NAME - ANSHIKA

UID - 22BCS15677

SECTION - 611

GROUP - “B”

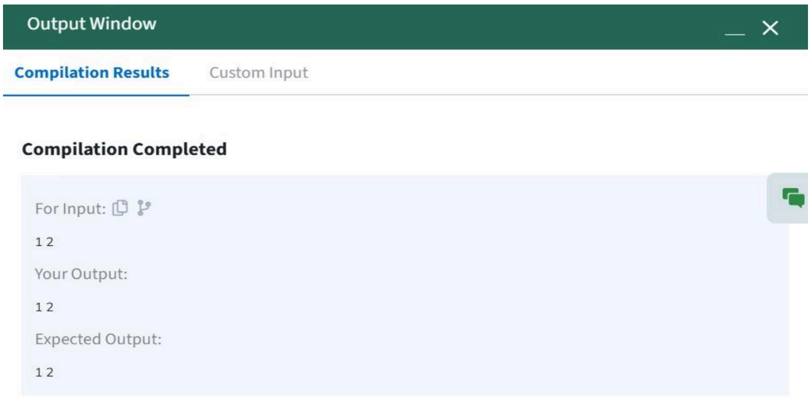
Ques 1. Print linked list

Code -

```
class Solution {
  public:

    void printList(Node *head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
    }
};
```

Output -



Compilation Completed

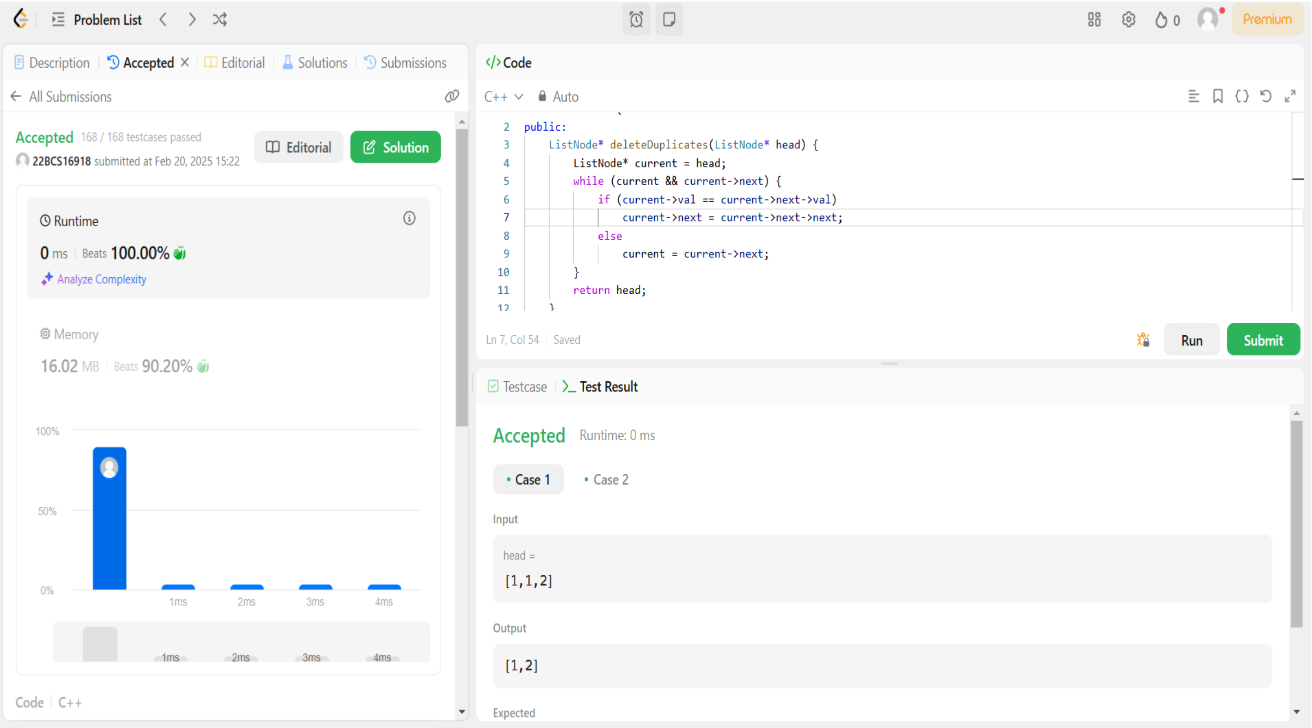
```
For Input:
1 2
Your Output:
1 2
Expected Output:
1 2
```

Ques 2. Remove duplicates from list

Code -

```
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;
            else
                current = current->next;
        }
        return head;
    }
};
```

Output -



Accepted 168 / 168 testcases passed

22BCS16918 submitted at Feb 20, 2025 15:22

0 ms | Beats 100.00%

16.02 MB | Beats 90.20%

```
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;
        while (current && current->next) {
            if (current->val == current->next->val)
                current->next = current->next->next;
            else
                current = current->next;
        }
        return head;
    }
```

Ln 7, Col 54 | Saved

Accepted Runtime: 0 ms

Input

head =

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

Output

```
[1,2]
```

Ques 3. Reverse linked list

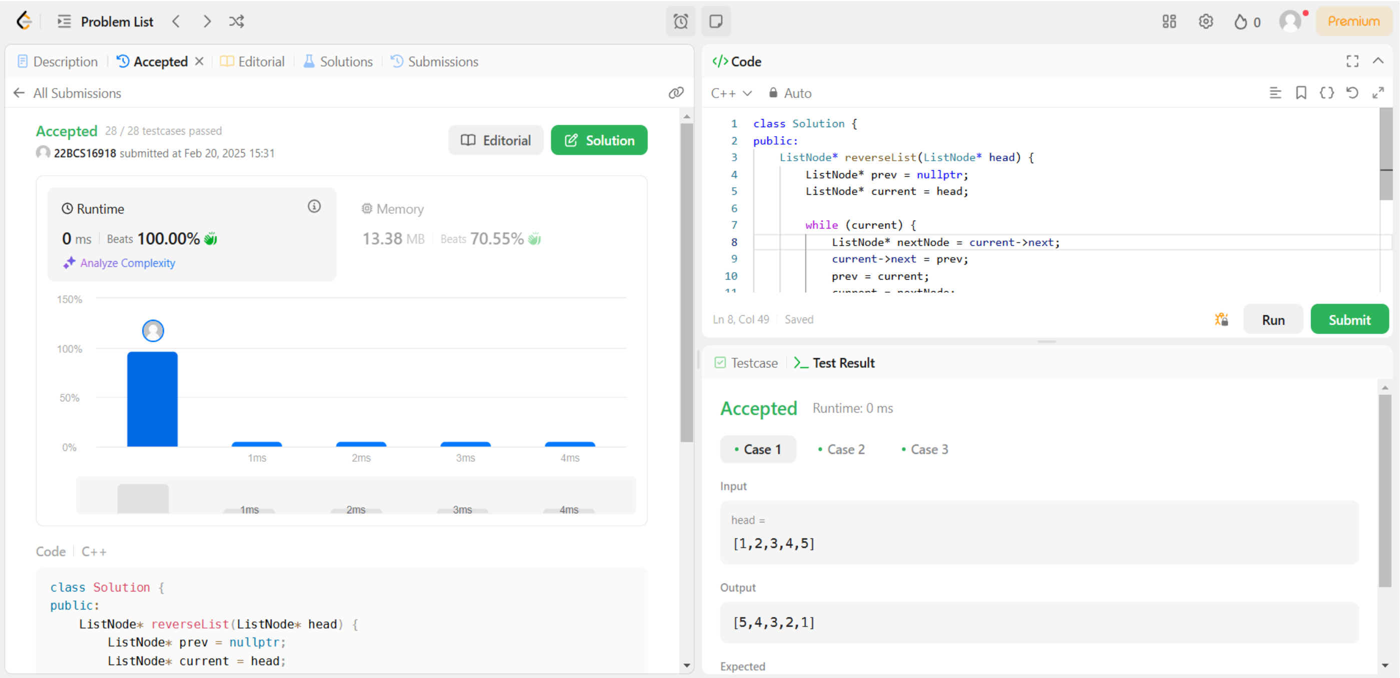
Code -

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;

        while (current) {
            ListNode* nextNode = current->next;
            current->next = prev;
            prev = current;
            current = nextNode;
        }

        return prev;
}
}
```

Output -



Accepted 28 / 28 testcases passed

22BCS16918 submitted at Feb 20, 2025 15:31

0 ms | Beats 100.00%

13.38 MB | Beats 70.55%

Code | C++

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;
```

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;

        while (current) {
            ListNode* nextNode = current->next;
            current->next = prev;
            prev = current;
```

Ln 8, Col 49 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =

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

Output

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

Expected

Ques 4. Delete the Middle Node of a Linked List

Code -

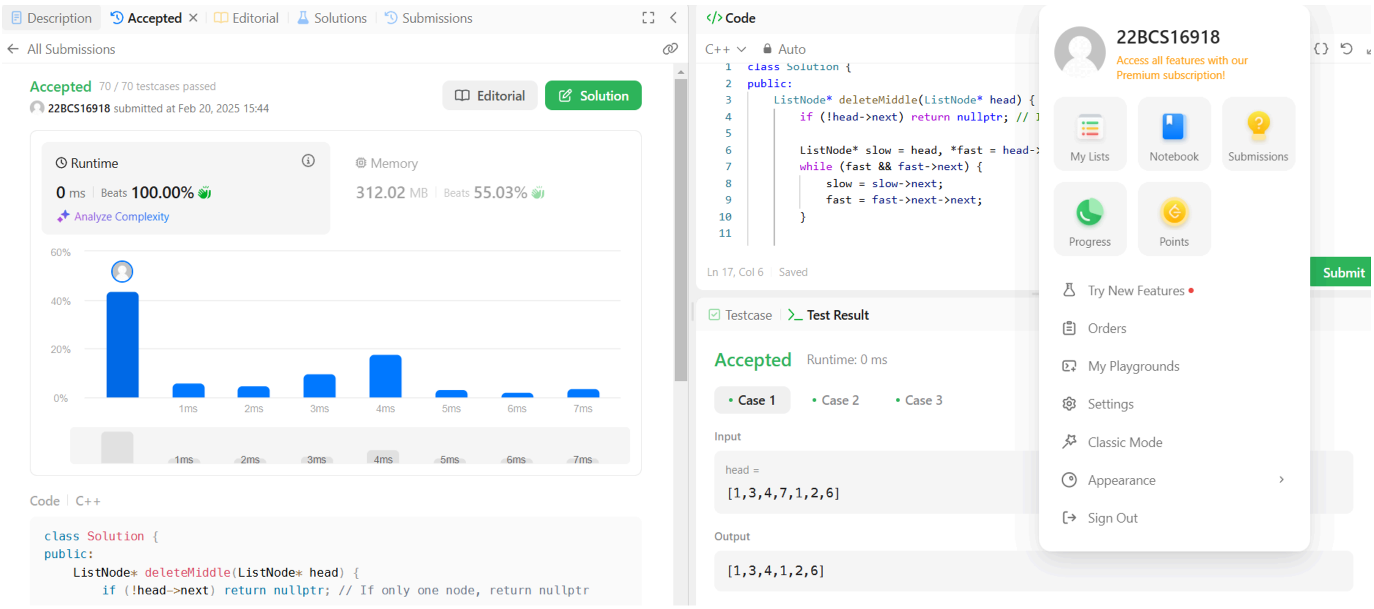
```
 class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head->next) return nullptr; // If only one node, return nullptr

        ListNode* slow = head, *fast = head->next->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        ListNode* temp = slow->next;
        slow->next = slow->next->next; // Skip middle node
        delete temp; // Free memory

        return head;
    }
};
```

Output -



Accepted 70 / 70 testcases passed

22BCS16918 submitted at Feb 20, 2025 15:44

0 ms | Beats 100.00%

312.02 MB | Beats 55.03%

Code | C++

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

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head->next) return nullptr; // I

        ListNode* slow = head, *fast = head->
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

```

Ln 17, Col 6 | Saved

22BCS16918

Access all features with our Premium subscription!

Accepted Runtime: 0 ms

Input

head =

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

Output

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

Ques 5. Merge two sorted list

Code -

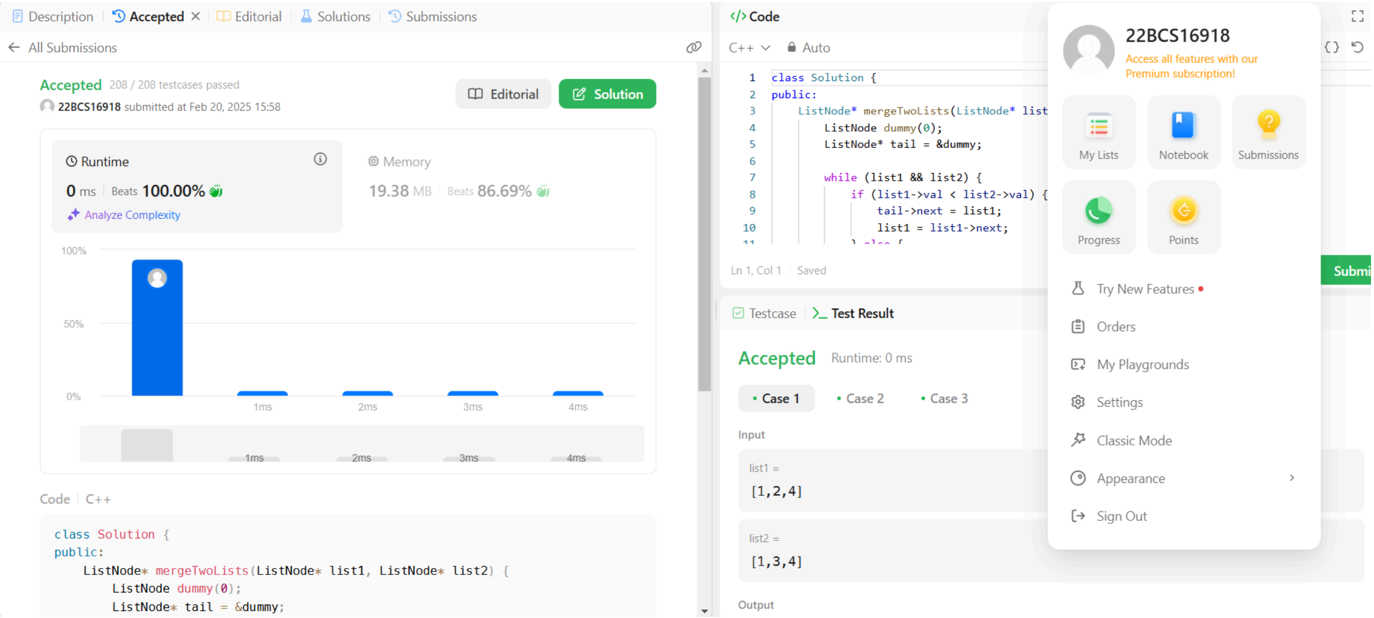
```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;

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

        tail->next = list1 ? list1 : list2;

        return dummy.next;
    }
};
```

Output -



Accepted 208 / 208 testcases passed

22BCS16918 submitted at Feb 20, 2025 15:58

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

19.38 MB | Beats 86.69%

Code | C++

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

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list
        ListNode dummy(0);
        ListNode* tail = &dummy;

        while (list1 && list2) {
            if (list1->val < list2->val) {
                tail->next = list1;
                list1 = list1->next;
```

Ln 1, Col 1 | Saved

22BCS16918

Access all features with our Premium subscription!

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

list1 =

[1,2,4]

list2 =

[1,3,4]

Output

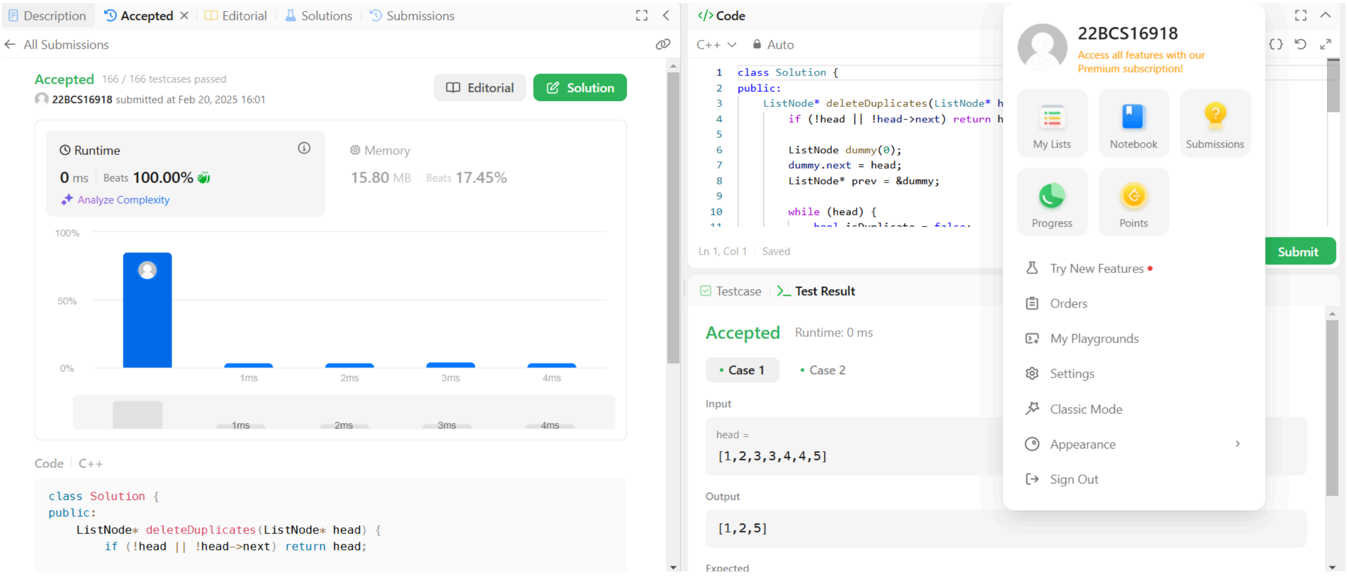
Ques 6. Remove Duplicates From sorted list 2

Code -

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

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;
        while (head) {
            bool isDuplicate = false;
            while (head->next && head->val == head->next->val) {
                isDuplicate = true;
                head = head->next;
            }
            if (isDuplicate) {
                prev->next = head->next;
            } else {
                prev = prev->next;
            }
            head = head->next;
        }
        return dummy.next;
    }
};
```

Output -



Accepted 166 / 166 testcases passed

22BCS16918 submitted at Feb 20, 2025 16:01

Runtime: 0 ms | Beats 100.00%

Memory: 15.80 MB | Beats 17.45%

Code | C++

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

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* h
        if (!head || !head->next) return h

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;

        while (head) {
            bool isDuplicate = false;
```

Ln 1, Col 1 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

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

Output

```
[1,2,5]
```

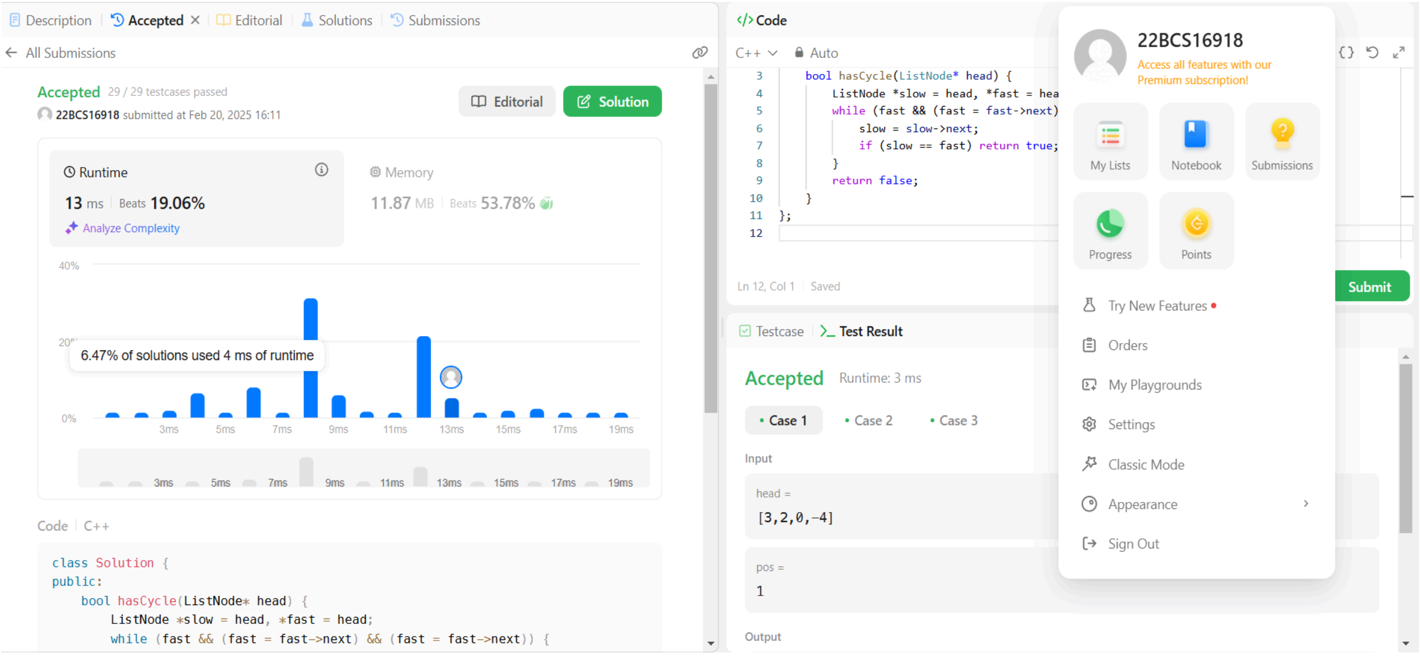
Expected

Ques 7. Detect a cycle in a linked list

Code -

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

Output -



Accepted 29 / 29 testcases passed

22BCS16918 submitted at Feb 20, 2025 16:11

13 ms | Beats 19.06%

11.87 MB | Beats 53.78%

6.47% of solutions used 4 ms of runtime

Code | C++

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode *slow = head, *fast = head;
        while (fast && (fast = fast->next) && (fast = fast->next)) {
```

```
    bool hasCycle(ListNode* head) {
        ListNode *slow = head, *fast = hea
        while (fast && (fast = fast->next)
            slow = slow->next;
            if (slow == fast) return true;
        }
        return false;
    }
};

```

Ln 12, Col 1 | Saved

22BCS16918

Access all features with our Premium subscription!

Accepted Runtime: 3 ms

• Case 1 • Case 2 • Case 3

Input

head =

[3,2,0,-4]

pos =

1

Output

Ques 8. Reverse Linked List II

Code -

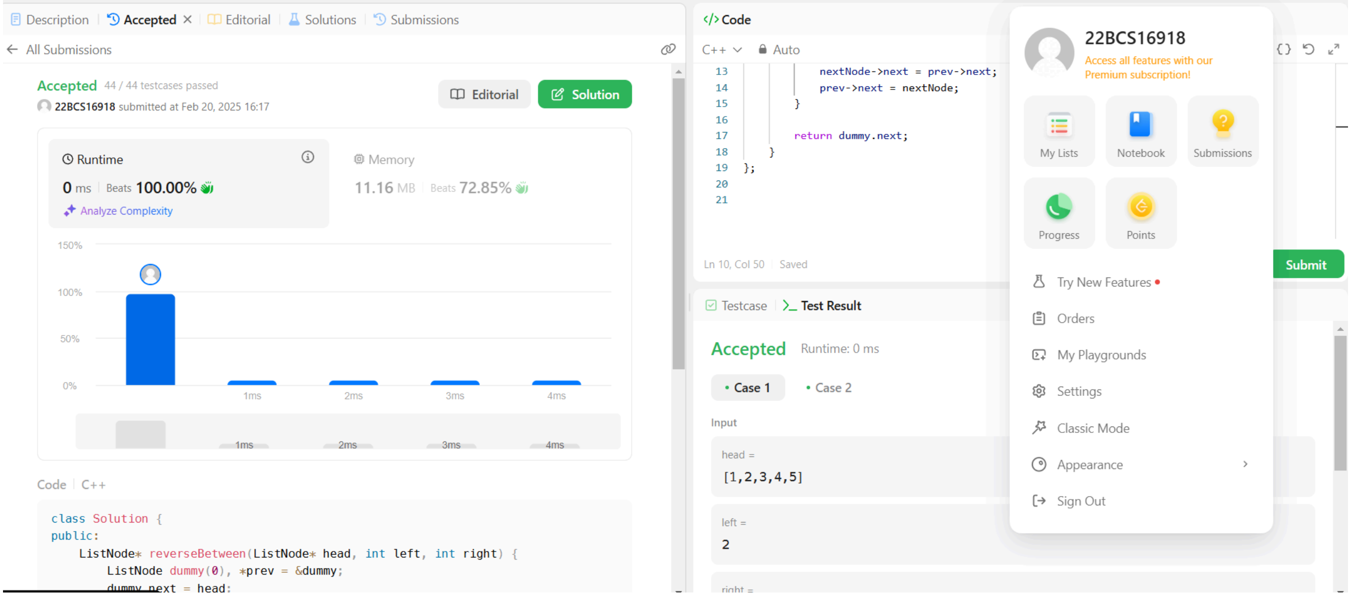
```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        ListNode dummy(0), *prev = &dummy;
        dummy.next = head;

        for (int i = 1; i < left; i++) prev = prev->next; // Move `prev` to `left-1`

        ListNode *curr = prev->next, *nextNode;
        for (int i = 0; i < right - left; i++) { // Reverse in-place
            nextNode = curr->next;
            curr->next = nextNode->next;
            nextNode->next = prev->next;
            prev->next = nextNode;
        }

        return dummy.next;
    }
};
```

Output -



Accepted 44 / 44 testcases passed

22BCS16918 submitted at Feb 20, 2025 16:17

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

11.16 MB | Beats 72.85%

Code | C++

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        ListNode dummy(0), *prev = &dummy;
        dummy.next = head;
```

```
                nextNode->next = prev->next;
                prev->next = nextNode;
            }

            return dummy.next;
        }
    };
```

Ln 10, Col 50 | Saved

22BCS16918

Access all features with our Premium subscription!

Accepted Runtime: 0 ms

Input

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

left =
2

right =

Ques 9 . Rotate a list

Code -

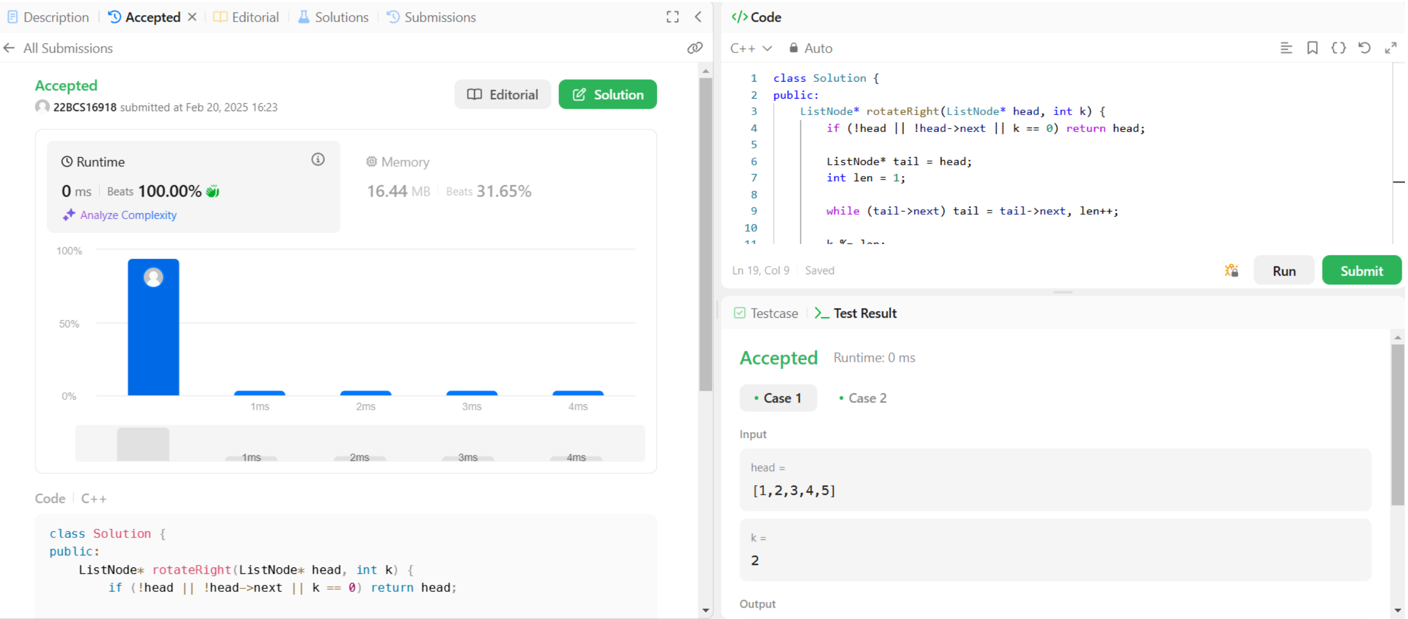
```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        ListNode* tail = head;
        int len = 1;
       while (tail->next) tail = tail->next, len++; // Find length & last node
        k %= len;
        if (k == 0) return head;

        tail->next = head; // Form a cycle
        for (int i = 0; i < len - k; i++) tail = tail->next; // Find new tail

        head = tail->next;
        tail->next = nullptr; // Break cycle

        return head;
    }
};
```

Output -



Accepted
22BCS16918 submitted at Feb 20, 2025 16:23

Runtime
0 ms | Beats 100.00%

Memory
16.44 MB | Beats 31.65%

Code | C++

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

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

        ListNode* tail = head;
        int len = 1;

        while (tail->next) tail = tail->next, len++;

        k %= len;
```

Ln 19, Col 9 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

k =
2

Output

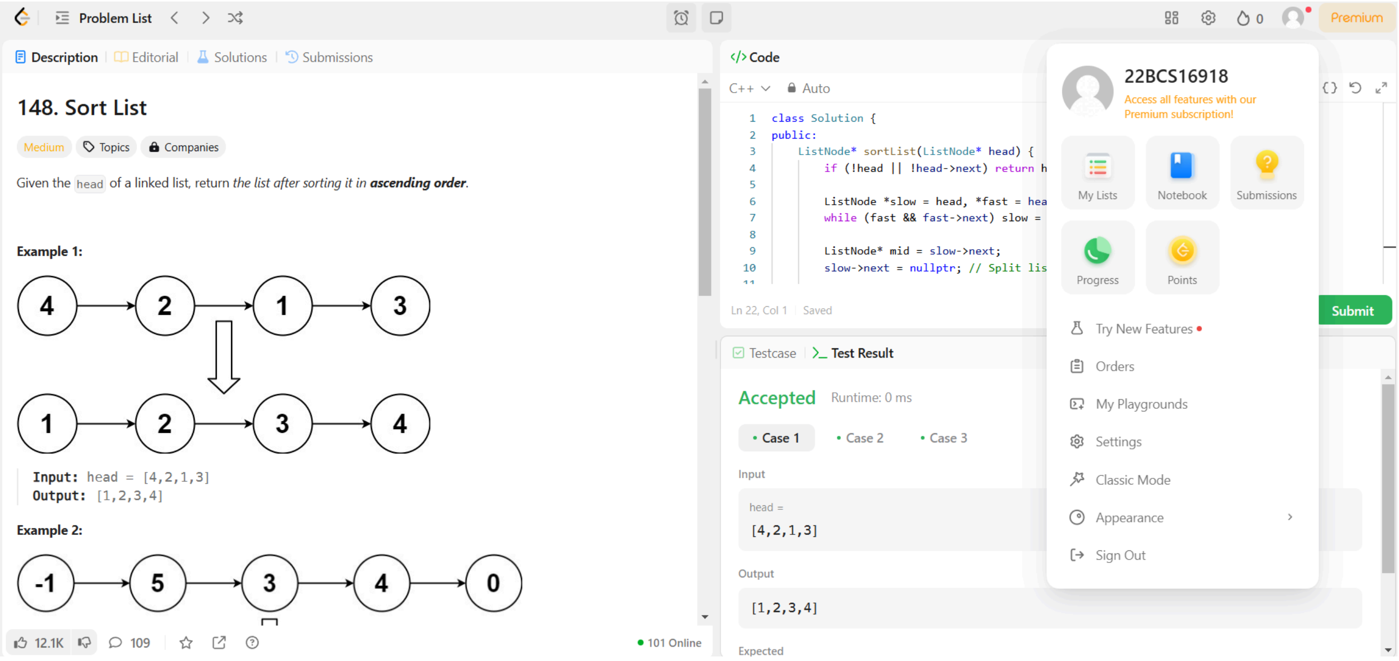
Ques 10. Sort list

Code -

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

        ListNode *slow = head, *fast = head->next;
        while (fast && fast->next) slow = slow->next, fast = fast->next->next;
        ListNode* mid = slow->next;
        slow->next = nullptr; // Split list
        return merge(sortList(head), sortList(mid));
    }
    ListNode* merge(ListNode* l1, ListNode* l2) {
        if (!l1 || !l2) return l1 ? l1 : l2;
        if (l1->val > l2->val) swap(l1, l2);
        l1->next = merge(l1->next, l2);
        return l1;
    }
};
```

Output -



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:

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

        ListNode *slow = head, *fast = hea
        while (fast && fast->next) slow =

        ListNode* mid = slow->next;
        slow->next = nullptr; // Split lis
```

Accepted Runtime: 0 ms

Input

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

Output

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

Expected

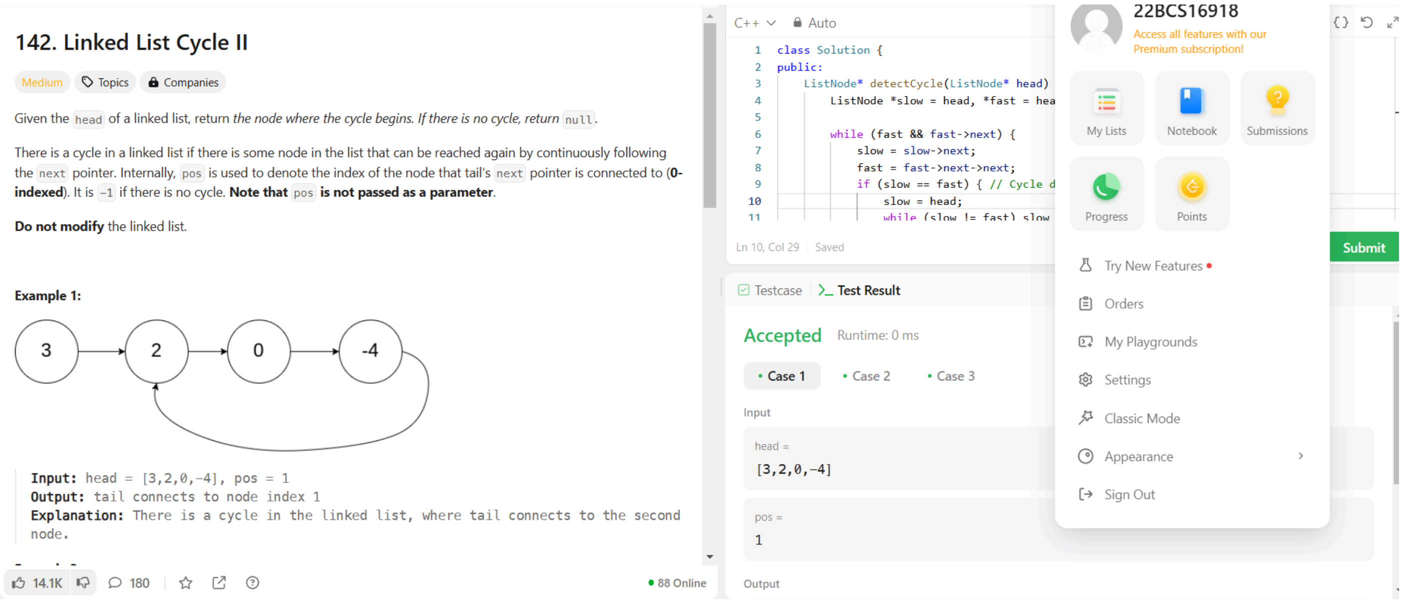
Ques 11. Detect a cycle in linked list 2

Code -

```
class Solution {
public:
    ListNode* detectCycle(ListNode* head) {
        ListNode *slow = head, *fast = head;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) { // Cycle detected
                slow = head;
                while (slow != fast) slow = slow->next, fast = fast->next;
                return slow; // Cycle start node
            }
        }
        return nullptr; // No cycle
    }
};
```

Output -



142. Linked List Cycle II

Medium · Topics · Companies

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

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 (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

Example 1:

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

```
class Solution {
public:
    ListNode* detectCycle(ListNode* head)
        ListNode *slow = head, *fast = hea

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) { // Cycle d
                slow = head;
                while (slow != fast) slow
```

Ln 10, Col 29 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

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

pos =
1

Output