

Name – Suman Kumar

UID – 22BCS15488

Section – 608/B

AP LAB ASSIGNMENT 2

1. Print Linked List

The screenshot shows a web-based IDE interface for solving the 'Print Linked List' problem. The left sidebar displays the 'Output Window' with 'Compilation Results' and 'Problem Solved Successfully' status. It shows 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 2 / 3', 'Accuracy: 66%', and 'Time Taken: 1.4'. The main editor area shows the Java code for the solution, which includes a linked list node definition and a function to print the list elements. The code is as follows:

```
1  // Node is defined as
2
3  class Node {
4      int data;
5      Node next;
6  }
7
8  // add code here
9
10 // Print elements of a Linked List on console
11 // Head pointer input could be NULL as well for empty list
12
13 class Solution {
14     // Function to display the elements of a Linked List in same line
15     void printList(Node head) {
16         // add code here
17         StringBuilder sb = new StringBuilder();
18         while(head != null){
19             sb.append(head.data).append(" ");
20             head = head.next;
21         }
22         System.out.print(sb.toString().trim());
23     }
24 }
```

2. Remove duplicates from a sorted list

The screenshot shows a web-based IDE interface for solving the 'Remove duplicates from a sorted list' problem. The left sidebar displays the 'Accepted' status with '168 / 168 testcases passed' and 'Suman Kumar submitted at Mar 07, 2025 20:29'. It also shows a runtime graph and a table of test cases. The main editor area shows the C++ code for the solution, which includes a linked list node definition and a function to remove duplicates. The code is as follows:

```
1  // Definition for singly-linked list.
2  struct ListNode {
3      int val;
4      ListNode *next;
5      ListNode() : val(0), next(nullptr) {}
6      ListNode(int x) : val(x), next(nullptr) {}
7      ListNode(int x, ListNode *next) : val(x), next(next) {}
8  };
9
10 class Solution {
11 public:
12     ListNode* deleteDuplicates(ListNode* head) {
13         ListNode* res = head;
14         while (head && head->next) {
15             if (head->val == head->next->val) {
16                 head->next = head->next->next;
17             } else {
18                 head = head->next;
19             }
20         }
21         return res;
22     }
23 }
```

3. Reverse a linked list

206. Reverse Linked List

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:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* node = nullptr;
        while (head != nullptr) {
            ListNode* temp = head->next;
            head->next = node;
            node = head;
            head = temp;
        }
        return node;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

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

Output:

4. Delete middle node of a list

2095. Delete the Middle Node of a Linked List

You are given the **head** of a linked list. **Delete the middle node**, and return the **head** of the modified linked list.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

For $n = 1, 2, 3, 4$, and 5 , the middle nodes are $0, 1, 1, 2$, and 2 , respectively.

Example 1:

Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation: The above figure represents the given linked list. The indices of the nodes are written below. Since $n = 7$, node 3 with value 7 is the middle node, which is marked in red. We return the new list after removing this node.

Example 2:

Input: head = [1,2,3,4]
Output: [1,2,4]
Explanation: The above figure represents the given linked list. For $n = 4$, node 2 with value 3 is the middle node, which is marked in red.

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr; // If only 1 node, return nullptr
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr; // To keep track of node before middle
        // Move fast by 2 steps and slow by 1 step
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }
        // Delete middle node
        prev->next = slow->next;
        delete slow; // Free memory
        return head;
    }
};
```

Case 1 Case 2 Case 3

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

5. Merge two sorted linked lists

21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists **list1** and **list2**.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return the **head** of the merged linked list.

Example 1:

Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []
Output: []

Example 3:

Input: list1 = [], list2 = [0]
Output: [0]

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* curr1 = list1;
        ListNode* curr2 = list2;
        ListNode* dummyNode = new ListNode(-1);
        ListNode* temp = dummyNode;
        while (curr1 && curr2) {
            if (curr1->val < curr2->val) {
                temp->next = curr1;
                temp = curr1;
                curr1 = curr1->next;
            } else {
                temp->next = curr2;
                temp = curr2;
                curr2 = curr2->next;
            }
        }
        if (curr1) temp->next = curr1;
        if (curr2) temp->next = curr2;
        return dummyNode->next;
    }
};
```

list1 = [1,2,4]
list2 = [1,3,4]
Output: [1,1,2,3,4,4]
Expected: [1,1,2,3,4,4]

6. Detect a cycle in a linked list

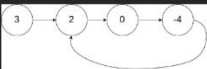
141. Linked List Cycle

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

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. **Note that `pos` is not passed as a parameter.**


Return `true` if there is a cycle in the linked list. Otherwise, return `false`.

Example 1:




Input: `head = [3,2,0,-4]`, `pos = 1`
Output: `true`
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input: `head = [1,2]`, `pos = 0`
Output: `true`
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:



Input: `head = [1]`, `pos = -1`
Output: `false`
Explanation: There is no cycle in the linked list.

Code:

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

Testcase / Test Result

Accepted Runtime: 0 ms

Case 1 **Case 2** **Case 3**

Input:

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

`pos =`
`1`

7. Rotate a list

Making the List Circular
Breaking the Cycle at the Correct Position

Code

Python3

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if head==None or head.next==None or k==0:
            return head
        l=1
        curr=head
        while curr.next:
            curr=curr.next
            l+=1
        r=k%l
        k=l-r
        curr.next=head
        while k>0:
            curr=curr.next
            k-=1
        head=curr.next
        curr.next = None
        return head
```

Python3

```
2 # class ListNode:
3 #     def __init__(self, val=0, next=None):
4 #         self.val = val
5 #         self.next = next
6 class Solution:
7     def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
8         if head==None or head.next==None or k==0:
9             return head
10        l=1
11        curr=head
12        while curr.next:
13            curr=curr.next
14            l+=1
15        r=k%l
16        k=l-r
17        curr.next=head
18        while k>0:
19            curr=curr.next
20            k-=1
21        head=curr.next
22        curr.next = None
23        return head
```

Testcase / Test Result

Accepted Runtime: 0 ms

Case 1 **Case 2**

Input:

`head =`
`[1,2,3,4,5]`

`k =`
`2`

8. Sort List

148. Sort List

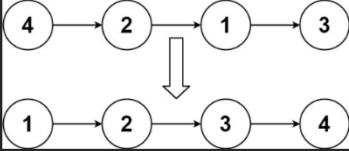
Medium

Topics

Companies

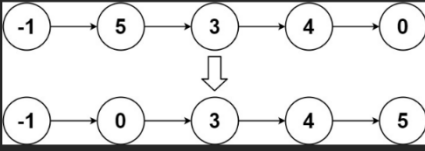
Given the head of a linked list, return the list after sorting it in ascending order.

Example 1:



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

Example 2:



Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]

Example 3:

Input: head = []
Output: []

12.2K 111 123 Online

Code

C++

```
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;
    }
};
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head = [4,2,1,3]

Output

[1,2,3,4]

9. Merge k sorted lists

23. Merge k Sorted Lists

Hard

Topics

Companies

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
1->4->5,
1->3->4,
2->6
merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:

Input: lists = []
Output: []

Example 3:

Input: lists = [[]]
Output: []

Constraints:

- $k == \text{lists.length}$
- $0 \leq k \leq 10^4$
- $0 \leq \text{lists[i].length} \leq 500$
- $-10^4 \leq \text{lists[i][j]} \leq 10^4$

20.1K 253 247 Online

Code

C++

```
return merge(left, right);
}

ListNode* merge(ListNode* l1, ListNode* l2) {
    ListNode* dummy = new ListNode(0);
    ListNode* curr = dummy;

    while (l1 && l2) {
        if (l1->val < l2->val) {
            curr->next = l1;
            l1 = l1->next;
        } else {
            curr->next = l2;
            l2 = l2->next;
        }
        curr = curr->next;
    }
    curr->next = l1 ? l1 : l2;
    return dummy->next;
}
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

lists = [[1,4,5],[1,3,4],[2,6]]

Output

[1,1,2,3,4,4,5,6]