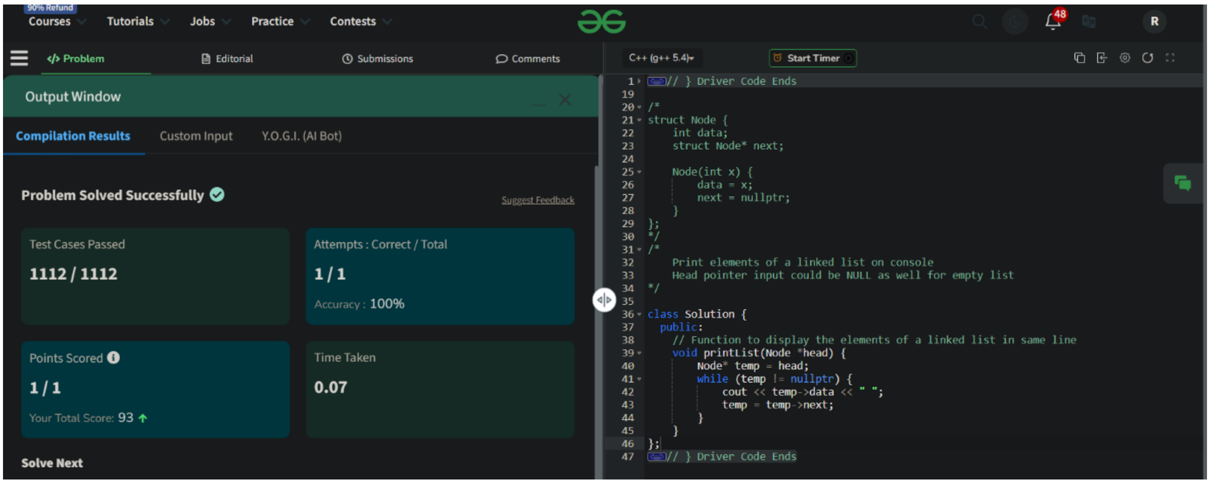
# ASSIGNMENT-3

## Q1) Print Linked List

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



Problem Solved Successfully

Test Cases Passed
1112 / 1112

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

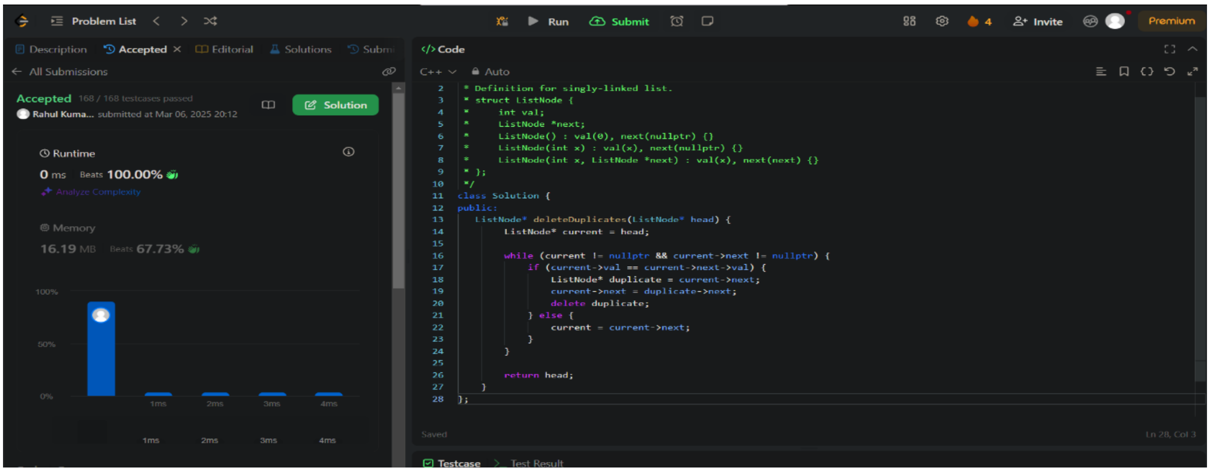
Points Scored
1 / 1
Your Total Score: 93

Time Taken
0.07

```
// } 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) {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
    }
};
// } Driver Code Ends
```

## Q2) Remove duplicates from a sorted list

```
/**
 * 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* current = head;

        while (current != nullptr && current->next != nullptr) {
            if (current->val == current->next->val) {
                ListNode* duplicate = current->next;
                current->next = duplicate->next;
                delete duplicate;
            } else {
                current = current->next;
            }
        }

        return head;
    }
};
```



Accepted 168 / 168 testcases passed

Rahul Kuma... submitted at Mar 06, 2025 20:12

Runtime

0 ms | Beats 100.00%

Analyze Complexity

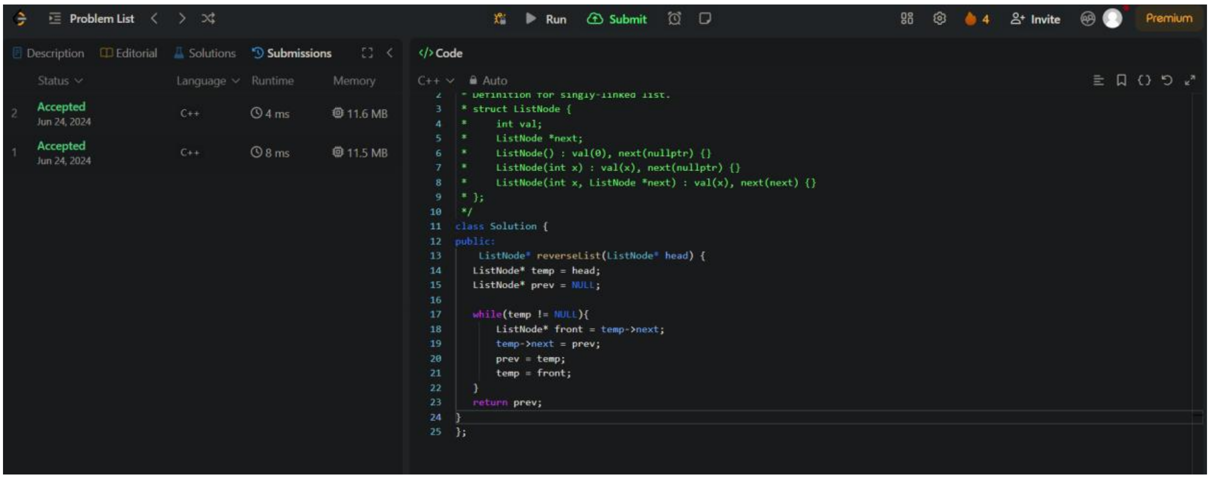
Memory

16.19 MB | Beats 67.73%

```
 * 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* current = head;

        while (current != nullptr && current->next != nullptr) {
            if (current->val == current->next->val) {
                ListNode* duplicate = current->next;
                current->next = duplicate->next;
                delete duplicate;
            } else {
                current = current->next;
            }
        }

        return head;
    }
};
```

**Q3)Reverse a linked list**

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

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

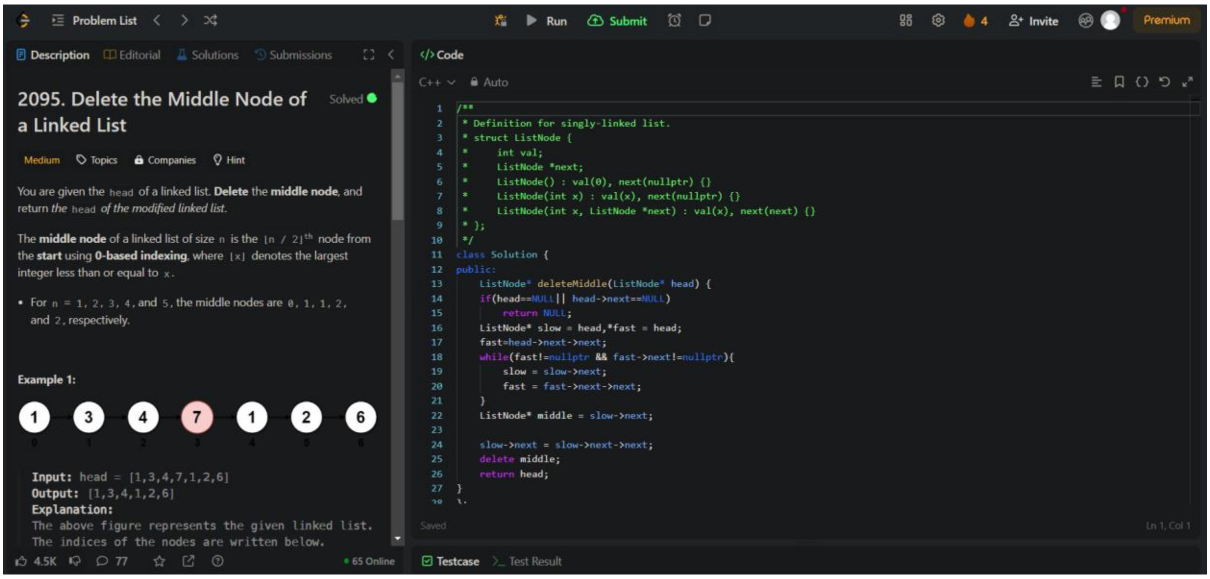


| | Status | Language | Runtime | Memory |
|---|---|---|---|---|
| 2 | Accepted<br>Jun 24, 2024 | C++ | 4 ms | 11.6 MB |
| 1 | Accepted<br>Jun 24, 2024 | C++ | 8 ms | 11.5 MB |

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

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

**Q4) Delete middle node of a list**

```
/**
 * 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* deleteMiddle(ListNode* head) {
    if(head==NULL|| head->next==NULL)
        return NULL;
    ListNode* slow = head,*fast = head;
    fast=head->next->next;
    while(fast!=nullptr && fast->next!=nullptr){
        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 `⌊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:**

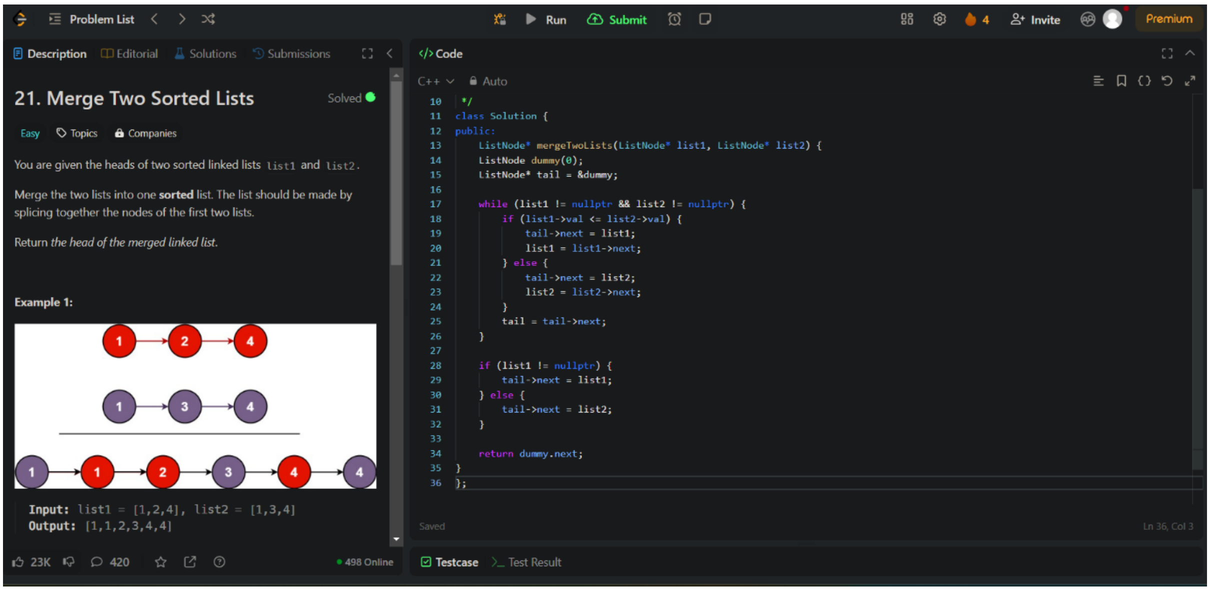
```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list.
The indices of the nodes are written below.
```

```
/**
 * 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* deleteMiddle(ListNode* head) {
    if(head==NULL|| head->next==NULL)
        return NULL;
    ListNode* slow = head,*fast = head;
    fast=head->next->next;
    while(fast!=nullptr && fast->next!=nullptr){
        slow = slow->next;
        fast = fast->next->next;
    }
    ListNode* middle = slow->next;

    slow->next = slow->next->next;
    delete middle;
    return head;
}
};
```

## Q5) Merge two sorted linked lists

```
/**
 * 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* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (list1 != nullptr && list2 != nullptr) {
        if (list1->val <= list2->val) {
            tail->next = list1;
            list1 = list1->next;
        } else {
            tail->next = list2;
            list2 = list2->next;
        }
        tail = tail->next;
    }

    if (list1 != nullptr) {
        tail->next = list1;
    } else {
        tail->next = list2;
    }

    return dummy.next;
}
};
```



21. Merge Two Sorted Lists

Solved

Easy

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.*

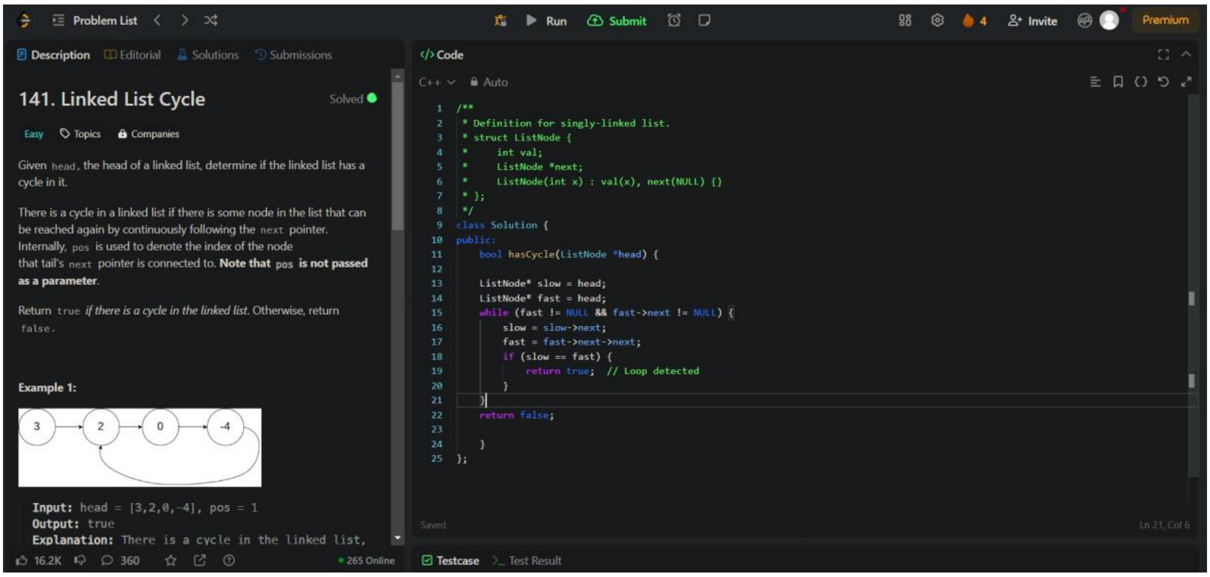
**Example 1:**

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

```
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (list1 != nullptr && list2 != nullptr) {
        if (list1->val <= list2->val) {
            tail->next = list1;
            list1 = list1->next;
        } else {
            tail->next = list2;
            list2 = list2->next;
        }
        tail = tail->next;
    }

    if (list1 != nullptr) {
        tail->next = list1;
    } else {
        tail->next = list2;
    }

    return dummy.next;
}
};
```

## Q6) Detect a cycle in a linked list

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     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;  // Loop detected
     }
  }
  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:**

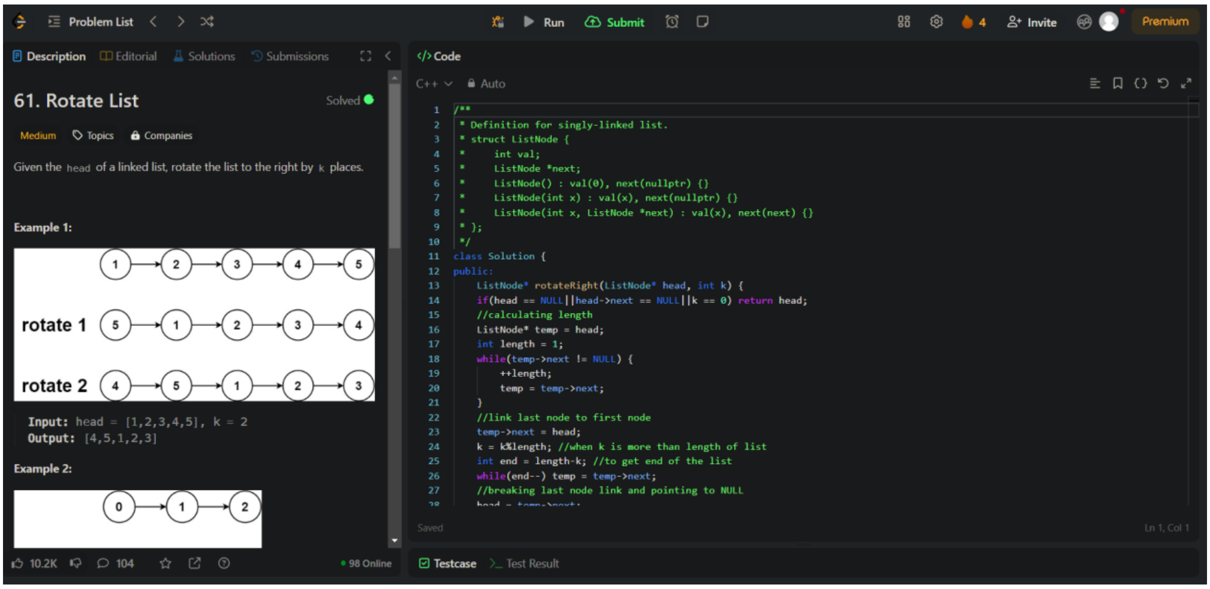
```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list,
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     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;  // Loop detected
        }
    }
    return false;

    }
};
```

## Q7) Rotate a list

```
/**
 * 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* rotateRight(ListNode* head, int k) {
```

```
    if(head == NULL||head->next == NULL||k == 0) return head;
    //calculating length
    ListNode* temp = head;
    int length = 1;
    while(temp->next != NULL) {
        ++length;
        temp = temp->next;
    }
    //link last node to first node
    temp->next = head;
    k = k%length; //when k is more than length of list
    int end = length-k; //to get end of the list
    while(end--) temp = temp->next;
    //breaking last node link and pointing to NULL
    head = temp->next;
    temp->next = NULL;

    return head;

    }
};
```



61. Rotate List

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

Example 1:

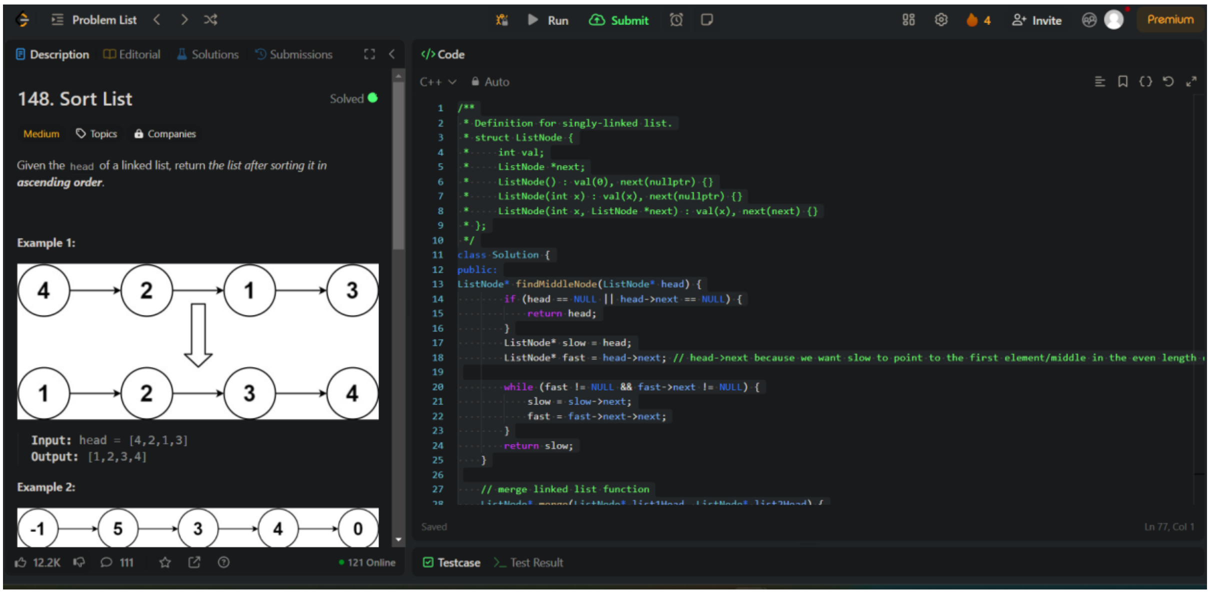
```
Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]
```

Example 2:

```
/**
 * 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* rotateRight(ListNode* head, int k) {
    if(head == NULL||head->next == NULL||k == 0) return head;
    //calculating length
    ListNode* temp = head;
    int length = 1;
    while(temp->next != NULL) {
        ++length;
        temp = temp->next;
    }
    //link last node to first node
    temp->next = head;
    k = k%length; //when k is more than length of list
    int end = length-k; //to get end of the list
    while(end--) temp = temp->next;
    //breaking last node link and pointing to NULL
    head = temp->next;
```

## Q9) Sort List

```
/**
 * 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* findMiddleNode(ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    ListNode* slow = head;
    ListNode* fast = head->next; // head->next because we want slow to point to the first element/middle in the even length case
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

// merge linked list function
ListNode* merge(ListNode* list1Head, ListNode* list2Head) {
    ListNode* dummyNode = new ListNode(-1); // can be any value
    ListNode* temp = dummyNode;

    while (list1Head != NULL && list2Head != NULL) {
        if (list1Head->val <= list2Head->val) {
            temp->next = list1Head;
            temp = list1Head;
            list1Head = list1Head->next;
        } else {
            temp->next = list2Head;
            temp = list2Head;
            list2Head = list2Head->next;
        }
    }

    // if list1 still has elements left
    while (list1Head != NULL) {
        temp->next = list1Head;
        temp = list1Head;
        list1Head = list1Head->next;
    }

    // if list2 still has elements left
    while (list2Head != NULL) {
        temp->next = list2Head;
        temp = list2Head;
        list2Head = list2Head->next;
    }
    return dummyNode->next;
}

// MergeSort recursive
ListNode* sortList(ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    ListNode* mid = findMiddleNode(head);
    ListNode* leftHead = head;
    ListNode* rightHead = mid->next;
    mid->next = NULL; // Disconnect the left and right halves

    leftHead = sortList(leftHead);
    rightHead = sortList(rightHead);
    return merge(leftHead, rightHead);
}
};
```



## 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:**

```
/**
 * 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* findMiddleNode(ListNode* head) {
        if (head == NULL || head->next == NULL) {
            return head;
        }
        ListNode* slow = head;
        ListNode* fast = head->next; // head->next because we want slow to point to the first element/middle in the even length

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

    // merge linked list function
    ListNode* merge(ListNode* list1Head, ListNode* list2Head) {
```