

ASSIGNMENT -3

Student Name: Tanisha Mahajan

UID: 22BCS11132

Branch: BE-CSE

Section/Group: 22BCS_TPP_607-B

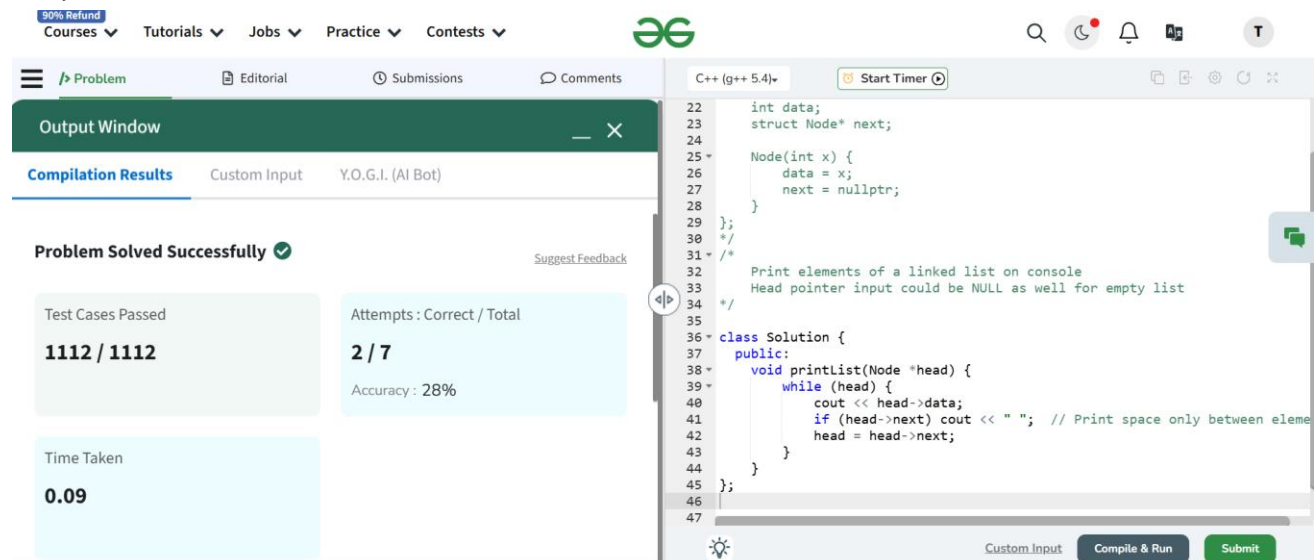
Semester: 6TH

Subject Name : AP-11

○ Linked List :

1. Print linked list :

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



The screenshot displays a C++ IDE interface. On the left, a sidebar shows navigation options: Courses, Tutorials, Jobs, Practice, and Contests. Below this, the 'Problem' tab is active, showing 'Compilation Results' for 'Custom Input' using 'Y.O.G.I. (AI Bot)'. A green checkmark indicates 'Problem Solved Successfully'. Test cases passed: 1112 / 1112. Attempts: 2 / 7. Accuracy: 28%. Time Taken: 0.09. The main editor shows C++ code for a linked list. The code defines a Node structure and a Solution class with a printList method. The printList method iterates through the linked list, printing each node's data followed by a space. The code is as follows:

```
22 int data;  
23 struct Node* next;  
24  
25 Node(int x) {  
26     data = x;  
27     next = nullptr;  
28 }  
29  
30 //  
31 /*  
32 Print elements of a linked list on console  
33 Head pointer input could be NULL as well for empty list  
34 */  
35  
36 class Solution {  
37 public:  
38     void printList(Node *head) {  
39         while (head) {  
40             cout << head->data;  
41             if (head->next) cout << " "; // Print space only between eleme  
42             head = head->next;  
43         }  
44     }  
45 };  
46  
47
```

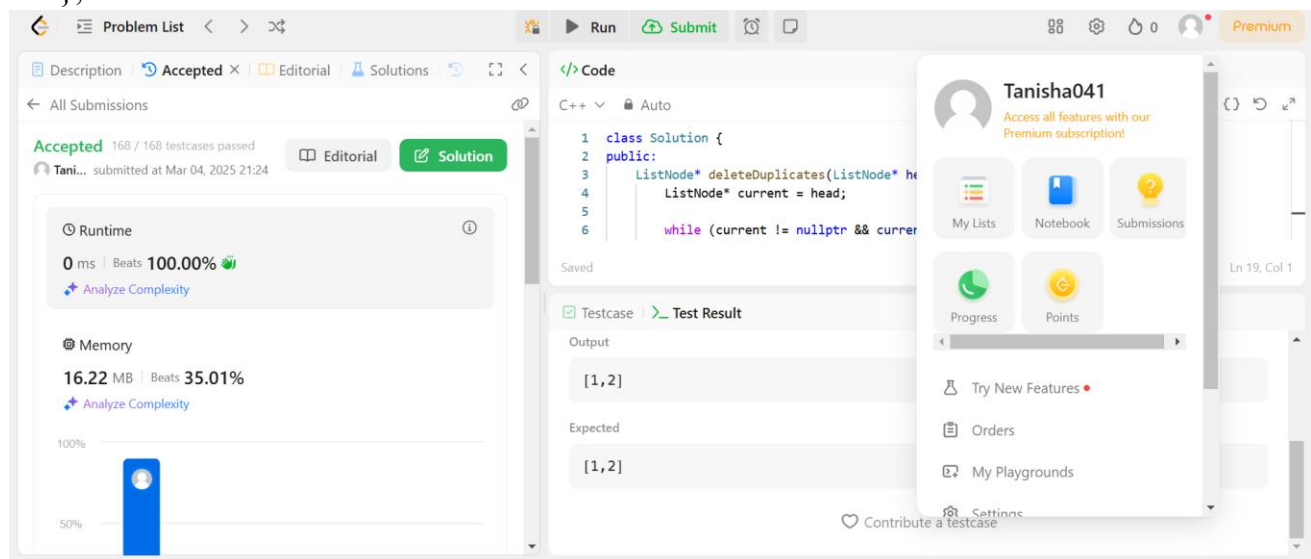
At the bottom of the IDE, there are buttons for 'Custom Input', 'Compile & Run', and 'Submit'.

2. Remove duplicates from the sorted list :

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

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

        return head;
    }
};
```



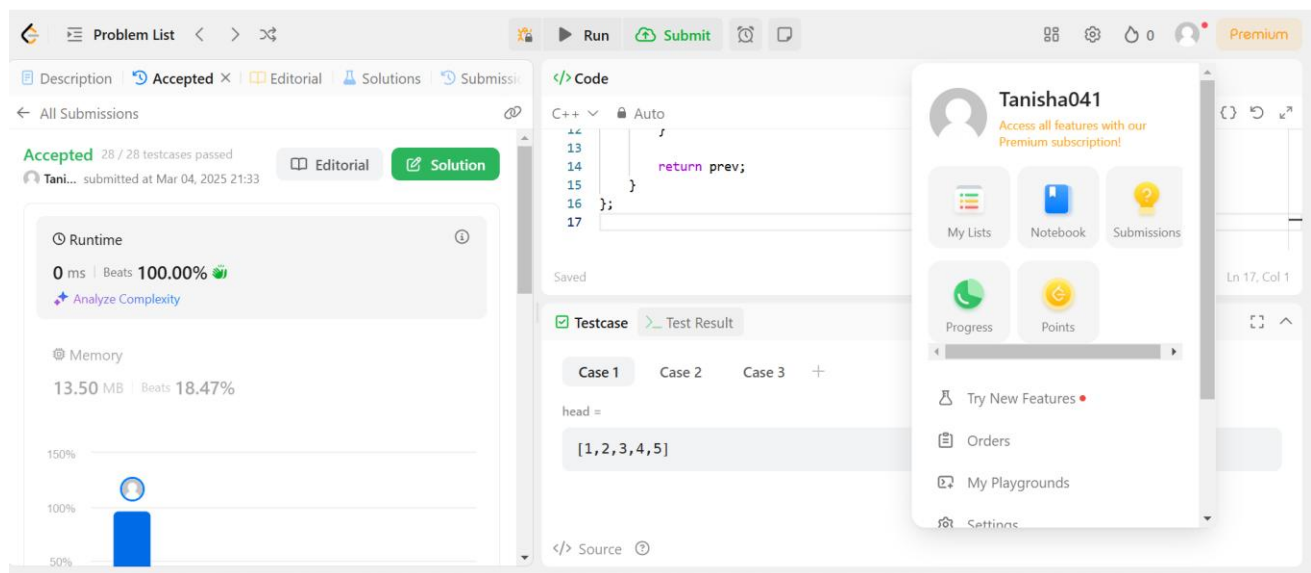
3. Reverse a Linked list :

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
```

```

ListNode* prev = nullptr;
ListNode* curr = head;
while (curr != nullptr) {
    ListNode* nextNode = curr->next;
    curr->next = prev;
    prev = curr;
    curr = nextNode;
}
return prev;
}
};

```



4. Delete middle node of linked list :

```

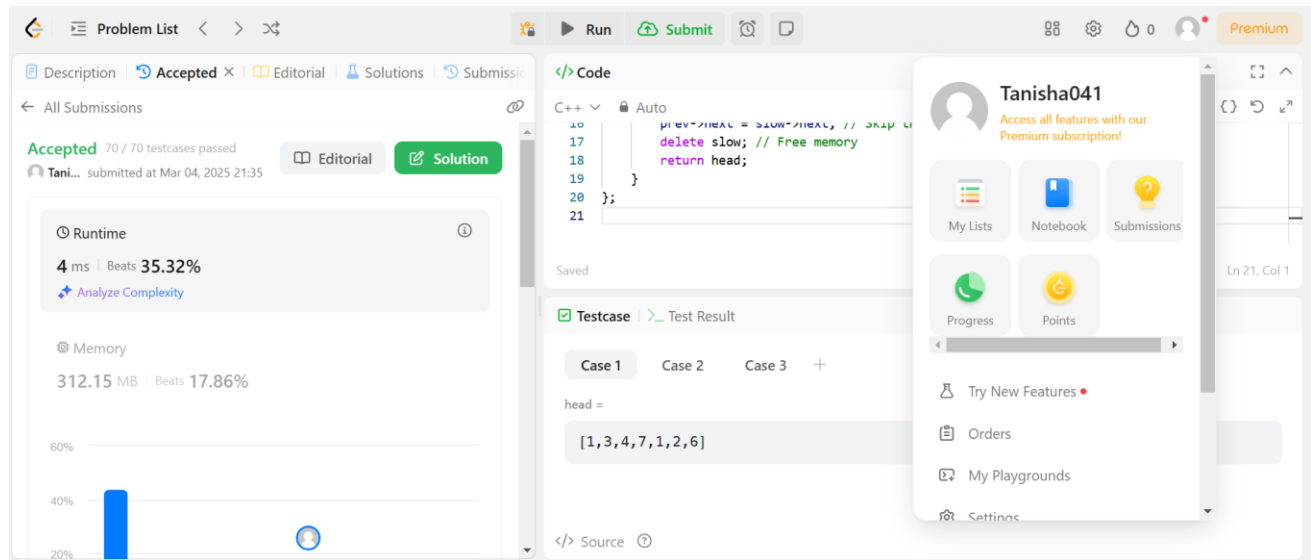
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr;
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;
        while (fast && fast->next) {
            prev = slow;

```

```

        slow = slow->next;
        fast = fast->next->next;
    }
    prev->next = slow->next;
    delete slow;
    return head;
}
};

```



The screenshot shows a coding platform interface with the following elements:

- Problem List:** A tab at the top left showing the current problem.
- Accepted:** A status indicator showing 70 / 70 testcases passed.
- Runtime:** 4 ms | Beats 35.32%.
- Memory:** 312.15 MB | Beats 17.86%.
- Code Editor:** A C++ code editor showing the implementation of a linked list deletion function. The code includes comments like `// Keep track of previous node` and `// Free memory`.
- Testcase:** A section showing the input array `[1, 3, 4, 7, 1, 2, 6]` and the output `head = [1, 3, 4, 7, 1, 2, 6]`.
- User Profile:** A sidebar on the right showing the user's name **Tanisha041**, a premium subscription status, and various navigation options like **My Lists**, **Notebook**, **Submissions**, **Progress**, **Points**, **Try New Features**, **Orders**, **My Playgrounds**, and **Settings**.

5. Merge two sorted linked lists :

```

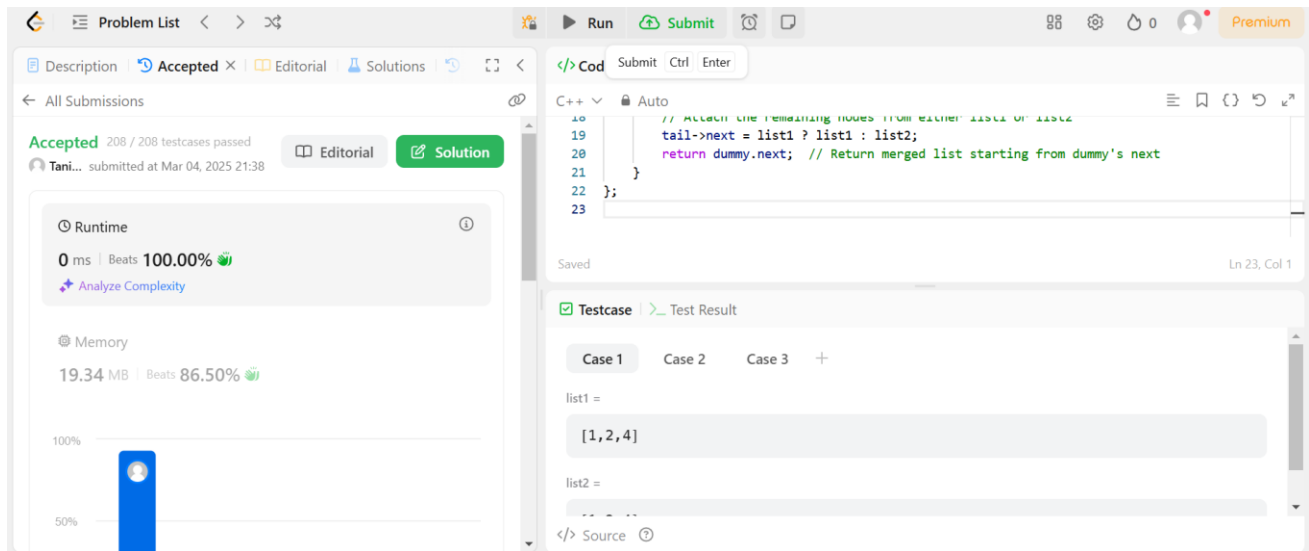
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while (list1 && list2) {
            if (list1->val < list2->val) {
                tail->next = list1;
                list1 = list1->next;
            }
            else {
                tail->next = list2;
            }
        }
    }
};

```

```

        list2 = list2->next;
    }
    tail = tail->next;
}
tail->next = list1 ? list1 : list2;
return dummy.next;
}
};

```



The screenshot displays a coding platform interface. On the left, the 'All Submissions' tab shows an 'Accepted' status for a submission by 'Tani...' at Mar 04, 2025 21:38. The runtime is 0 ms (Beats 100.00%) and memory is 19.34 MB (Beats 86.50%). The main editor shows a C++ solution for merging two sorted linked lists. The code uses a dummy node and a tail pointer to merge list1 and list2. The test case section shows list1 = [1, 2, 4] and list2 = [3, 5, 6], with the expected merged list being [1, 2, 3, 4, 5, 6].

6. Detect a cycle in a linked list::

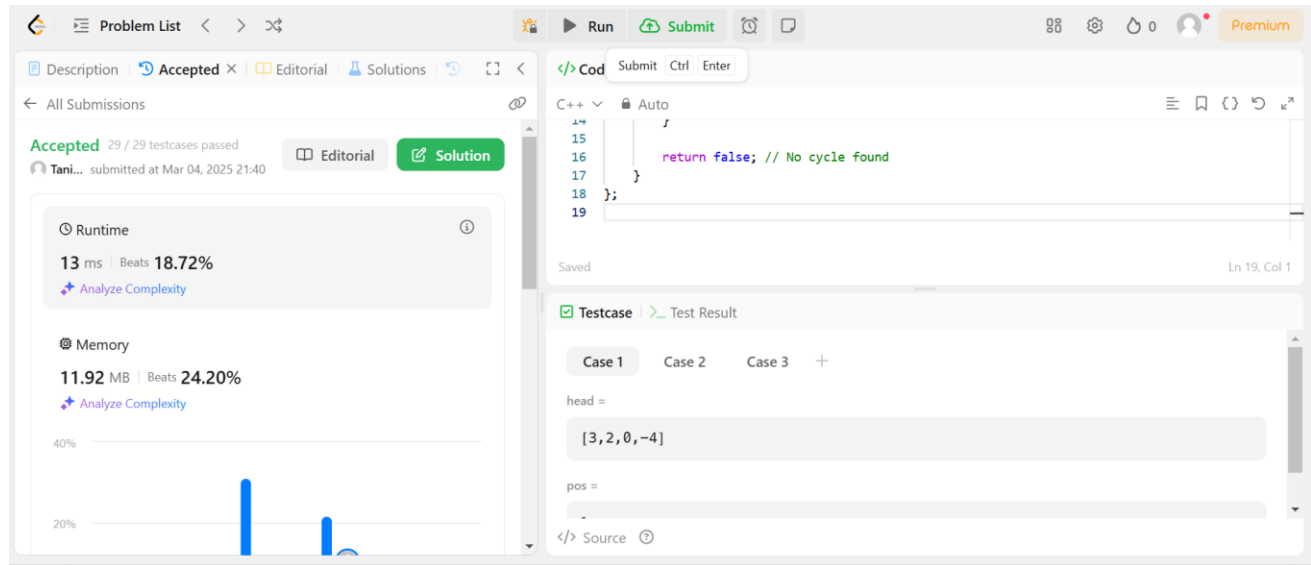
```

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

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;

```

```
        if (slow == fast) return true;
    }
    return false;
}
};
```



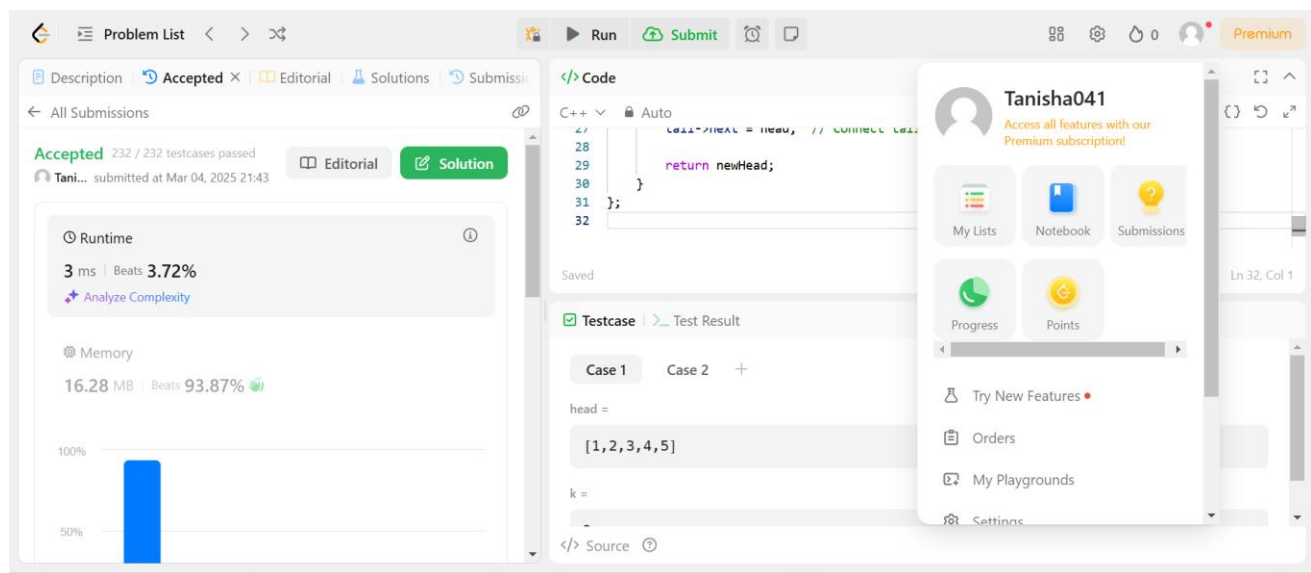
7. Rotate a list:

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        int length = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            length++;
        }
        k = k % length;
        if (k == 0) return head;
        ListNode* newTail = head;
```

```

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

```



8. Sort a list:

```

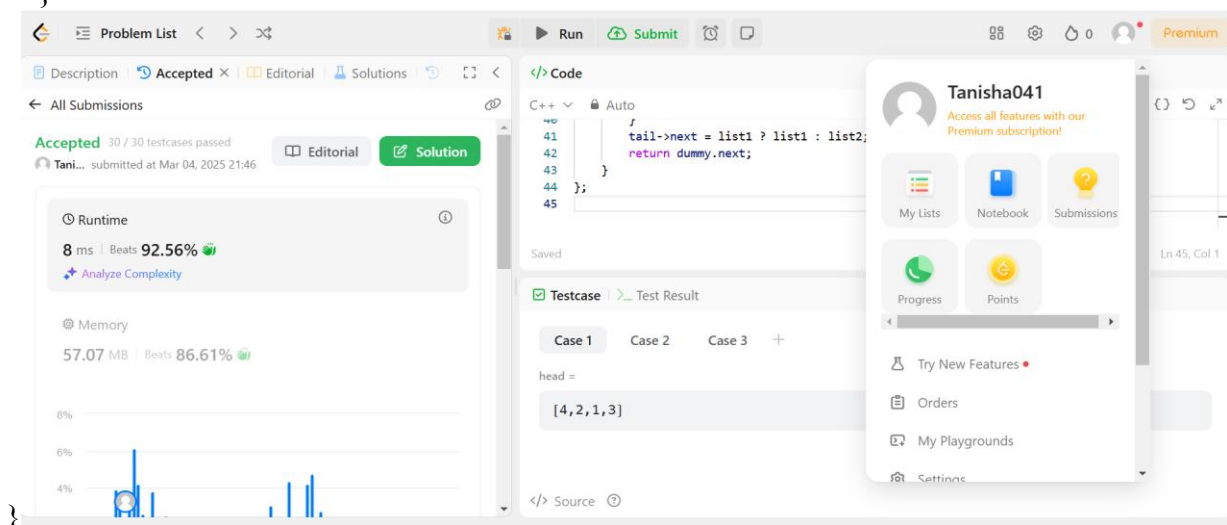
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;
        while (fast && fast->next) {
            prev = slow;

```

```

        slow = slow->next;
        fast = fast->next->next;
    }
    prev->next = nullptr;
    ListNode* left = sortList(head);
    ListNode* right = sortList(slow);
    return merge(left, right);
}
private:
    ListNode* merge(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while (list1 && list2) {
            if (list1->val < list2->val) {
                tail->next = list1;
                list1 = list1->next;
            } else {
                tail->next = list2;
                list2 = list2->next;
            }
            tail = tail->next;
        }
        tail->next = list1 ? list1 : list2;
        return dummy.next;
    }
}

```

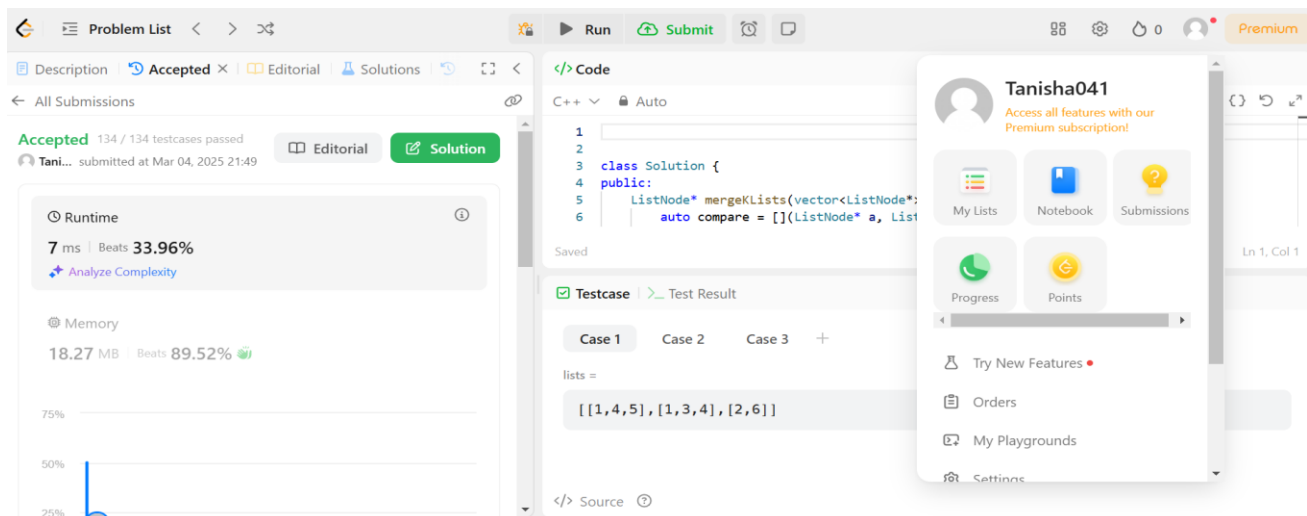


The screenshot displays a coding platform interface. On the left, the 'Runtime' section shows '8 ms' and 'Beats: 92.56%'. The 'Memory' section shows '57.07 MB' and 'Beats: 86.61%'. The 'Code' section shows the C++ implementation of the merge sort algorithm for a linked list. The 'Testcase' section shows the input '[4,2,1,3]' and the output '[1,2,3,4]'. The sidebar on the right shows the user profile 'Tanisha041' and navigation options like 'My Lists', 'Notebook', 'Submissions', 'Progress', and 'Points'.

9. Merge k sorted lists :

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*> lists) {
        auto compare = [](ListNode* a, ListNode* b) { return a->val > b->val; };
        priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> minHeap(compare);

        for (ListNode* list : lists) {
            if (list) minHeap.push(list);
        }
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while (!minHeap.empty()) {
            ListNode* node = minHeap.top();
            minHeap.pop();
            tail->next = node;
            tail = tail->next;
            if (node->next) minHeap.push(node->next);
        }
        return dummy.next;
    }
};
```



The screenshot displays a coding platform interface with the following components:

- Problem List:** Shows the problem is "Accepted" with 134 / 134 testcases passed. The user "Tani..." submitted it on Mar 04, 2025 at 21:49.
- Runtime:** 7 ms, Beats 33.96%.
- Memory:** 18.27 MB, Beats 89.52%.
- Code Editor:** Contains the C++ solution for "Merge k sorted lists". The code defines a `Solution` class with a `mergeKLists` method that uses a min-heap to merge the lists.
- Testcase:** Shows the input lists as `[[1,4,5], [1,3,4], [2,6]]`.
- User Profile:** A sidebar on the right shows the user "Tanisha041" with a premium subscription. It includes links to "My Lists", "Notebook", "Submissions", "Progress", and "Points".