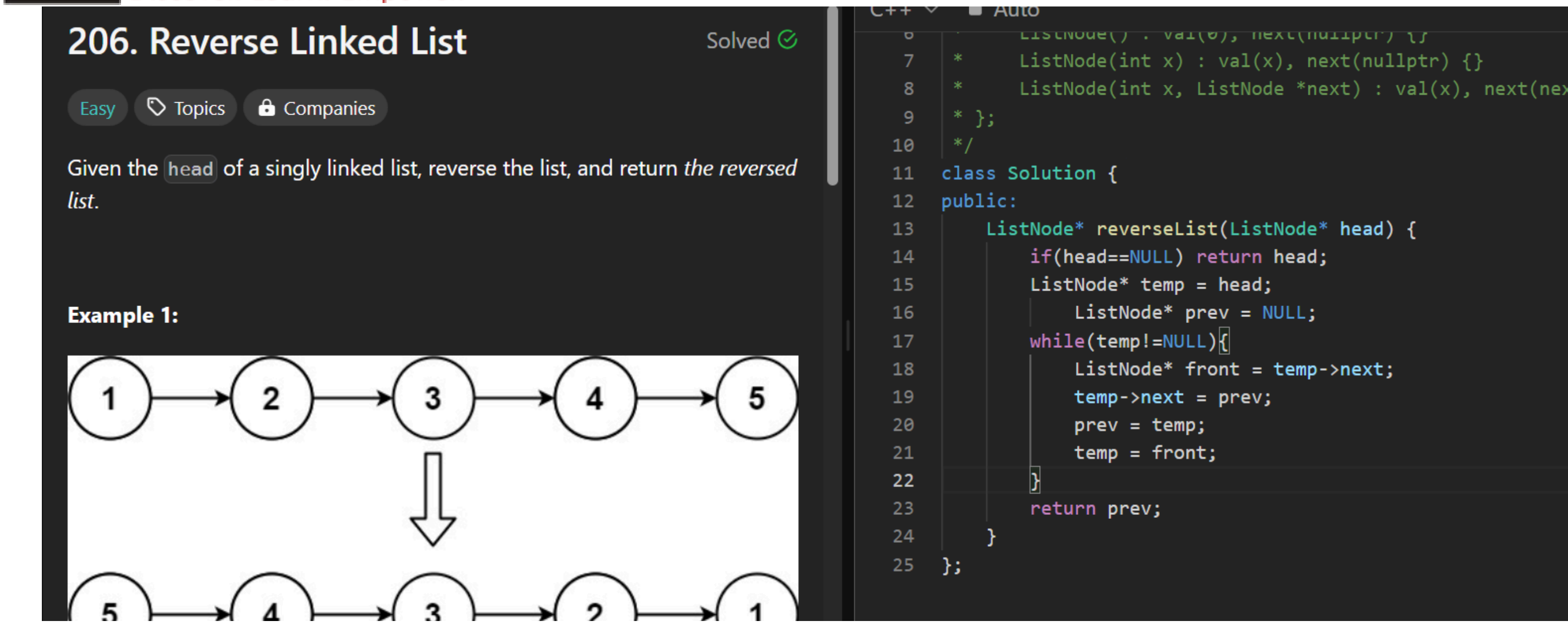
```
[1,2]
```

Contribute a testcase

Question-3: Reverse a linked list: Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Answer:

```
class Solution {
public:
ListNode*
reverseList(ListNode*
head) {
if(head==NULL)
return head;
ListNode* temp =
head;
ListNode* prev =
NULL;
while(temp!=NULL){
ListNode* front =
temp->next;
temp->next = prev;
prev = temp;
temp = front;
}
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:

```
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(ne
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(head==NULL) return head;
        ListNode* temp = head;
            ListNode* prev = NULL;
        while(temp!=NULL){
            ListNode* front = temp->next;
            temp->next = prev;
            prev = temp;
            temp = front;
        }
        return prev;
    }
};
```

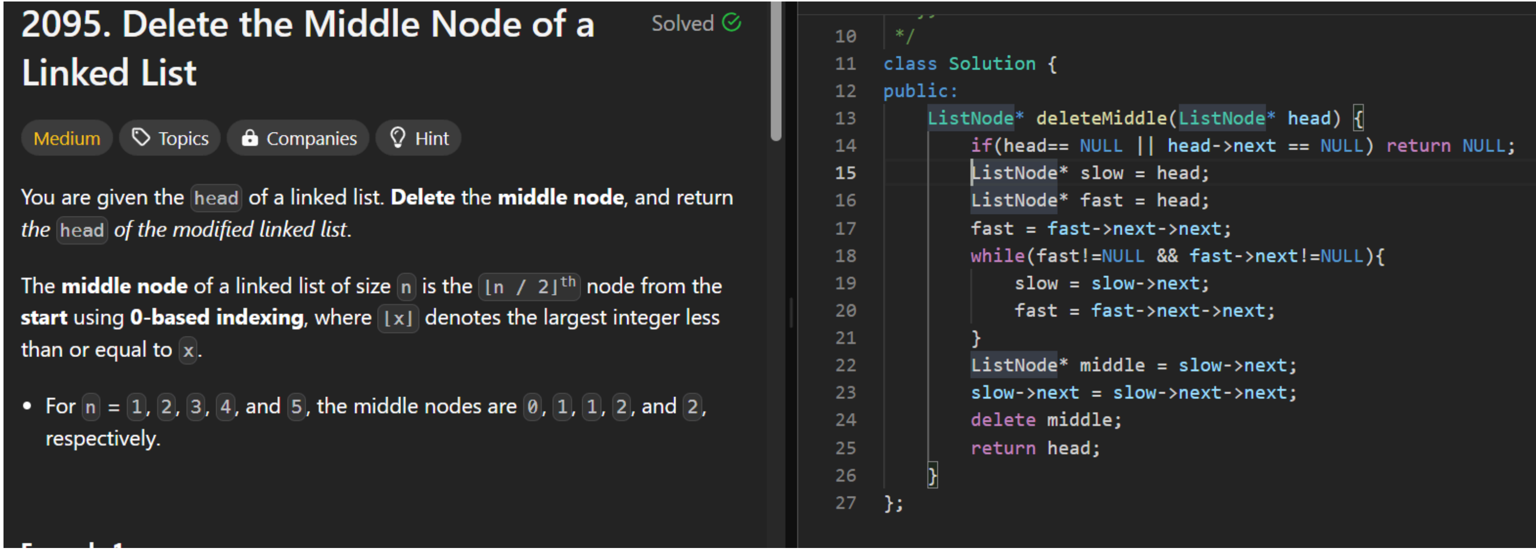
Question-4: Delete middle node of a list:

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.

Answer:

```
class Solution {
public:
ListNode* deleteMiddle(ListNode* head) {
if(head== NULL || head->next == NULL) return NULL;
ListNode* slow = head;
ListNode* fast = head;
fast = fast->next->next;
while(fast!=NULL && fast->next!=NULL){
slow = slow->next;
fast = fast->next->next;
}
ListNode* middle = slow->next;
slow->next = slow->next->next;
delete middle;
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 $\lfloor n / 2 \rfloor^{th}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to `x`.

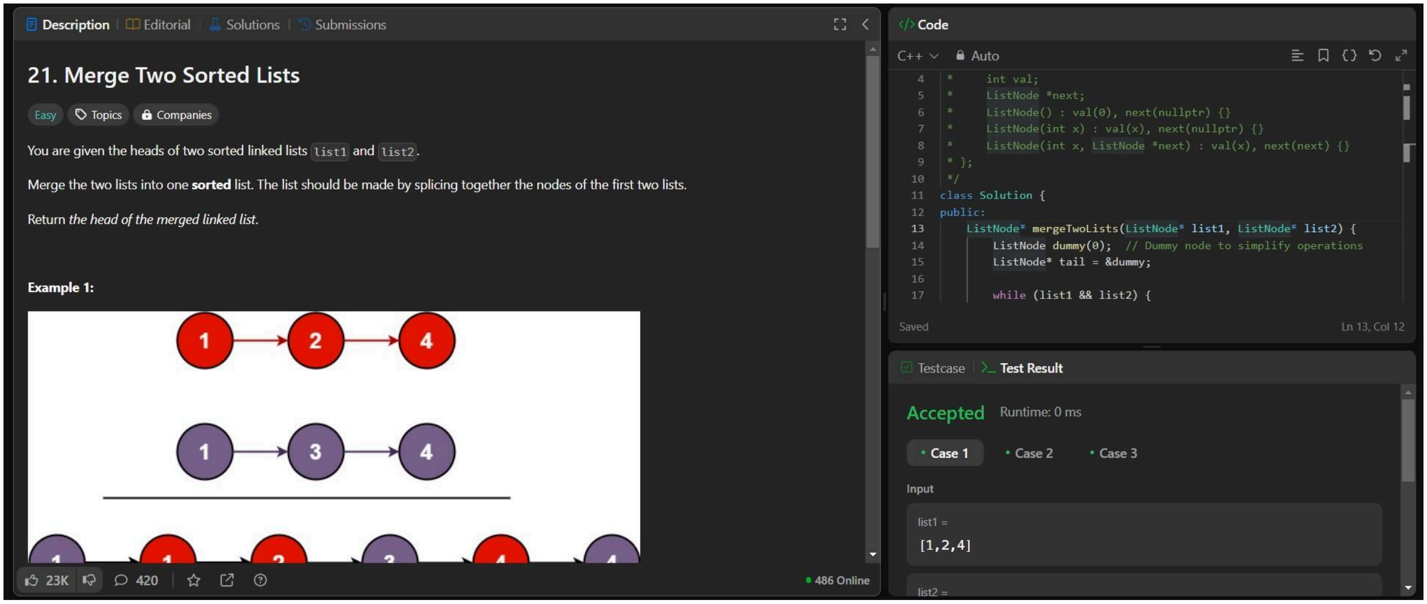
- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

```
 */
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(head== NULL || head->next == NULL) return NULL;
        ListNode* slow = head;
        ListNode* fast = head;
        fast = fast->next->next;
        while(fast!=NULL && fast->next!=NULL){
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* middle = slow->next;
        slow->next = slow->next->next;
        delete middle;
        return head;
    }
};
```

Question-5: Merge two sorted linked 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*.

Answer:class Solution { public:

```
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0); // Dummy node to simplify operations
    ListNode* tail = &dummy;
    while (list1 && list2) {
      if (list1->val < list2->val) {
        tail->next = list1;
        list1 = list1->next;
      } else {
        tail->next  =  list2;
        list2             =
        list2->next;
      }
      tail = tail->next;
    }

    // Append remaining nodes
    tail->next = list1 ? list1 : list2;

    return dummy.next;
    }
};
```



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:

```
 *     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* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);  // Dummy node to simplify operations
        ListNode* tail = &dummy;

        while (list1 && list2) {
```

Accepted Runtime: 0 ms

Input

list1 =
[1,2,4]

list2 =

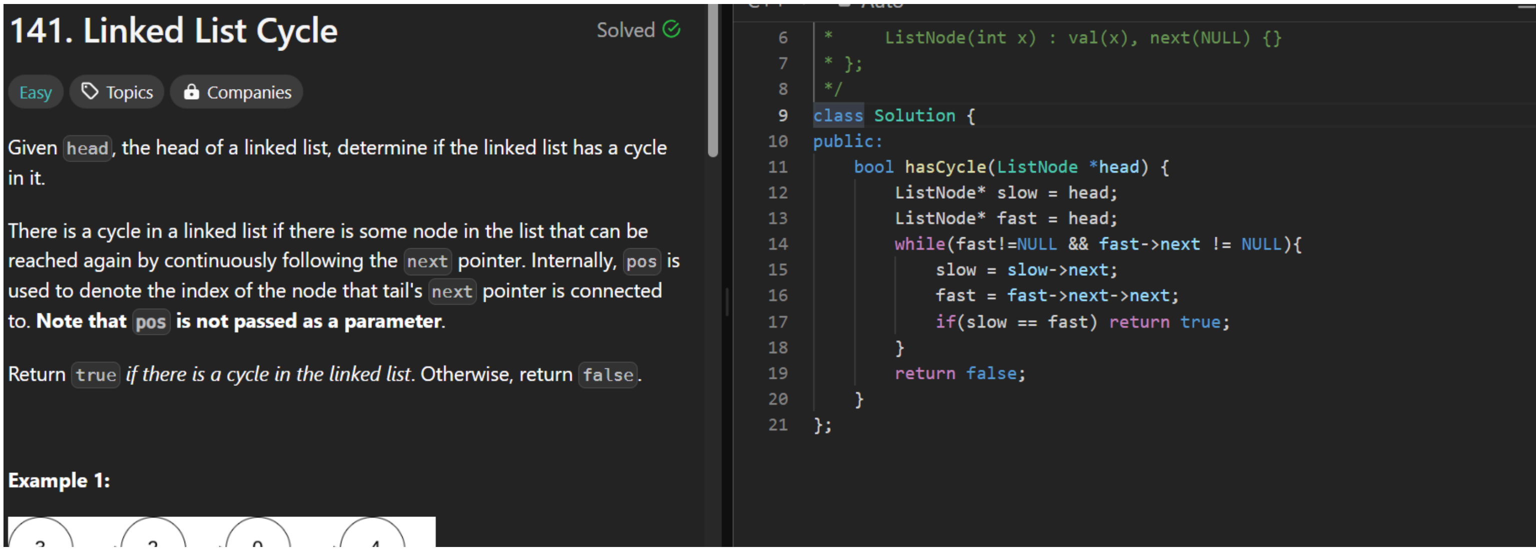
Question-6: Detect a cycle in a linked list:

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.

Answer:

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



141. Linked List Cycle

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`.

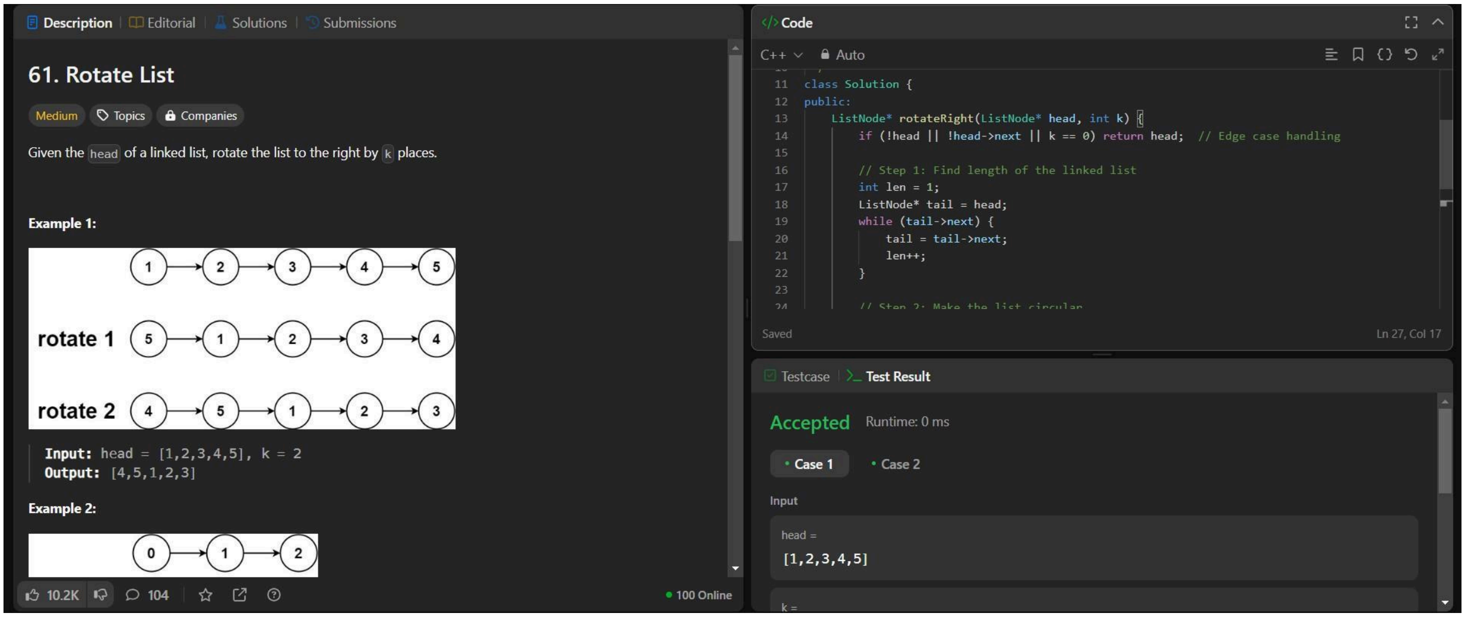
Example 1:

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

Question-7: Rotate List: Given the head of a linked list, rotate the list to the right by k places.

Answer:

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head; // Edge case handling

        // Step 1: Find length of the linked list
        int len = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            len++;
        }

        // Step 2: Make the list circular
        tail->next = head;

        // Step 3: Compute the new tail position
        k = k % len; // Optimize k (rotation beyond length is redundant)
        int stepsToNewHead = len - k; // Find the new head position
        ListNode* newTail = head;

        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail->next;
        }

        // Step 4: Break the cycle and set the new head
        head = newTail->next;
        newTail->next = nullptr;

        return head;

    }
};
```



61. Rotate List

Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:

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

Example 2:

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;  // Edge case handling

        // Step 1: Find length of the linked list
        int len = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            len++;
        }

        // Step 2: Make the list circular
```

Accepted Runtime: 0 ms

Input

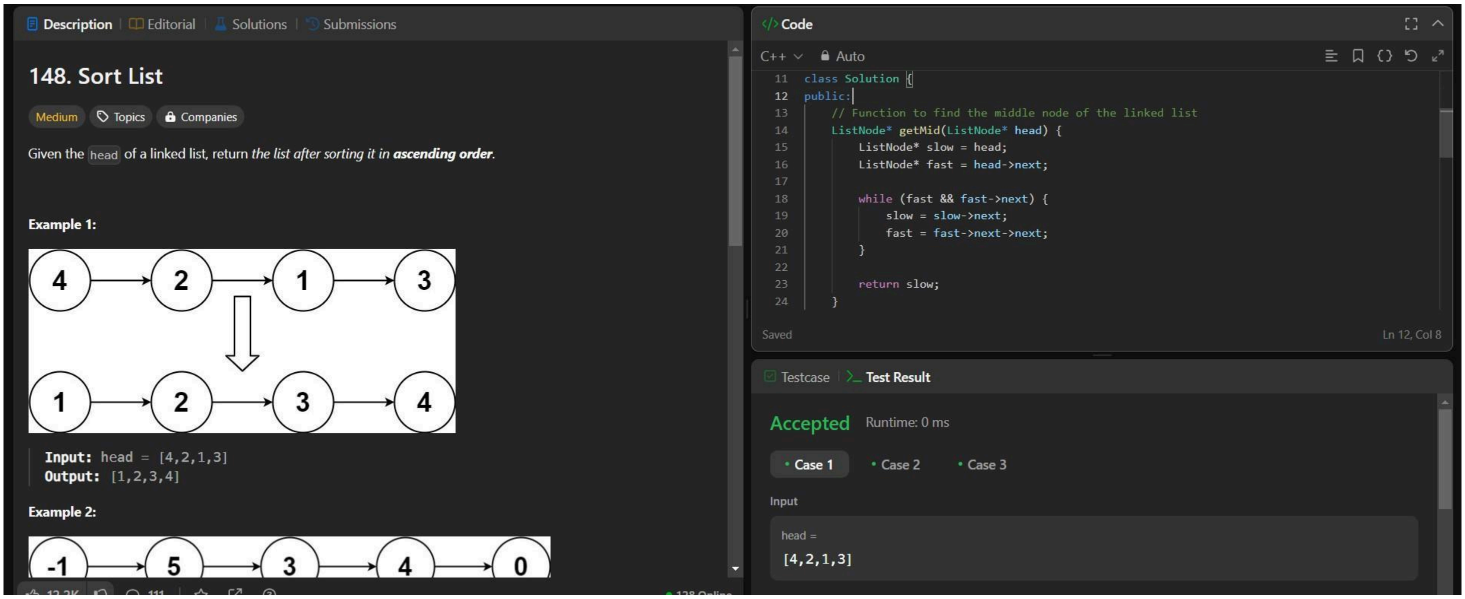
head =
[1,2,3,4,5]

k =

Question-8: Sort List: Given the head of a linked list, return *the list after sorting it* in **ascending order**.

Answer:

```
class Solution {
public:
    // Function to find the middle node of the linked list
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        return slow;
    }

    // Function to merge two sorted lists
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
while (l1 && l2) {
            if (l1->val < l2->val) {
                tail->next = l1;
                l1 = l1->next;
            } else {
                tail->next = l2;
                l2 = l2->next;
            }
            tail = tail->next;
        }

        if (l1) tail->next = l1;
        if (l2) tail->next = l2;

        return dummy.next;
    }

    // Function to sort the linked list using Merge Sort
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        ListNode* mid = getMid(head);
        ListNode* rightHead = mid->next;
        mid->next = nullptr;

        ListNode* left = sortList(head);
        ListNode* right = sortList(rightHead);

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



Description | Editorial | Solutions | Submissions

148. Sort List

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

Example 2:

Code

C++ | Auto

```
class Solution {
public:
    // Function to find the middle node of the linked list
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        return slow;
    }
```

Saved | Ln 12, Col 8

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

head =
[4,2,1,3]

Question-9: Merge k sorted lists:

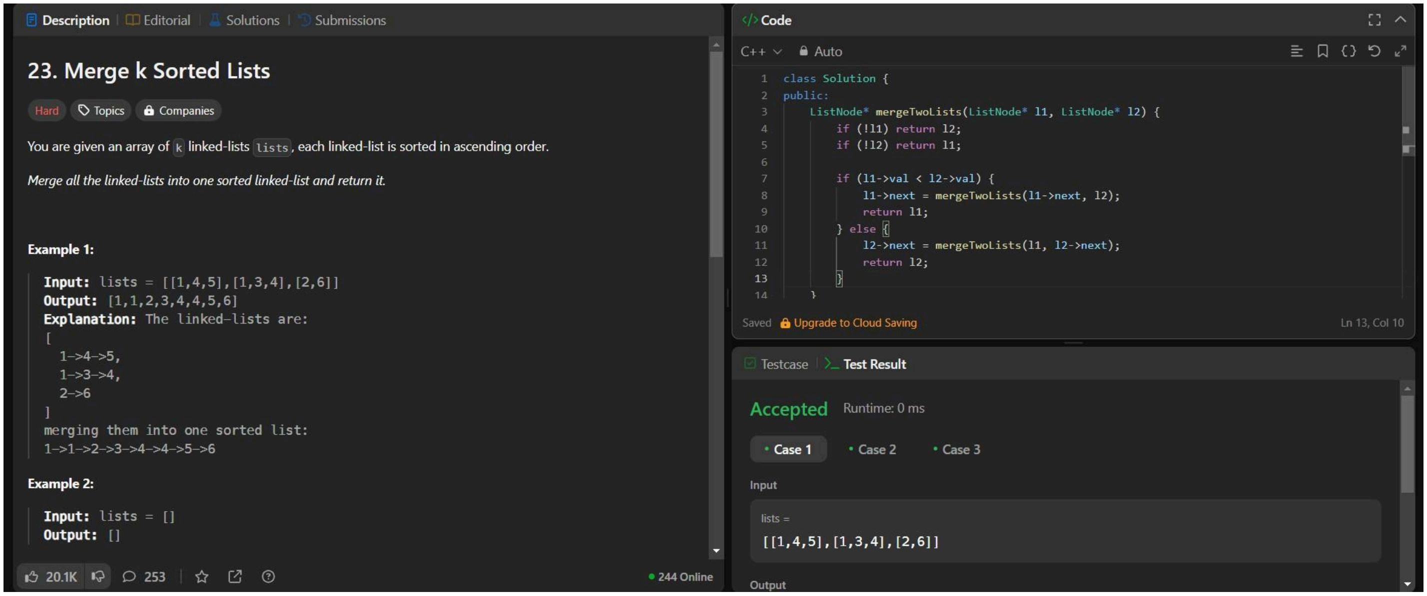
You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

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

Answer:

```
class Solution {
public:
   ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
      if (!l1) return l2;
      if (!l2) return l1;


      if (l1->val < l2->val) {
         l1->next = mergeTwoLists(l1->next, l2);
         return l1;
      } else {
         l2->next = mergeTwoLists(l1, l2->next);
         return l2;
      }
   }

   ListNode* mergeKLists(vector<ListNode*>& lists) {
      if (lists.empty()) return nullptr;
      int n = lists.size();
```

```
        while (n > 1) {int

        j=0;

          for (int i = 0; i < n / 2; i++) {
            lists[i] = mergeTwoLists(lists[i], lists[n - i - 1]);
          }
          n = (n + 1) / 2; // Reduce the number of lists
        }

        return lists[0];
      }
    };
```



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: []
```

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }
```

Accepted Runtime: 0 ms

Input

lists =
[[1,4,5],[1,3,4],[2,6]]

Output