

Assignment 3

Name- Anirudh Gagneja

UID- 22BCS16527

Class- 609-B

Question 1) Print Linked List

Code)

```
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; } } };
```

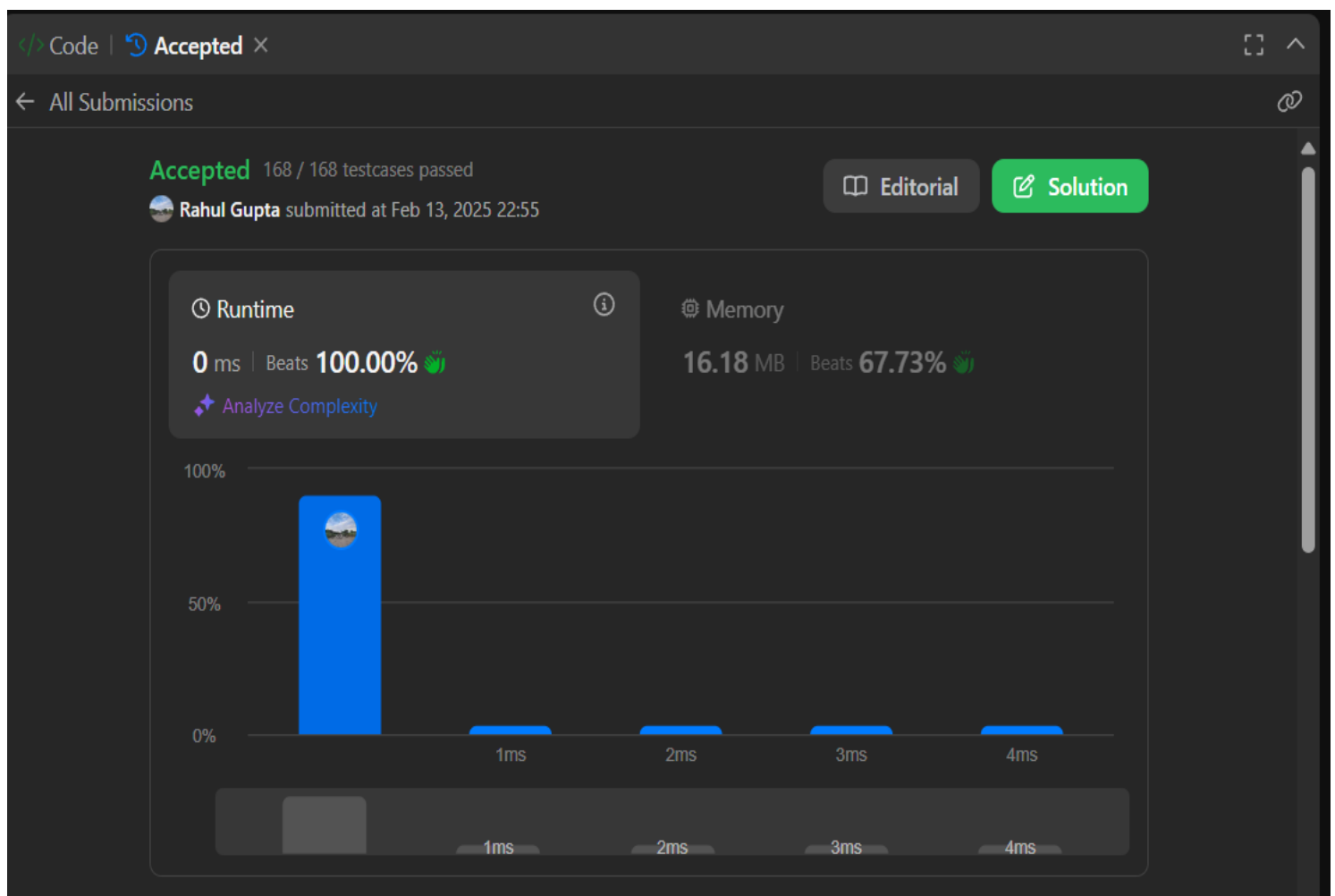
The screenshot displays a coding platform interface with a dark theme. The top navigation bar includes links for Courses, Tutorials, Jobs, Practice, and Contests. The main header shows the problem name, 'Print Linked List', and the language 'C++ (g++ 5.4)'. The left sidebar contains tabs for 'Output Window', 'Compilation Results', and 'Custom Input'. The 'Compilation Results' tab is active, showing a green checkmark and the text 'Problem Solved Successfully'. Below this, a table displays test case results: 1112/1112 test cases passed, 1/3 attempts correct, 33% accuracy, 0/1 points scored, and a time taken of 0.05 seconds. The right sidebar shows the code editor with the C++ code for the linked list printing function. The code is as follows:

```
1 // Driver Code Ends  
19 /*  
20 struct Node {  
21     int data;  
22     struct Node* next;  
23  
24     Node(int x) {  
25         data = x;  
26         next = nullptr;  
27     }  
28 };  
29 */  
30 /*  
31     Print elements of a linked list on console  
32     Head pointer input could be NULL as well for empty list  
33 */  
34  
35 class Solution {  
36 public:  
37     // Function to display the elements of a linked list in same line  
38     void printList(Node *head) {  
39         Node *temp=head;  
40         while(temp){  
41             cout<<temp->data<<" ";  
42             temp=temp->next;  
43         }  
44     }  
45 };  
46  
47 // } Driver Code Ends
```

Question 2) [Remove Duplicates from Sorted List](#)

Code)

```
class Solution {  
public:  
    ListNode* deleteDuplicates(ListNode* head) {  
        ListNode* current = head;  
        while (current && current->next) {  
            if (current->val == current->next->val) {  
                current->next = current->next->next; // Skip duplicate node  
            } else {  
                current = current->next; // Move to the next node  
            }  
        }  
        return head; } };
```



Question 3) [Reverse Linked List](#)

Code)

```
class Solution {  
public:  
    ListNode* reverseList(ListNode* head) {  
        ListNode* prev = nullptr;  
        ListNode* curr = head;  
  
        while (curr) {  
            ListNode* nextNode = curr->next;  
            curr->next = prev;  
            prev = curr;  
            curr = nextNode;  
        }  
        return prev; } };
```

The screenshot displays a coding platform interface for a C++ solution. The left sidebar shows the submission status as 'Accepted' with 28/28 testcases passed. The user 'Rahul Gupta' submitted the solution on Mar 05, 2025 at 14:41. The runtime is 0 ms, which beats 100.00% of other solutions. The memory usage is 13.47 MB, which beats 39.77% of other solutions. A bar chart visualizes the runtime performance. The main editor shows the C++ code for the 'reverseList' function. The bottom panel shows the test case selection area with 'Case 1' selected.

Submission Details:

- Status: Accepted
- Testcases: 28 / 28 passed
- User: Rahul Gupta
- Submitted: Mar 05, 2025 14:41
- Runtime: 0 ms | Beats 100.00%
- Memory: 13.47 MB | Beats 39.77%

Code:

```
1 class Solution {  
2 public:  
3     ListNode* reverseList(ListNode* head) {  
4         ListNode* prev = nullptr;  
5         ListNode* curr = head;  
6  
7         while (curr) {  
8             ListNode* nextNode = curr->next;  
9             curr->next = prev;  
10            prev = curr;  
11            curr = nextNode;  
12        }  
13  
14        return prev;  
15    }  
16 };  
17
```

Testcase Selection:

- Case 1
- Case 2
- Case 3
- +

Question 4) [Delete the Middle Node of a Linked List](#)

Code)

```
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;  
        return head; };
```

Description | **Accepted** | Editorial | Solutions | Submissions

← All Submissions

Accepted 70 / 70 testcases passed

Rahul Gupta submitted at Mar 05, 2025 14:48

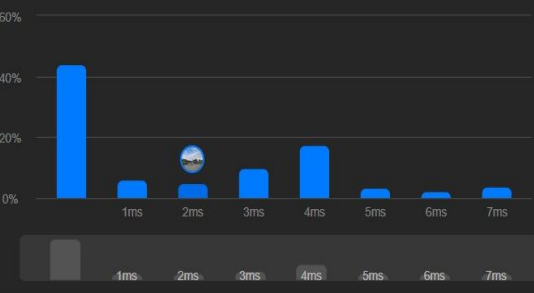
Runtime

2 ms | Beats 49.98%

Analyze Complexity

Memory

312.08 MB | Beats 55.20%



Time Interval	Percentage
1ms	~45%
2ms	~5%
3ms	~10%
4ms	~15%
5ms	~5%
6ms	~2%
7ms	~5%

Code | C++

class Solution {

Code

C++ v Auto

```
1 class Solution {  
2 public:  
3     ListNode* deleteMiddle(ListNode* head) {  
4         if (!head || !head->next) {  
5             return nullptr;  
6         }  
7         ListNode* slow = head;  
8         ListNode* fast = head;  
9         ListNode* prev = nullptr;  
10        while (fast && fast->next) {  
11            prev = slow;  
12            slow = slow->next;  
13            fast = fast->next->next; }  
14        prev->next = slow->next;  
15        return head;  
16    }  
17 }  
18 };  
19
```

Saved

Ln 12, Col 25

Testcase | Test Result

Case 1 Case 2 Case 3 +

</> Source

Question 5) [Merge Two Sorted Lists](#)

Code)

```
class Solution {
```

```
public:
```

```
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
```

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

```
        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;
```

```
        } else if (list2) {
```

```
            current->next = list2; }
```

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

The screenshot displays a coding platform interface with two main panels. The left panel shows the submission status: 'Accepted' with 208/208 testcases passed. The user 'Rahul Gupta' submitted the solution on Feb 13, 2025, at 22:58. The runtime is 3 ms, beating 4.22% of other solutions. The memory usage is 19.40 MB, beating 86.51% of other solutions. A bar chart shows the runtime distribution across different time intervals. The right panel shows the C++ code for the solution, which is a class 'Solution' with a public method 'mergeTwoLists'. The code uses a dummy node and a current pointer to merge two sorted linked lists. The bottom of the interface shows a 'Testcase' tab with 'Case 1' selected.

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dummy = new ListNode();
        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;
        } else if (list2) {
            current->next = list2;
        }
        return dummy->next;
    }
};
```

Question 6) [Linked List Cycle](#)

```
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; } };
```

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Accepted' tab is selected, showing submission details for 'Rahul Gupta' submitted at Mar 04, 2025 19:02. The runtime is 8 ms, beating 80.86% of solutions, and memory is 11.98 MB, beating 24.25%. A bar chart shows the distribution of runtime results. The main editor on the right contains the C++ code for the 'hasCycle' function, which uses Floyd's Cycle-Finding algorithm. The code is as follows:

```
1 class Solution {
2 public:
3     bool hasCycle(ListNode* head) {
4         if (head == NULL || head->next == NULL) {
5             return false; }
6         ListNode* slow = head;
7         ListNode* fast = head->next;
8         while (fast != slow) {
9             if (fast->next == NULL || fast->next->next == NULL) {
10                 return false; }
11             slow = slow->next;
12             fast = fast->next->next; }
13     return true; }
```

At the bottom, the 'Testcase' tab is active, showing 'Case 1' selected. The source code editor at the very bottom shows the beginning of the class definition: `class Solution {`.

Question 7) [Rotate List](#)

Code)

```
class Solution {  
public:  
    ListNode* rotateRight(ListNode* head, int k) {  
        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;  
        return head; } };
```

The screenshot displays a coding platform interface with two main panels. The left panel shows the submission status: 'Accepted' with 232/232 testcases passed. The user 'Rahul Gupta' submitted the solution on Mar 05, 2025 at 14:56. The runtime is 0 ms, beating 100.00% of other solutions. The memory usage is 16.23 MB, beating 93.87% of other solutions. A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming. The right panel shows the C++ code for the 'rotateRight' function, which implements the logic to rotate a linked list by k nodes. The code is saved and the user is currently viewing the 'Testcase' tab.

Submission Details:

- Status: Accepted (232 / 232 testcases passed)
- Submitted by: Rahul Gupta
- Submitted at: Mar 05, 2025 14:56
- Runtime: 0 ms | Beats 100.00%
- Memory: 16.23 MB | Beats 93.87%

Code Snippet:

```
class Solution {  
public:  
    ListNode* rotateRight(ListNode* head, int k) {  
        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;  
        return head; } };
```

Question 8) [Sort List](#)

Code)

```
#include <iostream>

using namespace std;

class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        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;
        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; } };
```

Description | Editorial | Solutions | Accepted × | Submissions

← All Submissions

Accepted 30 / 30 testcases passed
Rahul Gupta submitted at Mar 05, 2025 14:59

Runtime
12 ms | Beats 78.58%
[Analyze Complexity](#)

Memory
56.90 MB | Beats 93.44%

8%
6%
4%
2%
0%

21ms 43ms 64ms

Code | C++

Code

C++ v Auto

```
1 #include <iostream>  
2 using namespace std;  
3 class Solution {  
4 public:  
5     ListNode* sortList(ListNode* head) {  
6         if (!head || !head->next) return head;  
7         ListNode* slow = head;  
8         ListNode* fast = head->next;  
9         while (fast && fast->next) {  
10             slow = slow->next;  
11             fast = fast->next->next;}  
12         ListNode* mid = slow->next;  
13         slow->next = nullptr;  
14         ListNode* left = sortList(head);  
15         ListNode* right = sortList(mid);  
16         return merge(left, right);  
17     }  
18     ListNode* merge(ListNode* l1, ListNode* l2) {  
19         ListNode dummy(0);  
20         ListNode* tail = &dummy;  
21         while (l1 && l2) {  
22             if (l1->val < l2->val) {  
23                 tail->next = l1;  
24                 l1 = l1->next;  
25             } else {  
26                 tail->next = l2;  
27                 l2 = l2->next;  
28             }  
29             tail = tail->next;  
30         }  
31         if (l1) tail->next = l1;  
32         if (l2) tail->next = l2;  
33         return dummy.next;  
34     }  
35 };
```

Saved [Upgrade to Cloud Saving](#) Ln 13, Col 30

Testcase | Test Result

Case 1 Case 2 Case 3 +

</> Source ⓘ

Question 9) [Merge k Sorted Lists](#)

Code)

```
#include <vector>

using namespace std;

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;
        return divideAndConquer(lists, 0, lists.size() - 1);
    }

    ListNode* divideAndConquer(vector<ListNode*>& lists, int left, int right) {
        if (left == right) return lists[left];
```

```

int mid = left + (right - left) / 2;

ListNode* l1 = divideAndConquer(lists, left, mid);

ListNode* l2 = divideAndConquer(lists, mid + 1, right);

return mergeTwoLists(l1, l2);

}

};

```

Description
Editorial
Solutions
Submissions
Accepted

All Submissions

Accepted 134 / 134 testcases passed
Rahul Gupta submitted at Mar 05, 2025 15:01

Runtime
0 ms | Beats 100.00%

Memory
18.57 MB | Beats 50.76%

Code

```

1 #include <vector>
2 using namespace std;
3 class Solution {
4 public:
5     ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
6         if (!l1) return l2;
7         if (!l2) return l1;
8
9         if (l1->val < l2->val) {
10             l1->next = mergeTwoLists(l1->next, l2);
11             return l1;
12         } else {
13             l2->next = mergeTwoLists(l1, l2->next);
14             return l2;
15         }
16     }
17
18     ListNode* mergeKLists(vector<ListNode*>& lists) {
19         if (lists.empty()) return nullptr;
20         return divideAndConquer(lists, 0, lists.size() - 1);
21     }
22
23     ListNode* divideAndConquer(vector<ListNode*>& lists, int left, int right) {
24         if (left == right) return lists[left];

```

Testcase
Test Result

Case 1 Case 2 Case 3 +

Source