DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CHANDIGARH UNIVERSITY

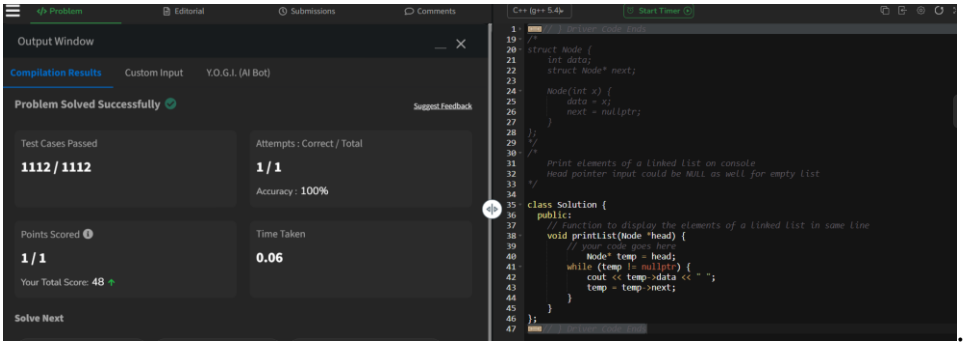
Discover. Learn. Empower.

Assignment

Student Name: Bhuvnesh Gautam
UID: 22BCS14051
Branch: BE - CSE
Section/Group: 22BCS_IOT-611 - B
Semester: 6th
Date of Performance: 14th Feb, 2025
Subject Name: Advance Programming

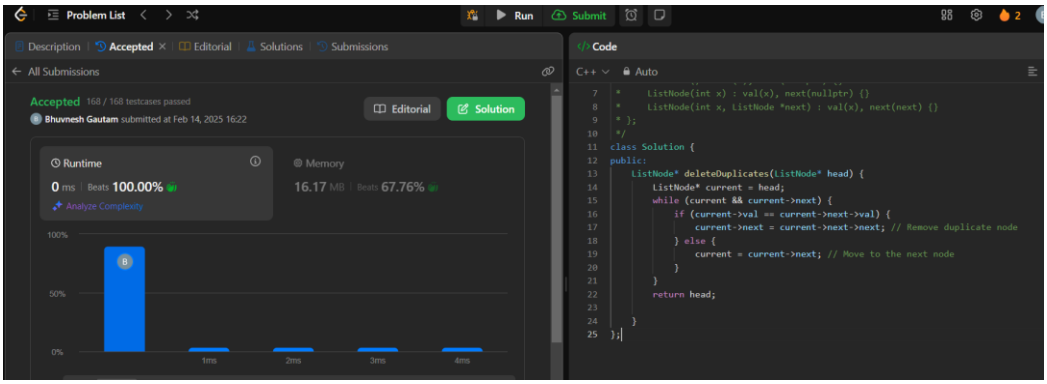
1. Print Linked List:

```
class Solution {
  public:
    void printList(Node *head) {
        // your code goes here
            Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
    }
};
```



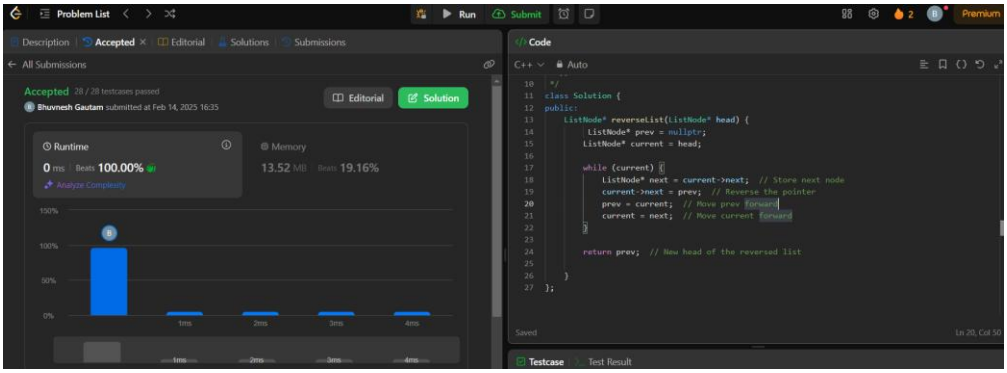
2. Remove Duplicates from Sorted List

```
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; // Remove duplicate node
            } else {
                current = current->next; // Move to the next node
            }
        }
    }
```



3. Reverse Linked List

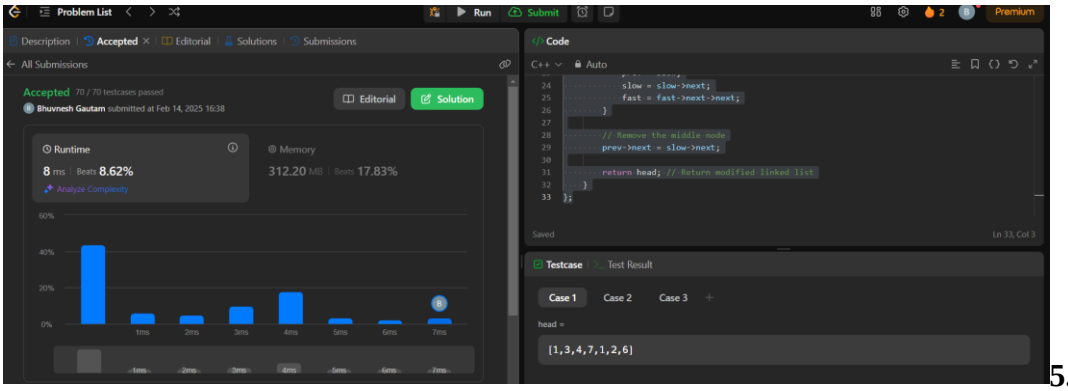
```
class Solution {
public:
   ListNode* reverseList(ListNode* head) {
       ListNode* prev = nullptr;
      ListNode* current = head;
      while (current) {
         ListNode* next = current->next;
         current->next = prev;
         prev = current;
         current = next;
      }
      return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



4. Delete the middle Node of Linked List

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

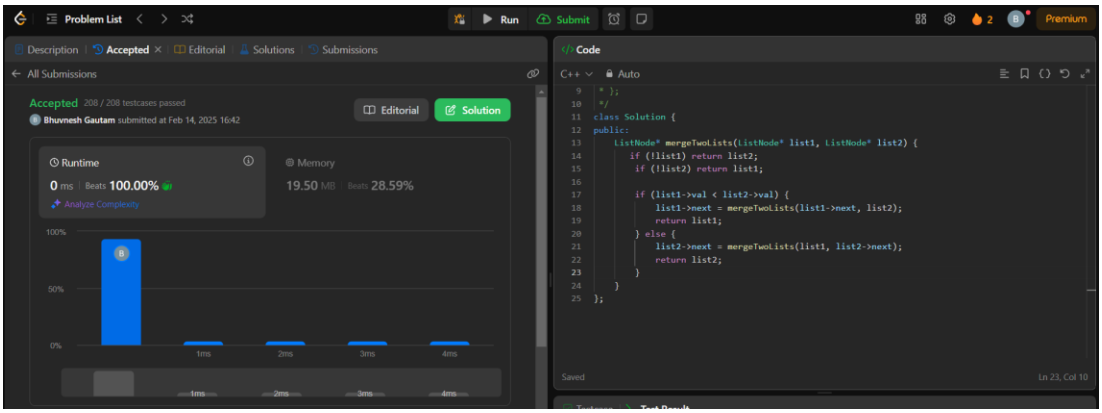
```
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        prev->next = slow->next;
        return head;
    }
};
```



5.

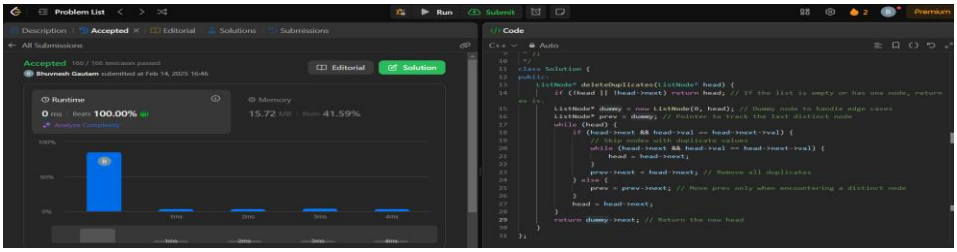
5. Merge two sorted list

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if (!list1) return list2;
        if (!list2) return list1;
        if (list1->val < list2->val) {
            list1->next = mergeTwoLists(list1->next, list2);
            return list1;
        } else {
            list2->next = mergeTwoLists(list1, list2->next);
            return list2;
        }
    }
};
```



6. Remove Duplicates from sorted list 2

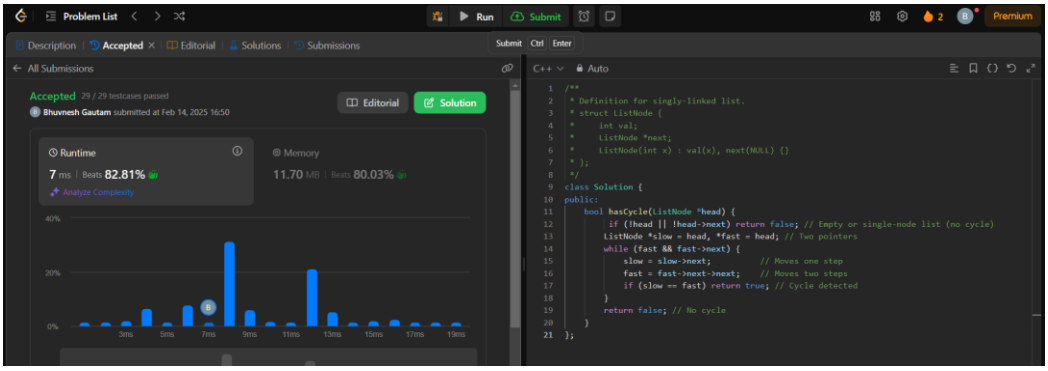
```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* dummy = new ListNode(0, head);
        ListNode* prev = dummy;
        while (head) {
            if (head->next && head->val == head->next->val) {
                // Skip nodes with duplicate values
                while (head->next && head->val == head->next->val) {
                    head = head->next;
                }
                prev->next = head->next;
            } else {
                prev = prev->next;
            }
            head = head->next;
        }
        return dummy->next;
    }
};
```



7. Linked List Cycle

```
class Solution {
```

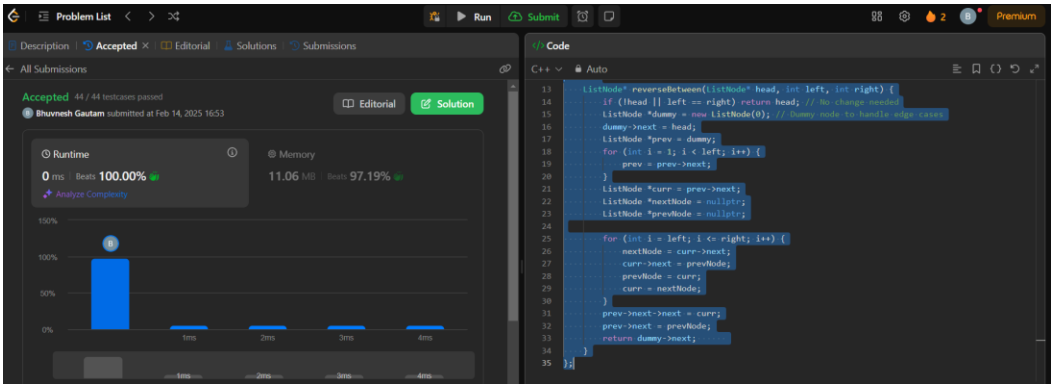
```
public:
    bool hasCycle(ListNode *head) {
        if (!head || !head->next) return false; // Empty or single-node list (no cycle)
        ListNode *slow = head, *fast = head; // Two pointers
        while (fast && fast->next) {
            slow = slow->next;          // Moves one step
            fast = fast->next->next;    // Moves two steps
            if (slow == fast) return true; // Cycle detected
        }
        return false; // No cycle
    }
};
```



8. Reverse Linked List 2

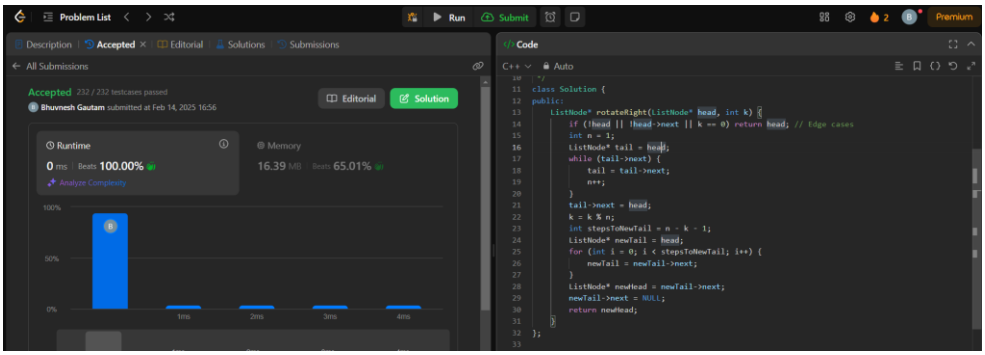
```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) return head; // No change needed
        ListNode *dummy = new ListNode(0); // Dummy node to handle edge cases
        dummy->next = head;
        ListNode *prev = dummy;
        for (int i = 1; i < left; i++) {
            prev = prev->next;
        }
        ListNode *curr = prev->next;
        ListNode *nextNode = nullptr;
        ListNode *prevNode = nullptr;

        for (int i = left; i <= right; i++) {
            nextNode = curr->next;
            curr->next = prevNode;
            prevNode = curr;
            curr = nextNode;
```

```
        }
        prev->next->next = curr;
        prev->next = prevNode;
        return dummy->next;
    }
};
```



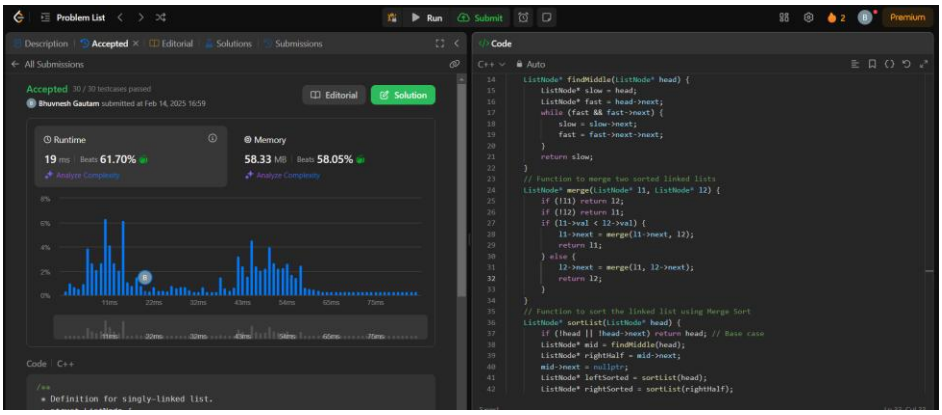
9. Rotate List

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head; // Edge cases
        int n = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            n++;
        }
        tail->next = head;
        k = k % n;
        int stepsToNewTail = n - k - 1;
        ListNode* newTail = head;
        for (int i = 0; i < stepsToNewTail; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = NULL;
        return newHead;
    }
};
```



10. Sort List

```
class Solution {
public:
    // Function to find the middle of the linked list
    ListNode* findMiddle(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 linked lists
    ListNode* merge(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;
        if (l1->val < l2->val) {
            l1->next = merge(l1->next, l2);
            return l1;
        } else {
            l2->next = merge(l1, l2->next);
            return l2;
        }
    }
    // Function to sort the linked list using Merge Sort
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head; // Base case
        ListNode* mid = findMiddle(head);
        ListNode* rightHalf = mid->next;
        mid->next = nullptr;
        ListNode* leftSorted = sortList(head);
        ListNode* rightSorted = sortList(rightHalf);
        return merge(leftSorted, rightSorted);
    }
};
```



11. Linked List Cycle 2

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        if (!head || !head->next) return nullptr;
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) break;
        }
        if (!fast || !fast->next) return nullptr;
        slow = head;
        while (slow != fast) {
            slow = slow->next;
            fast = fast->next;
        }
        return slow;
    }
};
```