DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Name- Pranav kumar Singh

UID-22BCS12894
Section-22BCS_IOT-606-B

Assignment-03

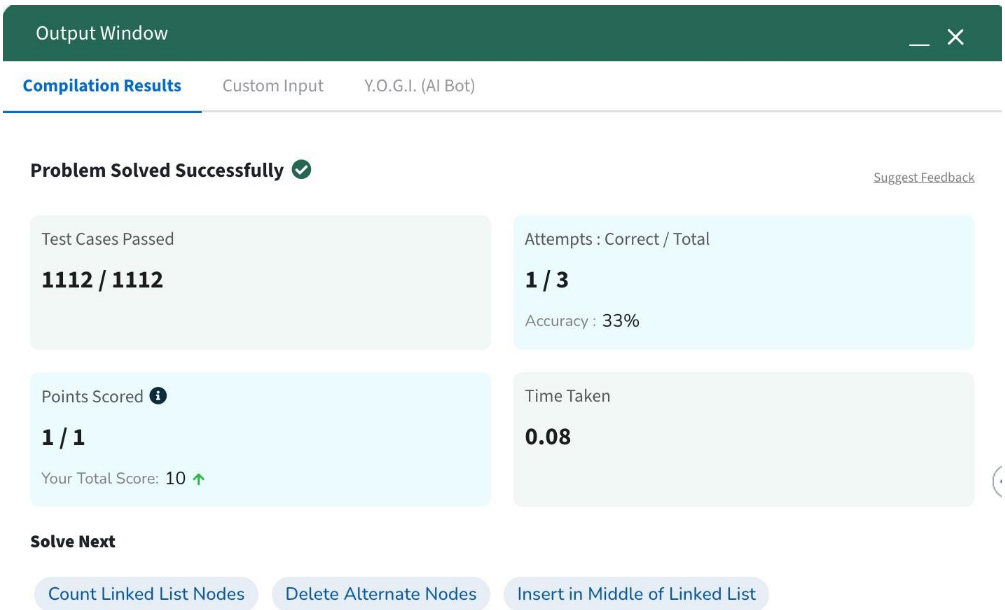
Advanced Programming Lab - 2 (22CSP-351)

Linked Lists

Question 1: Print Linked List

Code:

```
class Solution {
  public:
    // Function to display the elements of a linked list in the same line
    void printList(Node *head) {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data;
            if (temp->next != NULL) cout << " "; // Print space if not the last element
            temp = temp->next;
        }
    }
};
```



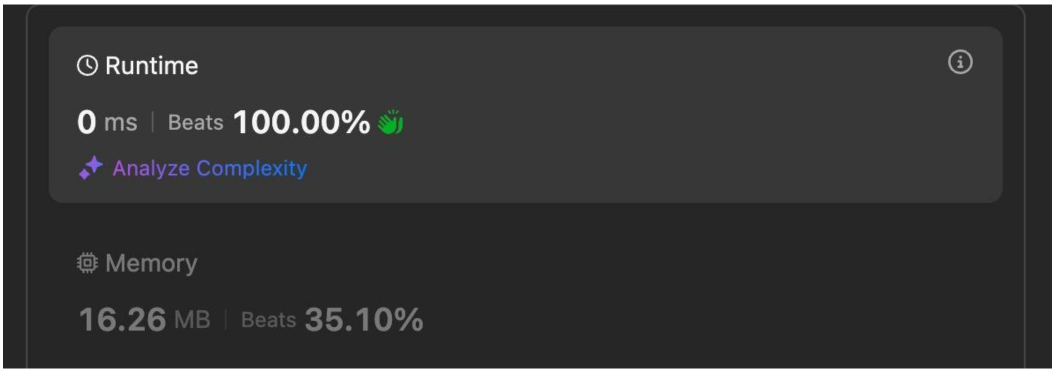
Question 2: Remove duplicates from a sorted list

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

        while (current != NULL && current->next != NULL) {
            if (current->val == current->next->val) {
                // Skip the duplicate node
```

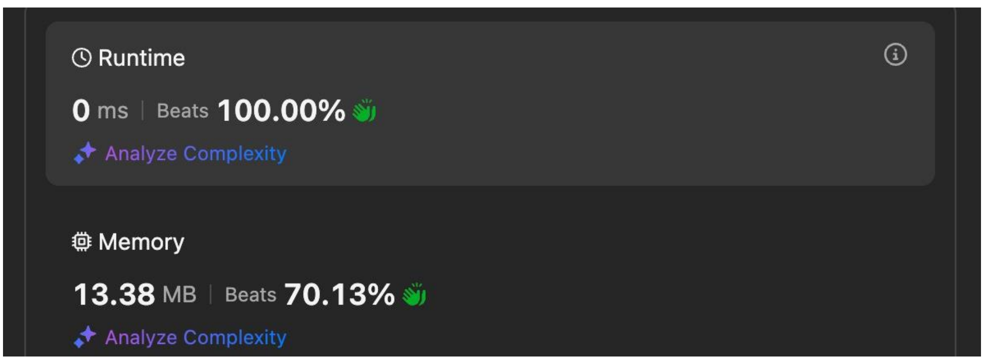
```
            current->next = current->next->next;
        } else {
            // Move to the next distinct node
            current = current->next;
        }
    }

    return head;
  }
};
```

Question 3: Reverse a linked list:

Code:




```
class Solution {
public:
  ListNode* reverseList(ListNode* head) {
    ListNode* prev = NULL;
    ListNode* current = head;
    ListNode* next = NULL;
    while (current != NULL) {
      next = current->next; // Store next node
      current->next = prev; // Reverse the current node's pointer
      prev = current;      // Move prev to current node
      current = next;      // Move current to next node
    }
    return prev; // New head of the reversed list
} };
```



Question 4: Delete middle node of a list

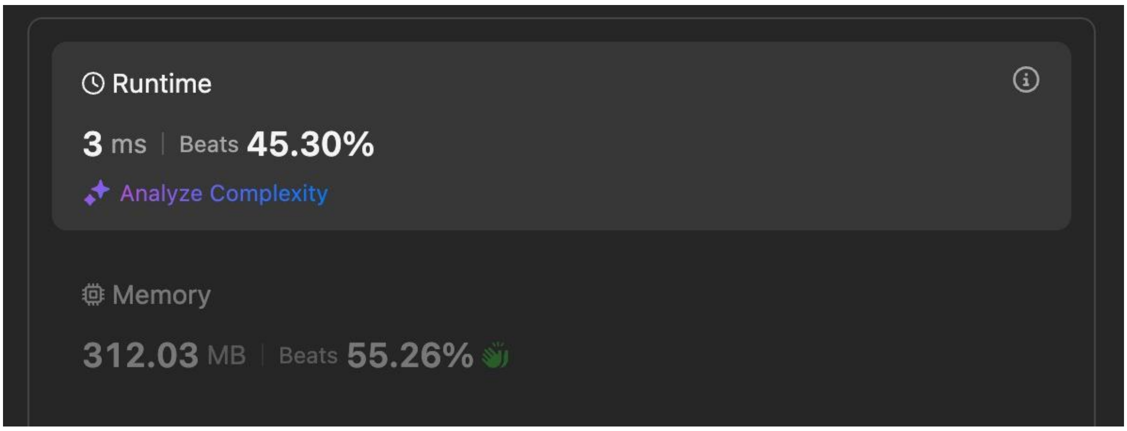
Code:

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        // If the list has only one node, return NULL (empty list)
        if (head == NULL || head->next == NULL)
            return NULL;

        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = NULL;

        // Use slow and fast pointers to find the middle node
        while (fast != NULL && fast->next != NULL) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        // Delete the middle node by skipping it
        prev->next = slow->next;
        delete slow;

        return head;
    }
};
```

Question 5: Merge two sorted linked lists

Code:

```
class Solution {
```



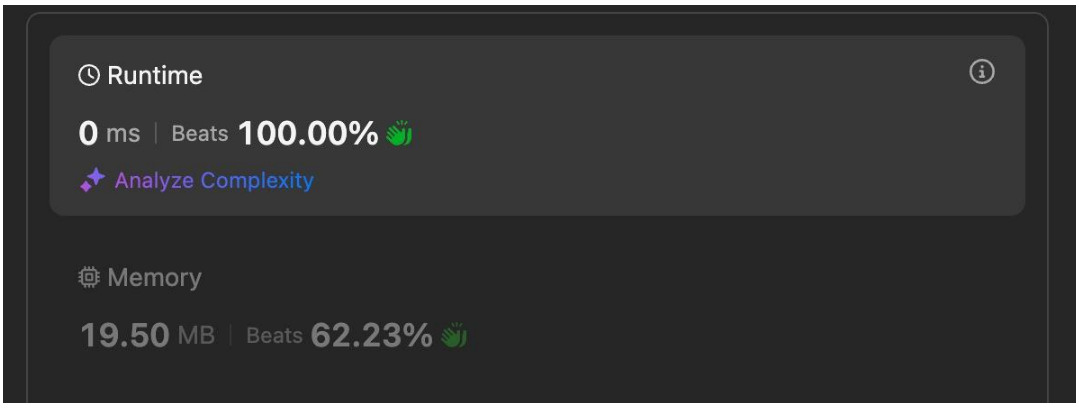

```
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        // If one list is empty, return the other list
        if (!list1) return list2;
        if (!list2) return list1;
        ListNode* dummy = new ListNode(-1);
        ListNode* current = dummy;
```

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

        if (list1) current->next = list1;
        if (list2) current->next = list2;

        return dummy->next; // Return the merged list starting from the first real node
    }
};
```

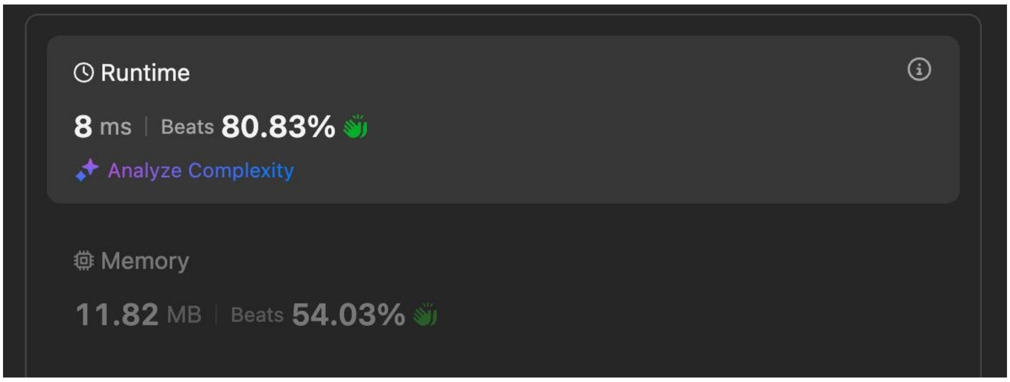
Question 6: Detect a cycle in a linked list

Code:

```
class Solution {
```



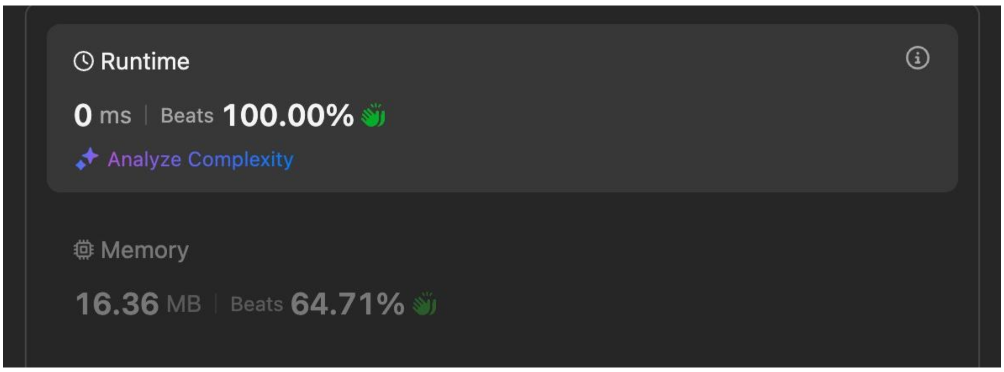

```
public:
    bool hasCycle(ListNode *head) {
        if (!head || !head->next) return false; // Edge case: empty list or single node without cycle

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

        while (fast && fast->next) {
            slow = slow->next;        // Move slow by 1 step
            fast = fast->next->next; // Move fast by 2 steps

            if (slow == fast) return true; // Cycle detected
        }

        return false; // No cycle found
    }
};
```



Question 7:Rotate a list

Code:

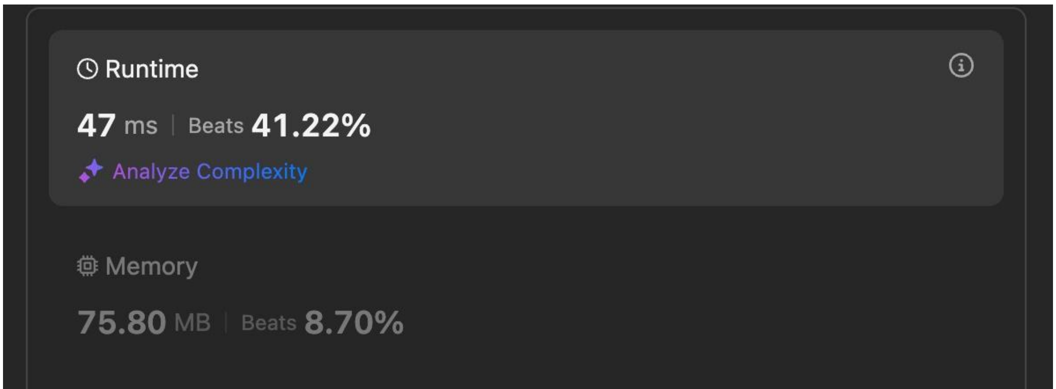
```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head; // Edge cases
        // Step 1: Find the length of the list
        int n = 1; // At least one node exists
        ListNode* tail = head;
        while (tail->next) {
            tail  =  tail->next;
            n++;
        }
        // Step 2: Optimize k
        k = k % n;
        if (k == 0) return head; // No rotation needed
        // Step 3: Find new tail (n-k-1) and new head (n-k)
        ListNode* newTail = head;
        for (int i = 0; i < n - k - 1; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next; // New head
        // Step 4: Rearrange pointers
        newTail->next = nullptr; // Break the old connection
        tail->next = head;       // Connect the old tail to old head
        return newHead; // Return the new head
    }
};
```



Question 8: Sort List

Code:

```
class Solution {
public:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode(0);
        ListNode* current = dummy;
        while (l1 && l2) {
            if (l1->val < l2->val) {
                current->next = l1;
                l1 = l1->next;
            } else {
                current->next = l2;
                l2 = l2->next;
            }
            current = current->next;
        }
        if (l1) current->next = l1;
        if (l2) current->next = l2;
        return dummy->next;
    }
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;

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



Question 9: Merge k sorted lists

Code:

```
#include <queue>
class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val; // Min-heap based on node values
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
        for (auto list : lists) {
            if (list) minHeap.push(list);
        }
        ListNode dummy(0); // Dummy node for ease of handling
        ListNode* tail = &dummy;
        while (!minHeap.empty()) {
            ListNode* smallest = minHeap.top();
            minHeap.pop();
            tail->next = smallest;
            tail = tail->next;
            if (smallest->next) {
                minHeap.push(smallest->next); // Add the next node to the heap
            }
        }
        return dummy.next; // Return the merged list
    }
};
```