



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Assignment-2

Student Name: Adarsh Tiwari

UID: 22BCS11352

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;
        }
    }
};
```

The screenshot displays a C++ development environment. On the left, the 'Output Window' shows 'Compilation Completed' for 'Case 1'. The 'Input' field contains '1 2', and the 'Your Output' field also shows '1 2', matching the 'Expected Output'. On the right, the code editor shows a C++ program. It defines a 'Node' struct with 'data' and 'next' pointers, a 'Node' constructor, and a 'Solution' class with a 'printList' function. The 'printList' function iterates through the linked list and prints each node's data followed by a space. The code is commented to show it's a driver code for testing the 'printList' function.

Question-2: Remove Duplicates from Sorted List

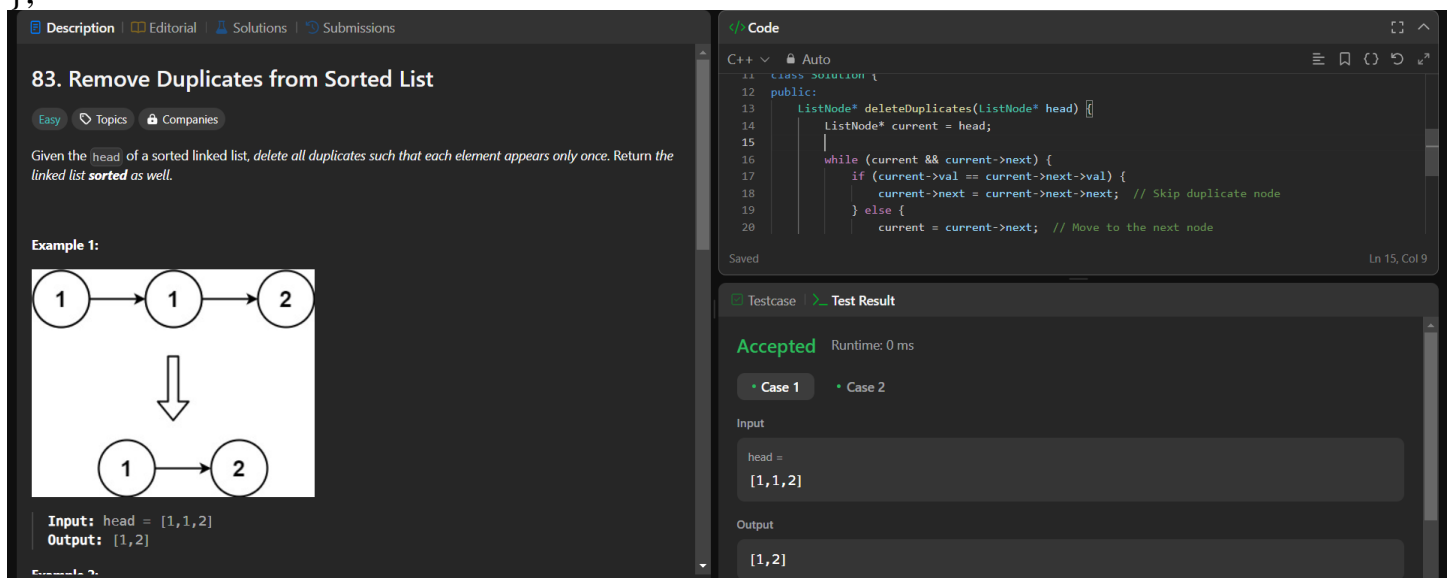
Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list sorted* as well.

Answer:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;

        while (current && current->next) {
            if (current->val == current->next->val) {
                current->next = current->next->next; // Skip duplicate node
            } else {
                current = current->next; // Move to the next node
            }
        }

        return head;
    }
};
```



The screenshot displays a coding interface for the problem "83. Remove Duplicates from Sorted List". On the left, the problem description states: "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." Below this, an example is shown: a linked list with nodes 1, 1, and 2 is transformed into a linked list with nodes 1 and 2. The input is given as "head = [1,1,2]" and the output is "[1,2]". On the right, the C++ code is shown, which is identical to the one provided in the text. The code is enclosed in a class named "Solution" with a public method "deleteDuplicates". The method uses a while loop to traverse the list, skipping duplicate nodes by updating the "next" pointer. The test result section shows "Accepted" with a runtime of 0 ms for Case 1.

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

Answer:

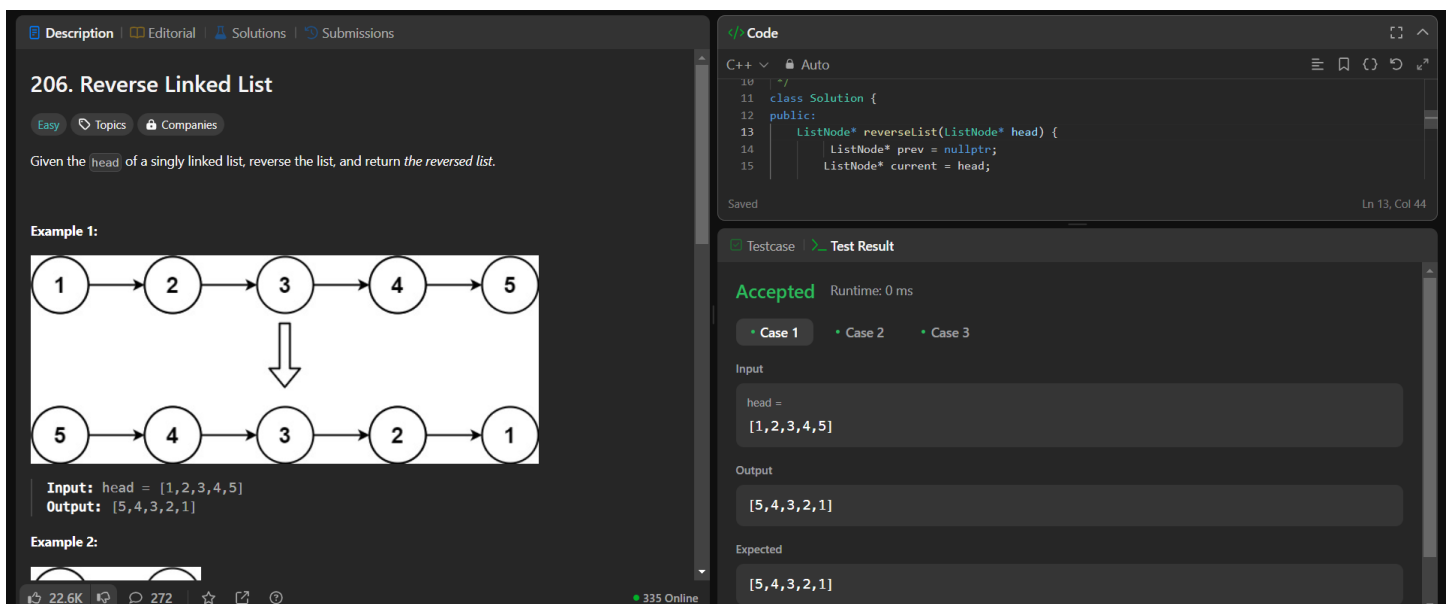
```
class Solution {
public:
```

```
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;

    while (current) {
        ListNode* nextNode = current->next; // Store next node
        current->next = prev;                // Reverse the link
        prev = current;                     // Move prev ahead
        current = nextNode;                 // Move current ahead
    }

    return prev; // New head of reversed list
}

};
```



206. Reverse Linked List

Easy Topics Companies

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:

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

Code:

```
C++
10 //
11 class Solution {
12 public:
13     ListNode* reverseList(ListNode* head) {
14         ListNode* prev = nullptr;
15         ListNode* current = head;
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

Output

[5,4,3,2,1]

Expected

[5,4,3,2,1]

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 $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

Answer:

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
```

```
if (!head || !head->next) return nullptr; // Edge case: If there's only one node, return NULL
```

```
ListNode* slow = head, *fast = head, *prev = nullptr;
```

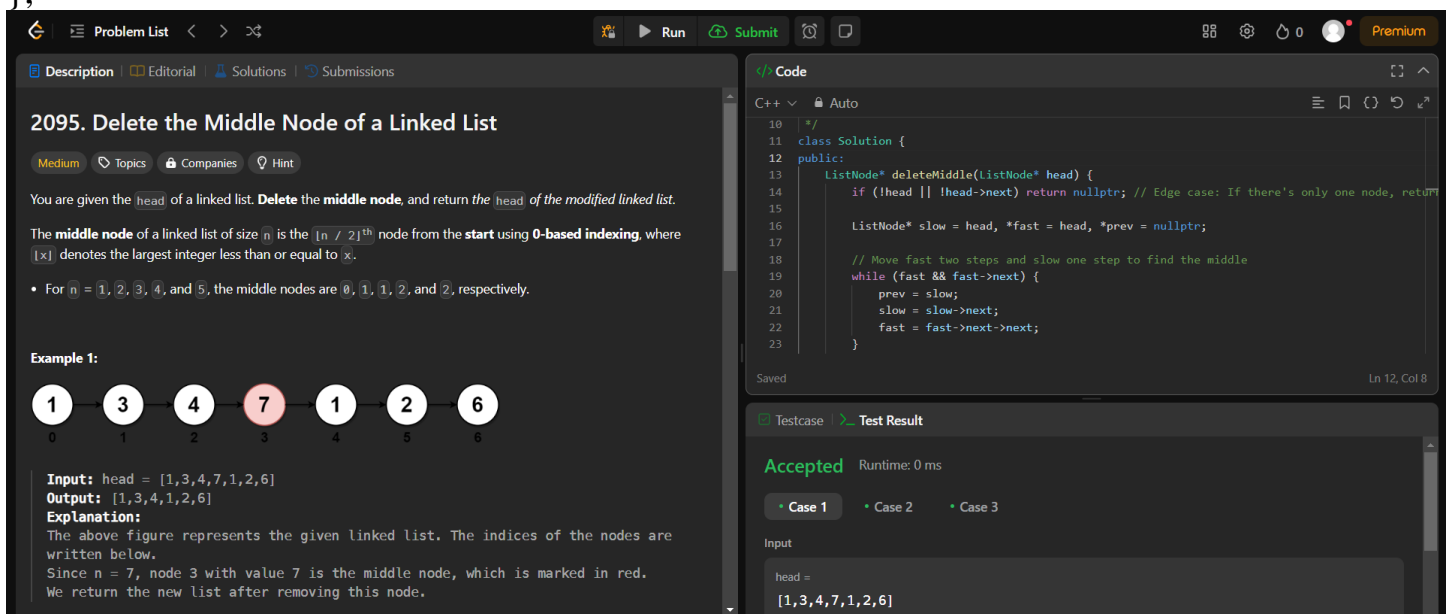
```
// Move fast two steps and slow one step to find the middle
```

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

```
// Remove the middle node
```

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

```
};
```



The screenshot shows a coding problem interface. On the left, the problem description for '2095. Delete the Middle Node of a Linked List' is visible, including a diagram of a linked list with nodes [1, 3, 4, 7, 1, 2, 6] where the node with value 7 is highlighted as the middle node to be deleted. On the right, the C++ code is displayed, implementing the solution using the slow and fast pointer technique. The code includes an edge case check for a single node and a loop to find the middle node. Below the code, the test result shows 'Accepted' with a runtime of 0 ms for Case 1.

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;
    }
};
```

Description
Editorial
Solutions
Submissions

21. Merge Two Sorted Lists

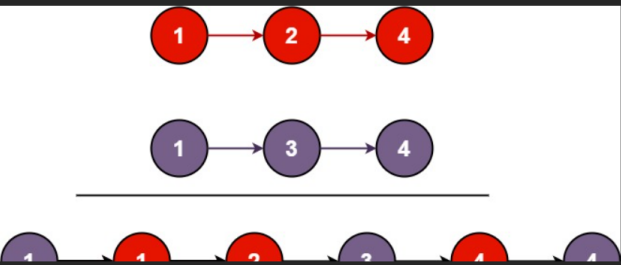
Easy
Topics
Companies

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

```

4  *   int val;
5  *   ListNode *next;
6  *   ListNode() : val(0), next(nullptr) {}
7  *   ListNode(int x) : val(x), next(nullptr) {}
8  *   ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
14         ListNode dummy(0); // Dummy node to simplify operations
15         ListNode* tail = &dummy;
16         while (list1 && list2) {

```

Saved
Ln 13, Col 12

Testcase
Test Result

Accepted
Runtime: 0 ms

Case 1
Case 2
Case 3

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) {
```

```
        if (!head || !head->next) return false; // Edge case: empty or single-node list
```

```
        ListNode* slow = head;
```

```
        ListNode* fast = head;
```

```
        while (fast && fast->next) {
```

```
            slow = slow->next;          // Move slow pointer one step
```

```
            fast = fast->next->next;    // Move fast pointer two steps
```

```
            if (slow == fast) return true; // Cycle detected
```

```
        }
```

```
        return false; // No cycle found
```

```
    }
```

```
};
```

Description

Editorial

Solutions

Submissions

141. Linked List Cycle

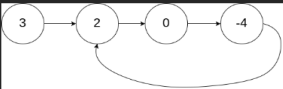
Easy Topics Companies

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:

16.2K 360 246 Online

Code

C++

Auto

```

1  class Solution {
2  public:
3      bool hasCycle(ListNode *head) {
4          if (!head || !head->next) return false; // Edge case: empty or single-node list
5
6          ListNode* slow = head;
7          ListNode* fast = head;
8
9          while (fast && fast->next) {
10             slow = slow->next;          // Move slow pointer one step
11             fast = fast->next->next;    // Move fast pointer two steps
12
13             if (slow == fast) return true; // Cycle detected
14         }
15         return false;
16     }
17 }

```

Saved Ln 21, Col 24

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head =

[3,2,0,-4]

pos =



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++;
        }

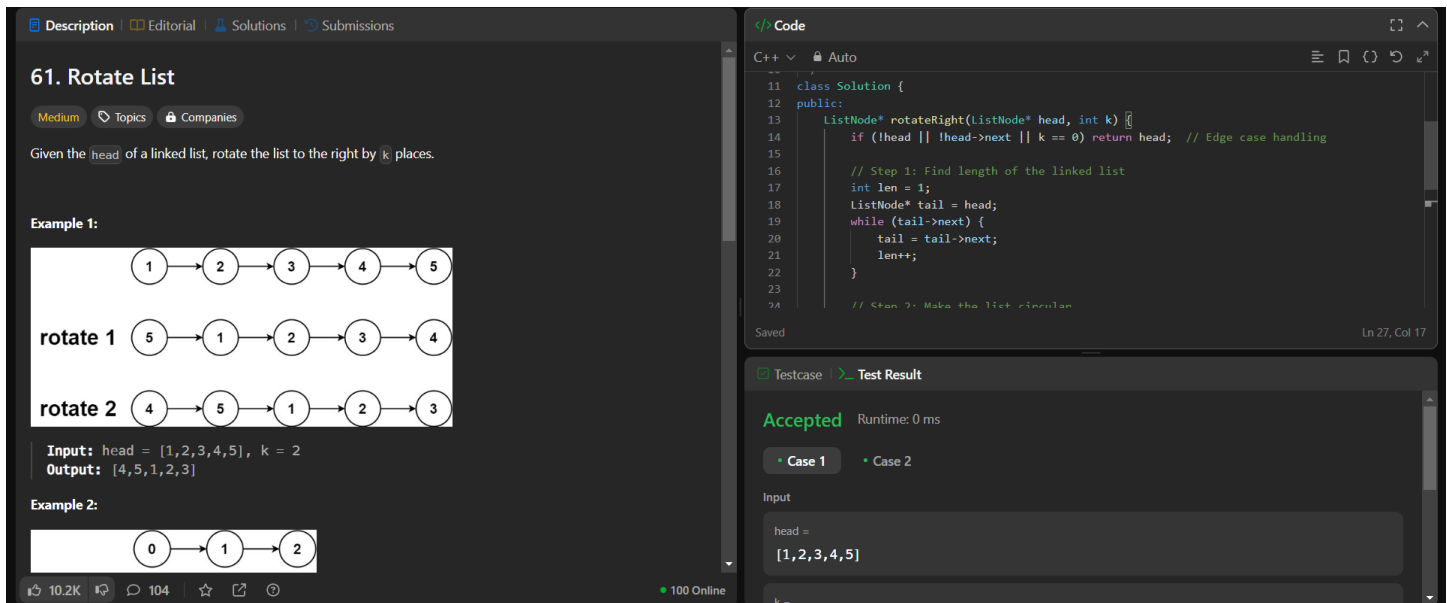
        // Step 2: Make the list circular
        tail->next = head;

        // 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

Medium Topics Companies

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

Example 1:

rotate 1

rotate 2

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

Example 2:

Input: `head = [0,1,2]`

```

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
    }
};

```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

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;
    }
};

```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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;

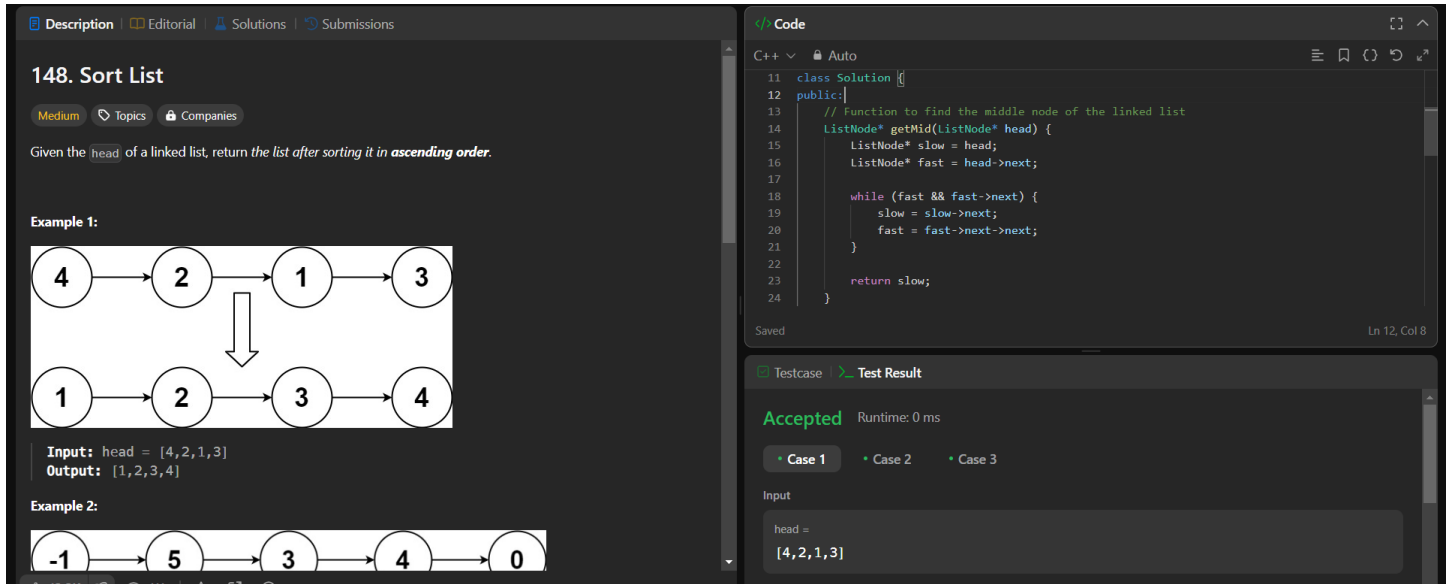
return dummy->next;
}

// 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);
}
};
```



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:

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

```

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;
    }
};

```

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];
}
};
```

Description | Editorial | Solutions | Submissions

23. Merge k Sorted Lists

Hard Topics Companies

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: `[]`

20.1K 253 244 Online

Code

```
1 class Solution {
2 public:
3     ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
4         if (!l1) return l2;
5         if (!l2) return l1;
6
7         if (l1->val < l2->val) {
8             l1->next = mergeTwoLists(l1->next, l2);
9             return l1;
10        } else {
11            l2->next = mergeTwoLists(l1, l2->next);
12            return l2;
13        }
14    }
15 }
```

Saved Upgrade to Cloud Saving Ln 13, Col 10

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

`lists =`

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

Output