

Assignment 3 (AP)

Problem 1 – Print Linked list

Solution –

```
class Solution {
public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        Node *temp=head;
        while(temp){
            cout<<temp->data<<" ";
            temp=temp->next;
        }
    }
};
```

Screenshot-

The screenshot displays a C++ IDE interface. On the left, a sidebar shows the 'Output Window' and 'Compilation Results'. The 'Compilation Results' section indicates 'Problem Solved Successfully' with a green checkmark. Below this, four statistics are shown: 'Test Cases Passed' (1112 / 1112), 'Attempts: Correct / Total' (1 / 1), 'Points Scored' (0 / 1), and 'Time Taken' (0.08). The 'Accuracy' is listed as 100%. The main editor area shows the C++ code for the 'printList' function, which iterates through a linked list and prints its elements in a single line. The code is enclosed in a 'class Solution' with a 'public' section. The IDE also shows a 'Start Timer' button and a 'Driver Code Ends' marker.

Test Cases Passed	Attempts: Correct / Total
1112 / 1112	1 / 1

Points Scored	Time Taken
0 / 1	0.08

Problem 2 – Remove Duplicate

Solution –

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

        while (head && head->next) {
            if (head->val == head->next->val) {
                head->next = head->next->next;
            } else {
                head = head->next;
            }
        }

        return res;
    }
};
```

Screenshot-

The screenshot displays a submission page on the left and a code editor on the right. The submission page shows a status of 'Accepted' with 168/168 testcases passed, submitted by 'Ayush' on Mar 05, 2025 at 14:42. Performance metrics include Runtime of 0 ms (Beats 100.00%) and Memory of 16.24 MB (Beats 35.16%). The code editor on the right shows the C++ implementation of the 'deleteDuplicates' function, which uses a while loop to traverse the linked list and remove duplicate nodes by skipping the next node if its value matches the current node's value.

```
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* res = head;

        while (head && head->next) {
            if (head->val == head->next->val) {
                head->next = head->next->next;
            } else {
                head = head->next;
            }
        }

        return res;
    }
};
```

Problem 3 – Reverse Linked list

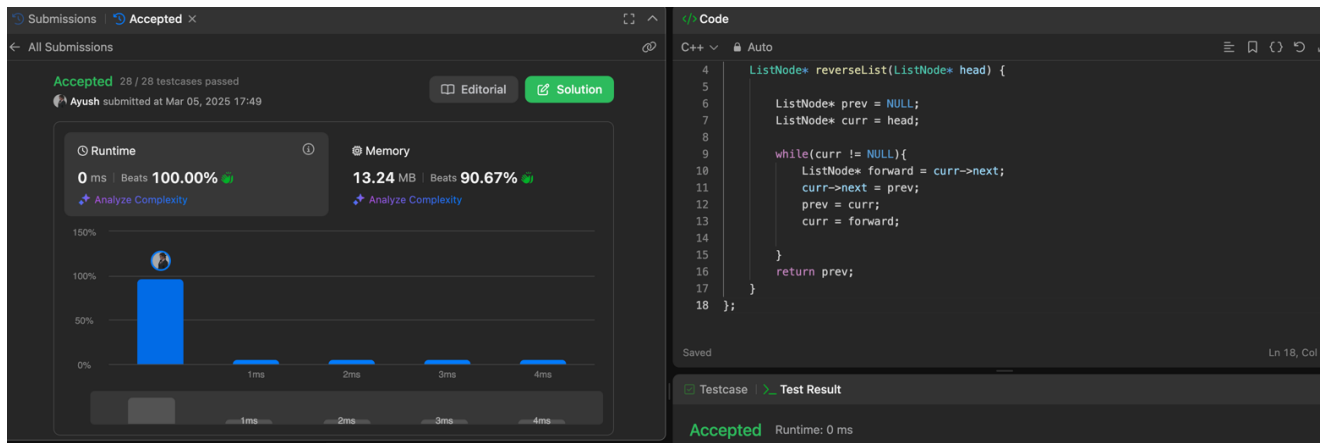
Solution –

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

        ListNode* prev = NULL;
        ListNode* curr = head;

        while(curr != NULL){
            ListNode* forward = curr->next;
            curr->next = prev;
            prev = curr;
            curr = forward;
        }
        return prev;
    }
};
```

Screenshot-

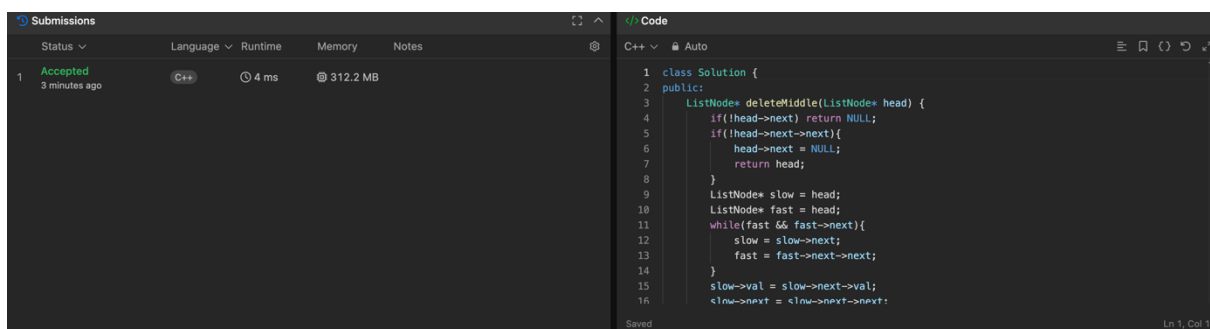


Problem 4 - Delete middle node of a list

Solution –

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(!head->next) return NULL;
        if(!head->next->next){
            head->next = NULL;
            return head;
        }
        ListNode* slow = head;
        ListNode* fast = head;
        while(fast && fast->next){
            slow = slow->next;
            fast = fast->next->next;
        }
        slow->val = slow->next->val;
        slow->next = slow->next->next;
        return head;
    }
};
```

Screenshot-



```
1 class Solution {
2 public:
3     ListNode* deleteMiddle(ListNode* head) {
4         if(!head->next) return NULL;
5         if(!head->next->next){
6             head->next = NULL;
7             return head;
8         }
9         ListNode* slow = head;
10        ListNode* fast = head;
11        while(fast && fast->next){
12            slow = slow->next;
13            fast = fast->next->next;
14        }
15        slow->val = slow->next->val;
16        slow->next = slow->next->next;
```

Problem 5 - Merge two sorted linked lists

Solution –

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dummy = new ListNode(0);
        ListNode* cur = dummy;

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

        cur->next = list1 ? list1 : list2;

        ListNode* head = dummy->next;
        delete dummy;
        return head;
    }
};
```

Screenshot-

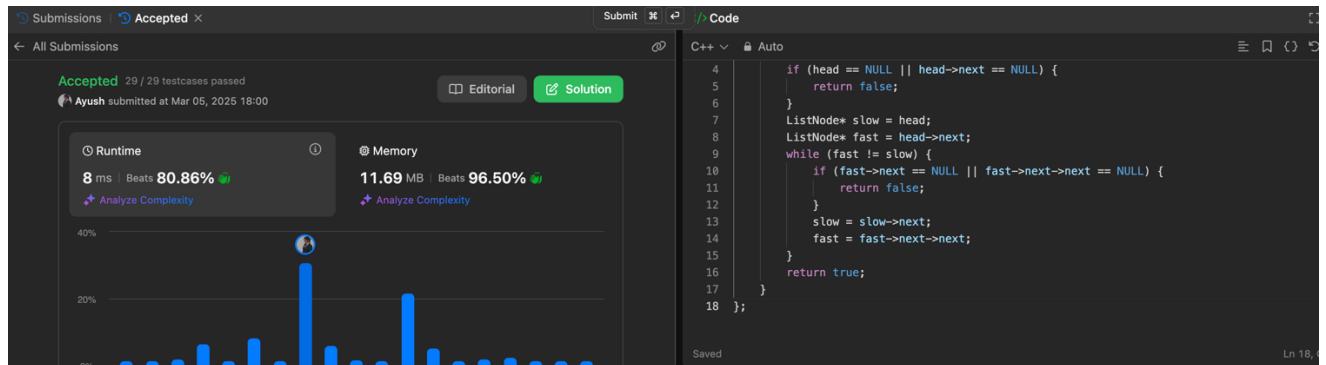
The screenshot displays a code editor interface with two main panels. The left panel shows the submission status: 'Accepted' with '208 / 208 testcases passed'. It also indicates the user 'Ayush' submitted the solution on 'Mar 05, 2025 17:57'. Performance metrics are shown: Runtime is '0 ms' (Beats 100.00%) and Memory is '19.43 MB' (Beats 62.23%). A blue bar chart is visible at the bottom of the left panel. The right panel shows the C++ code for the solution, which is identical to the one provided in the text above. The code is written in a dark-themed editor with line numbers 1 through 16. The status bar at the bottom right indicates 'Ln 24, Col 3'.

Problem 6 - Detect a cycle in a linked list:

Solution –

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

Screenshot-



Problem 7- Rotate a list:

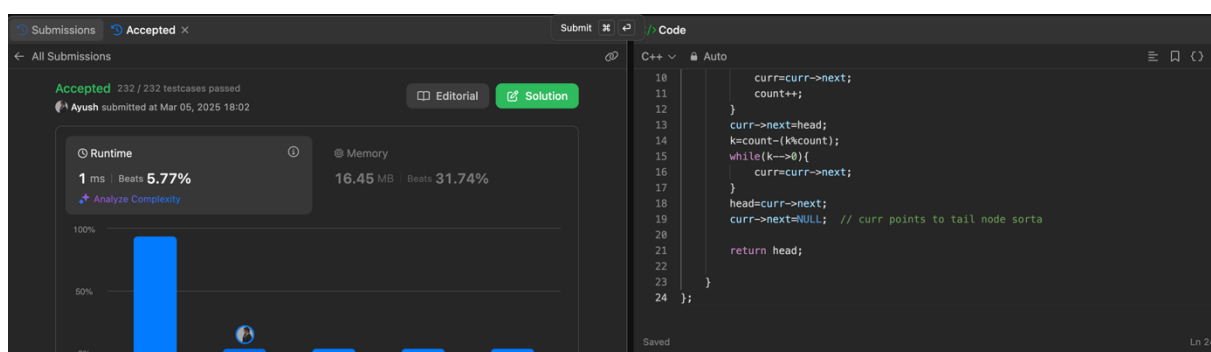
Solution –

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

        ListNode* curr=head;
        int count=1;
        while(curr->next!=NULL){
            curr=curr->next;
            count++;
        }
        curr->next=head;
        k=count-(k%count);
        while(k-->0){
            curr=curr->next;
        }
        head=curr->next;
        curr->next=NULL; // curr points to tail node sorta

        return head;
    }
};
```

Screenshot-



Problem 8- Sort list

Solution –

```
#include <iostream>
using namespace std;
//takesoumen collection
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        // Find the middle using slow and fast pointers
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        ListNode* mid = slow->next;
        slow->next = nullptr;

        // Recursively split and merge
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);

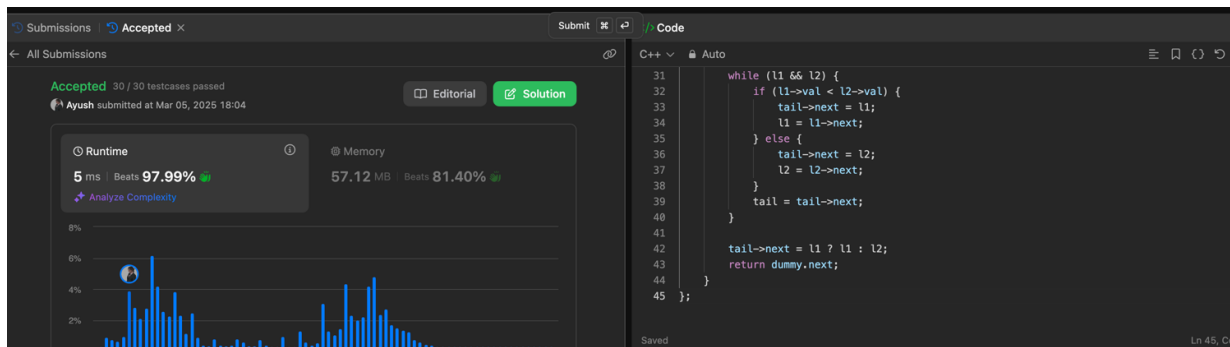
        return merge(left, right);
    }

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* 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;
    }
};
```

Screenshot-



Problem 9- Merge k sorted lists

Solution –

```

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) {
            return nullptr;
        }

        while (lists.size() > 1) {
            vector<ListNode*> temp;
            for (size_t i = 0; i < lists.size(); i += 2) {
                ListNode* l1 = lists[i];
                ListNode* l2 = i + 1 < lists.size() ? lists[i + 1] : nullptr;
                temp.push_back(mergeLists(l1, l2));
            }
            lists = move(temp);
        }

        return lists[0];
    }

private:
    ListNode* mergeLists(ListNode* l1, ListNode* l2) {
        ListNode dummy;
        ListNode* node = &dummy;

        while (l1 && l2) {
            if (l1->val > l2->val) {
                node->next = l2;
                l2 = l2->next;
            } else {
                node->next = l1;
                l1 = l1->next;
            }
        }
    }
}

```

```

        node = node->next;
    }

    node->next = l1 ? l1 : l2;
    return dummy.next;
}
};

```

Screenshot-

The screenshot shows a submission page for a C++ problem. The submission is marked as "Accepted" with 134/134 test cases passed. The user "Ayush" submitted it on Mar 05, 2025 at 18:06. The runtime is 0 ms (Beats 100.00%) and memory is 19.05 MB (Beats 22.95%). The code is displayed on the right side of the interface.

Runtime: 0 ms | Beats 100.00%
Memory: 19.05 MB | Beats 22.95%

Code:

```

26         while (l1 && l2) {
27             if (l1->val > l2->val) {
28                 node->next = l2;
29                 l2 = l2->next;
30             } else {
31                 node->next = l1;
32                 l1 = l1->next;
33             }
34             node = node->next;
35         }
36
37         node->next = l1 ? l1 : l2;
38         return dummy.next;
39     }
40 };

```