



Assignment-3

Name – Vedant Aggarwal

UID – 22BCS13945

Semester - 6

Date - 06-03-2025

Subject - Advanced Programming Lab-2

Subject Code - 22CSP-351

- **Linked Lists:**

1. **Print Linked List-**

```
class Solution {
public:
    void printList(Node *head) {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) cout << " "; // Print
space only if not the last node
            temp = temp->next; // Move to the next node
(Fix for infinite loop)
        }
        cout << endl; // Print newline for correct
formatting
    }
};
```

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1112 / 1112

Attempts : Correct / Total
1 / 3
Accuracy : 33%

Points Scored
1 / 1
Your Total Score: 10

Time Taken
0.08

Solve Next

Count Linked List Nodes Delete Alternate Nodes Insert in Middle of Linked List

2. Remove duplicates from a sorted list-

The screenshot shows a LeetCode submission for the problem "Remove Duplicates from a Sorted List". The submission is in C++ and has a status of "Accepted". The code is as follows:

```
1 class Solution {
2 public:
3     ListNode* deleteDuplicates(ListNode* head) {
4         ListNode* current = head;
5
6         while (current && current->next) {
7             if (current->val == current->next->val) {
8                 current->next = current->next->next; // Skip duplicate node
9             } else {
10                 current = current->next; // Move to next node
11             }
12         }
13
14         return head;
15     }
16 };
```

The test case shows an input of `head = [1,1,2]` and an output of `[1,2]`. The runtime is 0 ms and the memory usage is 16.2 MB.

3. Reverse a linked list-

The screenshot shows a LeetCode submission for the problem "Reverse a linked list". The submission is in C++ and has a status of "Accepted". The code is as follows:

```
1 class Solution {
2 public:
3     ListNode* reverseList(ListNode* head) {
4         ListNode* prev = nullptr;
5         ListNode* current = head;
6
7         while (current) {
8             ListNode* nextNode = current->next; // Store next node
9             current->next = prev; // Reverse the link
10            prev = current; // Move prev ahead
11            current = nextNode; // Move current ahead
12        }
13
14        return prev; // New head of the reversed list
15    }
16 };
17
```

The test case shows an input of `head = [1,2,3,4,5]`. The runtime is 0 ms and the memory usage is 13.3 MB.

4. Delete middle node of a list-

The screenshot shows the LeetCode interface for the problem "Delete the Middle Node of a Linked List". The submission is accepted, with a runtime of 0 ms and memory usage of 312.2 MB. The code is written in C++ and implements a solution to delete the middle node of a linked list.

```
1 class Solution {
2 public:
3     ListNode* deleteMiddle(ListNode* head) {
4         if (!head || !head->next) return nullptr; // Edge case: Only one node
5
6         ListNode* slow = head;
7         ListNode* fast = head;
8         ListNode* prev = nullptr; // To track the node before the middle
9
10        while (fast && fast->next) {
11            prev = slow;
12            slow = slow->next;
13            fast = fast->next->next;
14        }
15
16        prev->next = slow->next; // Remove the middle node
17
18        return head;
19    }
20 };
```

The test result shows the submission is accepted for all three test cases.

5. Merge two sorted linked lists-

The screenshot shows the LeetCode interface for the problem "Merge Two Sorted Linked Lists". The submission is accepted, with a runtime of 0 ms and memory usage of 19.7 MB. The code is written in C++ and implements a solution to merge two sorted linked lists.

```
1 class Solution {
2 public:
3     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
4         ListNode dummy(0); // Dummy node to simplify edge cases
5         ListNode* current = &dummy;
6
7         while (list1 && list2) {
8             if (list1->val < list2->val) {
9                 current->next = list1;
10                list1 = list1->next;
11            } else {
12                current->next = list2;
13                list2 = list2->next;
14            }
15            current = current->next;
16        }
17
18        // Attach the remaining nodes (if any)
19        if (list1) current->next = list1;
20        if (list2) current->next = list2;
21    }
22 };
23
```

The test result shows the submission is accepted for all three test cases.

6. Detect a cycle in a linked list-

The screenshot shows a LeetCode submission for the problem "Detect a cycle in a linked list". The submission is in C++ and has a runtime of 6 ms and a memory usage of 12 MB. The code is as follows:

```
1 class Solution {
2 public:
3     bool hasCycle(ListNode *head) {
4         ListNode* slow = head;
5         ListNode* fast = head;
6
7         while (fast && fast->next) {
8             slow = slow->next; // Move slow one step
9             fast = fast->next->next; // Move fast two steps
10
11             if (slow == fast) return true; // Cycle detected
12         }
13
14         return false; // No cycle
15     }
16 };
17
```

The test result shows "Accepted" with a runtime of 2 ms. The input is "head = [3,2,0,-4]".

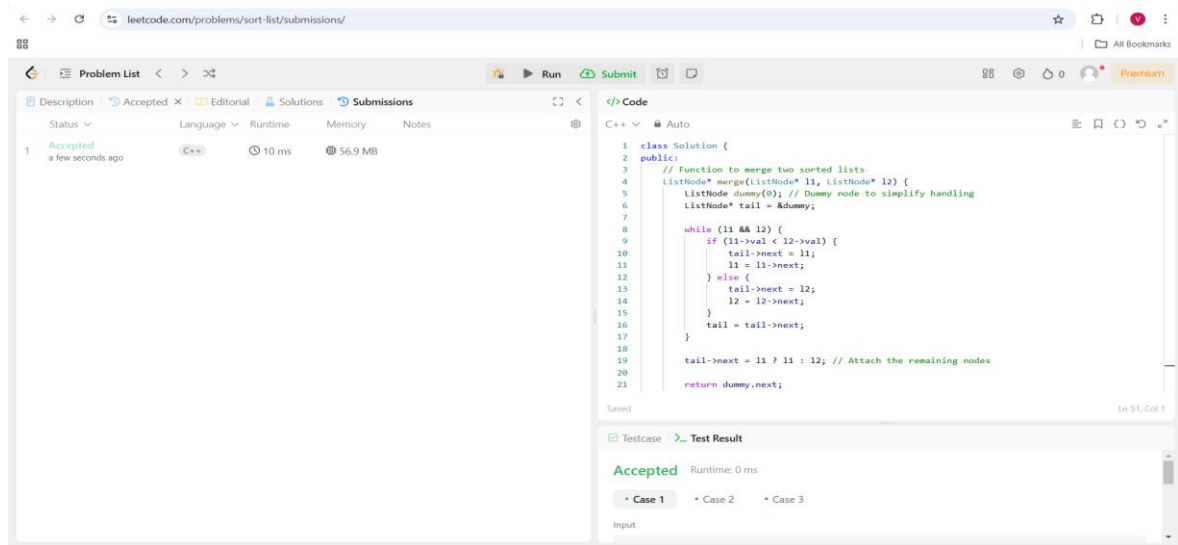
7. Rotate a list-

The screenshot shows a LeetCode submission for the problem "Rotate a list". The submission is in C++ and has a runtime of 0 ms and a memory usage of 16.3 MB. The code is as follows:

```
1 class Solution {
2 public:
3     ListNode* rotateRight(ListNode* head, int k) {
4         if (!head || !head->next || k == 0) return head; // Edge cases
5
6         // Step 1: Compute the length of the list
7         int length = 1;
8         ListNode* tail = head;
9         while (tail->next) {
10             tail = tail->next;
11             length++;
12         }
13
14         // Step 2: Optimize k
15         k = k % length;
16         if (k == 0) return head; // No rotation needed
17
18         // Step 3: Find the new tail (length - k - 1 th node)
19         ListNode* newTail = head;
20         for (int i = 0; i < length - k - 1; i++) {
21             newTail = newTail->next;
22         }
23
24         // Step 4: Set the new head and break the cycle
25         ListNode* newHead = newTail->next;
26         newTail->next = nullptr;
27     }
28 };
29
```

The test result shows "Accepted" with a runtime of 0 ms.

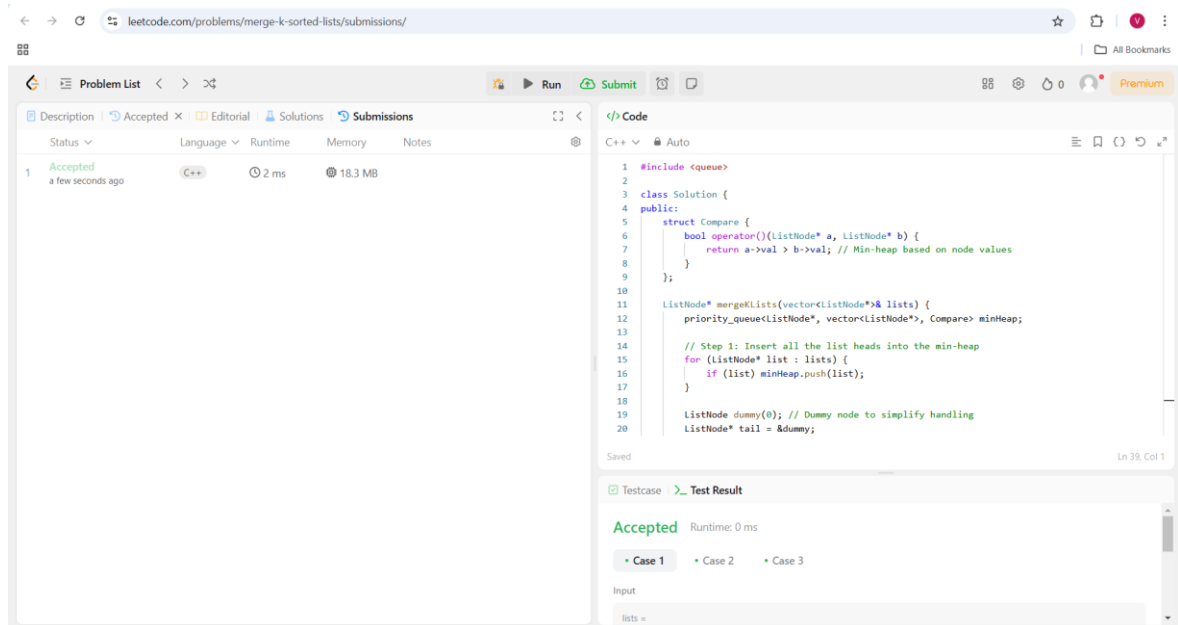
8. Sort List-



The screenshot shows a LeetCode submission for the "Sort List" problem. The submission is in C++ and has a runtime of 10 ms and memory usage of 56.9 MB. The code implements a merge sort algorithm for a linked list.

```
1 class Solution {
2 public:
3     // Function to merge two sorted lists
4     ListNode* merge(ListNode* l1, ListNode* l2) {
5         ListNode dummy(0); // Dummy node to simplify handling
6         ListNode* tail = &dummy;
7
8         while (l1 && l2) {
9             if (l1->val < l2->val) {
10                 tail->next = l1;
11                 l1 = l1->next;
12             } else {
13                 tail->next = l2;
14                 l2 = l2->next;
15             }
16             tail = tail->next;
17         }
18         tail->next = l1 ? l1 : l2; // Attach the remaining nodes
19
20         return dummy->next;
21 }
```

9. Merge k sorted lists-



The screenshot shows a LeetCode submission for the "Merge k sorted lists" problem. The submission is in C++ and has a runtime of 2 ms and memory usage of 18.3 MB. The code uses a min-heap to merge k sorted linked lists.

```
1 #include <queue>
2
3 class Solution {
4 public:
5     struct Compare {
6         bool operator()(ListNode* a, ListNode* b) {
7             return a->val > b->val; // Min-heap based on node values
8         }
9     };
10
11     ListNode* mergeKLists(vector<ListNode*>& lists) {
12         priority_queue<ListNode*, vector<ListNode*&, Compare> minHeap;
13
14         // Step 1: Insert all the list heads into the min-heap
15         for (ListNode* list : lists) {
16             if (list) minHeap.push(list);
17         }
18
19         ListNode dummy(0); // Dummy node to simplify handling
20         ListNode* tail = &dummy;
```