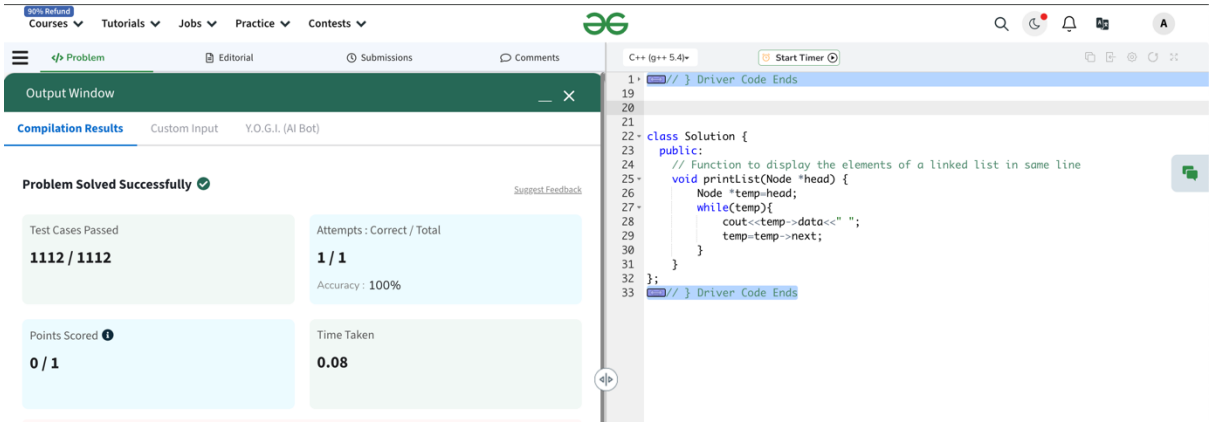
# Assignment 3 (AP)

**Problem 1** – Print Linked list

**Solution –**

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

Screenshot-

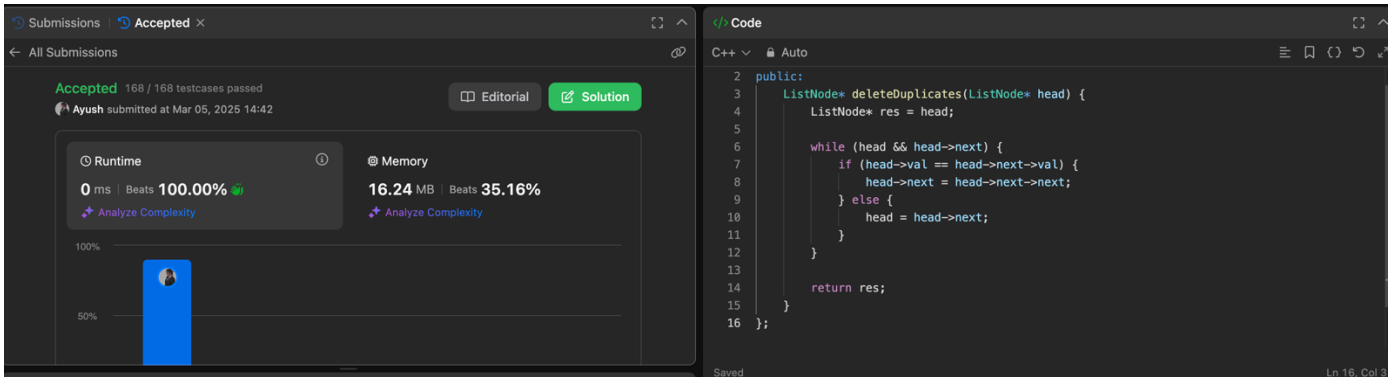


**Problem 2** – Remove Duplicate

**Solution –**

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

      while (head && head->next) {
         if (head->val == head->next->val) {
            head->next = head->next->next;
         } else {
            head = head->next;
         }
      }

      return res;
   }
};
```

Screenshot-

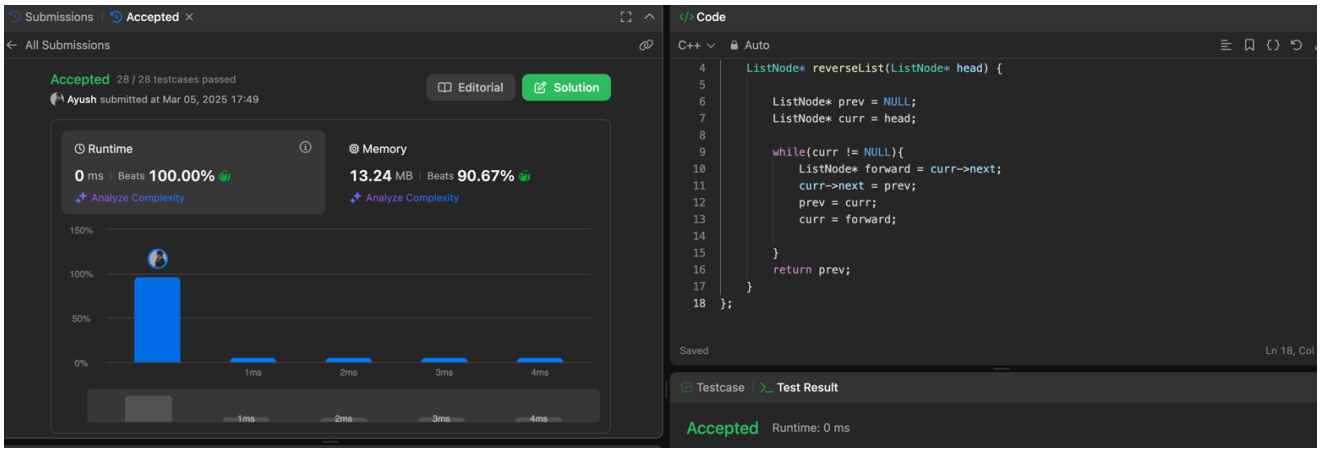


**Problem 3** – Reverse Linked list

**Solution –**

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

      ListNode* prev = NULL;
      ListNode* curr = head;

      while(curr != NULL){
         ListNode* forward = curr->next;
         curr->next = prev;
         prev = curr;
         curr = forward;

      }
      return prev;
   }
};
```

Screenshot-

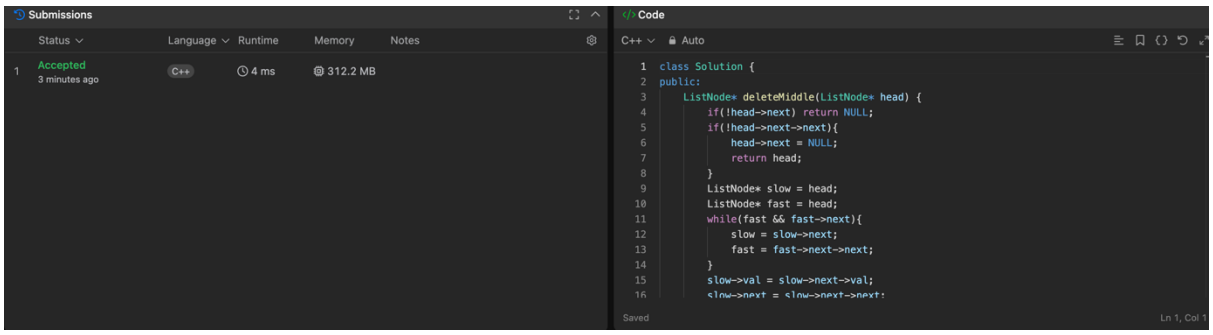


**Problem 4 - Delete middle node of a list**

**Solution –**

```
class Solution {
public:
   ListNode* deleteMiddle(ListNode* head) {
      if(!head->next) return NULL;
      if(!head->next->next){
         head->next = NULL;
         return head;
      }
      ListNode* slow = head;
      ListNode* fast = head;
      while(fast && fast->next){
         slow = slow->next;
         fast = fast->next->next;
      }
      slow->val = slow->next->val;
      slow->next = slow->next->next;
      return head;
   }
};
```

Screenshot-

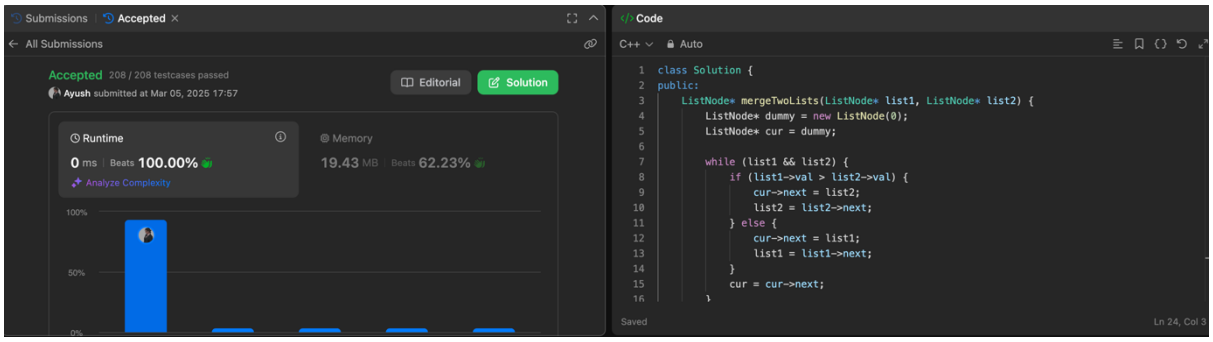


**Problem 5 - Merge two sorted linked lists**

**Solution –**

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

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

      cur->next = list1 ? list1 : list2;

      ListNode* head = dummy->next;
      delete dummy;
      return head;
   }
};
```

Screenshot-

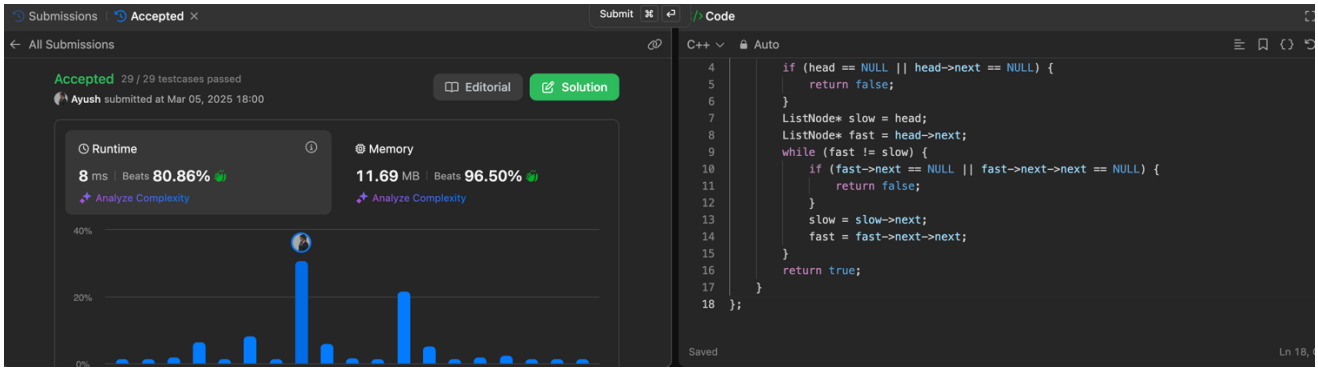


**Problem 6 - Detect a cycle in a linked list:**

**Solution –**

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

Screenshot-

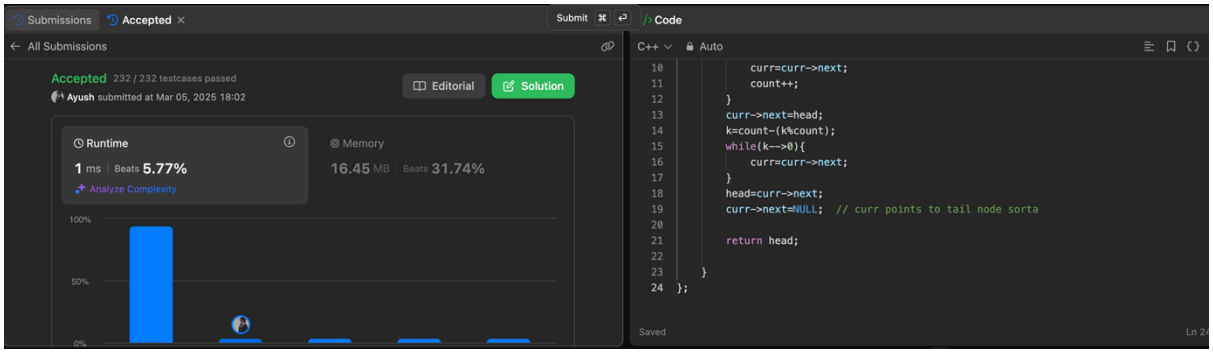


**Problem 7- Rotate a list:**

**Solution –**

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

      ListNode* curr=head;
      int count=1;
      while(curr->next!=NULL){
         curr=curr->next;
         count++;
      }
      curr->next=head;
      k=count-(k%count);
      while(k-->0){
         curr=curr->next;
      }
      head=curr->next;
      curr->next=NULL;  // curr points to tail node sorta

      return head;

   }
};
```

Screenshot-

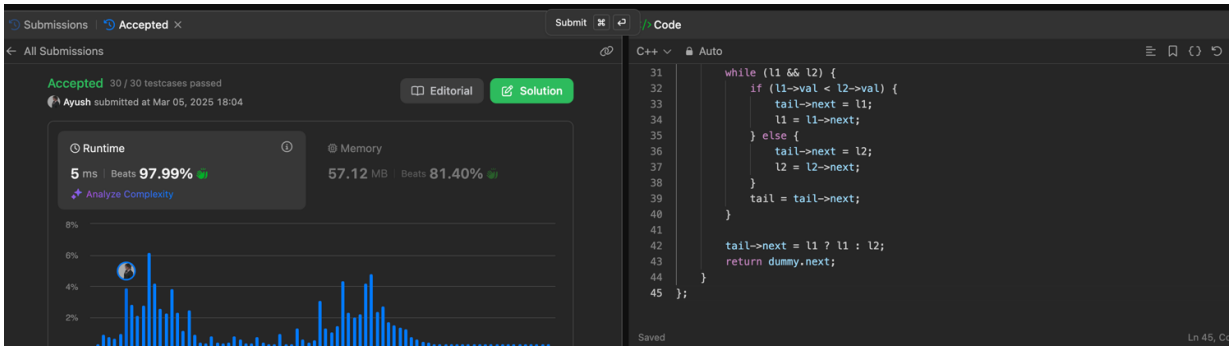


**Problem 8- Sort list**

**Solution –**

```
#include <iostream>
using namespace std;
//takesoumen collection
class Solution {
public:
   ListNode* sortList(ListNode* head) {
      if (!head || !head->next) return head;

      // Find the middle using slow and fast pointers
      ListNode* slow = head;
      ListNode* fast = head->next;
      while (fast && fast->next) {
         slow = slow->next;
         fast = fast->next->next;
      }

      ListNode* mid = slow->next;
      slow->next = nullptr;

      // Recursively split and merge
      ListNode* left = sortList(head);
      ListNode* right = sortList(mid);

      return merge(left, right);
   }

   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;
   }
};
```

Screenshot-




**Problem 9- Merge k sorted lists**

**Solution –**

```
class Solution {
public:
   ListNode* mergeKLists(vector<ListNode*>& lists) {
      if (lists.empty()) {
         return nullptr;
      }

      while (lists.size() > 1) {
         vector<ListNode*> temp;
         for (size_t i = 0; i < lists.size(); i += 2) {
            ListNode* l1 = lists[i];
            ListNode* l2 = i + 1 < lists.size() ? lists[i + 1] : nullptr;
            temp.push_back(mergeLists(l1, l2));
         }
         lists = move(temp);
      }

      return lists[0];
   }

private:
   ListNode* mergeLists(ListNode* l1, ListNode* l2) {
      ListNode dummy;
      ListNode* node = &dummy;

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

```
        node = node->next;
    }

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

Screenshot-