

# Assignment-3

Student Name: Unnati Srivastava

UID: 22BCS13475

Branch: BE-CSE

Section/Group: IOT-610/B

Semester: 6<sup>th</sup>

Date Of Performance: 5/3/25

Subject Name: Advanced Programming II

Subject Code: 22CSP-351

## QUES 1: Print Linked List

**Solution:**

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

The screenshot shows a C++ IDE interface. On the left, the 'Output Window' displays 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It states 'Problem Solved Successfully' with a green checkmark. Below this, statistics are shown: 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 0.07'. At the bottom, there are buttons for 'Solve Next' and 'Node at a given index in linked list', 'Delete Alternate Nodes', and 'Insert in Middle of Linked List'. On the right, the code editor shows the C++ code for the 'Print Linked List' problem, including the 'Node' struct, the 'printList' function, and the 'Solution' class. The code is highlighted in blue, and the 'Driver Code Ends' are visible at the top and bottom of the code block.

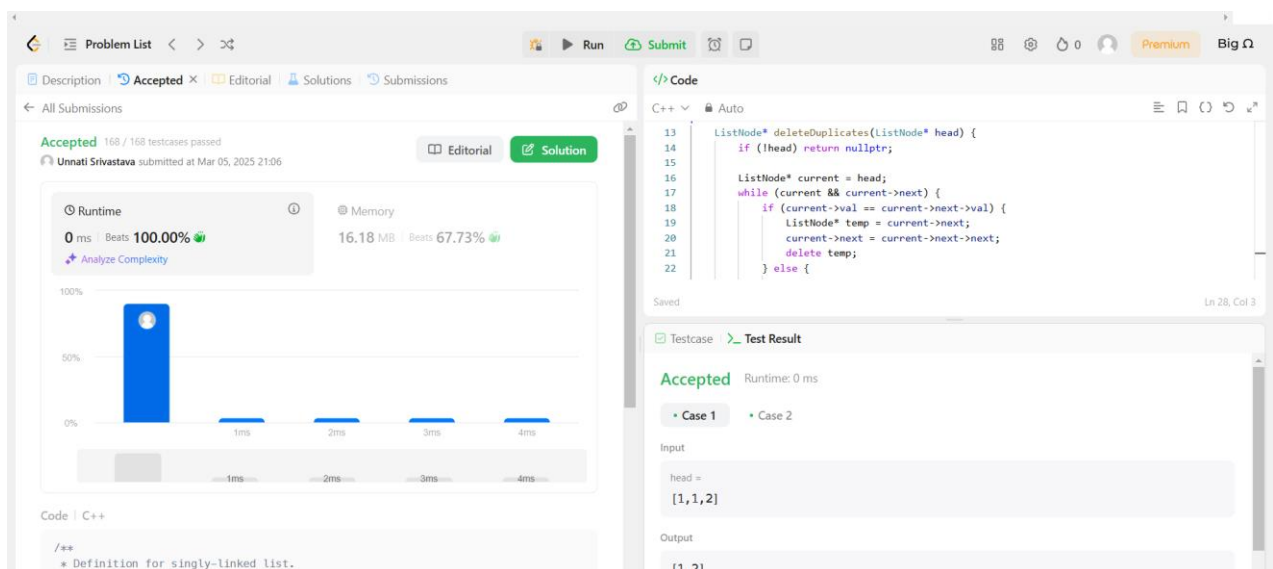
## QUES 2: Remove duplicates from a sorted list

**Solution:**

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

```
if (!head) return nullptr;
```

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

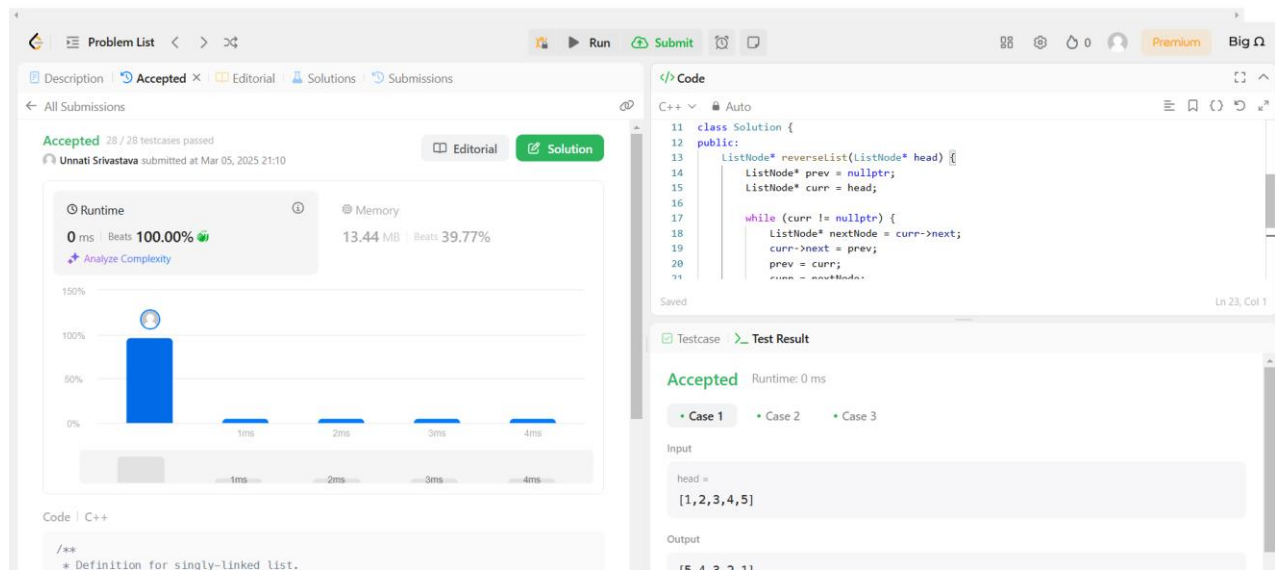


### **QUES 3: Reverse a linked list**

#### **Solution:**

```
class Solution {  
public:  
    ListNode* reverseList(ListNode* head) {  
        ListNode* prev = nullptr;  
        ListNode* curr = head;  
  
        while (curr != nullptr) {  
            ListNode* nextNode = curr->next;  
            curr->next = prev;  
            prev = curr;  
            curr = nextNode;  
        }  
  
        return prev;  
    }  
};
```

```
};
```



#### **QUES 4: Delete middle node of a list**

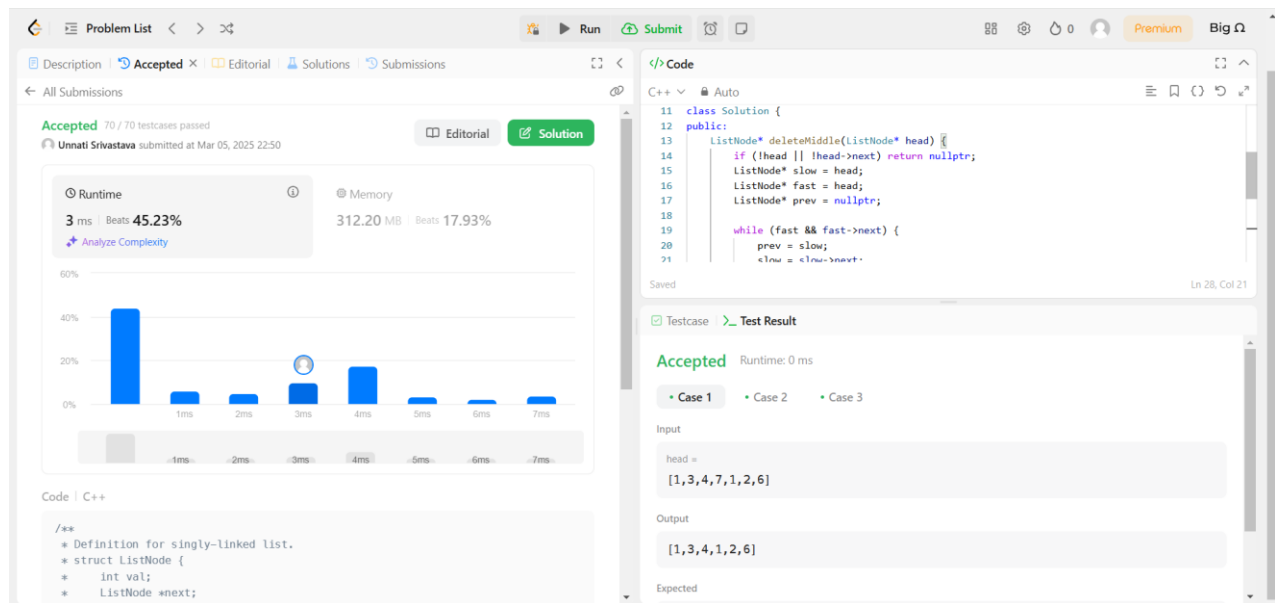
##### **Solution:**

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr;
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;

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

        prev->next = slow->next;
        delete slow;

        return head;
    }
};
```



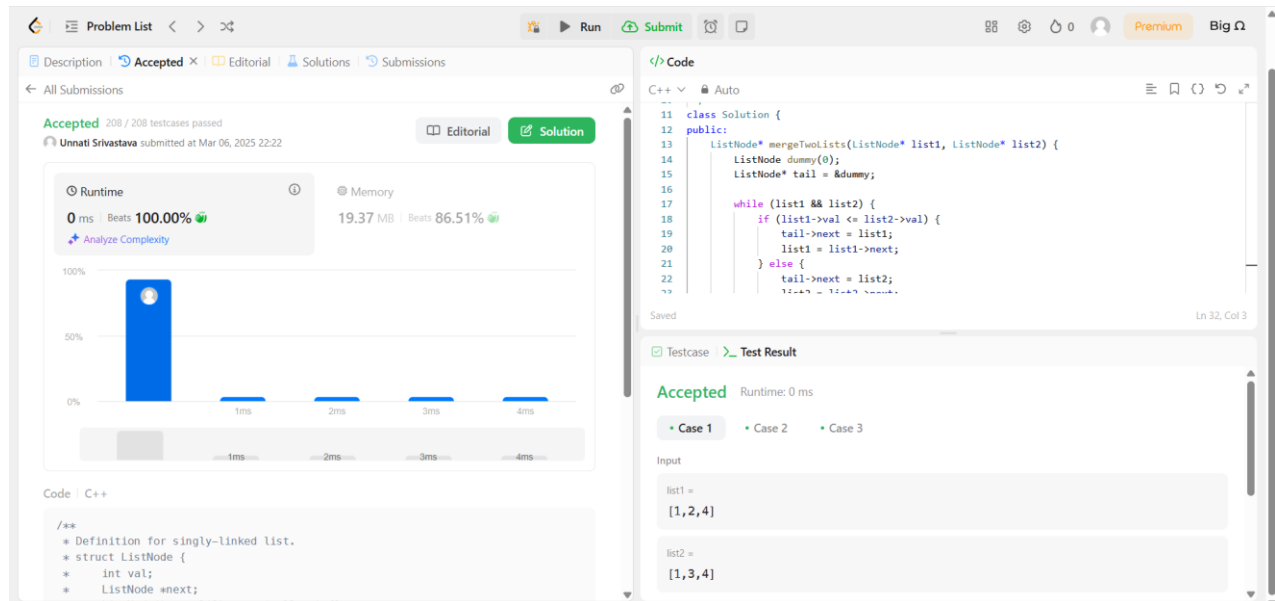
## QUES 5: Merge Two Sorted Lists

### **Solution:**

```
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;
        }
        if (list1) tail->next = list1;
        if (list2) tail->next = list2;

        return dummy.next;
    }
};
```



## QUES 6: Detect a cycle in a linked list

### Solution:

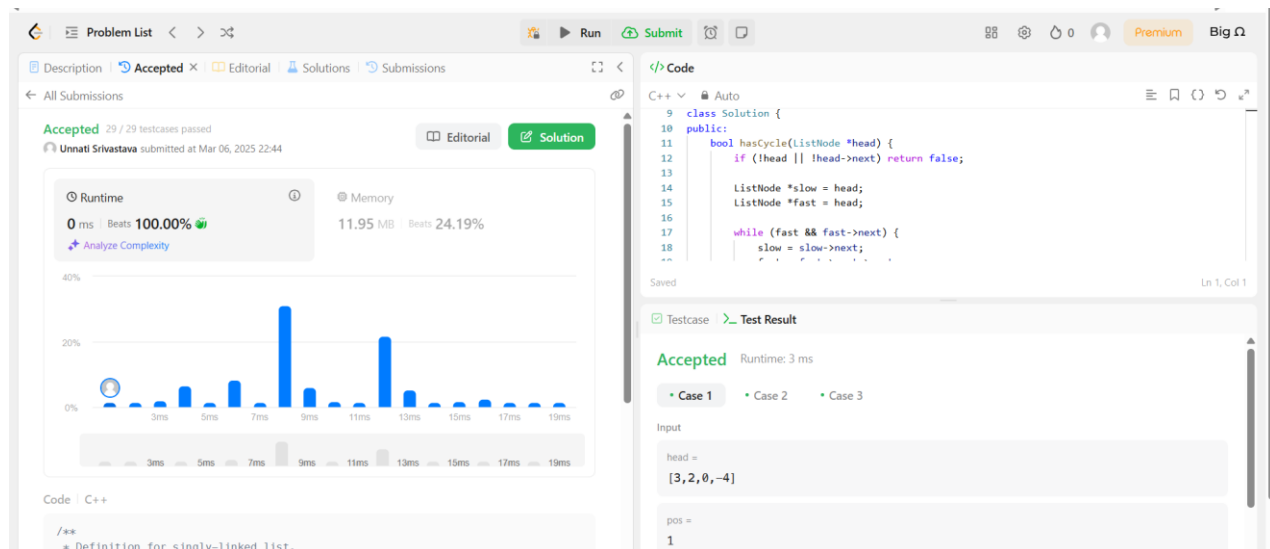
```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (!head || !head->next) return false;

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

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

            if (slow == fast) return true;
        }

        return false;
    }
};
```



## QUES 7: Rotate a list

### **Solution:**

```

class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        int length = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            length++;
        }
        tail->next = head;
        k = k % length;
        int stepsToNewHead = length - k;
        ListNode* newTail = head;
        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        return newHead;
    }
};

```

**61. Rotate List**

Medium Topics Companies

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

**Example 1:**

1 → 2 → 3 → 4 → 5

**rotate 1** 5 → 1 → 2 → 3 → 4

**rotate 2** 4 → 5 → 1 → 2 → 3

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

**Example 2:**

0 → 1 → 2

10.2K 104 122 Online

**Code**

```

11 class Solution {
12 public:
13     ListNode* rotateRight(ListNode* head, int k) {
14         if (!head || !head->next || k == 0) return head;
15         int length = 1;
16         ListNode* tail = head;
17         while (tail->next) {
18             tail = tail->next;
19             length++;
20         }

```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

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

k =  
2

Output

## **QUES 8: Sort list**

### **Solution:**

```

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

```

```

        ListNode* mid = getMiddle(head);
        ListNode* left = head;
        ListNode* right = mid->next;
        mid->next = nullptr;

```

```

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

```

```

        return merge(left, right);
    }

```

private:

```

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

```

```

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;

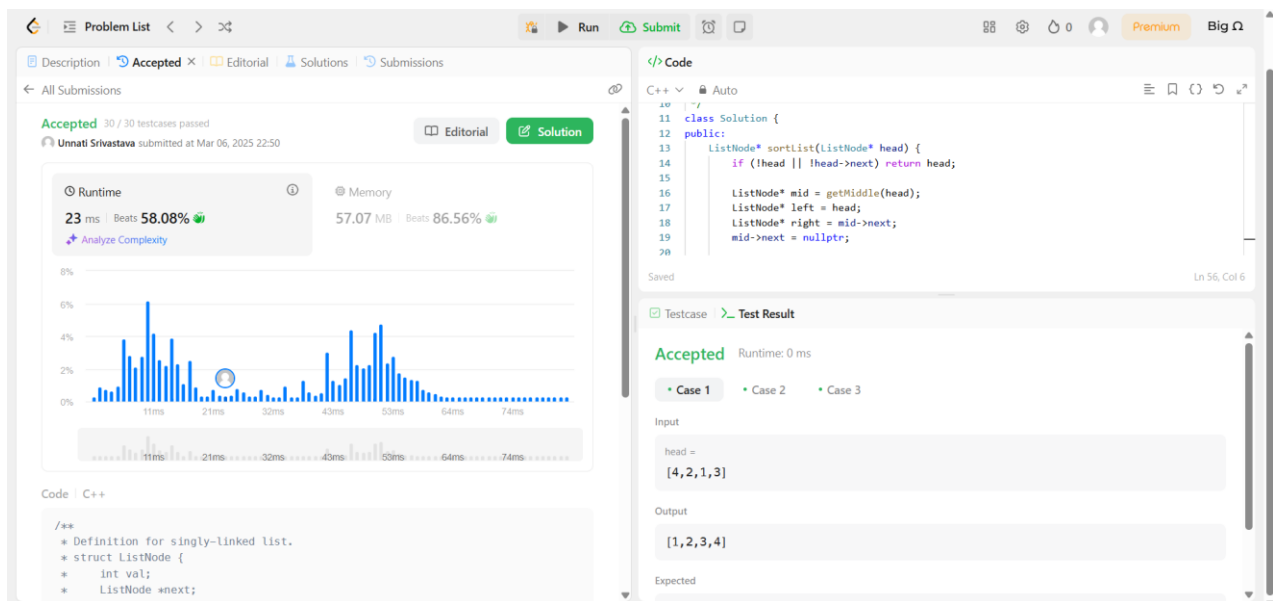
```

```

while (l1 && l2) {
    if (l1->val < l2->val) {
        tail->next = l1;
        l1 = l1->next;
    } else {
        tail->next = l2;
        l2 = l2->next;
    }
    tail = tail->next;
}

tail->next = l1 ? l1 : l2;
return dummy.next;
}
};

```



## QUES 9: Merge k sorted lists

### **Solution:**

```

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        auto compare = [](ListNode* a, ListNode* b) {
            return a->val > b->val;
        };

        priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> minHeap(compare);

        for (auto list : lists) {
            if (list) minHeap.push(list);
        }
    }
};

```



```

ListNode dummy(0);
ListNode* tail = &dummy;

while (!minHeap.empty()) {
    ListNode* node = minHeap.top();
    minHeap.pop();

    tail->next = node;
    tail = tail->next;

    if (node->next) minHeap.push(node->next);
}

return dummy.next;
}
};

```

The screenshot displays a C++ submission on a coding platform. The submission is titled "Accepted" and shows performance metrics: Runtime 3 ms (Beats 64.88%), Memory 18.47 MB (Beats 66.07%). A bar chart shows the runtime distribution across test cases. The code is a C++ implementation of a linked list merging algorithm using a min-heap. The test results show the input lists as [[1,4,5], [1,3,4], [2,6]] and the output as [1,1,2,3,4,4,5,6].

**Runtime:** 3 ms | Beats: 64.88%

**Memory:** 18.47 MB | Beats: 66.07%

**Testcase:** Accepted | Runtime: 0 ms

**Case 1:** Input: lists = [[1,4,5], [1,3,4], [2,6]] | Output: [1,1,2,3,4,4,5,6]