

Ap-Assignment 3

Name: Harmandeep Singh

UID:22BCS14975

Leetcode Profile: <https://leetcode.com/u/Harman001/>

Q1.Print Linkedlist:

a.Code:

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

The screenshot shows the LeetCode interface for the 'Print Linked List' problem. The left pane displays the 'Output Window' with 'Compilation Results' indicating 'Problem Solved Successfully'. It shows 'Test Cases Passed: 1112 / 1112', 'Attempts: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 0.07'. The right pane shows the C++ code for the solution, which is identical to the code provided in the text above. The code defines a Node struct and a printList function that traverses the linked list and prints its elements on the same line.

Q2.Remove Duplicates from Sorted list

```

a. 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;
    }
};

```

b. Output

The screenshot displays a code editor interface for a C++ solution. The left sidebar shows the submission status: "Accepted" with 169/168 testcases passed, submitted on Feb 13, 2025. The runtime is 0 ms, beating 100.00% of other solutions. The memory usage is 16.29 MB, beating 35.32%. A bar chart shows the runtime distribution, with the solution being the fastest. The code editor shows the following C++ code:

```

10  */
11  class Solution {
12  public:
13      ListNode* deleteDuplicates(ListNode* head) {
14          ListNode* current = head;
15
16          while (current != nullptr && current->next != nullptr) {
17              if (current->val == current->next->val) {
18                  ListNode* temp = current->next;
19                  current->next = current->next->next;
20                  delete temp;
21              } else {
22                  current = current->next;
23              }
24          }
25      }
26  };

```

The right sidebar shows the test result for "Case 1". The input is "head = [1,1,2]" and the output is "[1,2]". The expected output is also "[1,2]".

Q3.Reverse Linkedlist

```

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;
    }

    ListNode* createLinkedList(const vector<int>& nums) {
        if (nums.empty()) return nullptr;

        ListNode* head = new ListNode(nums[0]);
        ListNode* current = head;
        for (size_t i = 1; i < nums.size(); i++) {
            current->next = new ListNode(nums[i]);
            current = current->next;
        }
        return head;
    }
}

```

206. Reverse Linked List Solved

Easy Topics Companies

Given the `head` of a singly linked list, reverse the list, and return the reversed list.

Example 1:

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

Example 2:

Input: `head = [1,2]`
Output: `[2,1]`

Runtime
0 ms | Beats 100.00%

Memory
13.27 MB | Beats 90.90%

Code | C++

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 */

```

Testcase | Test Result

Q4.Delete Middle Node

```

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 displays the LeetCode submission page for the 'Delete Middle Node' problem. The code is written in C++ and is accepted. The runtime is 0 ms, and the memory usage is 312.14 MB, which beats 17.99% of other submissions. The test results show that the solution passed all test cases.

Runtime: 0 ms | Beats 100.00%

Memory: 312.14 MB | Beats 17.99%

Testcase 1: Test Result

Accepted Runtime: 0 ms

Case 1

Input: head = [1,3,4,7,1,2,6]

Output: [1,3,4,1,2,6]

Expected: [1,3,4,1,2,6]

Q5.Merge two Sorted linkedlist

```

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;
    }

    void printList(ListNode* head) {
        while (head) {
            std::cout << head->val << " -> ";
            head = head->next;
        }
        std::cout << "NULL" << std::endl;
    }
};

```

The screenshot displays a code submission interface. On the left, a table shows the submission status: 'Accepted' 9 hours ago, C++ language, 0 ms runtime, and 19.6 MB memory. The main panel shows the submission details for 'Harmandeep' submitted on Feb 13, 2025, at 10:59. It indicates that 208 out of 208 test cases passed. The runtime is 0 ms, beating 100.00% of other submissions, and the memory usage is 19.63 MB, beating 10.67%. A bar chart shows the runtime distribution, with a single bar at 0 ms. Below the chart, the C++ code is displayed, defining a singly-linked list structure and a mergeTwoLists function.

Submission Details:

- Status: Accepted (208 / 208 testcases passed)
- Submitted by: Harmandeep (submitted at Feb 13, 2025 10:59)
- Language: C++
- Runtime: 0 ms | Beats 100.00%
- Memory: 19.63 MB | Beats 10.67%

Code (C++):

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };

```

Q6.Remove duplicates from sorted list2

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (!head) return nullptr;

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prev = dummy;

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

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

The screenshot displays the LeetCode interface for the problem "Remove Duplicates from Sorted List II". The left sidebar shows the submission status as "Accepted" with 166/166 testcases passed. The user's submission is by "Harma..." and was submitted on Feb 13, 2025 at 19:40. The runtime is 0 ms, which is 100.00% better than other submissions. The memory usage is 15.75 MB, which is 41.61% better. A bar chart shows the user's performance compared to other submissions. The right side shows the C++ code for the solution, which uses a dummy node and a while loop to remove duplicates. The bottom section shows the test results, indicating that the solution is accepted for all test cases.

```
Accepted 166 / 166 testcases passed
Harma... submitted at Feb 13, 2025 19:40
Editorial Solution

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
15.75 MB | Beats 41.61%

Code | C++
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 */

14     if (!head) return nullptr;
15
16     ListNode* dummy = new ListNode(0);
17     dummy->next = head;
18     ListNode* prev = dummy;
19
20     while (head) {
21         if (head->next && head->val == head->next->val) {
22             while (head->next && head->val == head->next->val) {
23                 head = head->next;
24             }
25             prev->next = head->next;
26         } else {
27             prev = prev->next;
28         }
29         head = head->next;
30     }
31
32     return dummy->next;
33 }
34
```

Accepted Runtime: 0 ms

Case 1 Case 2

Q7.Detect Cycle in Linkedlist

```
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;
    }

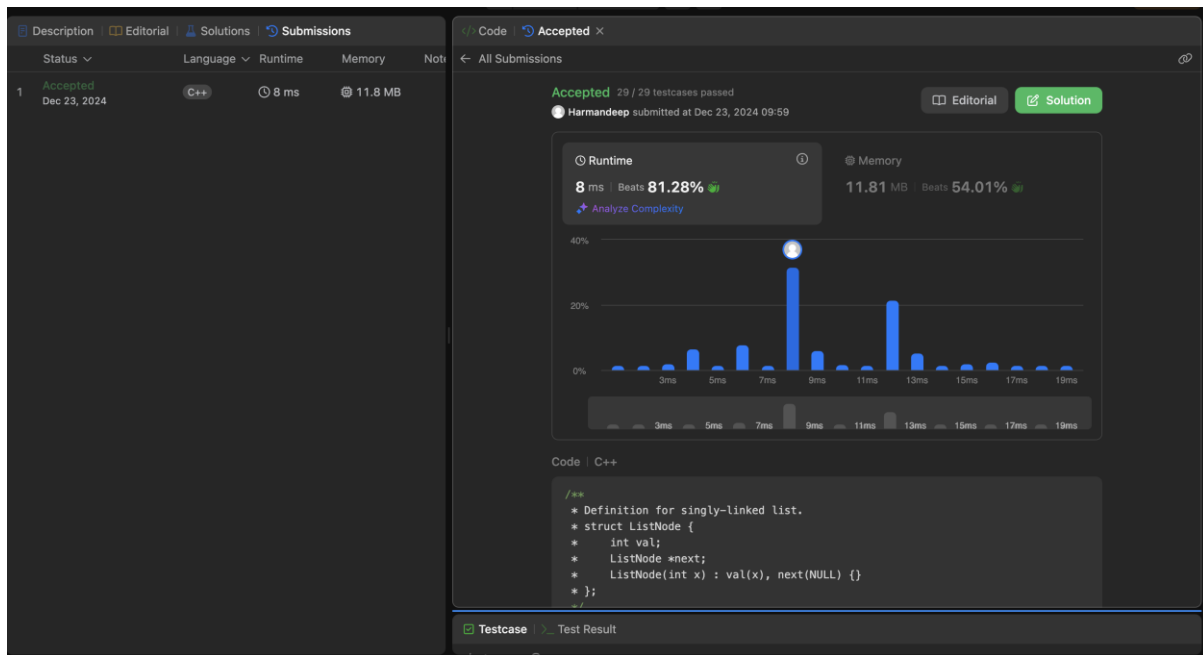
    ListNode* createCycleList(const vector<int>& nums, int pos) {
        if (nums.empty()) return nullptr;

        ListNode* head = new ListNode(nums[0]);
        ListNode* current = head;
        ListNode* cycleNode = nullptr;

        for (size_t i = 1; i < nums.size(); i++) {
            current->next = new ListNode(nums[i]);
            current = current->next;
            if (static_cast<int>(i) == pos) {
                cycleNode = current;
            }
        }

        if (pos >= 0) {
            current->next = cycleNode;
        }

        return head;
    }
};
```



Q8.Reverse LinkedList 2

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) return head;

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prev = dummy;

        for (int i = 1; i < left; i++) {
            prev = prev->next;
        }

        ListNode* curr = prev->next;
        ListNode* next = nullptr;

        ListNode* prevReversed = nullptr;
        for (int i = 0; i <= right - left; i++) {
            next = curr->next;
            curr->next = prevReversed;
            prevReversed = curr;
            curr = next;
        }

        prev->next->next = curr;
```



```

prev->next = prevReversed;

return dummy->next;
}
};

```

The screenshot displays a LeetCode submission for a C++ solution. The left sidebar provides submission statistics: 'Accepted' status, 44/44 testcases passed, a runtime of 0 ms (100.00% beats), and a memory usage of 11.16 MB (73.57% beats). A bar chart visualizes the runtime distribution. The main area shows the C++ code for reversing a linked list between left and right indices. The test result section indicates 'Accepted' for Case 1 with input head=[1,2,3,4,5], left=2, and right=4.

Q9.Rotate a list

```

class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;

```

```

        int n = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            n++;
        }

```

```

        k = k % n;
        if (k == 0) return head;

```

```

        ListNode* newTail = head;
        for (int i = 0; i < n - k - 1; i++) {
            newTail = newTail->next;
        }

```

```

ListNode* newHead = newTail->next;
newTail->next = nullptr;
tail->next = head;

return newHead;
}
};

```

Submission Details:

- Status: Accepted
- Testcases: 232 / 232 passed
- Runtime: 0 ms | Beats 100.00%
- Memory: 16.44 MB | Beats 31.94%

Code (C++):

```

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
};

class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;

        int n = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            n++;
        }
    }
};

```

Test Result:

Accepted Runtime: 0 ms

Case 1: Input: head = [1,2,3,4,5], k = 2. Output: [4,5,1,2,3]

Q10.Detect a cycle in Linkedlist 2

```

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

        ListNode* slow = head;
        ListNode* fast = head;

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

        if (!fast || !fast->next) return nullptr;
    }
};

```

```

slow = head;
while (slow != fast) {
    slow = slow->next;
    fast = fast->next;
}

return slow;
}

void createCycle(ListNode* head, int pos) {
    if (pos == -1) return;

    ListNode* cycleNode = head;
    ListNode* tail = head;
    int index = 0;

    while (tail->next) {
        if (index == pos) cycleNode = tail;
        tail = tail->next;
        index++;
    }

    tail->next = cycleNode;
}
};

```

Accepted
18 / 18 testcases passed
Harma... submitted at Feb 13, 2025 19:50

Runtime
11 ms | Beats 19.48%

Memory
11.47 MB | Beats 24.87%

Bar chart showing runtime distribution across test cases. The x-axis represents runtime in milliseconds (3ms to 13ms), and the y-axis represents the percentage of test cases (0% to 30%). The distribution shows a peak at 11ms.

Code | C++

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 * };
 */

```

Code
C++

```

2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9  class Solution {
10 public:
11     ListNode *detectCycle(ListNode *head) {
12         if (!head || !head->next) return nullptr;
13
14         ListNode* slow = head;
15         ListNode* fast = head;

```

Testcase
Test Result

Accepted
Runtime: 0 ms

Case 1
Case 2
Case 3

Input
head =
[3,2,0,-4]
pos =
1

Output
tail connects to node index 1