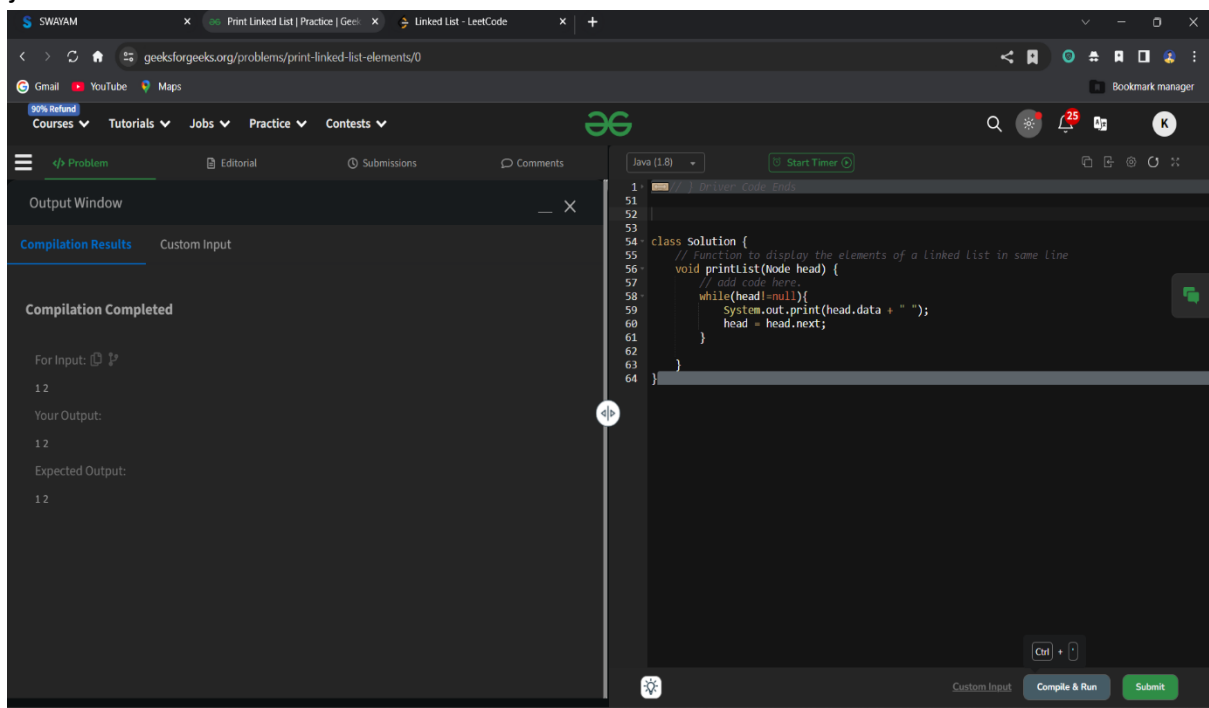


Assignment no.3

Ques 1. [Print linked list](#)

```
class Solution {  
    // Function to display the elements of a linked list in same line  
    void printList(Node head) {  
        // add code here.  
        while(head!=null){  
            System.out.print(head.data + " ");  
            head = head.next;  
        }  
    }  
}
```



```

public ListNode deleteDuplicates(ListNode head) {
    ListNode res = head;

    while (head != null && head.next != null) {
        if (head.val == head.next.val) {
            head.next = head.next.next;
        } else {
            head = head.next;
        }
    }

    return res;
}
}

```

The screenshot shows a web browser displaying the LeetCode problem page for "83. Remove Duplicates from Sorted List". The problem description states: "Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well." An example shows a linked list with nodes 1, 1, 2 being transformed into a linked list with nodes 1, 2. The input is head = [1,1,2] and the output is [1,2].

On the right side of the screenshot, the Java code for the solution is displayed in a code editor. The code is as follows:

```

class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode res = head;

        while (head != null && head.next != null) {
            if (head.val == head.next.val) {
                head.next = head.next.next;
            } else {
                head = head.next;
            }
        }

        return res;
    }
}

```

Below the code editor, the "Test Result" section shows that the solution is "Accepted" with a runtime of 0 ms. There are two test cases, "Case 1" and "Case 2", both of which passed.

Ques 3. Reverse a linked list] (<https://leetcode.com/problems/reverse-linked-list/>)

```

class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode node = null;
    }
}

```

```

while (head != null) {
    ListNode temp = head.next;
    head.next = node;
    node = head;
    head = temp;
}

return node;
}
}

```

206. Reverse Linked List

Easy Topics Companies

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

Example 1:

Diagram showing a linked list with nodes 1, 2, 3, 4, 5. An arrow points down to the reversed list with nodes 5, 4, 3, 2, 1.

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

Example 2:

Diagram showing a linked list with nodes 1, 2, 3, 4, 5. An arrow points down to the reversed list with nodes 5, 4, 3, 2, 1.

Code:

```

1 class Solution {
2     public ListNode reverseList(ListNode head) {
3         ListNode node = null;
4
5         while (head != null) {
6             ListNode temp = head.next;
7             head.next = node;
8             node = head;
9             head = temp;
10        }
11
12        return node;
13    }
14 }

```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

Ques 4. [Delete middle node of a list](#)

```

class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if(head == null) return null;

```

```

ListNode prev = new ListNode(0);

prev.next = head;

ListNode slow = prev;

ListNode fast = head;

while(fast != null && fast.next != null){

    slow = slow.next;

    fast = fast.next.next;

}

slow.next = slow.next.next;

return prev.next;

}

```

2095. Delete the Middle Node of a Linked List Solved

Medium Topics Companies Hint

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 $\lfloor n / 2 \rfloor^{\text{th}}$ 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.

```

class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if(head == null) return null;
        ListNode prev = new ListNode(0);
        prev.next = head;
        ListNode slow = prev;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        slow.next = slow.next.next;
        return prev.next;
    }
}

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Ques 5. [Merge two sorted linked lists](#)

```

class Solution {

    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

        if(list1!=null && list2!=null){

```

```

        if(list1.val<list2.val){
            list1.next=mergeTwoLists(list1.next,list2);
            return list1;
        }
        else{
            list2.next=mergeTwoLists(list1,list2.next);
            return list2;
        }
    }
}

if(list1==null)
    return list2;

return list1;

}
}

```

The screenshot shows the LeetCode interface for the problem '21. Merge Two Sorted Lists'. The problem description states: '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.' An example is provided with two lists: List 1 (1, 2, 4) and List 2 (1, 3, 4), which are merged into a new list (1, 2, 3, 4, 4). The code editor on the right shows a Java solution that implements the merge logic using recursive calls. The test results at the bottom show 'Accepted' with a runtime of 0 ms for three test cases.

Ques 6. [Remove duplicates from sorted lists 2](#)

```

class Solution {
    public ListNode deleteDuplicates(ListNode head) {

```

```
ListNode ans = new ListNode(1000, head); // Dummy node to handle edge cases
```

```
ListNode cur = ans;
```

```
while (cur.next != null && cur.next.next != null) {
```

```
    if (cur.next.val == cur.next.next.val) { // Check if duplicates exist
```

```
        int val = cur.next.val;
```

```
        while (cur.next != null && cur.next.val == val) { // Skip all duplicates
```

```
            cur.next = cur.next.next;
```

```
        }
```

```
    } else {
```

```
        cur = cur.next; // Move to the next node
```

```
    }
```

```
}
```

```
return ans.next; // Return the modified list starting after the dummy node
```

```
}
```

```
}
```

The screenshot displays the LeetCode interface for problem 82, "Remove Duplicates from Sorted List II". The problem description states: "Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well." Example 1 shows an input list [1,2,3,3,4,4,5] being transformed into an output list [1,2,5]. Example 2 shows an input list [1,1,1,2,3] being transformed into an output list [2,3]. The code pane on the right contains the following Java code:

```
1 class Solution {
2     public ListNode deleteDuplicates(ListNode head) {
3         ListNode ans = new ListNode(1000, head); // Dummy node to handle edge cases
4         ListNode cur = ans;
5
6         while (cur.next != null && cur.next.next != null) {
7             if (cur.next.val == cur.next.next.val) { // Check if duplicates exist
8                 int val = cur.next.val;
9                 while (cur.next != null && cur.next.val == val) { // Skip all duplicates
10                     cur.next = cur.next.next;
11                 }
12             } else {
13                 cur = cur.next; // Move to the next node
14             }
15         }
16         return ans.next; // Return the modified list starting after the dummy node
17     }
18 }
19 }
```

Ques 7. [Detect a cycle in a linked list](#)

```

public class Solution {

    public boolean hasCycle(ListNode head) {

        ListNode fast = head;

        ListNode slow = head;

        while (fast != null && fast.next != null) {

            fast = fast.next.next;

            slow = slow.next;

            if (fast == slow) {

                return true;

            }

        }

        return false;

    }

}

```

The screenshot shows the LeetCode interface for problem 141, "Linked List Cycle". The problem description states: "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." An example diagram shows a linked list with nodes 3, 2, 0, -4, where the tail (-4) points back to node 2, creating a cycle. The input is head = [3,2,0,-4], pos = 1, and the output is true.

The Java solution is as follows:

```

1 public class Solution {
2     public boolean hasCycle(ListNode head) {
3         ListNode fast = head;
4         ListNode slow = head;
5
6         while (fast != null && fast.next != null) {
7             fast = fast.next.next;
8             slow = slow.next;
9
10            if (fast == slow) {
11                return true;
12            }
13        }
14
15        return false;
16    }
17 }

```

The test result shows "Accepted" with a runtime of 0 ms.

Ques 8. [Reverse linked list 2](#)

```
class Solution {  
    public ListNode reverseBetween(ListNode head, int left, int right) {  
        if (head == null || left == right) {  
            return head;  
        }  
  
        ListNode dummy = new ListNode(0);  
        dummy.next = head;  
        ListNode prev = dummy;  
  
        for (int i = 0; i < left - 1; i++) {  
            prev = prev.next;  
        }  
  
        ListNode cur = prev.next;  
  
        for (int i = 0; i < right - left; i++) {  
            ListNode temp = cur.next;  
            cur.next = temp.next;  
            temp.next = prev.next;  
            prev.next = temp;  
        }  
        return dummy.next;  
    }  
}
```


92. Reverse Linked List II

Medium

Given the head of a singly linked list and two integers left and right where left ≤ right, reverse the nodes of the list from position left to position right, and return the reversed list.

Example 1:

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

Example 2:

```

1 class Solution {
2     public ListNode reverseBetween(ListNode head, int left, int right) {
3         if (head == null || left == right) {
4             return head;
5         }
6
7         ListNode dummy = new ListNode(0);
8         dummy.next = head;
9         ListNode prev = dummy;
10
11         for (int i = 0; i < left - 1; i++) {
12             prev = prev.next;
13         }
14
15         ListNode cur = prev.next;
16
17         for (int i = 0; i < right - left; i++) {
18             ListNode temp = cur.next;
19             cur.next = temp.next;
20         }
21     }
22 }

```

Accepted

Runtime: 0 ms

Case 1 Case 2

Ques 9. [rotate a list](#)

```
class Solution {
```

```
    public ListNode rotateRight(ListNode head, int k) {
```

```
        if(head==null)
```

```
            return head;
```

```
        int size=size(head);
```

```
        if(k==0 || k%size==0)
```

```
            return head;
```

```
        if(k>size){
```

```
            k=k%size;
```

```
        }
```

```
        k=size-k;
```

```

        ListNode temp=head;
        while(temp.next!=null){
            temp=temp.next;
        }
        ListNode tail=temp;
        temp=head;
        while(k!=1)
        {
            temp=temp.next;
            k--;
        }

        tail.next=head;
        head=temp.next;
        temp.next=null;
        return head;

    }

    public int size(ListNode head){
        int size=0;
        while(head!=null){
            head=head.next;
            size++;
        }return size;
    }
}

```

leetcode.com/problems/rotate-list/

Problem List < > < >

Description Editorial Solutions Submissions

61. Rotate List

Medium Topics Companies

Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:

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

Example 2:

10.2K 101 0 Online

Code Accepted

```

1 class Solution {
2     public ListNode rotateRight(ListNode head, int k) {
3         if(head==null)
4             return head;
5
6         int size=size(head);
7
8         if(k==0 || k%size==0)
9             return head;
10
11        if(k>size){
12            k=k%size;
13        }
14
15        k=size-k;
16
17        ListNode temp=head;
18        while(temp.next!=null){
19            temp=temp.next;
20        }
21    }
22 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

Ques 10. [Sort List](#)

```
public class Solution {
```

```
    public ListNode sortList(ListNode head) {
```

```
        if (head == null || head.next == null)
```

```
            return head;
```

```
        // step 1. cut the list to two halves
```

```
        ListNode prev = null, slow = head, fast = head;
```

```
        while (fast != null && fast.next != null) {
```

```
            prev = slow;
```

```
            slow = slow.next;
```

```
            fast = fast.next.next;
```

```
        }
```

```
        prev.next = null;
```

```

// step 2. sort each half
ListNode l1 = sortList(head);
ListNode l2 = sortList(slow);

// step 3. merge l1 and l2
return merge(l1, l2);
}

ListNode merge(ListNode l1, ListNode l2) {
    ListNode l = new ListNode(0), p = l;

    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            p.next = l1;
            l1 = l1.next;
        } else {
            p.next = l2;
            l2 = l2.next;
        }
        p = p.next;
    }

    if (l1 != null)
        p.next = l1;

```

```

    if (l2 != null)
        p.next = l2;

    return l.next;
}
}

```

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]`

```

1 public class Solution {
2     public ListNode sortList(ListNode head) {
3         if (head == null || head.next == null)
4             return head;
5         // step 1. cut the list to two halves
6         ListNode prev = null, slow = head, fast = head;
7         while (fast != null && fast.next != null) {
8             prev = slow;
9             slow = slow.next;
10            fast = fast.next.next;
11        }
12        prev.next = null;
13        // step 2. sort each half
14        ListNode l1 = sortList(head);
15    }
16 }

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Ques 11. [Detect a cycle in a linked list 2](#)

```

public class Solution {

    public ListNode detectCycle(ListNode head) {

        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) break;

        }

        if (fast == null || fast.next == null) return null;

        while (head != slow) {

```

```
        head = head.next;
        slow = slow.next;
    }
    return head;
}
```

142. Linked List Cycle II

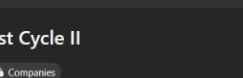
MediumTopicsCompanies

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.

Code

JavaAuto

```
1 public class Solution {
2     public ListNode detectCycle(ListNode head) {
3         ListNode slow = head, fast = head;
4         while (fast != null && fast.next != null) {
5             slow = slow.next;
6             fast = fast.next.next;
7             if (slow == fast) break;
8         }
9         if (fast == null || fast.next == null) return null;
10        while (head != slow) {
11            head = head.next;
12            slow = slow.next;
13        }
14        return head;
15    }
16 }
```

SavedLn 16, Col 2

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2Case 3

Input