**LC#19:REMOVE nth NODE FROM END OF THE LIST**

BRUTE FORCE:

* Count the occurrence first and then subtract with that given ‘n’ such that we can find the node to be deleted.
* **Edge:** if target==0 return head.next(remove first and return)

**APPROACH:**

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

ListNode temp = head;

int count = 0;

while(temp!=null)

{

count++;

temp = temp.next;

}

int res = count -n;

temp = head;

ListNode prev = null;

if(res ==0)

return head.next;

int i =1;

while(i<res)

{

temp= temp.next;

i++;

}

temp.next = temp.next.next;

return head; }}

**OPTIMAL:**

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

int count = 0;

ListNode fast = head, slow = head;

while(count<n)

{

fast = fast.next;

count++;

}

if (fast == null) {

return head.next;

}

while(fast.next!=null)

{

fast = fast.next;

slow = slow.next;

}

slow.next=slow.next.next;

return head;

}

}

**LC#876: MIDDLE OF THE LINKED LIST**

**APPROACH : FAST TWICE SLOW ONCE -> when fast is null then slow will be in the middle of the linked list**

CODE:

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode middleNode(ListNode head) {

        ListNode fast = head , slow = head;

        while(fast!=null)

        {

            fast = fast.next;

            if(fast!=null)

            {

                fast = fast.next;

                slow = slow.next;

            }

        }

        return slow;}}

**LC#2095:DELETE THE MIDDLE OF THE LINKED LIST**

class Solution {

    public ListNode deleteMiddle(ListNode head) {

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

            return null;

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

        while(fast!=null)

        {

            fast = fast.next;

            if(fast!=null)

            {

                prev = slow;

                fast = fast.next;

                slow = slow.next;

            }

        }

        prev.next = slow.next;

        return head;

    }

}