

NAME - ANSHIKA

UID - 22BCS16918

SECTION - 611

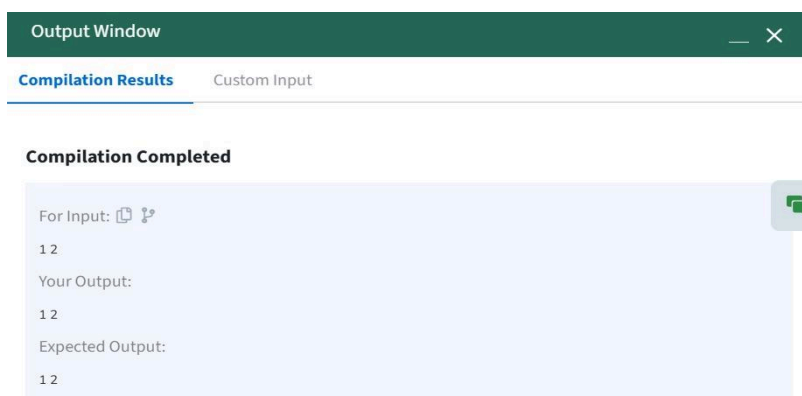
GROUP - "B"

Ques 1. Print linked list

Code -

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

Output -



Ques 2. Remove duplicates from 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;
            else
                current = current->next;
        }
        return head;
    }
};
```

Output -

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

- Problem List:** A sidebar on the left showing the problem status as "Accepted" with 168/168 testcases passed. The submission ID is 228CS16918, submitted on Feb 20, 2025 at 15:22. It includes buttons for "Editorial" and "Solution".
- Runtime and Memory:** The runtime is 0 ms, beating 100.00% of solutions. The memory usage is 16.02 MB, beating 90.20% of solutions. A bar chart shows the runtime distribution across different time intervals (1ms, 2ms, 3ms, 4ms).
- Code Editor:** The main area shows the C++ code for the solution, which is a class `Solution` with a public method `deleteDuplicates`. The code uses a while loop to traverse the list and remove duplicates by skipping the next node if its value is the same as the current node's value.
- Testcase and Test Result:** The bottom right section shows the test results for "Case 1". The input is `head = [1,1,2]` and the output is `[1,2]`, which matches the expected result. The runtime for this test case is 0 ms.

Ques 3. Reverse linked list

Code -

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;

        while (current) {
            ListNode* nextNode = current->next;
            current->next = prev;
            prev = current;
            current = nextNode;
        }

        return prev;
    }
}
```

Output -

The screenshot displays a coding platform interface for a C++ solution. The top navigation bar includes 'Problem List', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The main content area shows the problem 'Reverse linked list' with a status of 'Accepted' and 28/28 testcases passed. A submission by user '228CS16918' is shown, submitted on Feb 20, 2025, at 15:31. The performance metrics indicate a runtime of 0 ms (Beats 100.00%) and memory usage of 13.38 MB (Beats 70.55%). A bar chart shows the runtime performance across different test cases. The code editor on the right contains the C++ code for reversing the linked list. The test result section shows the input 'head = [1,2,3,4,5]' and the output '[5,4,3,2,1]', which matches the expected result.

Runtime Performance:

Test Case	Runtime (ms)	Memory (MB)
Case 1	0	13.38
Case 2	0	13.38
Case 3	0	13.38

Test Result:

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Expected: [5,4,3,2,1]

Ques 4. Delete the Middle Node of a Linked List

Code -

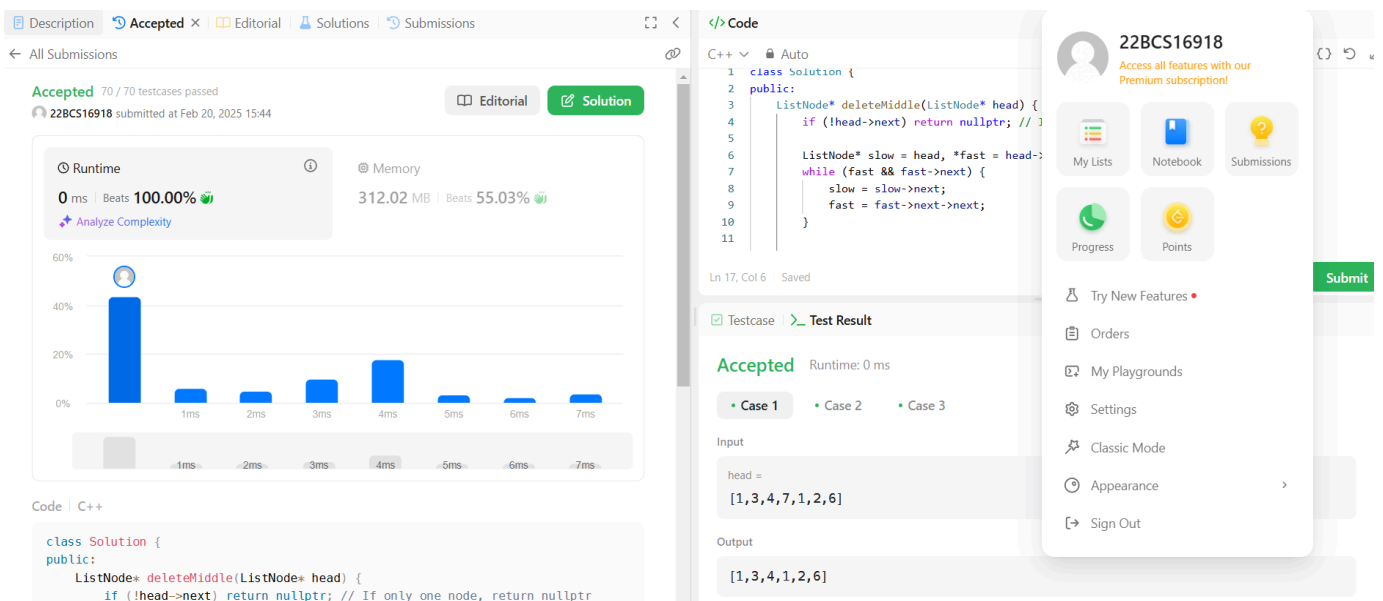
```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head->next) return nullptr; // If only one node, return nullptr

        ListNode* slow = head, *fast = head->next->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        ListNode* temp = slow->next;
        slow->next = slow->next->next; // Skip middle node
        delete temp; // Free memory

        return head;
    }
};
```

Output -



Ques 5. Merge two sorted list

Code -

```
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;
    }
};
```

Output -

The screenshot displays a coding platform interface for a C++ solution. The top navigation bar includes links for Description, Accepted (208 / 208 testcases passed), Editorial, Solutions, and Submissions. The user's profile, 22BCS16918, is shown in the top right corner with a premium subscription badge. The main content area is divided into three sections: a performance summary, a code editor, and a test result panel.

Performance Summary:

- Runtime:** 0 ms, Beats 100.00% (Analyze Complexity)
- Memory:** 19.38 MB, Beats 86.69%

Code Editor: The code is written in C++ and implements a function to merge two sorted linked lists. The code is as follows:

```
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;
    }
};
```

Test Result Panel:

- Accepted** Runtime: 0 ms
- Case 1:** [1,2,4]
- Case 2:** [1,3,4]
- Case 3:** [1,3,4]

User Profile Sidebar:

- 22BCS16918
- Access all features with our Premium subscription!
- My Lists
- Notebook
- Submissions
- Progress
- Points
- Try New Features
- Orders
- My Playgrounds
- Settings
- Classic Mode
- Appearance
- Sign Out

Ques 6. Remove Duplicates From sorted list 2

Code -

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

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;
        while (head) {
            bool isDuplicate = false;
            while (head->next && head->val == head->next->val) {
                isDuplicate = true;
                head = head->next;
            }
            if (isDuplicate) {
                prev->next = head->next;
            } else {
                prev = prev->next;
            }
            head = head->next;
        }
        return dummy.next;
    }
};
```

Output -

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

- Top Bar:** Navigation tabs for Description, Accepted (selected), Editorial, Solutions, and Submissions.
- Submission Status:** Shows 'Accepted' with 166/166 testcases passed. The user ID is 22BCS16918, submitted at Feb 20, 2025 16:01.
- Performance Metrics:**
 - Runtime:** 0 ms, Beats 100.00%.
 - Memory:** 15.80 MB, Beats 17.45%.
- Complexity Graph:** A bar chart showing the runtime performance relative to other submissions, with a single bar at 100%.
- Code Editor:** Displays the C++ code for the solution, which is the same as provided in the code block above.
- Testcase Results:** Shows 'Accepted' with a runtime of 0 ms. Two test cases are listed: Case 1 and Case 2.
- Input/Output:**
 - Input:** head = [1,2,3,3,4,4,5]
 - Output:** [1,2,5]
 - Expected:** (The output matches the expected result).
- User Profile Sidebar:** On the right, it shows the user's profile (22BCS16918) and a list of navigation options: My Lists, Notebook, Submissions, Progress, Points, Try New Features, Orders, My Playgrounds, Settings, Classic Mode, Appearance, and Sign Out.

Ques 7. Detect a cycle in a linked list

Code -

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

Output -

The screenshot displays a coding platform interface for a C++ solution. At the top, a submission status bar shows 'Accepted' with 29/29 testcases passed, submitted by user 22BCS16918 on Feb 20, 2025. Below this, a performance graph shows runtime (13 ms, 19.06% beats) and memory (11.87 MB, 53.78% beats). A bar chart indicates that 6.47% of solutions used 4 ms of runtime. The code editor shows the following C++ code:

```
bool hasCycle(ListNode* head) {
    ListNode *slow = head, *fast = head;
    while (fast && (fast = fast->next) && (fast = fast->next)) {
        slow = slow->next;
        if (slow == fast) return true;
    }
    return false;
};
```

The test result panel shows 'Accepted' with a runtime of 3 ms for Case 1. The input is 'head = [3,2,0,-4]' and 'pos = 1'. The output is empty. The sidebar on the right shows the user profile for 22BCS16918, with options for My Lists, Notebook, Submissions, Progress, and Points. A 'Submit' button is visible at the bottom right.

Ques 8. Reverse Linked List II

Code -

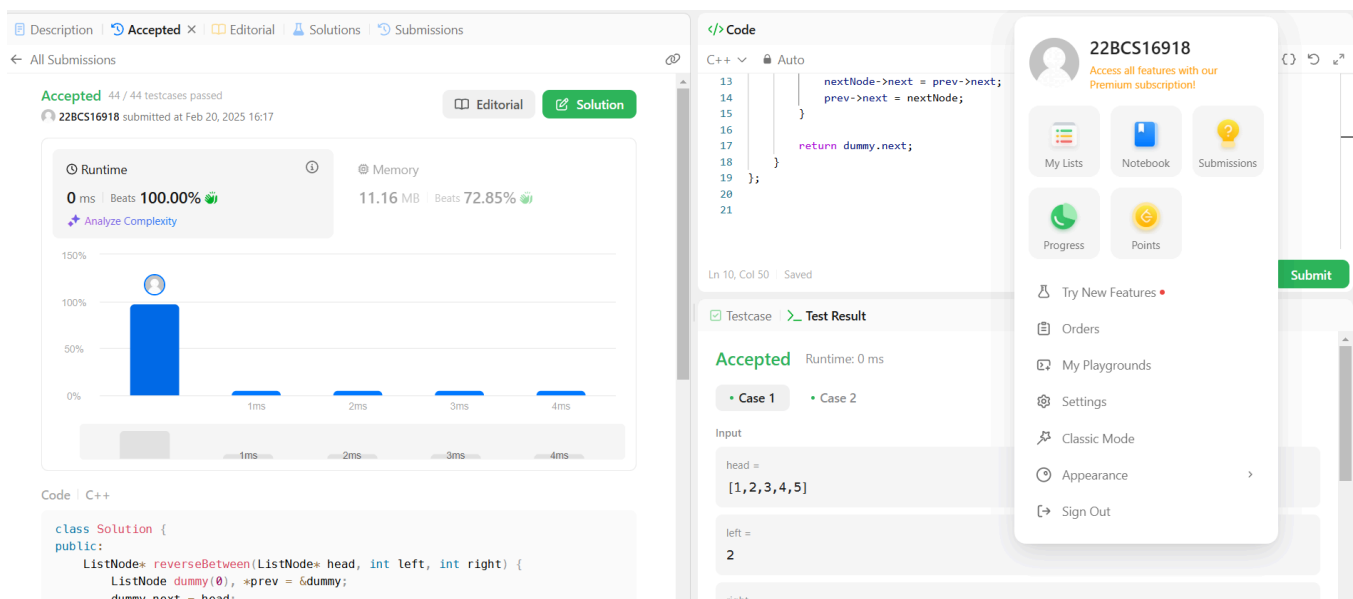
```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        ListNode dummy(0), *prev = &dummy;
        dummy.next = head;

        for (int i = 1; i < left; i++) prev = prev->next; // Move `prev` to `left-1`

        ListNode *curr = prev->next, *nextNode;
        for (int i = 0; i < right - left; i++) { // Reverse in-place
            nextNode = curr->next;
            curr->next = nextNode->next;
            nextNode->next = prev->next;
            prev->next = nextNode;
        }

        return dummy.next;
    }
};
```

Output -



Ques 9 . Rotate a list

Code -

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        ListNode* tail = head;
        int len = 1;
        while (tail->next) tail = tail->next, len++; // Find length & last node
        k %= len;
        if (k == 0) return head;

        tail->next = head; // Form a cycle
        for (int i = 0; i < len - k; i++) tail = tail->next; // Find new tail

        head = tail->next;
        tail->next = nullptr; // Break cycle

        return head;
    }
};
```

Output -

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

- Navigation Bar:** Includes tabs for Description, Accepted (selected), Editorial, Solutions, and Submissions. A left arrow and 'All Submissions' link are also present.
- Submission Status:** Shows 'Accepted' in green, submitted by user '22BCS16918' on Feb 20, 2025 at 16:23. Buttons for 'Editorial' and 'Solution' are available.
- Performance Metrics:**
 - Runtime:** 0 ms, Beats 100.00% (indicated by a green circle).
 - Memory:** 16.44 MB, Beats 31.65%.
 - A link to 'Analyze Complexity' is provided.
- Bar Chart:** A chart showing runtime performance across different test cases, with a single blue bar reaching 100%.
- Code Editor:** Displays the C++ code for the 'rotateRight' function. 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;
5
6         ListNode* tail = head;
7         int len = 1;
8
9         while (tail->next) tail = tail->next, len++;
10
11         tail->next = head;
12
13         for (int i = 0; i < len - k; i++) tail = tail->next;
14
15         head = tail->next;
16         tail->next = nullptr;
17
18         return head;
19     }
20 }
```
- Testcase Section:** Shows 'Accepted' with a runtime of 0 ms. Two test cases are listed: 'Case 1' and 'Case 2'.
- Input:**
 - head = [1,2,3,4,5]
 - k = 2
- Output:** A field for the output of the function.

Ques 10. Sort list

Code -

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

        ListNode *slow = head, *fast = head->next;
        while (fast && fast->next) slow = slow->next, fast = fast->next->next;
        ListNode* mid = slow->next;
        slow->next = nullptr; // Split list
        return merge(sortList(head), sortList(mid));
    }
    ListNode* merge(ListNode* l1, ListNode* l2) {
        if (!l1 || !l2) return l1 ? l1 : l2;
        if (l1->val > l2->val) swap(l1, l2);
        l1->next = merge(l1->next, l2);
        return l1;
    }
};
```

Output -

The screenshot displays a coding platform interface for the problem "148. Sort List". The problem description states: "Given the head of a linked list, return the list after sorting it in ascending order." Example 1 shows an input linked list [4, 2, 1, 3] being transformed into an output [1, 2, 3, 4]. Example 2 shows an input [-1, 5, 3, 4, 0] being transformed into an output [1, 2, 3, 4]. The code editor on the right contains the C++ solution provided in the previous block. The test result section shows "Accepted" with a runtime of 0 ms. A user profile overlay for "22BCS16918" is visible on the right side of the interface.

Ques 11. Detect a cycle in linked list 2

Code -

```
class Solution {
public:
    ListNode* detectCycle(ListNode* head) {
        ListNode *slow = head, *fast = head;

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) { // Cycle detected
                slow = head;
                while (slow != fast) slow = slow->next, fast = fast->next;
                return slow; // Cycle start node
            }
        }
        return nullptr; // No cycle
    }
};
```

Output -

142. Linked List Cycle II

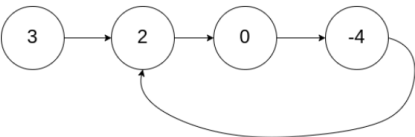
Medium Topics Companies

Given the `head` of a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (0-indexed). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter.**

Do not modify the linked list.

Example 1:



```
graph LR; 3((3)) --> 2((2)); 2 --> 0((0)); 0 --> -4((-4)); -4 --> 2;
```

Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.

14.1K 180 88 Online

```
1 class Solution {
2 public:
3     ListNode* detectCycle(ListNode* head) {
4         ListNode *slow = head, *fast = head;
5
6         while (fast && fast->next) {
7             slow = slow->next;
8             fast = fast->next->next;
9             if (slow == fast) { // Cycle detected
10                 slow = head;
11                 while (slow != fast) slow = slow->next, fast = fast->next;
12                 return slow; // Cycle start node
13             }
14         }
15         return nullptr; // No cycle
16     }
17 }
```

Ln 10, Col 29 Saved

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head = [3,2,0,-4]

pos = 1

Output

22BCS16918

Access all features with our Premium subscription!

My Lists Notebook Submissions

Progress Points

Try New Features

Orders

My Playgrounds

Settings

Classic Mode

Appearance

Sign Out

Submit

