AP Assignment-2

Ques 1

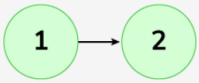
Print Linked List

Difficulty: **Basic** Accuracy: **60.71%** Submissions: **142K+** Points: **1**

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

Examples:

Input: LinkedList : 1 -> 2



Output: 1 2

Explanation: The linked list contains two elements 1 and 2.The elements are printed in a single line.

Input: Linked List : 49 -> 10 -> 30

Output Window

Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed	Attempts : Correct / Total
1112 / 1112	**1 / 1**
	Accuracy : 100%
Points Scored	Time Taken
1 / 1	**0.1**
Your Total Score: 63 ↑	

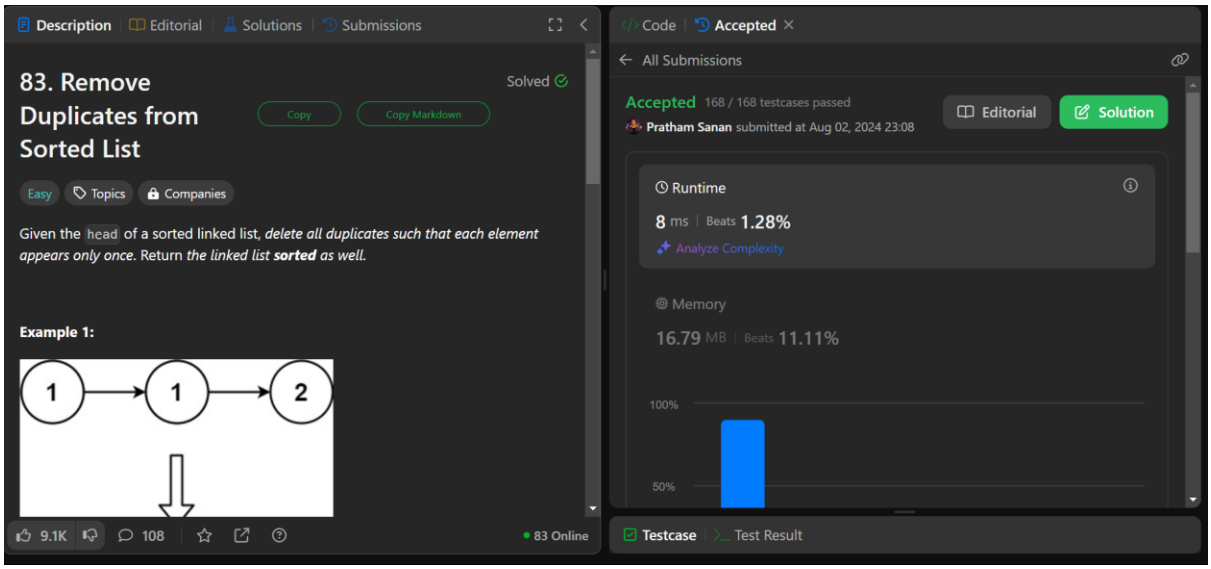
Solve Next

Count Linked List Nodes | Delete Alternate Nodes

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

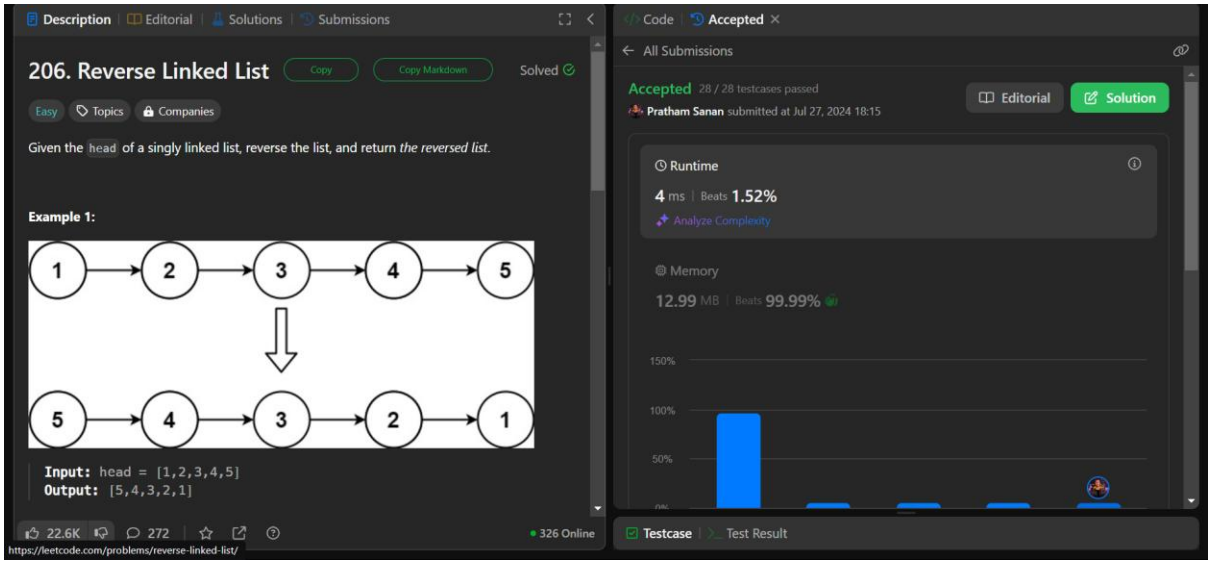
Custom Input | Compile & Run | Submit

Ques 2



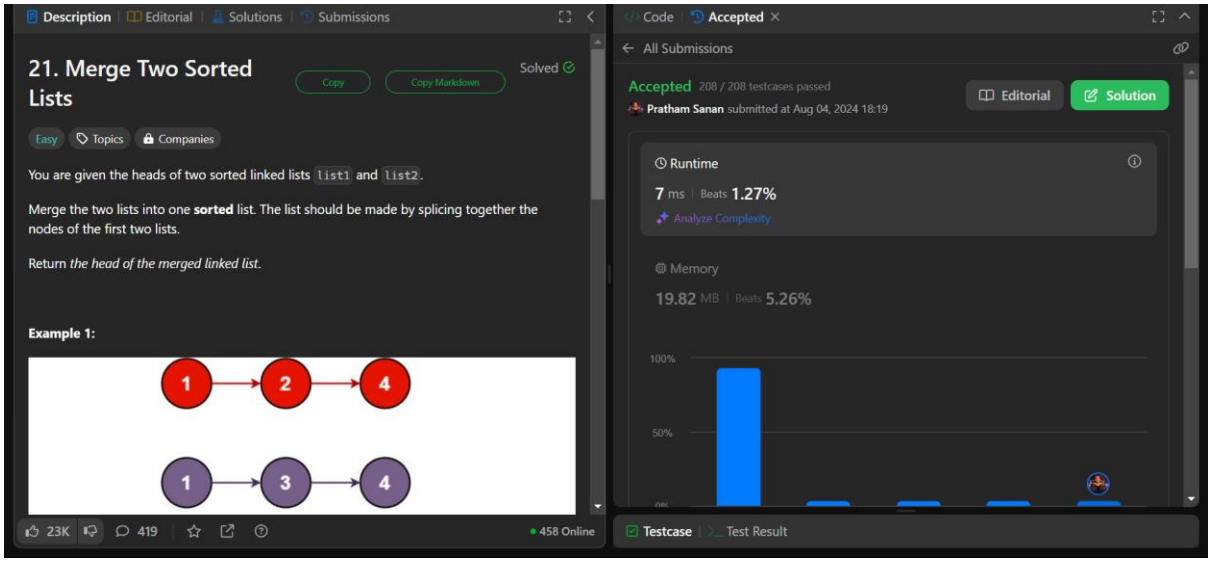

```
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head == NULL){
            return NULL;
        }

        ListNode* curr = head;

        while(curr !=NULL){

            if((curr -> next != NULL) && (curr -> val == curr -> nex
                ListNode* next_next = curr -> next -> next;
                ListNode* nodeToDelete = curr -> next;
                delete(nodeToDelete);
                curr -> next = next_next;
            }
            else{
                curr = curr -> next;
            }
        }
        return head;
    }
```

Ques 3



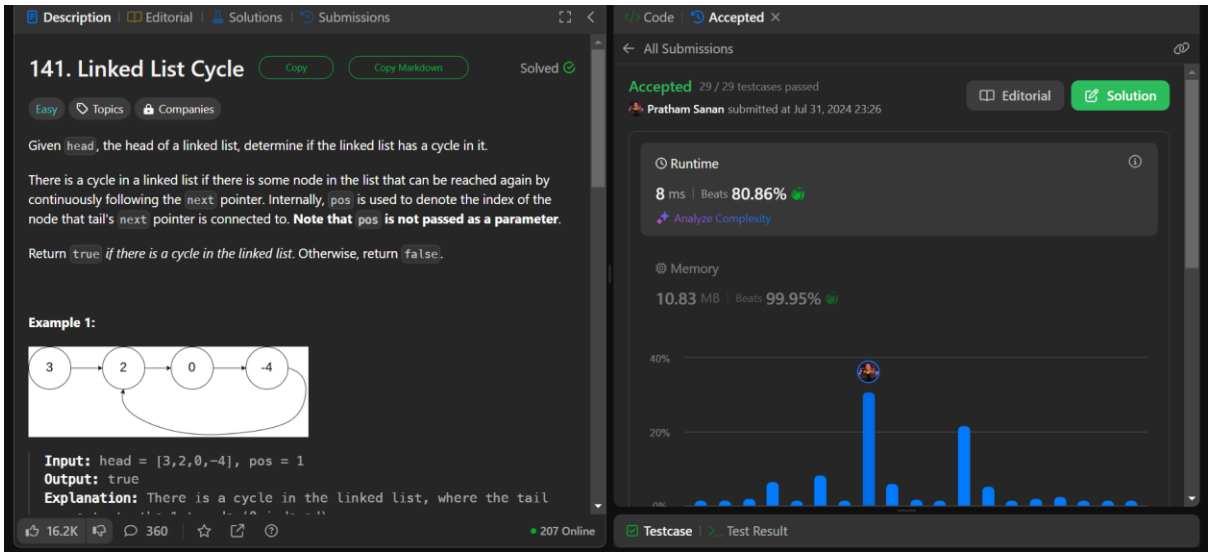

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(head == NULL || head -> next == NULL){
            return head;
        }

        ListNode* prev = NULL;
        ListNode* curr = head;
        ListNode* frwd = NULL;

        while(curr != NULL){
            frwd = curr -> next;
            curr -> next = prev;
            prev = curr;
            curr = frwd;
        }
        return prev;
    }
};
```

Ques 5



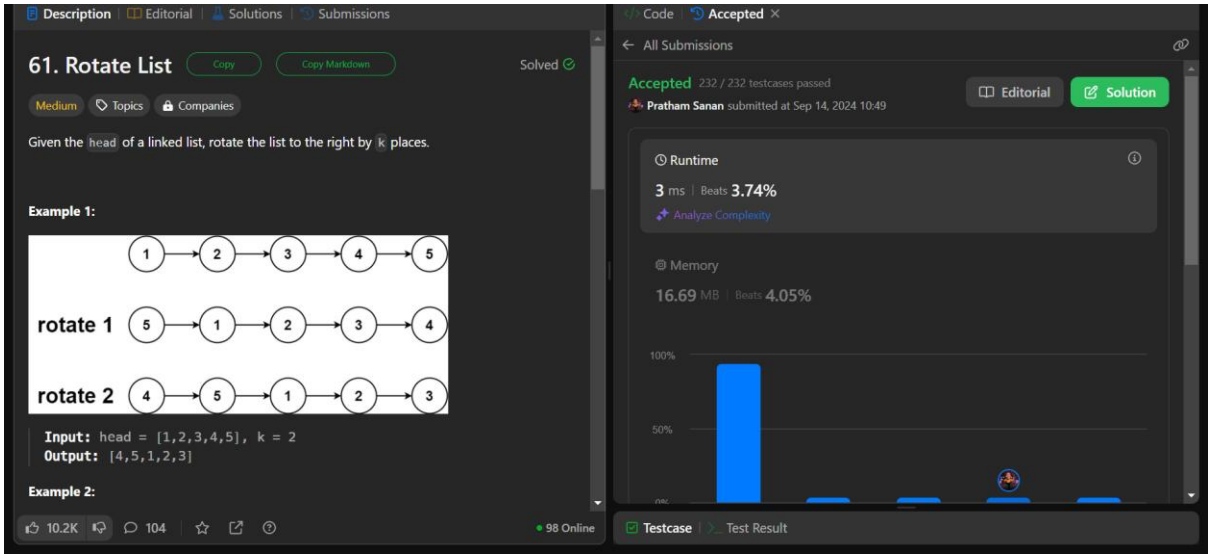

```cpp
ListNode* solve(ListNode* list1, ListNode* list2){

    if(list1 -> next == NULL){
        list1 -> next = list2;
        return list1;
    }

    ListNode* curr1 = list1;
    ListNode* next1 = curr1 -> next;
    ListNode* curr2 = list2;
    ListNode* next2 = curr2 -> next;

    while(next1 != NULL && curr2 != NULL){
        if((curr2 -> val >= curr1 -> val) && (curr2 -> val <= next
            curr1 -> next = curr2;
            next2 = curr2 -> next;
            curr2 -> next = next1;

            curr1 = curr2;
            curr2 = next2;
        }
        else{
            // curr1 aur next1 ko aage badana hai
            curr1 = next1;
            next1 = next1 -> next;
            if(next1 == NULL){
                curr1 -> next = curr2;
                return list1;
            }
        }
    }
    return list1;
}

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

        if(list1 -> val <= list2 -> val){
            return solve(list1, list2);
        }
        else{
            return solve(list2, list1);
        }
    }
};
```

Ques 6



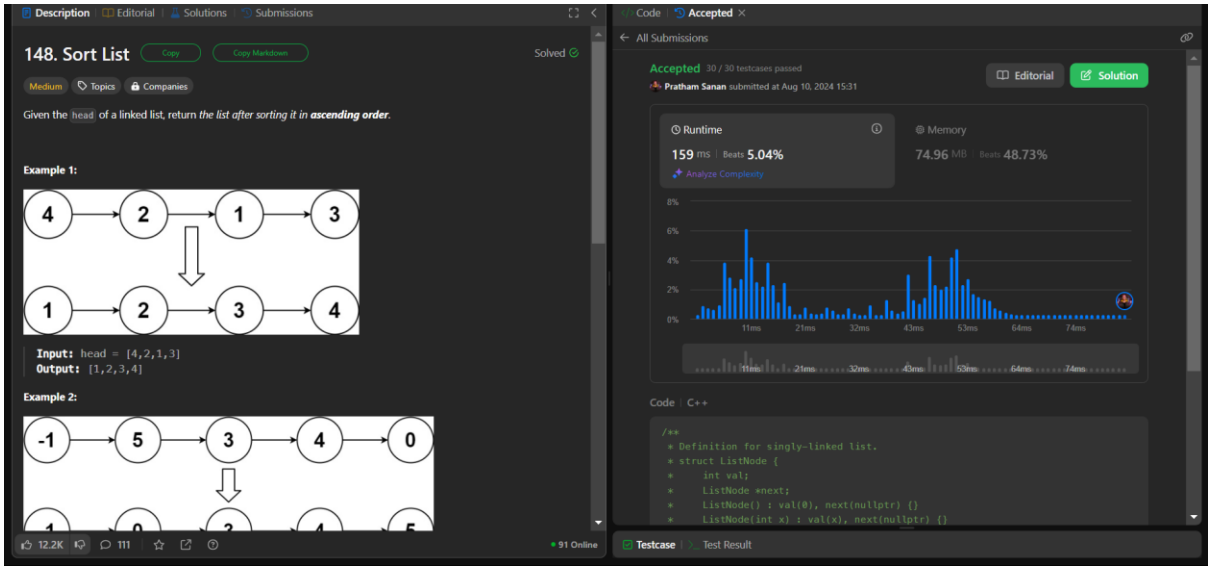

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(head == NULL){
            return NULL;
        }

        ListNode* slow = head;
        ListNode* fast = head;

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

            slow = slow->next;

            if(slow == fast){
                return slow;
            }
        }

        return NULL;
```

Ques 7




```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {

        if (head == NULL || head->next == NULL) {
            return head;
        }

        int lgth=1;
        ListNode* temp = head;
        while(temp->next){
            temp = temp->next;
            lgth++;
        }

        temp->next = head;

        k = k%lgth;
        int stepsToNewHead = lgth-k;
        ListNode* newTail = head;
        for(int i=1;i<stepsToNewHead;i++){
            newTail = newTail->next;
        }

        ListNode* newHead = newTail->next;
        newTail->next = NULL;

        return newHead;
    }
};
```

Ques 8




```cpp
ListNode* findMid(ListNode* head){
    ListNode* slow = head;
    ListNode* fast = head->next;

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

ListNode* merge(ListNode* left, ListNode* right){
    if(left == NULL){
        return right;
    }
    if(right == NULL){
        return left;
    }

    ListNode* ans = new ListNode(-1);
    ListNode* temp = ans;

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

    while(left != NULL){
        temp -> next=left;
        temp = left;
        left = left->next;
    }

    while(right != NULL){
        temp->next = right;
        temp = right;
        right = right->next;
    }

    ans = ans->next;
    return ans;
}

class Solution {
public:
    ListNode* sortList(ListNode* head) {
        //base case
        if(head == NULL || head->next == NULL){
            return head;
        }

        ListNode* mid = findMid(head);

        ListNode* left = head;
        ListNode* right = mid->next;
        mid->next = NULL;

        left = sortList(left);
        right = sortList(right);

        ListNode* result = merge(left, right);

        return result;
    }
};
```