ASSIGNMENT 2

Student Name: Atharva Deshmukh **UID: 22BCS12930**
Branch: B.E-C.S.E **Section/Group: 606-B**
Semester: 6th **Date of Performance: 06/03/25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22CSP-351**

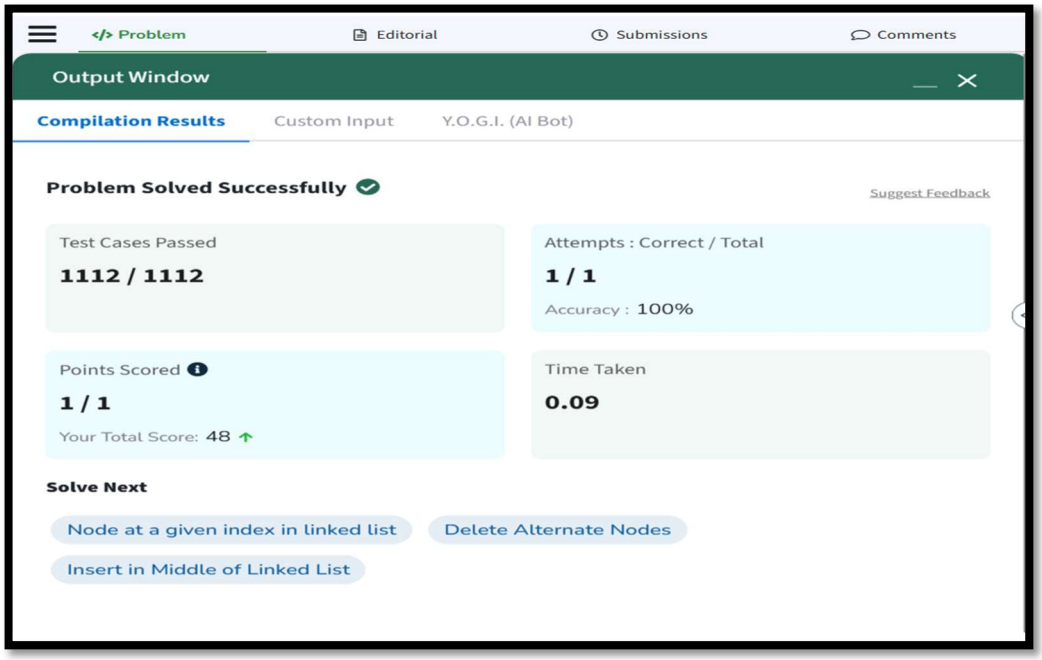
1. Print Linked List

Given a linked list. Print all the elements of the linked list separated by space followed.

Code Snippet

```
class Solution {
  public:
    void printList(Node *head) {
       Node* temp=head;
       while(temp){
          cout<<temp->data<<" ";
          temp=temp->next;
       }
    }
};
```

Submission



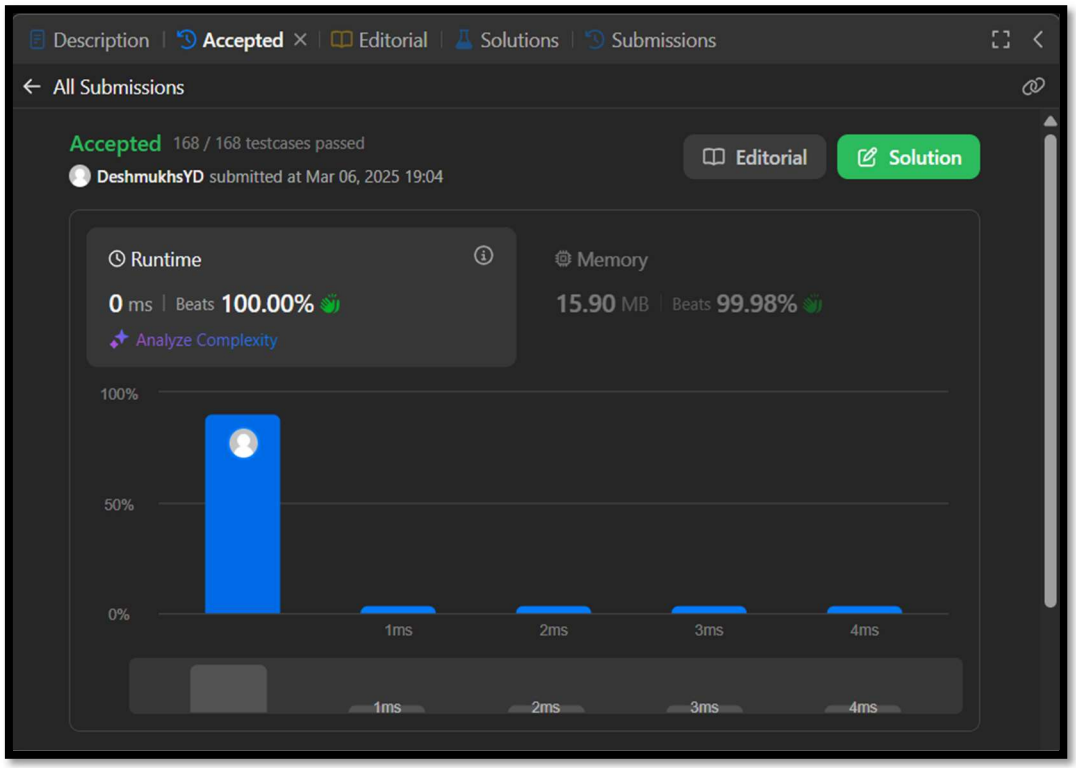
2. **Remove Duplicates from Sorted List**

Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Code Snippet

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        while (curr && curr->next) {
            if (curr->val == curr->next->val) {
                curr->next = curr->next->next;
            } else {
                curr = curr->next;
            }
        }
        return head;
    }
};
```

Submission



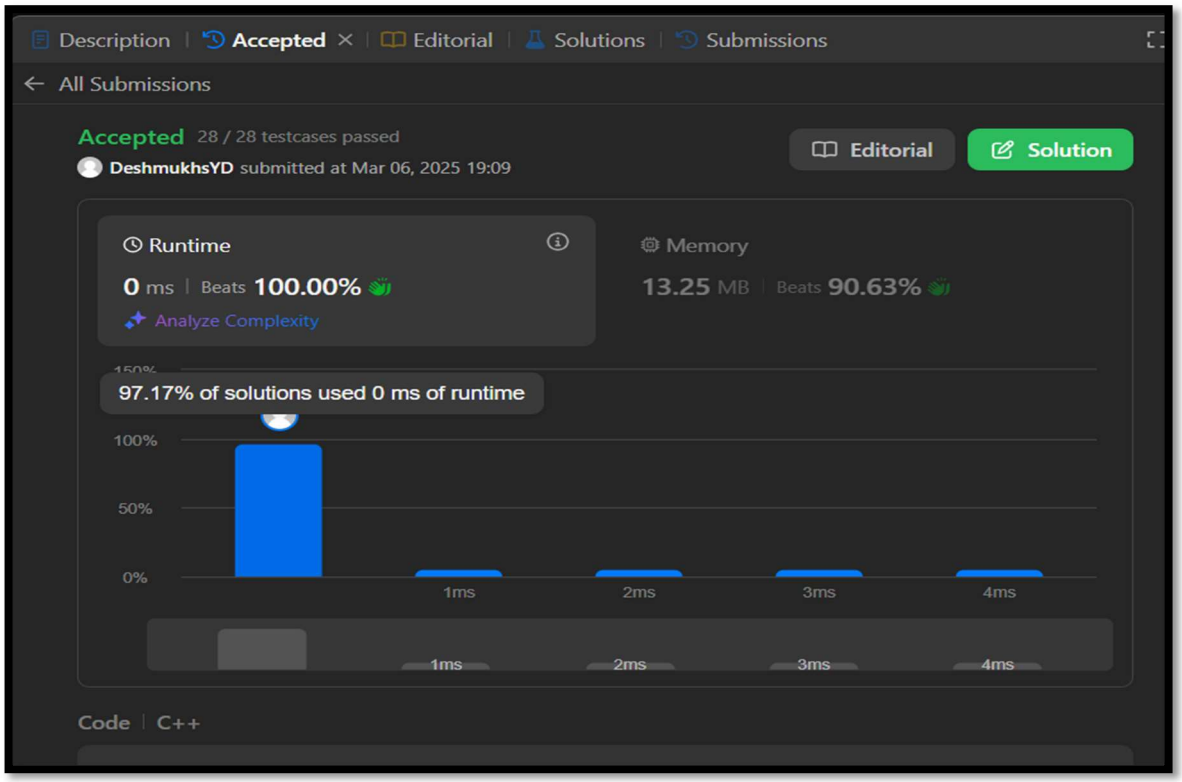
3. **Reverse Linked List**

Given the head of a singly linked list, reverse the list, and return the reversed list.

Code Snippet

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

Submissions



4. Delete the Middle Node of a Linked List

You are given the head of a linked list. Delete the middle node, and return the head of the modified linked list.

The middle node of a linked list of size n is the ⌊n / 2⌋th node from the start using 0-based indexing, where ⌊x⌋ denotes the largest integer less than or equal to x.

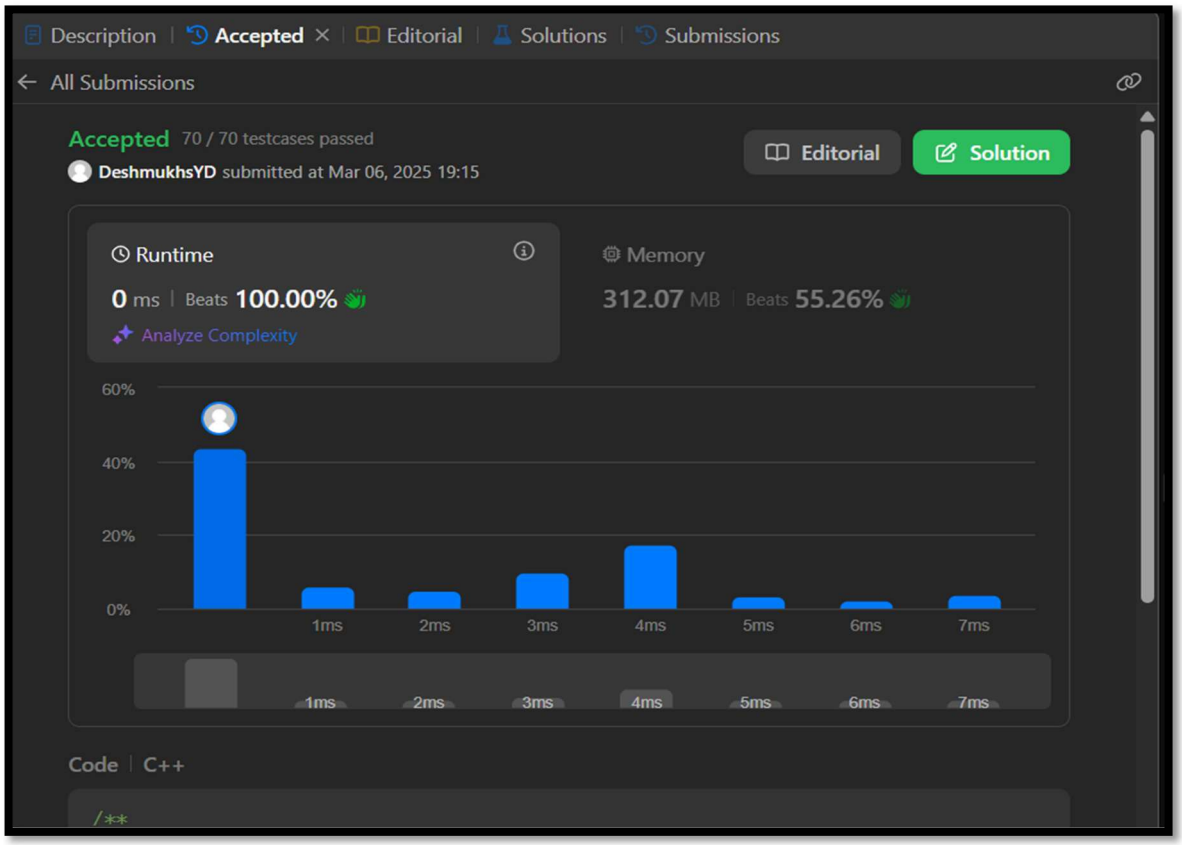
For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively.

Code Snippet

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

   }
};
```

Submission



5. Merge Two Sorted Lists

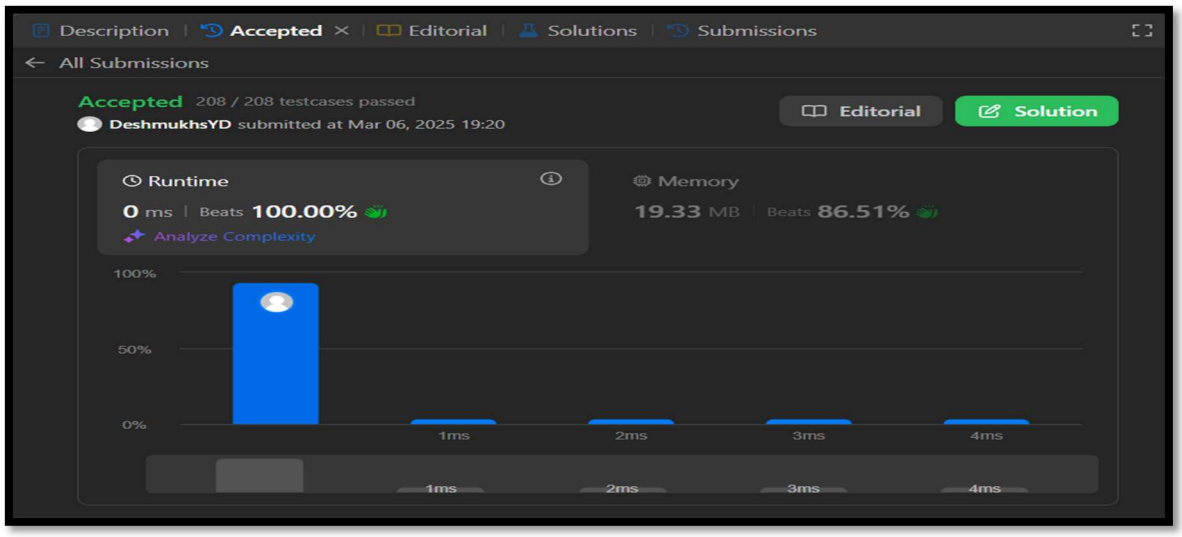
You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.
Return the head of the merged linked list.

Code Snippet

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

Submission



6. **Linked List Cycle**

Given head, the head of a linked list, determine if the linked list has a cycle in it.

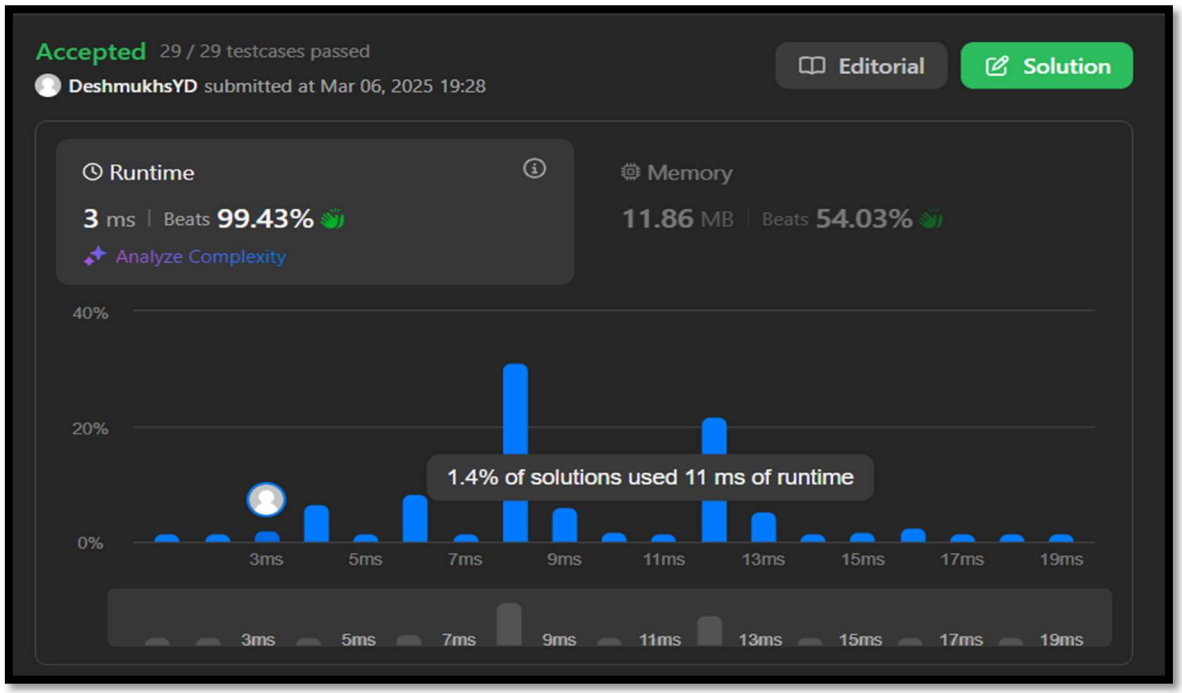
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Code Snippet

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

Submission



7. **Rotate List**

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

Code Snippet

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

```
        int len = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            len++;
        }
        k %= len;
        if (k == 0) return head;
        tail->next = head;
        int stepsToNewHead = len - k;
        ListNode* newTail = head;
        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        return newHead;

    }
};
```

Submission

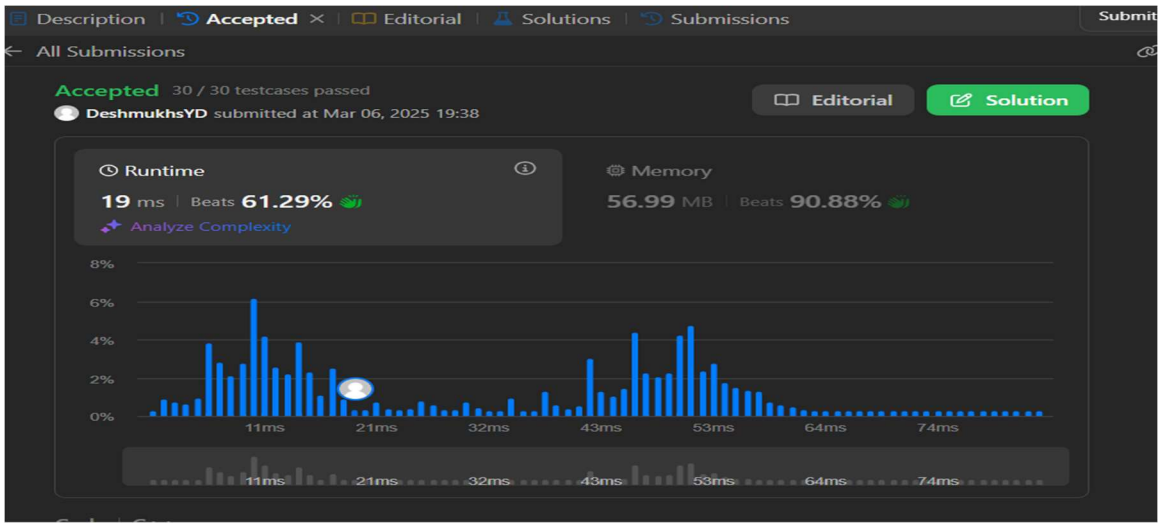
8. **Sort List**

Given the head of a linked list, return the list after sorting it in ascending order.

Code Snippet

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* mid = getMid(head);
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);
        return merge(left, right);
    }
private:
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head, *fast = head, *prev = nullptr;
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }
        prev->next = nullptr;
        return slow;
    }
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0), *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;
    }
};
```

Submission



9. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Code Snippet

```
#include <queue>
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        auto cmp = [](ListNode* a, ListNode* b) { return a->val > b->val; };
        priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> minHeap(cmp);
        for (ListNode* list : lists) {
            if (list) minHeap.push(list);
        }
        ListNode dummy(0), *tail = &dummy;
        while (!minHeap.empty()) {
            ListNode* node = minHeap.top();
            minHeap.pop();
            tail->next = node;
            tail = node;
            if (node->next) minHeap.push(node->next);
        }
        return dummy.next;
    };
```

Submission