

ASSIGNMENT 3

ADVANCE PROGRAMMING LAB-II

Date: 06/03/25

Submitted by: Shivansh Rattan

UID: 22BCS16351

Section: 610-B

Submitted to: Pratima mam

1. Print Linked List: <https://www.geeksforgeeks.org/problems/print-linked-list-elements/0>

Code:

```
class Solution {
public:
    void printList(Node *head) {
        Node* current = head;

        while (current != nullptr) {
            cout << current->data;

            if (current->next != nullptr) {
                cout << " ";
            }

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

Output:

The screenshot displays a coding platform interface with a green header bar. The main content area is divided into two panels. The left panel, titled 'Output Window', shows 'Compilation Results' for a problem solved successfully. It includes statistics: 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 1 / 4', 'Accuracy: 25%', 'Points Scored: 1 / 1', and 'Your Total Score: 7'. The right panel shows the C++ code for the 'printList' function, which iterates through a linked list and prints its elements. The code is enclosed in a class 'Solution' with a public method 'printList'. The interface also features a 'Solve Next' button at the bottom left and 'Compile & Run' and 'Submit' buttons at the bottom right.

2. Remove duplicates from a sorted list: <https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/>

Code:

```
</> Code
C++ Auto
8  *   ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         if (head == nullptr) {
15             return nullptr;
16         }
17
18         ListNode* current = head;
19
20         while (current != nullptr && current->next != nullptr) {
21             if (current->val == current->next->val) {
22                 current->next = current->next->next;
23             } else {
24                 current = current->next;
25             }
26         }
27
28         return head;
29     }
30 }
```

Output:

The screenshot displays the LeetCode submission interface for the problem 'Remove Duplicates from a Sorted List'. The submission is marked as 'Accepted' with 168/168 testcases passed. The user 'Shivansh Rattan' submitted the solution on Mar 06, 2025 at 23:07. The performance metrics are as follows:

- Runtime:** 0 ms, Beats 100.00% (Analyze Complexity)
- Memory:** 16.07 MB, Beats 90.02%

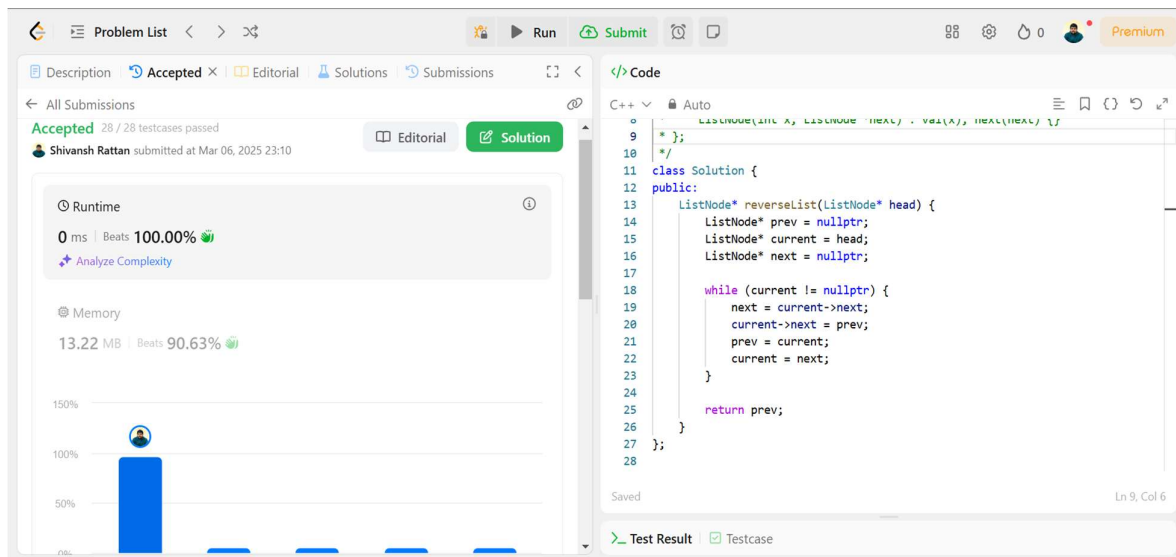
A performance graph is shown with a blue bar representing the user's performance, which is at the top of the chart, indicating a top performance. The code editor on the right shows the same C++ code as in the previous block, with line numbers 8 through 28. The bottom of the interface shows tabs for 'Test Result' and 'Testcase'.

3. Reverse a linked list: <https://leetcode.com/problems/reverse-linked-list/description/>

Code:

```
</> Code
C++ v Auto
ListNode* reverseList(ListNode* head) {
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
};
```

Output:



4. Delete middle node of a list: <https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/>

Code:

```
</> Code
C++ Auto
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         if (!head || !head->next) return nullptr;
15
16         ListNode *slow = head, *fast = head, *prev = nullptr;
17
18         while (fast && fast->next) {
19             prev = slow;
20             slow = slow->next;
21             fast = fast->next->next;
22         }
23
24         prev->next = slow->next;
25         delete slow;
26         return head;
27     }
28 };
```

Output:

The screenshot shows the LeetCode submission interface. On the left, the 'Accepted' status is confirmed with '70 / 70 testcases passed'. The submission was made by Shivansh Rattan on March 06, 2025, at 23:12. The performance metrics are: Runtime 0 ms (Beats 100.00%) and Memory 311.98 MB (Beats 83.41%). A bar chart at the bottom shows the user's performance relative to others. On the right, the C++ code for the 'deleteMiddle' function is displayed, matching the code provided in the previous block. The code is saved and the test results are shown as 'Accepted' with a runtime of 0 ms.

5. Merge two sorted linked lists: <https://leetcode.com/problems/merge-two-sorted-lists/description/>

Code:

```
</> Code
C++ v Auto
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
14         ListNode* dummy = new ListNode(0);
15         ListNode* current = dummy;
16
17         while (list1 != nullptr && list2 != nullptr) {
18             if (list1->val < list2->val) {
19                 current->next = list1;
20                 list1 = list1->next;
21             } else {
22                 current->next = list2;
23                 list2 = list2->next;
24             }
25             current = current->next;
26         }
27
28         if (list1 != nullptr) {
29             current->next = list1;
30         } else if (list2 != nullptr) {
31             current->next = list2;
32         }
33
34         return dummy->next;
35     }
36 };
```

Output:

The screenshot displays the LeetCode interface for the 'Merge Two Sorted Lists' problem. On the left, the 'Submissions' tab is active, showing a list of submissions. The top submission by Shivansh Rattan is highlighted, indicating it was 'Accepted' with a runtime of 0 ms (Beats 100.00%) and a memory usage of 19.48 MB (Beats 62.23%). The right side of the image shows the code editor for this submission, which contains the same C++ code as shown in the previous block. The code is a class solution that merges two sorted linked lists by comparing their values and appending the smaller one to the result list.

6. Detect a cycle in a linked list: <https://leetcode.com/problems/linked-list-cycle/description/>

Code:

```
</> Code
C++ v Auto
8  */
9  class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         if (!head || !head->next) return false;
13
14         ListNode *slow = head, *fast = head;
15
16         while (fast && fast->next != nullptr) {
17             slow = slow->next;
18             fast = fast->next->next;
19             if (slow == fast)
20                 return true;
21         }
22
23         return false;
24     }
25 };
```

Output:

The screenshot shows the LeetCode interface for the 'Detect a Cycle in a Linked List' problem. The submission status is 'Accepted' with 29/29 testcases passed. The user, Shivansh Rattan, submitted the solution on March 06, 2025, at 23:16. The performance metrics are 8 ms runtime (80.83% beats) and 11.96 MB memory (24.19% beats). A bar chart shows the user's performance relative to others. The code editor on the right displays the C++ solution using Floyd's Cycle-Finding algorithm.

```
</> Code
C++ v Auto
5  * ListNode *next;
6  * ListNode(int x) : val(x), next(NULL) {}
7  *
8  */
9  class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         if (!head || !head->next) return false;
13
14         ListNode *slow = head, *fast = head;
15
16         while (fast && fast->next != nullptr) {
17             slow = slow->next;
18             fast = fast->next->next;
19             if (slow == fast)
20                 return true;
21         }
22
23         return false;
24     }
25 }
```

7. Rotate a list: <https://leetcode.com/problems/rotate-list/description/>

Code:

```
</> Code
C++ v Auto
11 class Solution {
12 public:
13     ListNode* rotateRight(ListNode* head, int k) {
14         if (head == nullptr || head->next == nullptr || k == 0) {
15             return head;
16         }
17
18         ListNode* temp = head;
19         int length = 1;
20         while (temp->next != nullptr) {
21             temp = temp->next;
22             length++;
23         }
24
25         k = k % length;
26         if (k == 0) {
27             return head;
28         }
29
30         temp->next = head;
31
32         ListNode* newTail = head;
33         for (int i = 0; i < length - k - 1; i++) {
34             newTail = newTail->next;
35         }
36
37         ListNode* newHead = newTail->next;
38
39         newTail->next = nullptr;
40
41         return newHead;
42     }
43 }
```

Saved Ln 14, Col 1

Output:

Problem List < > < Run Submit < > Premium

Description Accepted x Editorial Solutions Submissions < >

All Submissions

Accepted 232 / 232 testcases passed
Shivansh Rattan submitted at Mar 06, 2025 23:19

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
16.24 MB | Beats 93.90%

100%
50%

Editorial Solution

```
</> Code
C++ v Auto
11 class Solution {
12 public:
13     ListNode* rotateRight(ListNode* head, int k) {
14         if (head == nullptr || head->next == nullptr || k == 0) {
15             return head;
16         }
17
18         ListNode* temp = head;
19         int length = 1;
20         while (temp->next != nullptr) {
21             temp = temp->next;
22             length++;
23         }
24
25         k = k % length;
26         if (k == 0) {
27             return head;
28         }
29
30         temp->next = head;
31
32         ListNode* newTail = head;
33         for (int i = 0; i < length - k - 1; i++) {
34             newTail = newTail->next;
35         }
36
37         ListNode* newHead = newTail->next;
38
39         newTail->next = nullptr;
40
41         return newHead;
42     }
43 }
```

Saved Ln 14, Col 1

Test Result Testcase

8. Sort List: <https://leetcode.com/problems/sort-list/description/>

Code:

```
</> Code
C++ v Auto
10  */
11  class Solution {
12  public:
13      ListNode* merge(ListNode* l1, ListNode* l2) {
14          ListNode* dummy = new ListNode(0);
15          ListNode* current = dummy;
16
17          while (l1 != nullptr && l2 != nullptr) {
18              if (l1->val < l2->val) {
19                  current->next = l1;
20                  l1 = l1->next;
21              } else {
22                  current->next = l2;
23                  l2 = l2->next;
24              }
25              current = current->next;
26          }
27
28          if (l1 != nullptr) {
29              current->next = l1;
30          } else {
31              current->next = l2;
32          }
33
34          return dummy->next;
35      }
36
37      ListNode* sortList(ListNode* head) {
38          if (head == nullptr || head->next == nullptr) {
39              return head;
40          }
41
42          ListNode *slow = head, *fast = head, *prev = nullptr;
43
44          while (fast != nullptr && fast->next != nullptr) {
45              prev = slow;
46              slow = slow->next;
```

Output:

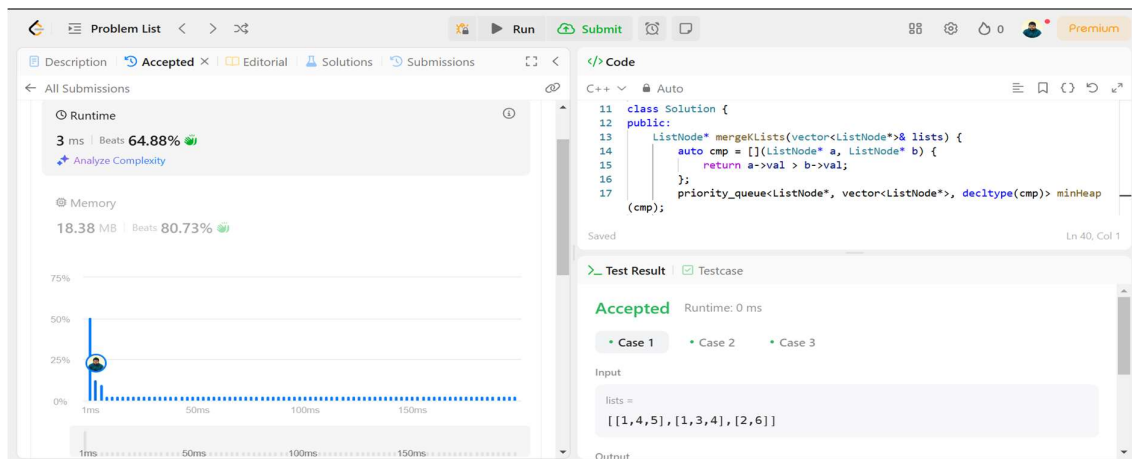
The screenshot displays the LeetCode submission page for the 'Sort List' problem. The submission is marked as 'Accepted' with 30/30 testcases passed. The user 'Shivansh Rattan' submitted it on Mar 06, 2025 at 23:22. The runtime is 51 ms, which beats 30.19% of other submissions. The memory usage is 75.56 MB, beating 46.24%. A bar chart at the bottom shows the distribution of runtime times, with a peak around 50 ms. The code editor on the right shows the C++ implementation, which includes a merge function and a sortList function that uses a slow-fast pointer technique to find the middle of the list and then merges the two halves.

9. Merge k sorted lists: <https://leetcode.com/problems/merge-k-sorted-lists/description/>

Code:

```
</> Code
C++ v Auto
11 class Solution {
12 public:
13     ListNode* mergeKLists(vector<ListNode*>& lists) {
14         auto cmp = [](ListNode* a, ListNode* b) {
15             return a->val > b->val;
16         };
17         priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> minHeap(cmp);
18
19         for (ListNode* list : lists) {
20             if (list != nullptr) {
21                 minHeap.push(list);
22             }
23         }
24
25         ListNode* dummy = new ListNode();
26         ListNode* current = dummy;
27
28         while (!minHeap.empty()) {
29             ListNode* node = minHeap.top();
30             minHeap.pop();
31
32             current->next = node;
33             current = current->next;
34
35             if (node->next != nullptr) {
36                 minHeap.push(node->next);
37             }
38         }
39
40         return dummy->next;
41     }
```

Output:



The End