ASSIGNMENT 3

ADVANCE PROGRAMMING LAB-II

Date: 06/03/25

Submitted by: Shivansh Rattan

UID: 22BCS16351

Section: 610-B

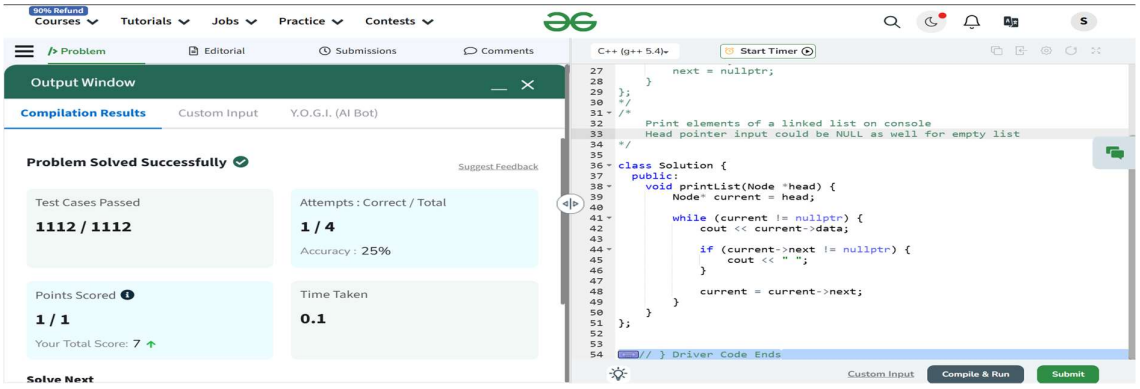
Submitted to: Pratima mam

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

Code:

```
class Solution {
  public:
    void printList(Node *head) {
        Node* current = head;

        while (current != nullptr) {
            cout << current->data;

            if (current->next != nullptr) {
                cout << " ";
            }

            current = current->next;
        }
    }
};
```

Output:

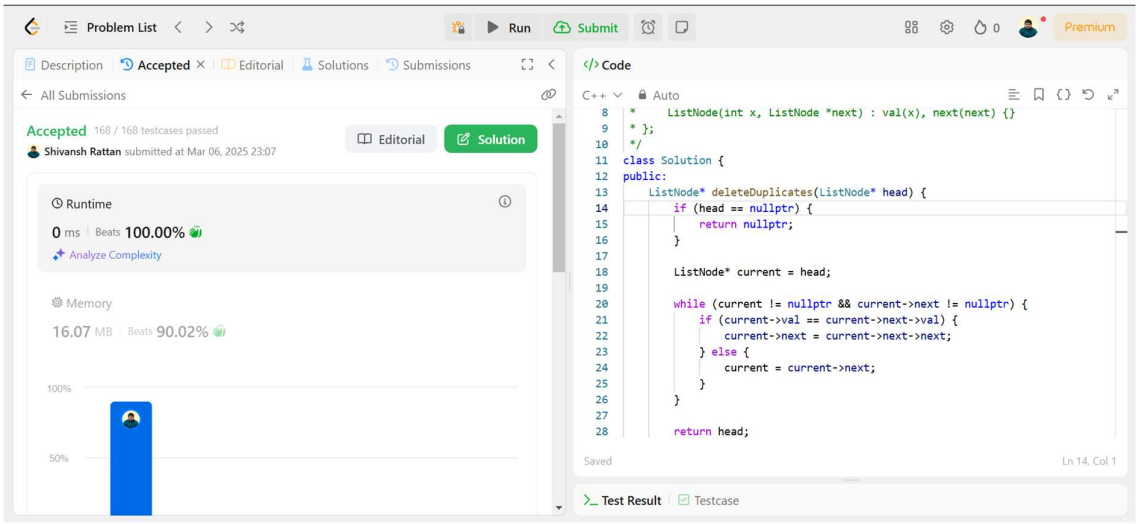


2. Remove duplicates from a sorted list: https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/

Code:

```
</> Code
C++ Auto
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (head == nullptr) {
            return nullptr;
        }

        ListNode* current = head;

        while (current != nullptr && current->next != nullptr) {
            if (current->val == current->next->val) {
                current->next = current->next->next;
            } else {
                current = current->next;
            }
        }

        return head;
```

Output:

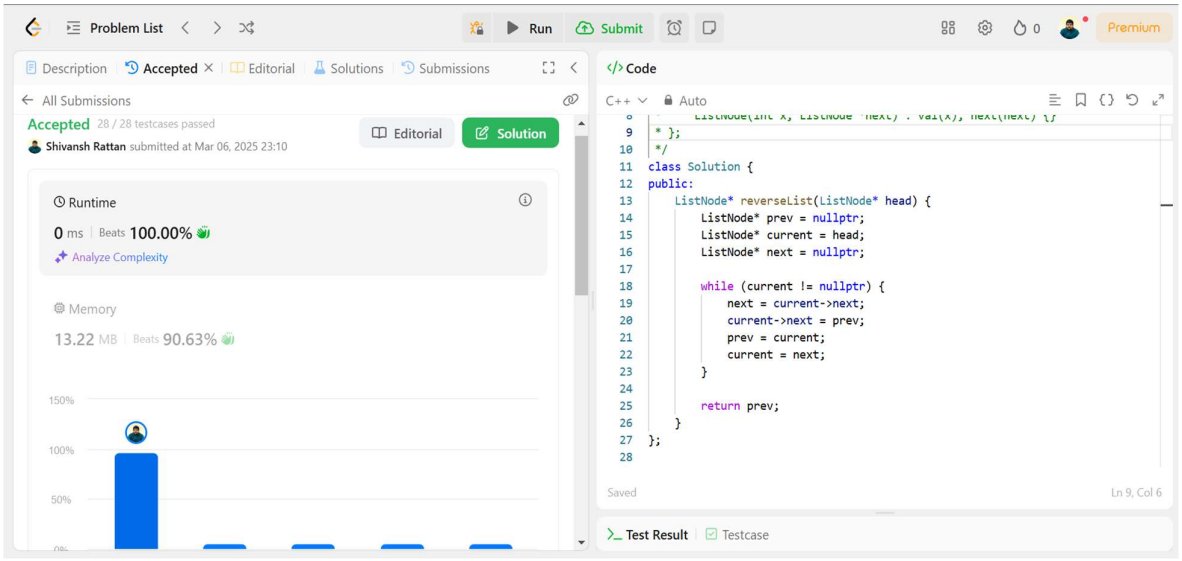


3. Reverse a linked list: https://leetcode.com/problems/reverse-linked-list/description/

Code:

```
</> Code
C++ Auto
 9  * };
10  */
11  class Solution {
12  public:
13      ListNode* reverseList(ListNode* head) {
14          ListNode* prev = nullptr;
15          ListNode* current = head;
16          ListNode* next = nullptr;
17
18          while (current != nullptr) {
19              next = current->next;
20              current->next = prev;
21              prev = current;
22              current = next;
23          }
24
25          return prev;
26      }
27  };
```

Output:

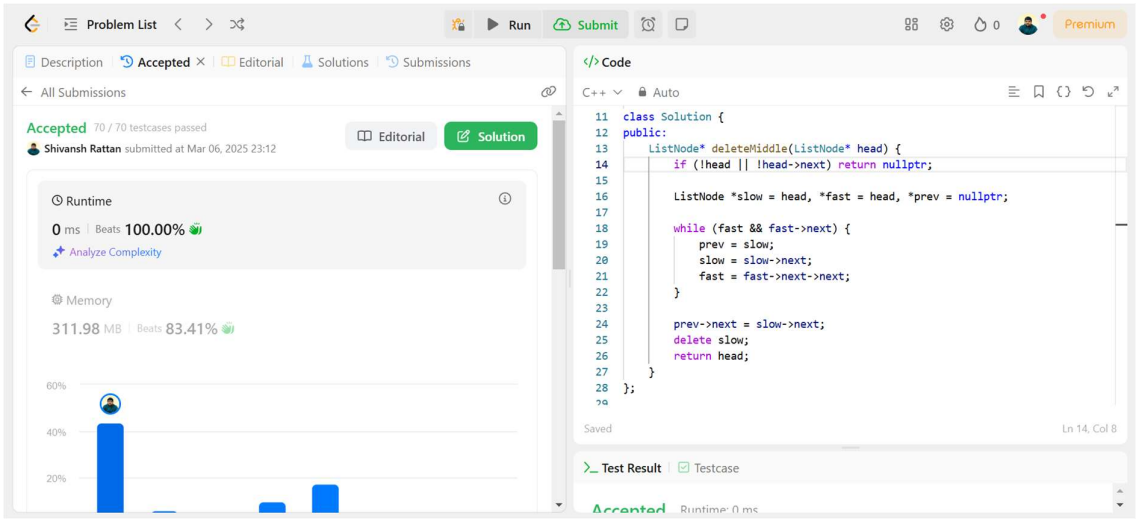


4. Delete middle node of a list: https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/

Code:

```cpp
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;
    }
};
```

Output:

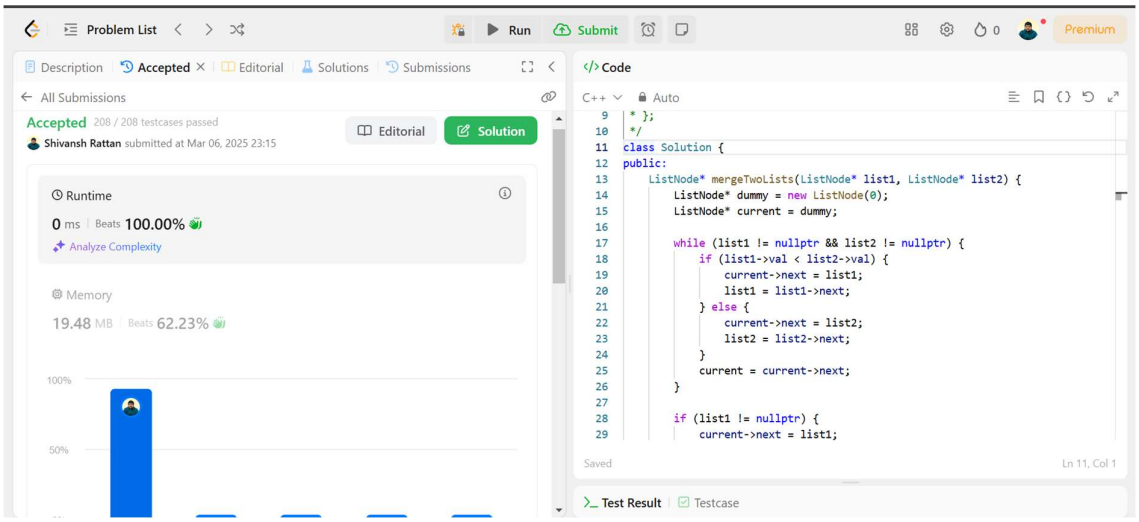


5. Merge two sorted linked lists: https://leetcode.com/problems/merge-two-sorted-lists/description/

Code:

```cpp
* };
*/
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dummy = new ListNode(0);
        ListNode* current = dummy;

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

        if (list1 != nullptr) {
            current->next = list1;
        } else if (list2 != nullptr) {
            current->next = list2;
        }

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

Output:

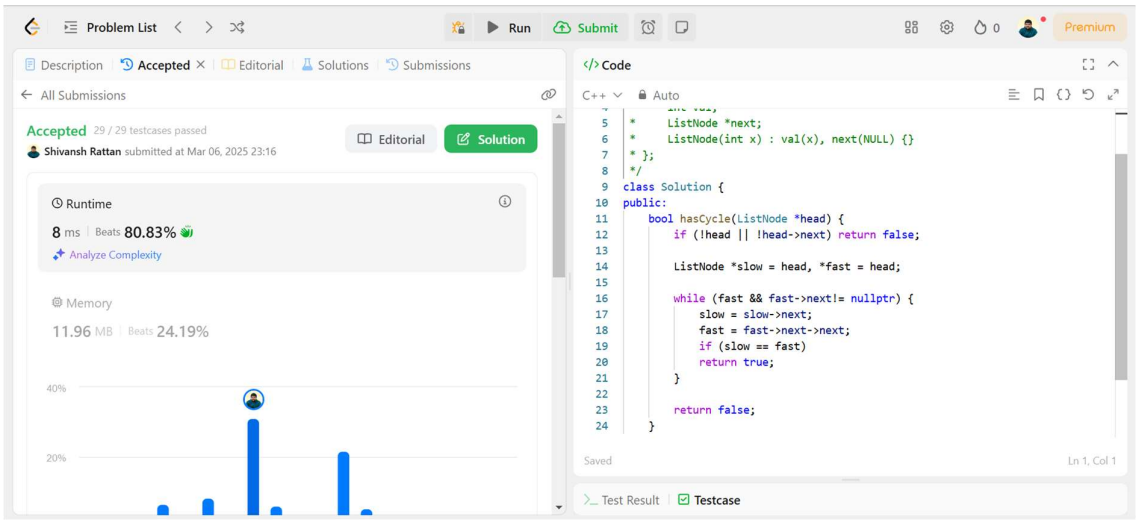


6. Detect a cycle in a linked list: https://leetcode.com/problems/linked-list-cycle/description/

Code:

```
*/
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (!head || !head->next) return false;

        ListNode *slow = head, *fast = head;

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

        return false;
    }
};
```

Output:

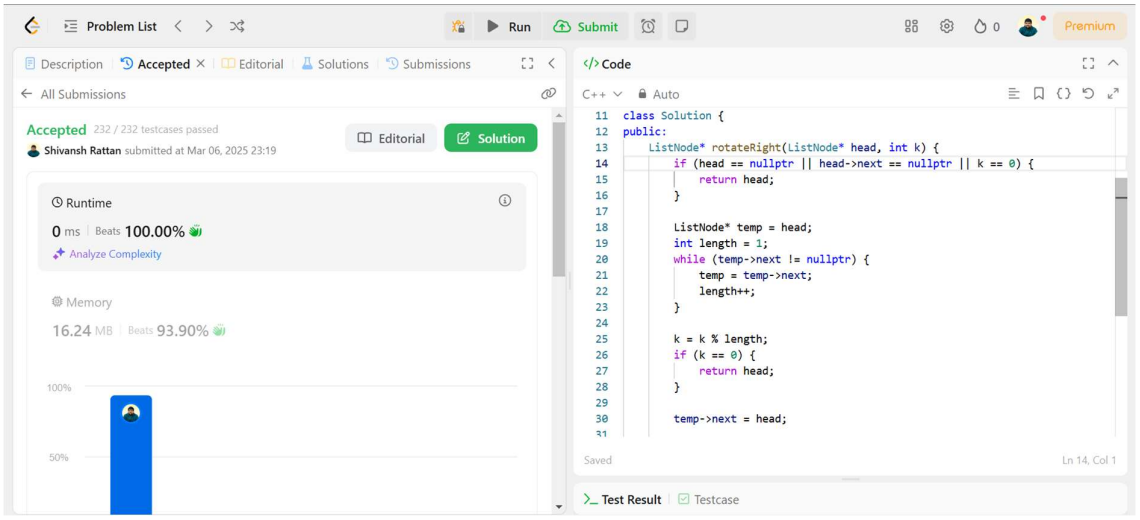


7. Rotate a list: https://leetcode.com/problems/rotate-list/description/

Code:

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

        ListNode* temp = head;
        int length = 1;
        while (temp->next != nullptr) {
            temp = temp->next;
            length++;
        }

        k = k % length;
        if (k == 0) {
            return head;
        }

        temp->next = head;

        ListNode* newTail = head;
        for (int i = 0; i < length - k - 1; i++) {
            newTail = newTail->next;
        }

        ListNode* newHead = newTail->next;

        newTail->next = nullptr;

        return newHead;
    }
```

Output:

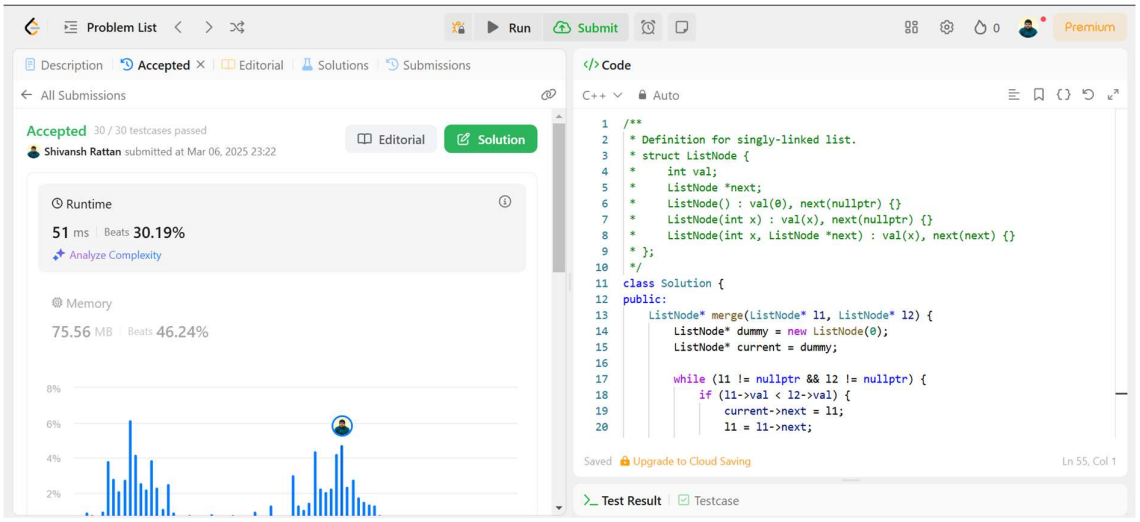


8. Sort List: https://leetcode.com/problems/sort-list/description/

Code:

```cpp
*/
class Solution {
public:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode(0);
        ListNode* current = dummy;

        while (l1 != nullptr && l2 != nullptr) {
            if (l1->val < l2->val) {
                current->next = l1;
                l1 = l1->next;
            } else {
                current->next = l2;
                l2 = l2->next;
            }
            current = current->next;
        }

        if (l1 != nullptr) {
            current->next = l1;
        } else {
            current->next = l2;
        }

        return dummy->next;
    }

    ListNode* sortList(ListNode* head) {
        if (head == nullptr || head->next == nullptr) {
            return head;
        }

        ListNode *slow = head, *fast = head, *prev = nullptr;

        while (fast != nullptr && fast->next != nullptr) {
            prev = slow;
            slow = slow->next;
```

Output:

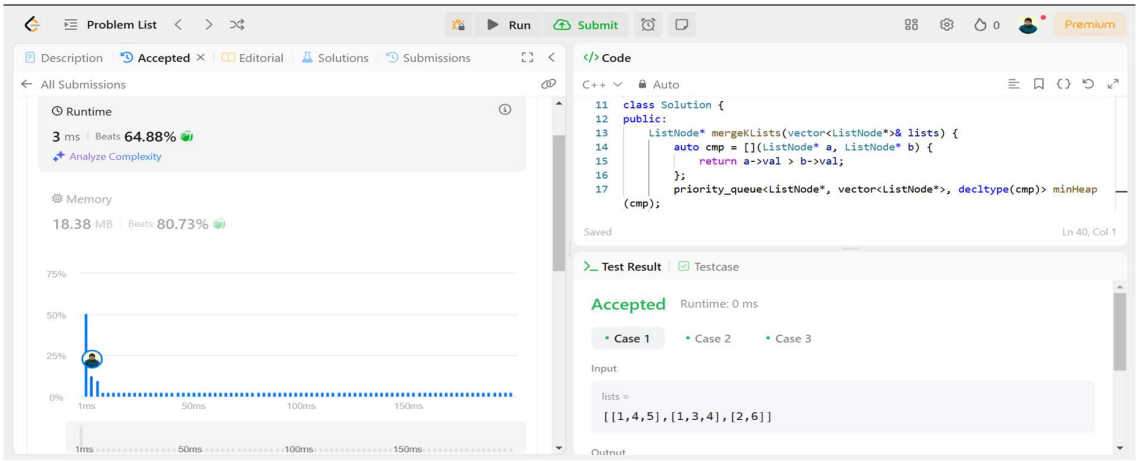


9. Merge k sorted lists: https://leetcode.com/problems/merge-k-sorted-lists/description/

Code:

```cpp
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 != nullptr) {
                minHeap.push(list);
            }
        }

        ListNode* dummy = new ListNode();
        ListNode* current = dummy;

        while (!minHeap.empty()) {
            ListNode* node = minHeap.top();
            minHeap.pop();

            current->next = node;
            current = current->next;

            if (node->next != nullptr) {
                minHeap.push(node->next);
            }
        }

        return dummy->next;
    }
```

Output:



The End