

Name – Ansh Kumar Singh

UID – 22BCS11057

SECTION- 608/B

Print Linked List:

```
class Solution {  
    // Function to display the elements of a linked list in same line  
    void printList(Node head) {  
        Node temp = head;  
  
        while (temp != null) {  
            System.out.print(temp.data + " ");  
            temp = temp.next;  
        }  
    }  
}
```

Problem Solved Successfully 

[Suggest Feedback](#)

Test Cases Passed

1112 / 1112

Attempts : Correct / Total

2 / 2

Accuracy : 100%

Time Taken

1.74

1. Remove duplicates from a sorted list:

```
class Solution {  
    public ListNode deleteDuplicates(ListNode head) {  
        ListNode current = head;  
  
        while (current != null && current.next != null) {  
            if (current.val == current.next.val) {  
                current.next = current.next.next; // Skip duplicate node  
            } else {  
                current = current.next; // Move to next node  
            }  
        }  
        return head;  
    }  
}
```

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

Output

```
[1,2]
```

Expected

```
[1,2]
```

2. Reverse a linked list:

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

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

        return prev; // New head of reversed list
    }
};
```

The screenshot shows the LeetCode submission interface for the problem "Reverse a linked list". The submission is marked as "Accepted" with a runtime of 0 ms. The code is written in C++ and implements the iterative reversal of a linked list. The performance metrics show a runtime of 0 ms (beats 100.00%) and a memory usage of 13.19 MB (beats 99.46%).

Runtime: 0 ms | Beats: 100.00%
Memory: 13.19 MB | Beats: 99.46%

Accepted 28 / 28 testcases passed
Ansh Kumar Singh submitted at Mar 07, 2025 22:02

Editorial Solution

```

15  ListNode* current = head;
16
17  while (current != nullptr) {
18      ListNode* nextNode = current->next; // Store next node
19      current->next = prev; // Reverse the link
20      prev = current; // Move prev forward
21      current = nextNode; // Move current forward
22  }
23
24  return prev; // New head of reversed list
25  }
26
27  };

```

Testcase Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

3. Delete middle node of a list:

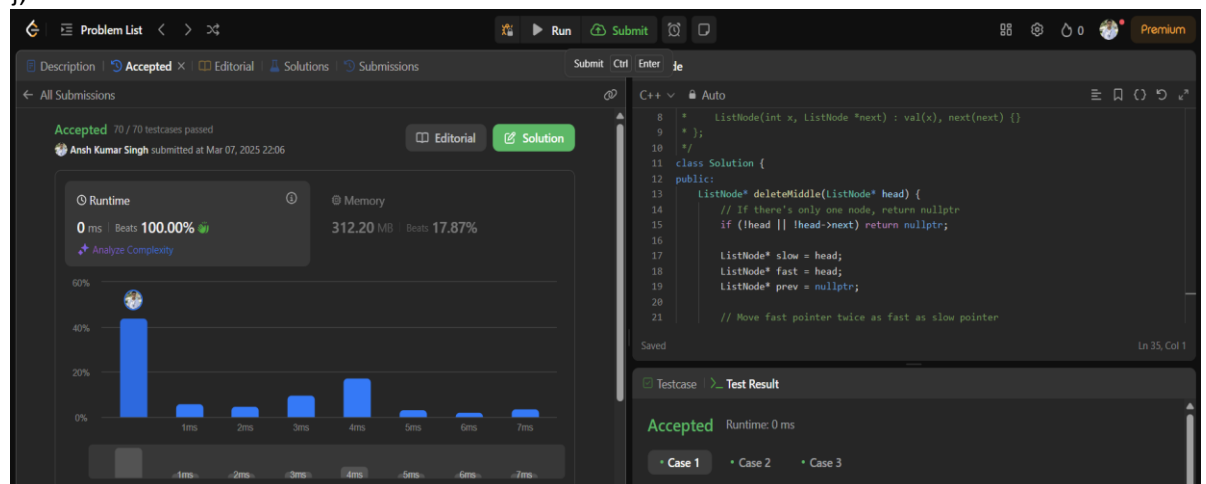
```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        // If there's only one node, return nullptr
        if (!head || !head->next) return nullptr;

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

        // Move fast pointer twice as fast as slow pointer
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        // Delete the middle node
        prev->next = slow->next;
        delete slow;

        return head;
    }
};
```



4. Merge two sorted linked lists:

```
class Solution {
public:
    ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode current = dummy;

        while (list1 != null && list2 != null) {
            if (list1->val < list2->val) {
```

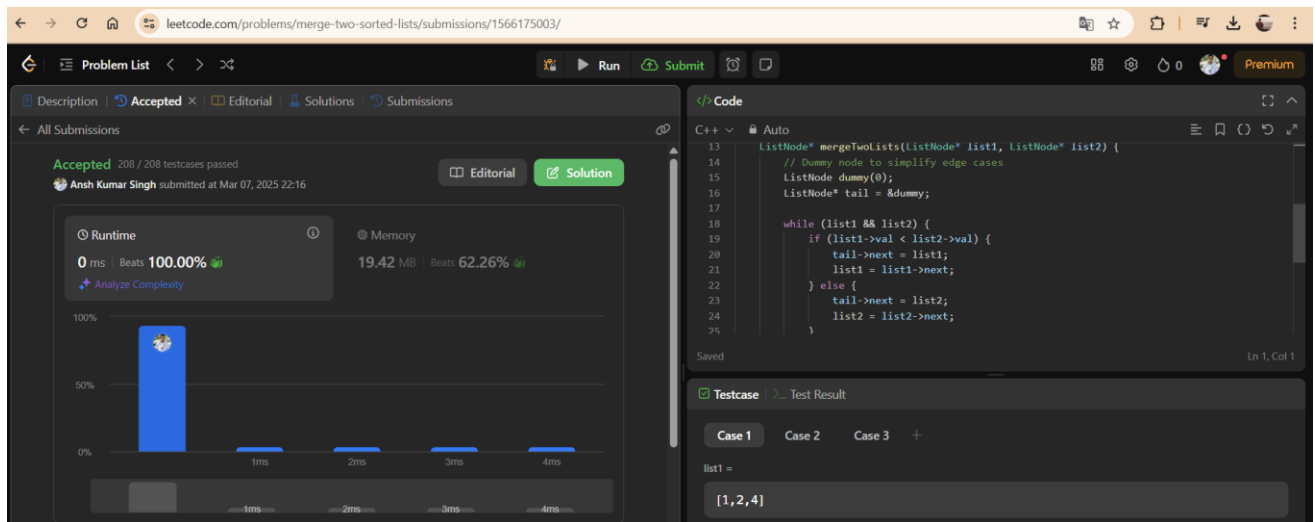
```

        current.next = list1;
        list1 = list1.next;
    } else {
        current.next = list2;
        list2 = list2.next;
    }
    current = current.next;
}

if (list1 != null) current.next = list1;
if (list2 != null) current.next = list2;

return dummy.next;
}
}

```



5. Detect a cycle in a linked list:

```

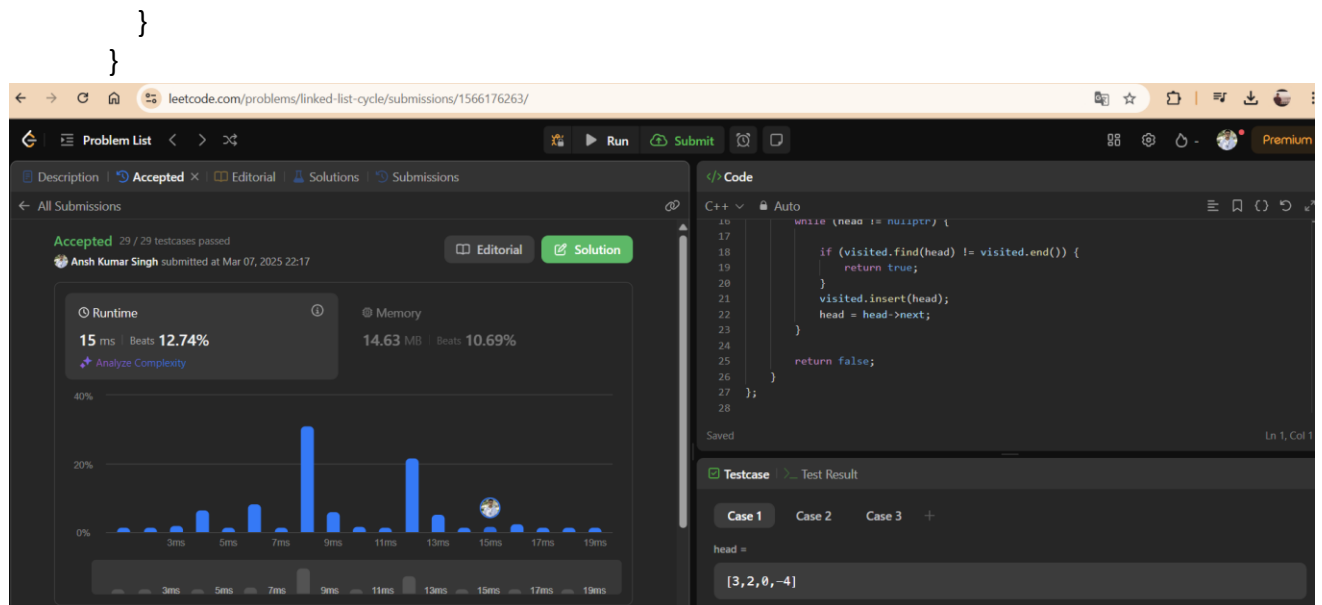
public class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null || head.next == null) return false;

        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast) return true;
        }
        return false;
    }
}

```



6. Rotate a list:

```

class Solution {
public:
    ListNode rotateRight(ListNode head, int k) {
        if (head == null || head.next == null || k == 0) return head;

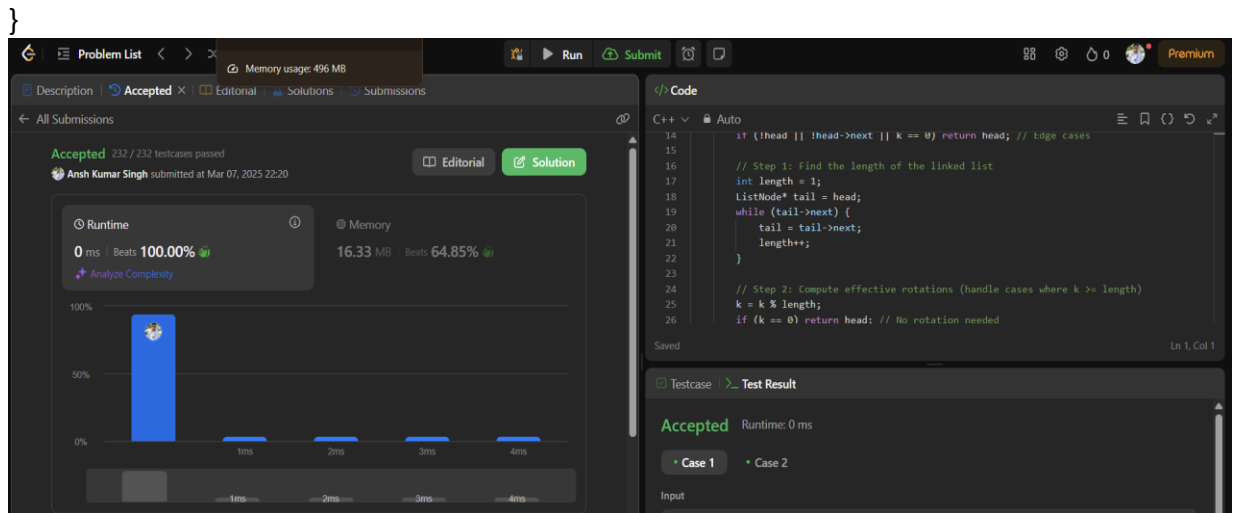
        ListNode temp = head;
        int length = 1;
        while (temp.next != null) {
            temp = temp.next;
            length++;
        }

        temp.next = head;

        int newTailIndex = length - k % length - 1;
        ListNode newTail = head;
        for (int i = 0; i < newTailIndex; i++) {
            newTail = newTail.next;
        }
        head = newTail.next;
        newTail.next = null;

        return head;
    }
}

```



7. Sort List:

```

class Solution {
public:
    ListNode sortList(ListNode head) {
        if (head == null || head.next == null) return head;
        ListNode mid = getMiddle(head);
        ListNode rightHead = mid.next;
        mid.next = null;

        ListNode left = sortList(head);
        ListNode right = sortList(rightHead);
        return merge(left, right);
    }

private:
    ListNode getMiddle(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

private:
    ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode curr = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {

```

```

        curr.next = l1;
        l1 = l1.next;
    } else {
        curr.next = l2;
        l2 = l2.next;
    }
    curr = curr.next;
}

curr.next = (l1 != null) ? l1 : l2;
return dummy.next;
}
}

```

The screenshot shows a LeetCode submission for the problem "Merge Two Sorted Lists". The submission is accepted, with a runtime of 0ms and memory usage of 19.42 MB. The code is written in C++ and implements a two-pointer approach to merge two sorted linked lists. The code is as follows:

```

ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    // Dummy node to simplify edge cases
    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;
        }
    }

    return dummy.next;
}

```

8. Merge k sorted lists:

```
import java.util.PriorityQueue;
```

```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) return null;
```

```

        PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);
        for (ListNode list : lists) {
            if (list != null) minHeap.add(list);
        }

```

```

        ListNode dummy = new ListNode(0);

```

```

ListNode curr = dummy;
while (!minHeap.isEmpty()) {
    ListNode smallest = minHeap.poll();
    curr.next = smallest;
    curr = curr.next;
    if (smallest.next != null) {
        minHeap.add(smallest.next);
    }
}
return dummy.next;
}

```

Accepted 29 / 29 testcases passed
Ansh Kumar Singh submitted at Mar 07, 2025 22:17

Runtime: 15 ms Beats 12.74%
Memory: 14.63 MB Beats 10.69%

Runtime Distribution: 0% 20% 40%
0ms 3ms 5ms 7ms 9ms 11ms 13ms 15ms 17ms 19ms

Code Editor (C++):

```

while (head != nullptr) {
    if (visited.find(head) != visited.end()) {
        return true;
    }
    visited.insert(head);
    head = head->next;
}
return false;
}

```

Testcase: [3, 2, 0, -4]