NAME :HARMAN SINGH

UID:22BCS14925

AP assignment

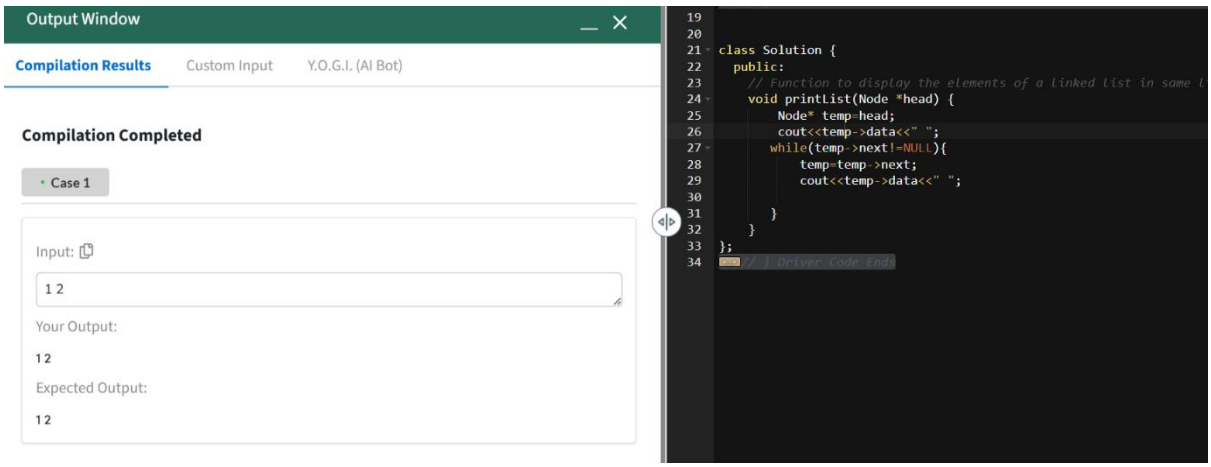
Linked Lists:

1. Print Linked List: https://www.geeksforgeeks.org/problems/print-linked-list-elements/0

Solution:

```
class Solution {
  public:

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

OUTPUT:

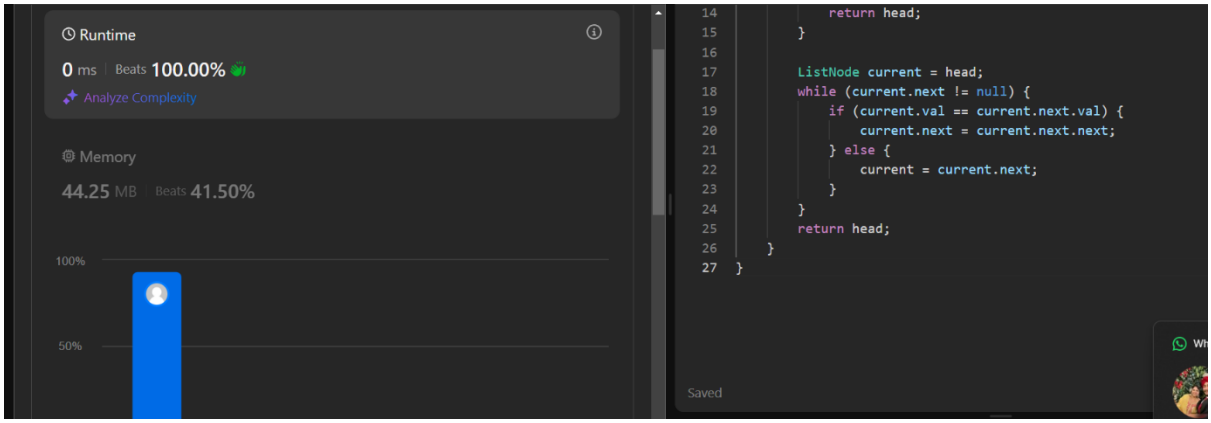


2. Remove duplicates from a sorted list: https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/

Solution:

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null) {
            return head;
        }
        ListNode current = head;
        while (current.next != null) {
            if (current.val == current.next.val) {
                current.next = current.next.next;
            } else {
                current = current.next;
        }
        }
        return head;
    }
}
```

OUTPUT:



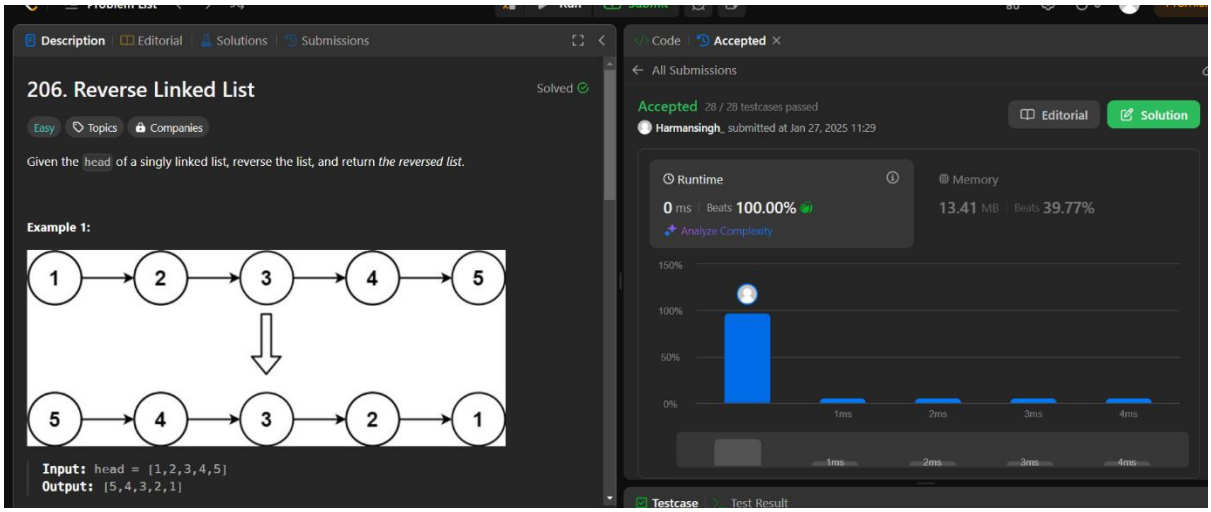
3. Reverse a linked list: https://leetcode.com/problems/reverse-linked-list/description/

   Solution:

   ```
   class Solution {
   public:
   ```

```
    ListNode* reverseList(ListNode* head) {
        ListNode *nextNode, *prevNode = NULL;
        while (head) {
            nextNode = head->next;
            head->next = prevNode;
            prevNode = head;
            head = nextNode;
        }
        return prevNode;
    }
};
```

OUTPUT:



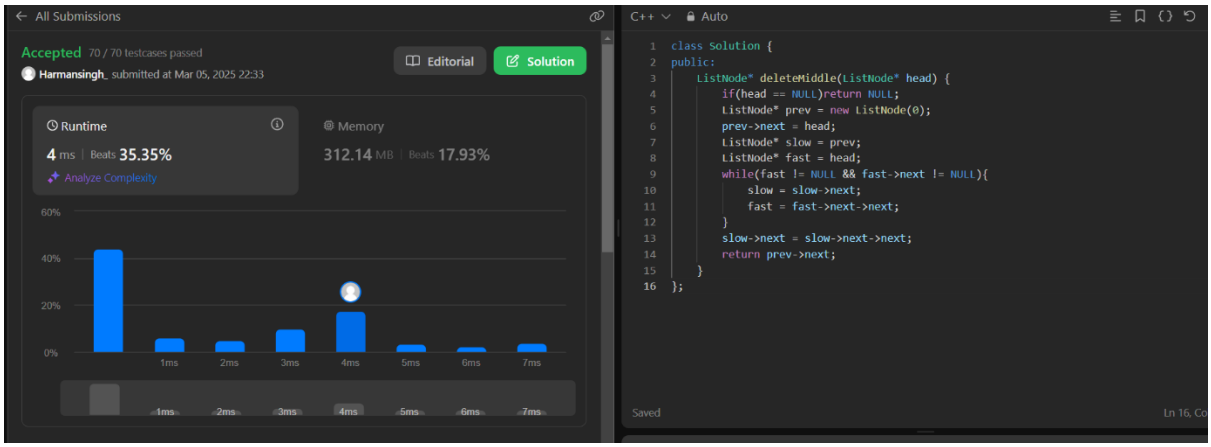
4. Delete middle node of a list: https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/

Solution:

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(head == NULL)return NULL;
        ListNode* prev = new ListNode(0);
        prev->next = head;
        ListNode* slow = prev;
        ListNode* fast = head;
```

```
        while(fast != NULL && fast->next != NULL){
            slow = slow->next;
            fast = fast->next->next;
        }
        slow->next = slow->next->next;
        return prev->next;
    }
};
```

Output:



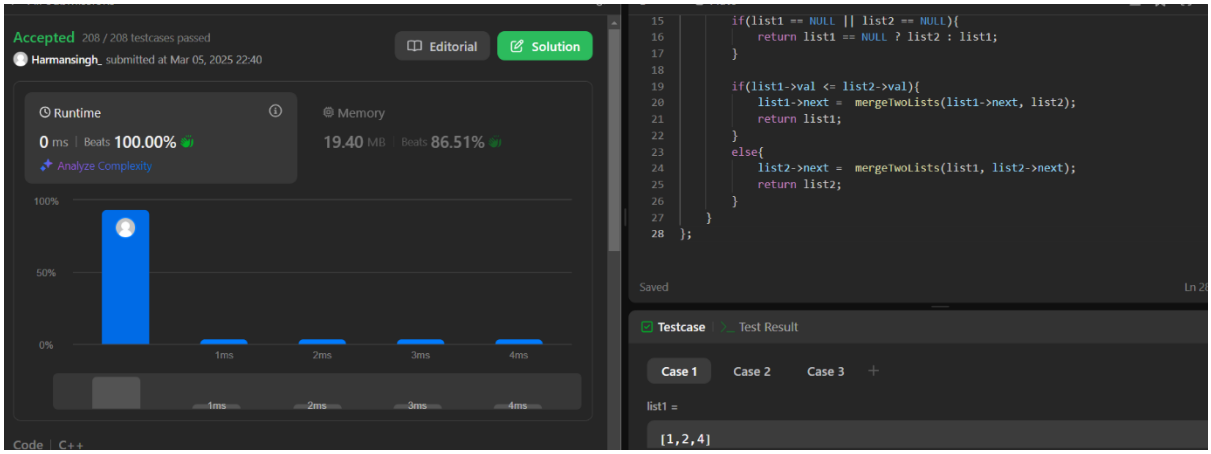
5. Merge two sorted linked lists: https://leetcode.com/problems/merge-two-sorted-lists/description/

Solution:

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1 == NULL || list2 == NULL){
            return list1 == NULL ? list2 : list1;
        }

        if(list1->val <= list2->val){
            list1->next =  mergeTwoLists(list1->next, list2);
            return list1;
```

```
        }
        else{
            list2->next =  mergeTwoLists(list1, list2->next);
            return list2;
        }
    }
};
```

OUTPUT:



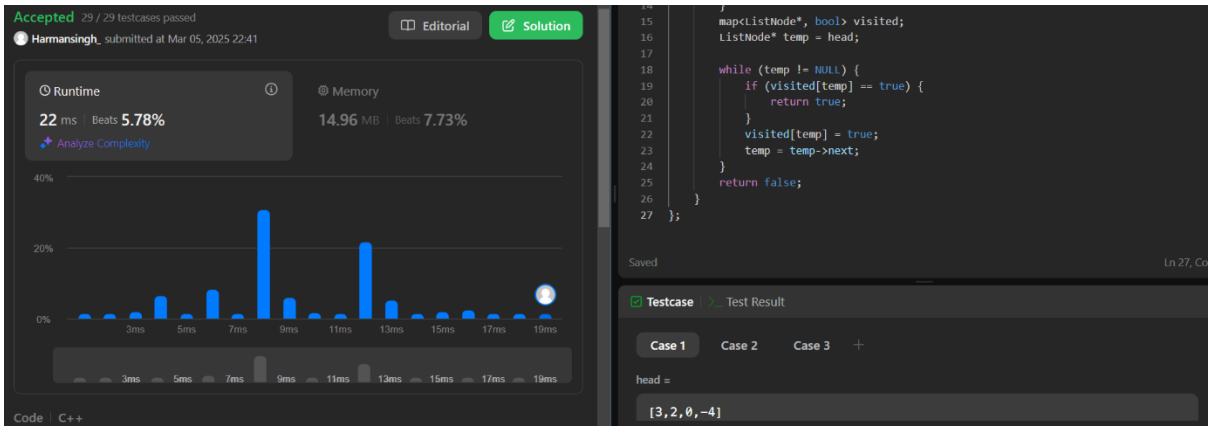
6. Detect a cycle in a linked list: https://leetcode.com/problems/linked-list-cycle/description/

Solution:

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        if (head == NULL) {
            return false;
        }
        map<ListNode*, bool> visited;
        ListNode* temp = head;

        while (temp != NULL) {
            if (visited[temp] == true) {
                return true;
            }
```

```
            visited[temp] = true;
            temp = temp->next;
        }
        return false;
    }
};
```

OUTPUT:



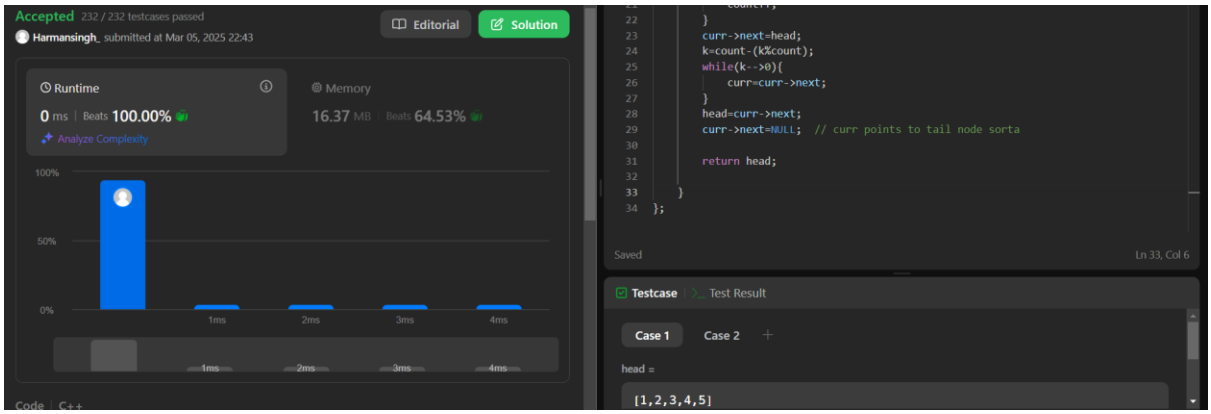
7. Rotate a list: https://leetcode.com/problems/rotate-list/description/

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;

    }
};
```

OUTPUT:



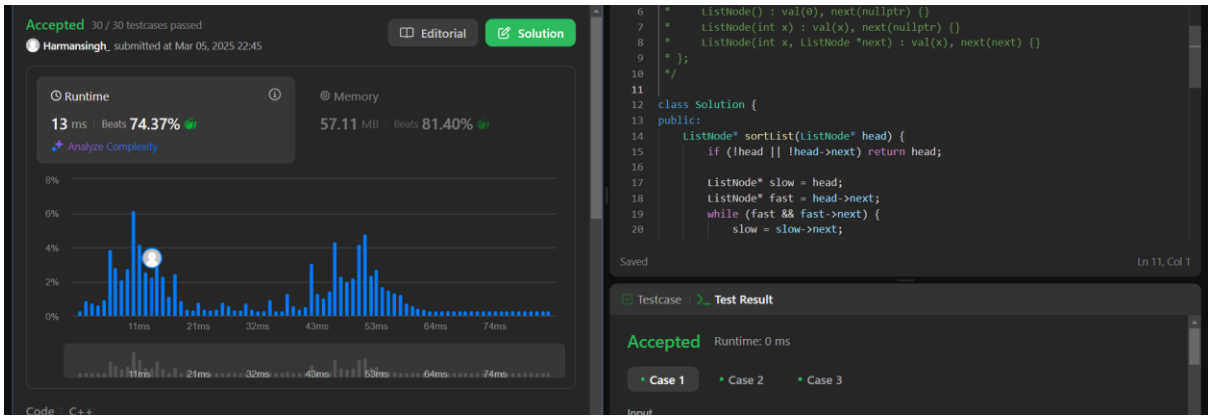
8. Sort List: https://leetcode.com/problems/sort-list/description/

Solution:

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

        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;


        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;
    }
};
```

OUTPUT:



9. Merge k sorted lists: https://leetcode.com/problems/merge-k-sorted-lists/description/

Solution:

```
class Solution {
public:
   ListNode* mergeKLists(vector<ListNode*>& lists) {
      priority_queue<pair<int,ListNode*>, vector<pair<int, ListNode*>> ,
greater<pair<int,ListNode*>>> pq;

      for(int i=0;i<lists.size();i++){
         if(lists[i])    pq.push({lists[i]->val , lists[i]});
      }
      ListNode* dummy = new ListNode(-1);
      ListNode* temp = dummy;

      while(!pq.empty()){
         auto it = pq.top();
         if(it.second->next){
            pq.push({it.second->next->val,it.second->next});
         }
         pq.pop();
         temp->next = it.second;
         temp=temp->next;
```

```
        }
        return dummy->next;
    }
};
```

OUTPUT: