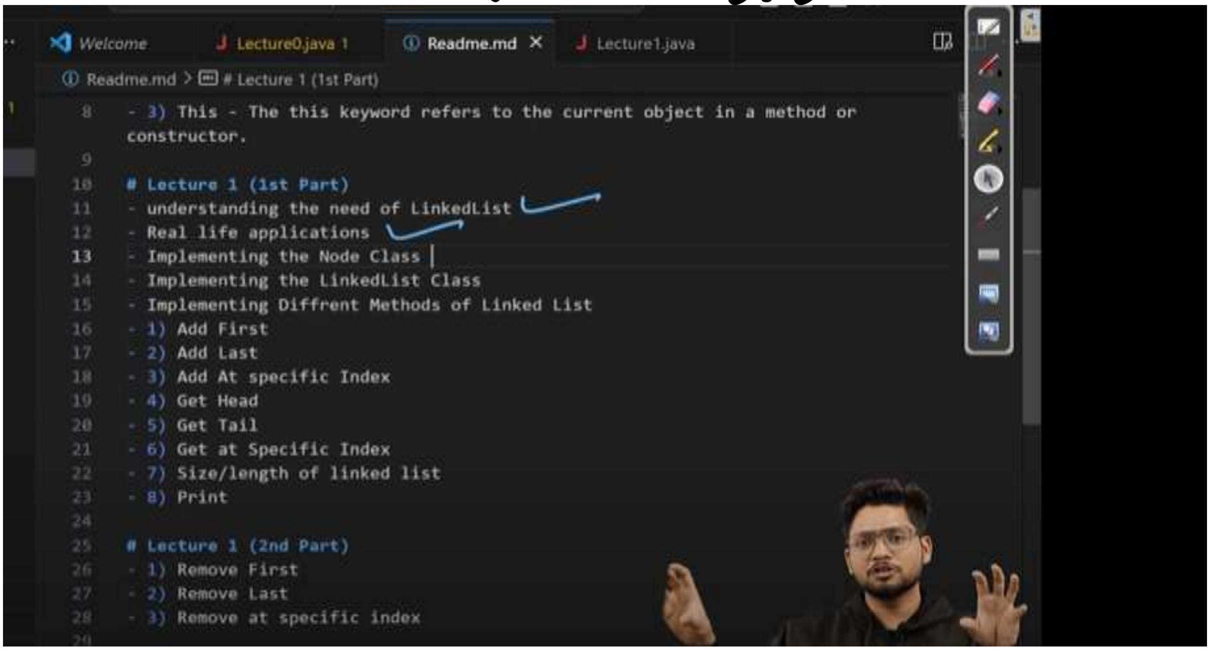
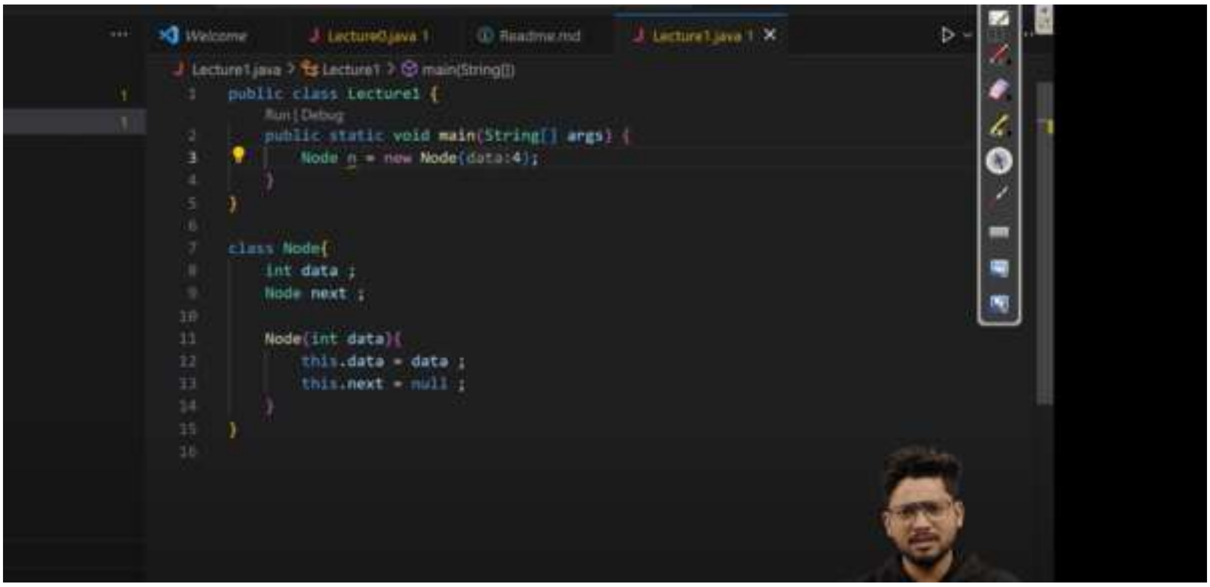
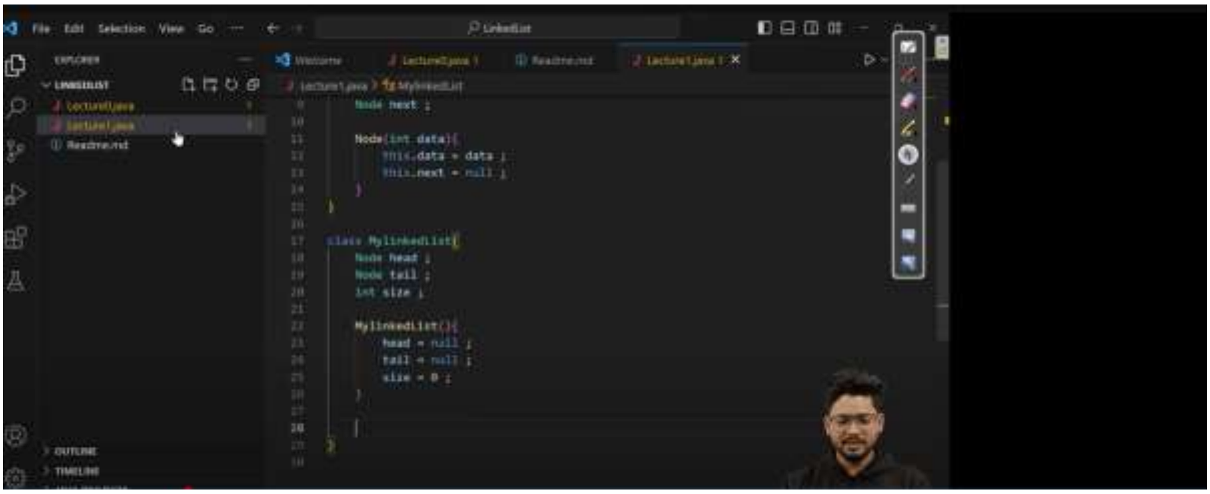
Linked List(gfg)



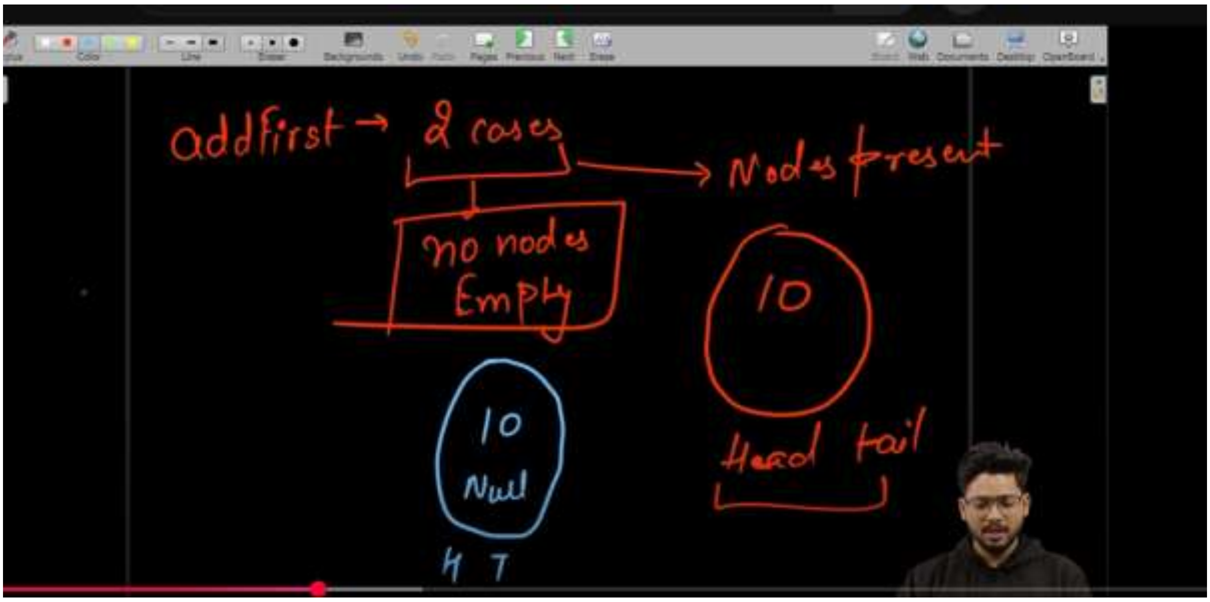
```
- 3) This - The this keyword refers to the current object in a method or
constructor.

# Lecture 1 (1st Part)
- understanding the need of LinkedList
- Real life applications
- Implementing the Node Class
- Implementing the LinkedList Class
- Implementing Diffrent Methods of Linked List
- 1) Add First
- 2) Add Last
- 3) Add At specific Index
- 4) Get Head
- 5) Get Tail
- 6) Get at Specific Index
- 7) Size/length of linked list
- 8) Print

# Lecture 1 (2nd Part)
- 1) Remove First
- 2) Remove Last
- 3) Remove at specific index
```

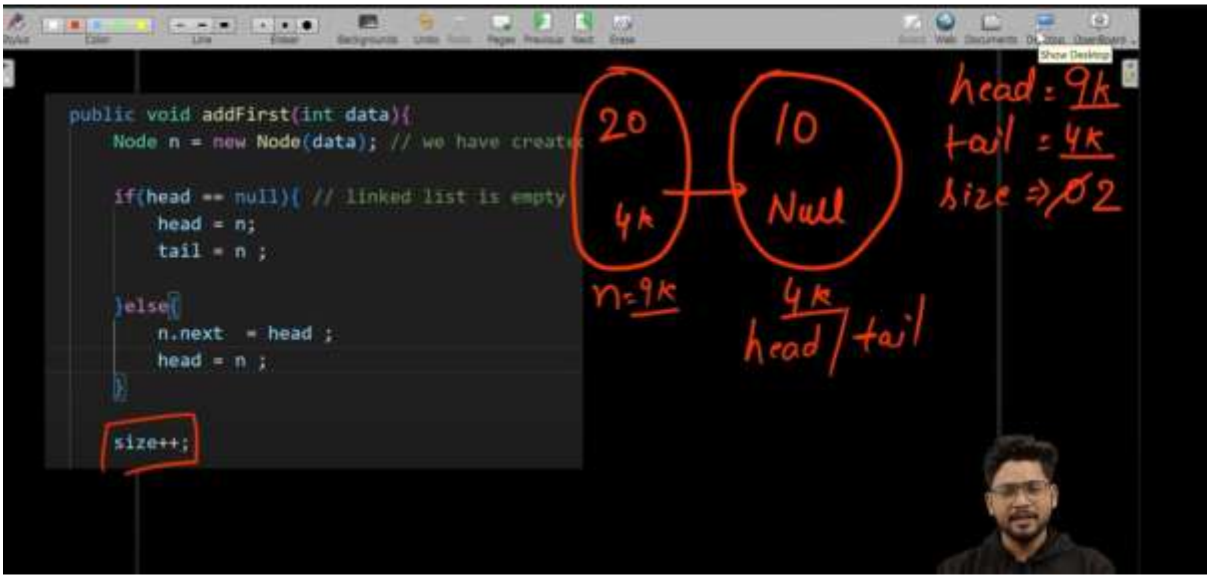


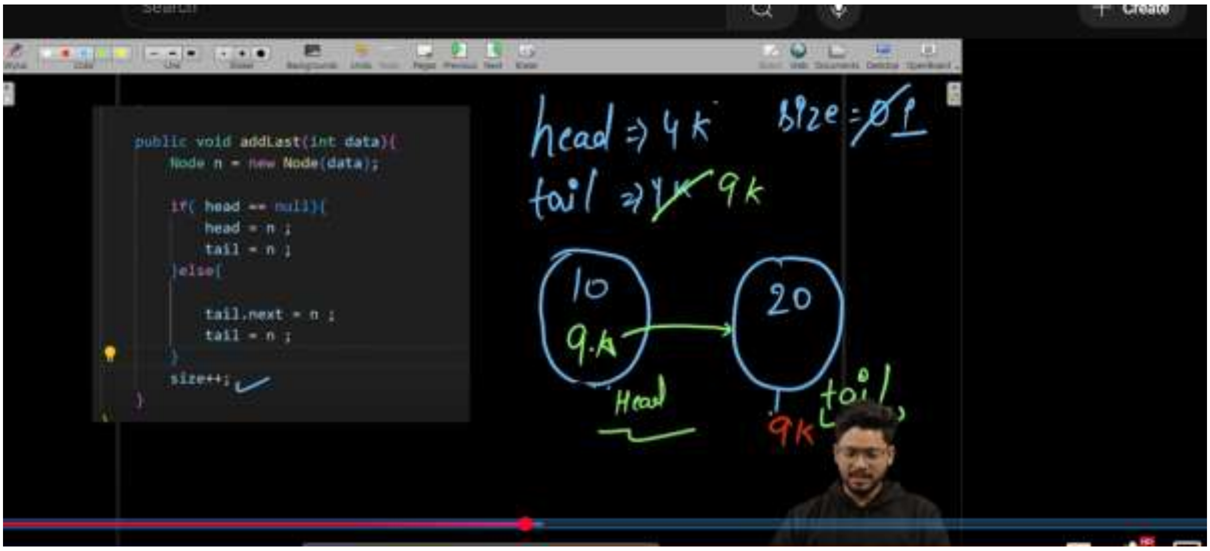
```java
public class Lecture1 {
    public static void main(String[] args) {
        Node n = new Node(data:4);
    }
}

class Node{
    int data ;
    Node next ;

    Node(int data){
        this.data = data ;
        this.next = null ;
    }
}
```

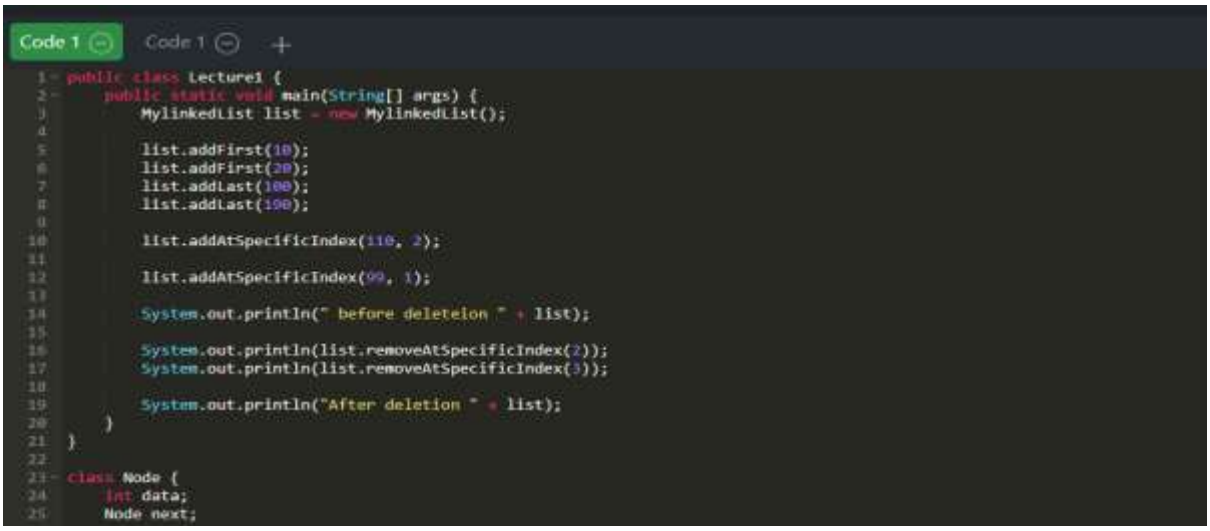


```java
        Node next ;

        Node(int data){
            this.data = data ;
            this.next = null ;
        }
    }

class MylinkedList{
        Node head ;
        Node tail ;
        int size ;

        MylinkedList(){
            head = null ;
            tail = null ;
            size = 0 ;
        }

        
}
```

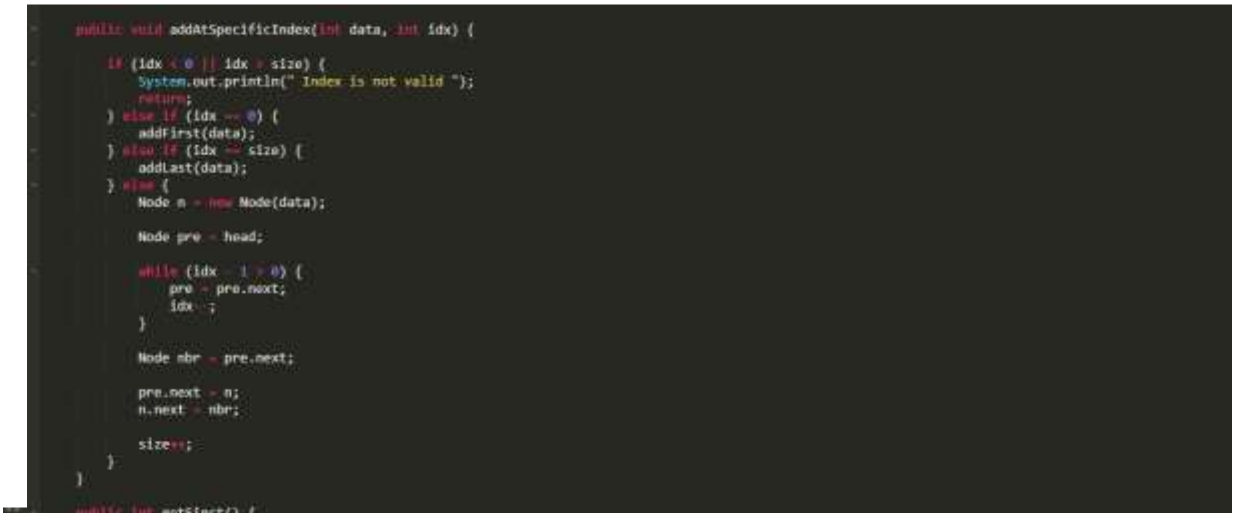



```
public void addFirst(int data){
    Node n = new Node(data); // we have create

    if(head == null){ // linked list is empty
        head = n;
        tail = n ;

    }else{
        n.next  = head ;
        head = n ;
    }

    size++;
```

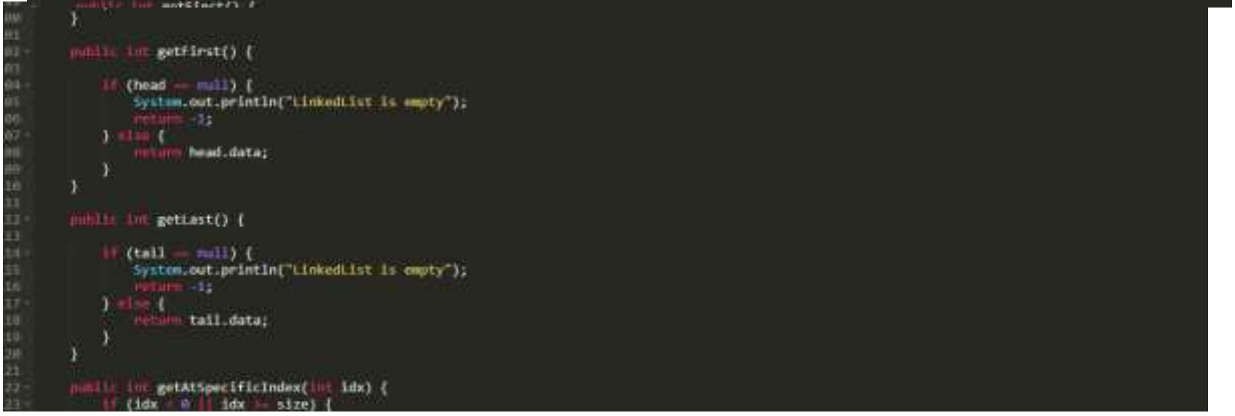

```
public void addLast(int data){
    Node n = new Node(data);

    if( head == null){
        head = n ;
        tail = n ;
    }else{

        tail.next = n ;
        tail = n ;
    }
    size++;
}
```



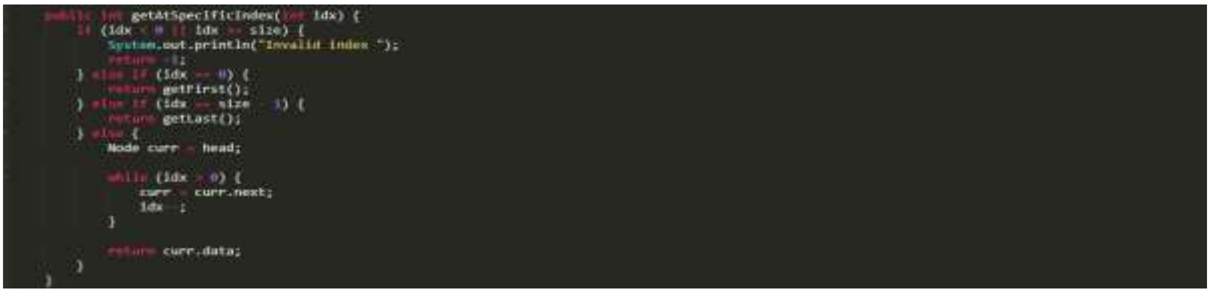
```
public class Lecture1 {
    public static void main(String[] args) {
        MylinkedList list = new MylinkedList();

        list.addFirst(10);
        list.addFirst(20);
        list.addLast(100);
        list.addLast(190);

        list.addAtSpecificIndex(110, 2);

        list.addAtSpecificIndex(99, 1);

        System.out.println(" before deleteion " + list);

        System.out.println(list.removeAtSpecificIndex(2));
        System.out.println(list.removeAtSpecificIndex(3));

        System.out.println("After deletion " + list);
    }
}

class Node {
    int data;
    Node next;
```



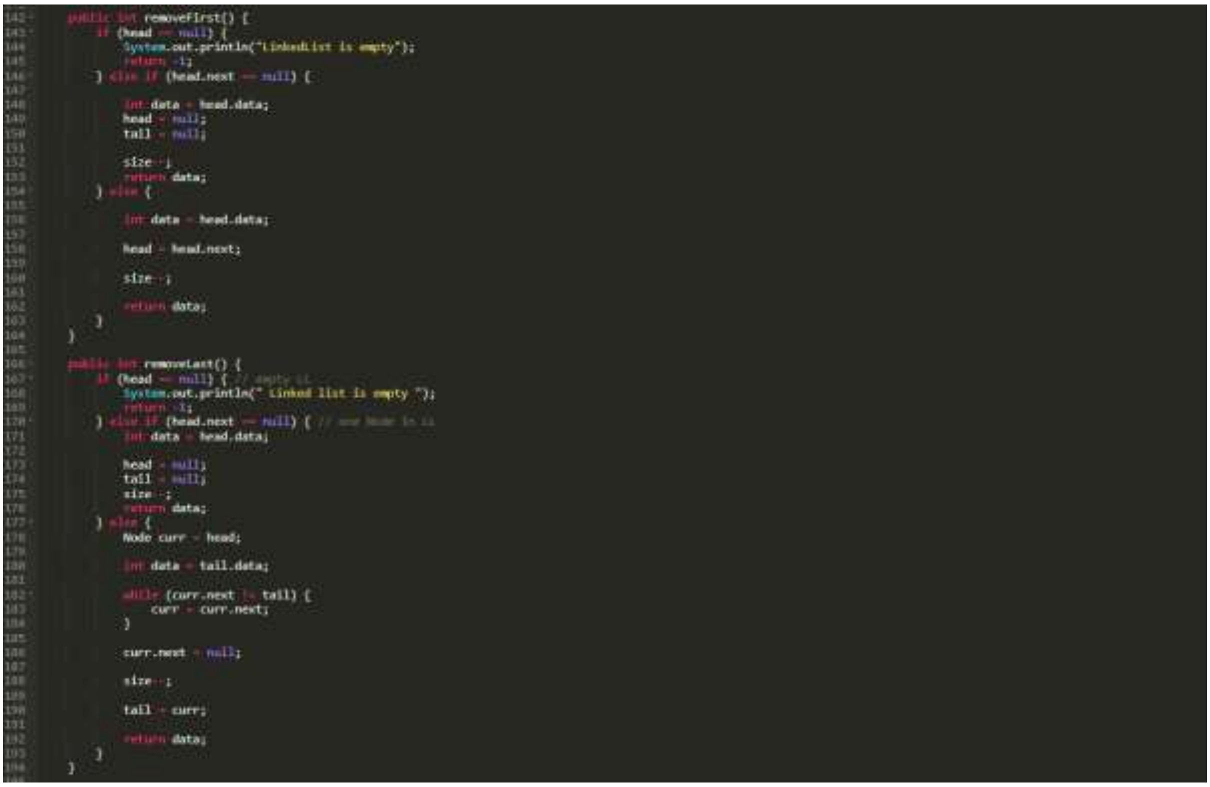
```
    public void addAtSpecificIndex(int data, int idx) {

        if (idx < 0 || idx > size) {
            System.out.println(" Index is not valid ");
            return;
        } else if (idx == 0) {
            addFirst(data);
        } else if (idx == size) {
            addLast(data);
        } else {
            Node n = new Node(data);

            Node pre = head;

            while (idx - 1 > 0) {
                pre = pre.next;
                idx--;
            }

            Node nbr = pre.next;

            pre.next = n;
            n.next = nbr;

            size++;
        }
    }
```



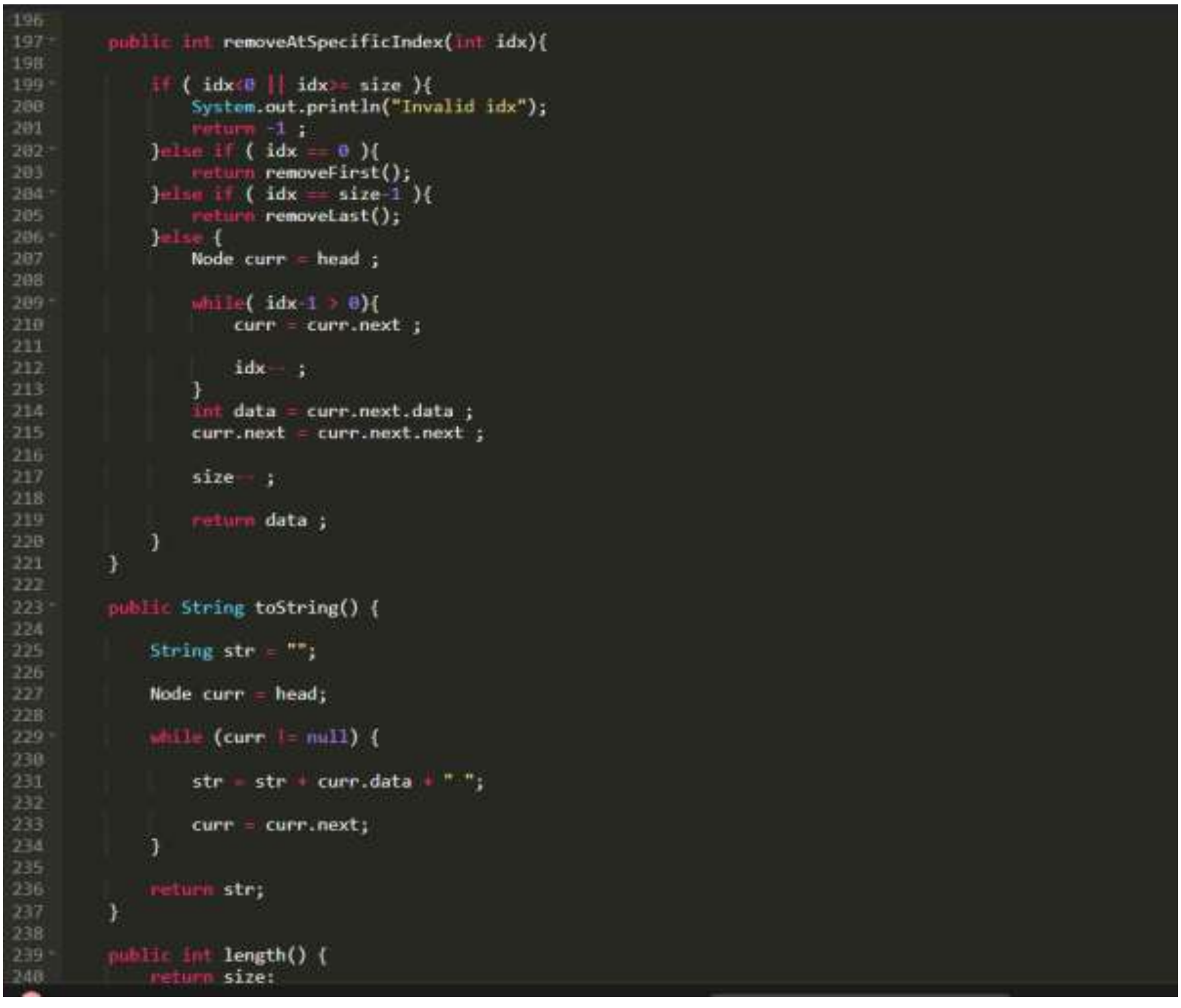
```
    }

    public int getFirst() {

        if (head == null) {
            System.out.println("LinkedList is empty");
            return -1;
        } else {
            return head.data;
        }
    }

    public int getLast() {

        if (tail == null) {
            System.out.println("LinkedList is empty");
            return -1;
        } else {
            return tail.data;
        }
    }

    public int getAtSpecificIndex(int idx) {
        if (idx < 0 || idx >= size) {
```



```
    public int getAtSpecificIndex(int idx) {
        if (idx < 0 || idx >= size) {
            System.out.println("Invalid index ");
            return -1;
        } else if (idx == 0) {
            return getFirst();
        } else if (idx == size - 1) {
            return getLast();
        } else {
            Node curr = head;

            while (idx > 0) {
                curr = curr.next;
                idx--;
            }

            return curr.data;
        }
    }
```

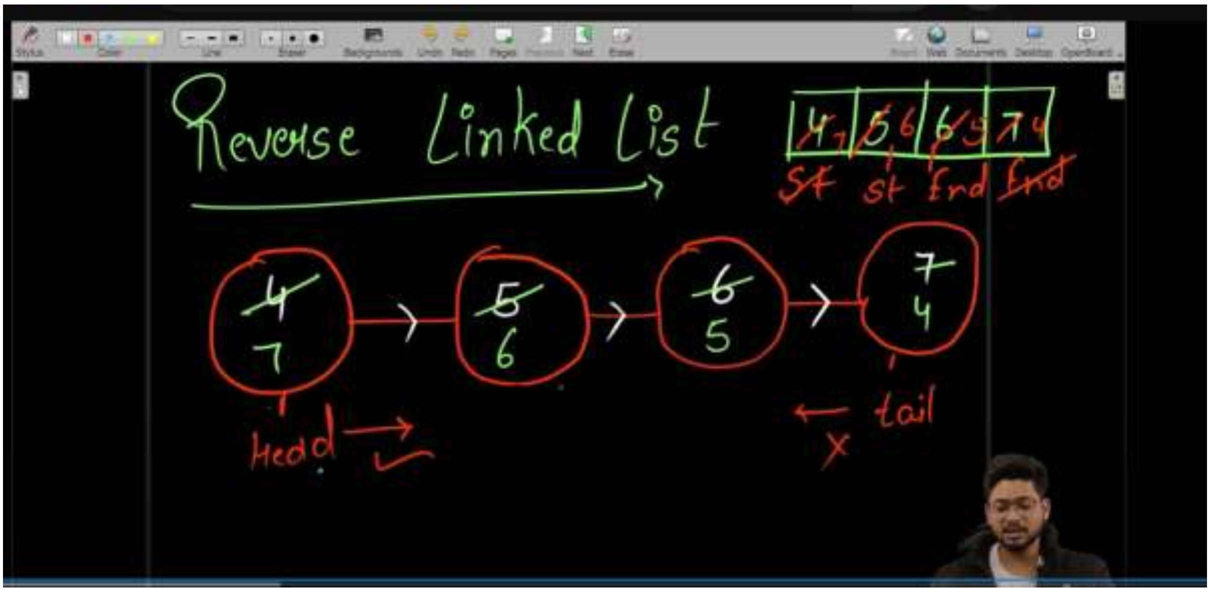


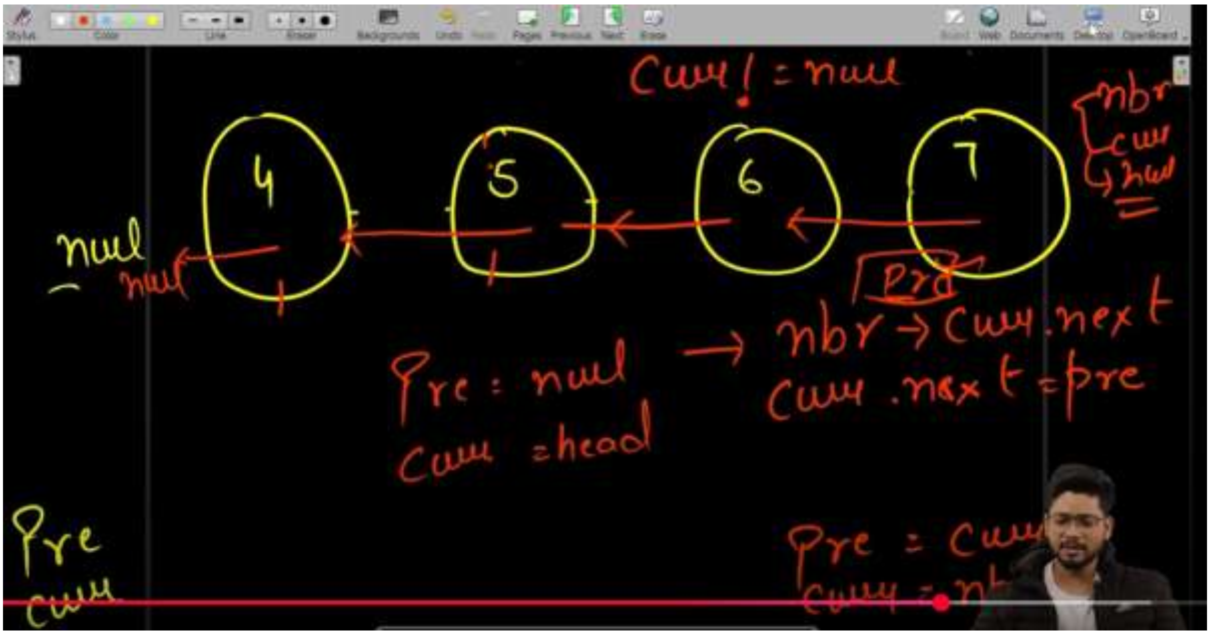
```java
    public int removeFirst() {
        if (head == null) {
            System.out.println("LinkedList is empty");
            return -1;
        } else if (head.next == null) {

            int data = head.data;
            head = null;
            tail = null;

            size--;
            return data;
        } else {

            int data = head.data;

            head = head.next;

            size--;

            return data;
        }
    }

    public int removeLast() {
        if (head == null) { // empty LL
            System.out.println(" Linked list is empty ");
            return -1;
        } else if (head.next == null) { // one Node in LL
            int data = head.data;

            head = null;
            tail = null;
            size--;
            return data;
        } else {
            Node curr = head;

            int data = tail.data;

            while (curr.next != tail) {
                curr = curr.next;
            }

            curr.next = null;

            size--;

            tail = curr;

            return data;
        }
    }
```

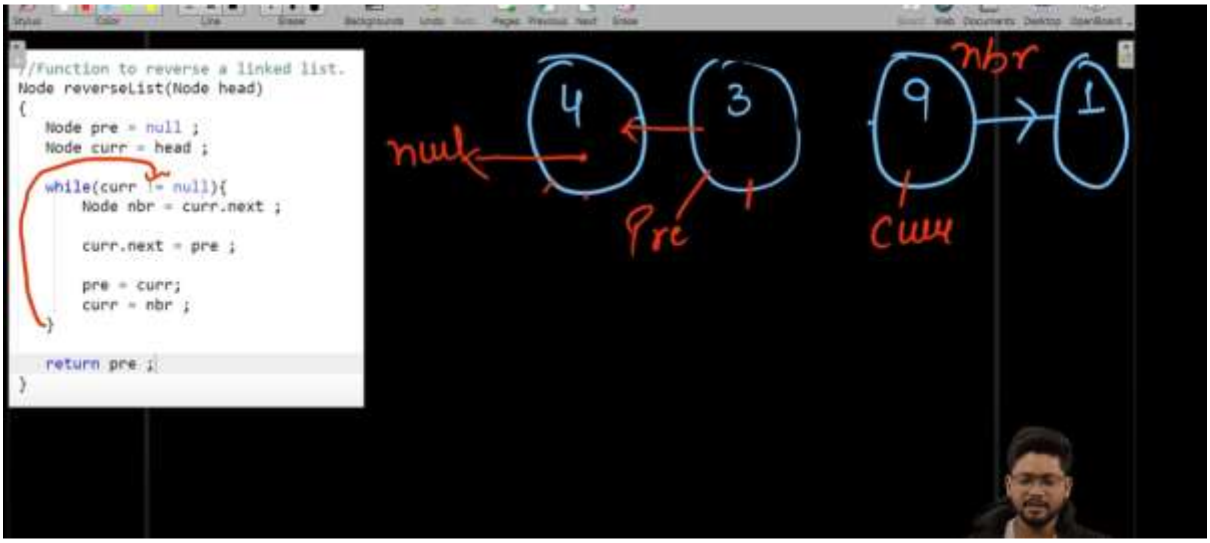


```java

    public int removeAtSpecificIndex(int idx){

        if ( idx<0 || idx>= size ){
            System.out.println("Invalid idx");
            return -1 ;
        }else if ( idx == 0 ){
            return removeFirst();
        }else if ( idx == size-1 ){
            return removeLast();
        }else {
            Node curr = head ;

            while( idx-1 > 0){
                curr = curr.next ;

                idx-- ;
            }
            int data = curr.next.data ;
            curr.next = curr.next.next ;

            size-- ;

            return data ;
        }
    }

    public String toString() {

        String str = "";

        Node curr = head;

        while (curr != null) {

            str = str + curr.data + " ";

            curr = curr.next;
        }

        return str;
    }

    public int length() {
        return size;
```

1. Reverse LinkedList

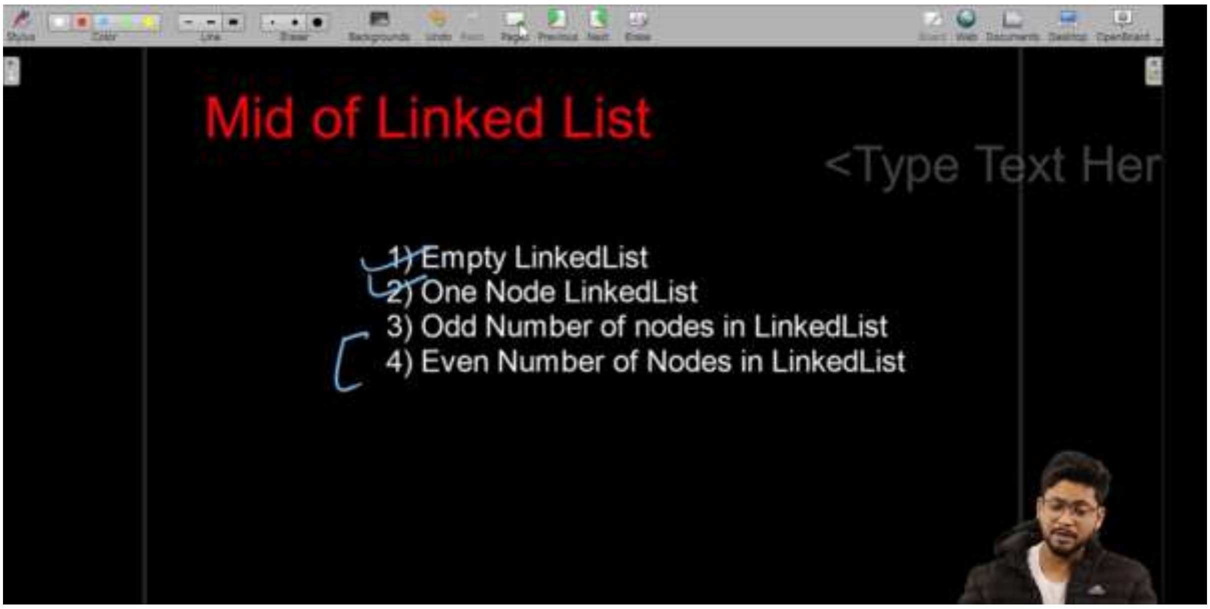






```
//Function to reverse a linked list.
Node reverseList(Node head)
{
    Node pre = null ;
    Node curr = head ;

    while(curr != null){
        Node nbr = curr.next ;

        curr.next = pre ;

        pre = curr;
        curr = nbr ;
    }

    return pre ;
}
```

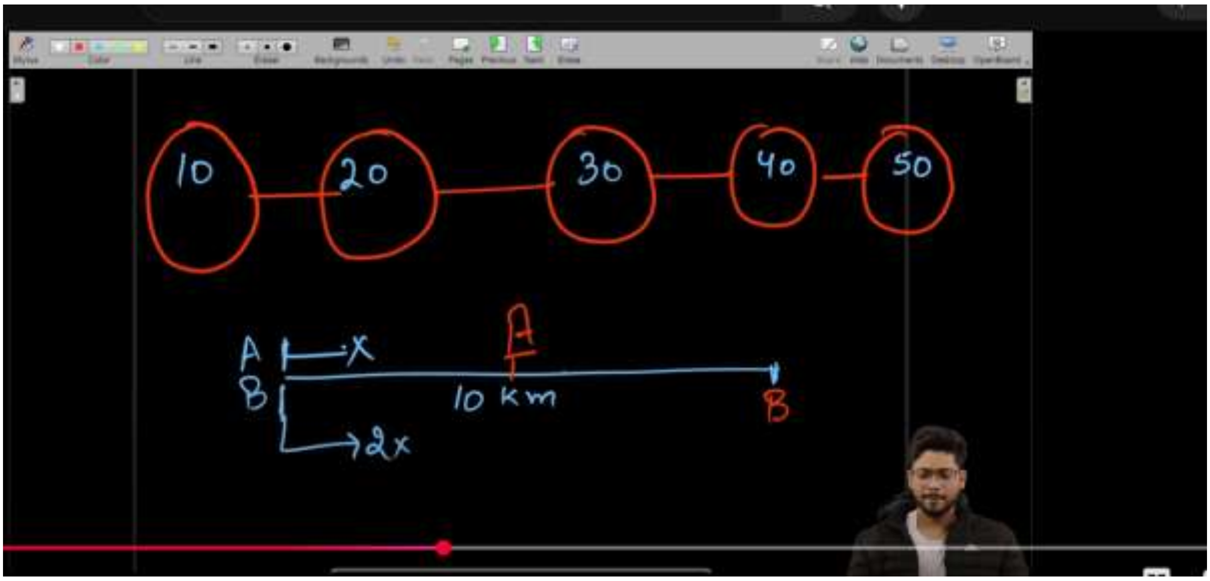
2.Middle Element in LinkedList

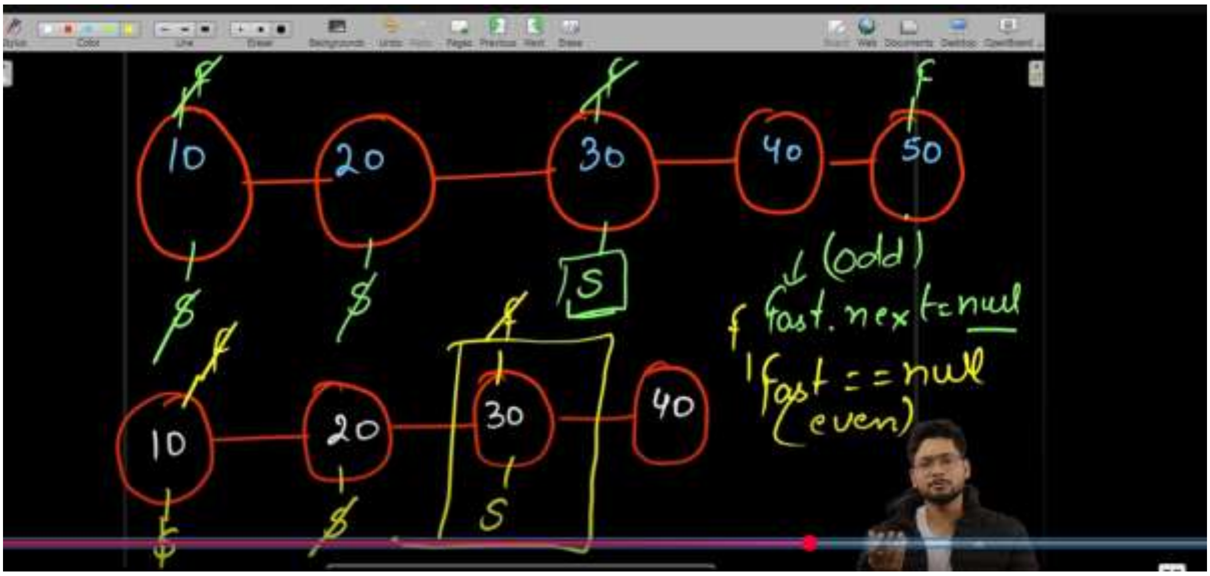


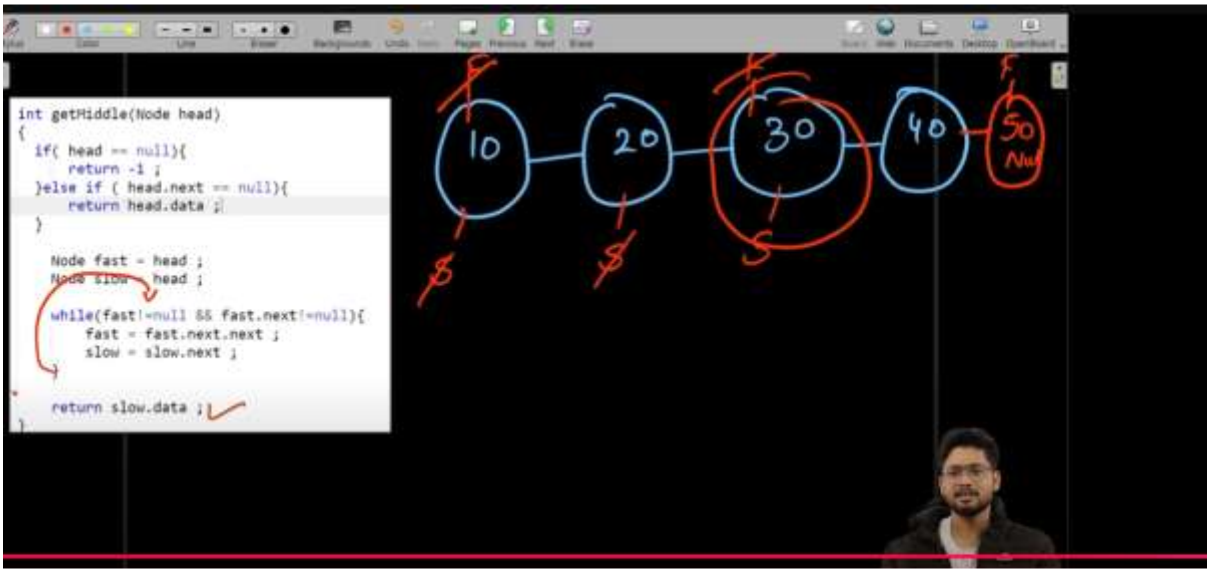
Mid of Linked List

<Type Text Her

1) Empty LinkedList
2) One Node LinkedList
3) Odd Number of nodes in LinkedList
4) Even Number of Nodes in LinkedList

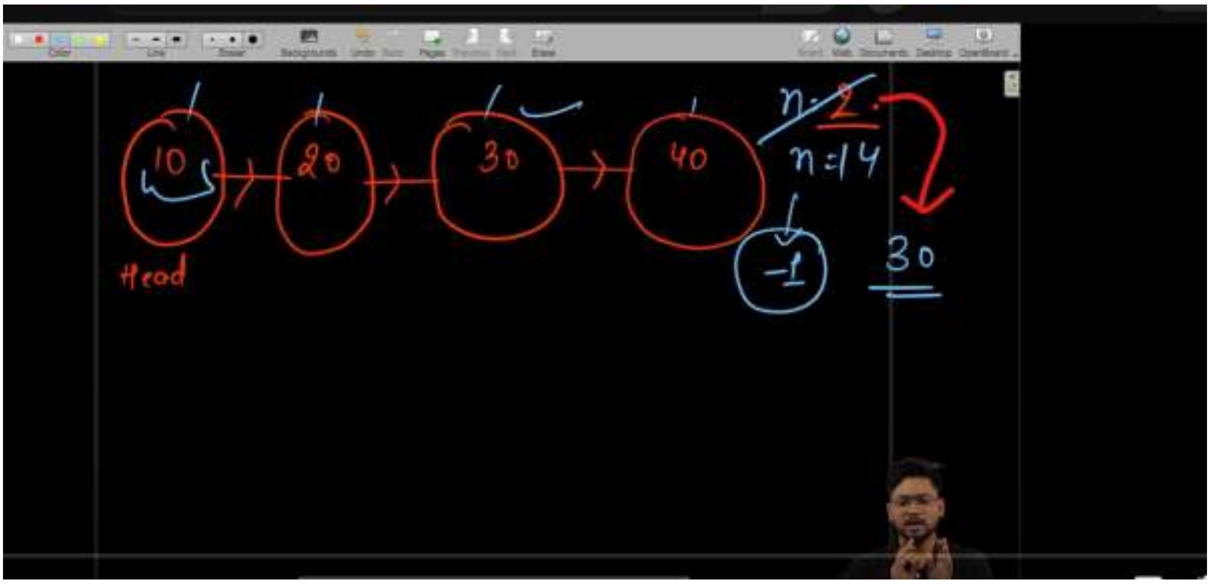


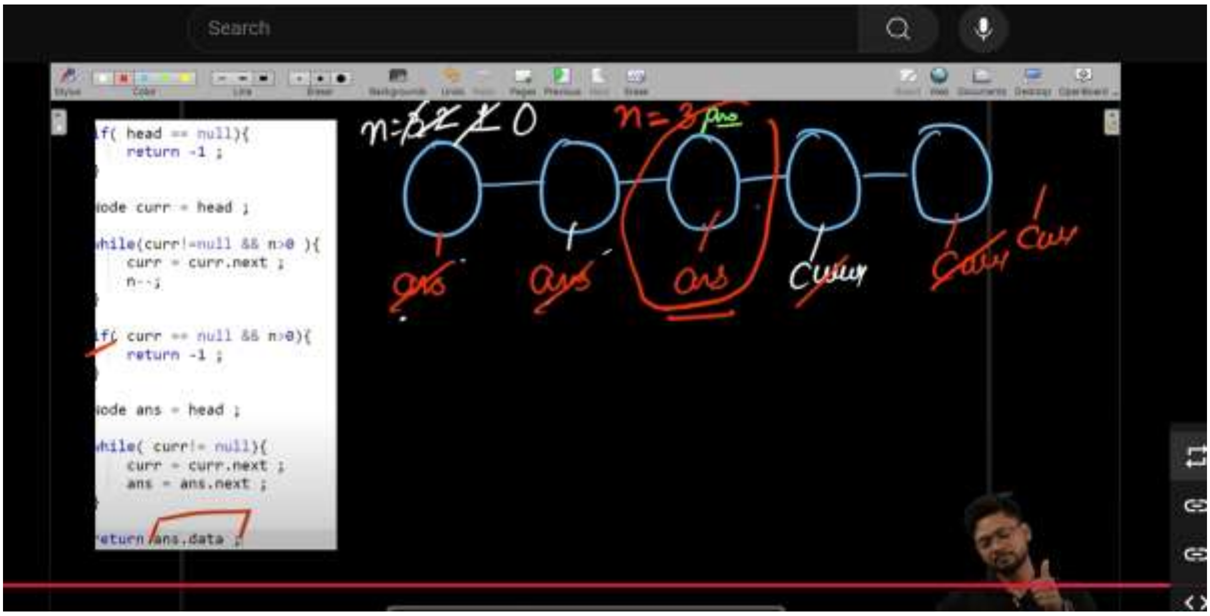




```
int getMiddle(Node head)
{
  if( head == null){
      return -1 ;
  }else if ( head.next == null){
      return head.data ;
  }

    Node fast = head ;
    Node slow = head ;

    while(fast!=null && fast.next!=null){
        fast = fast.next.next ;
        slow = slow.next ;
    }

    return slow.data ;
}
```

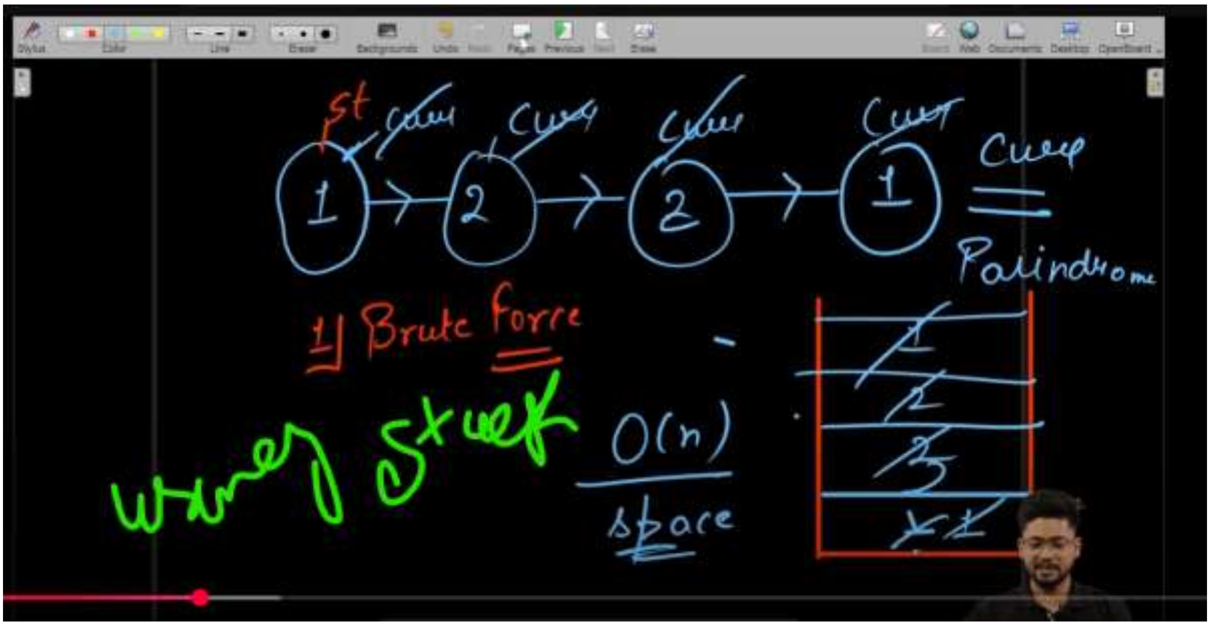
3.Nth Node from End of LinkedList

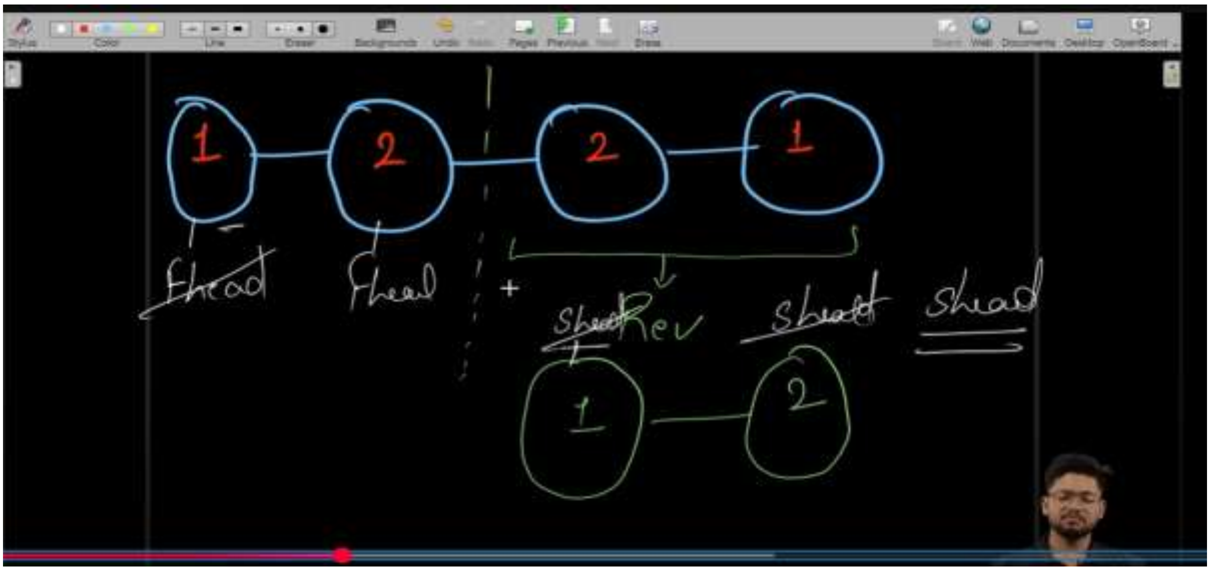


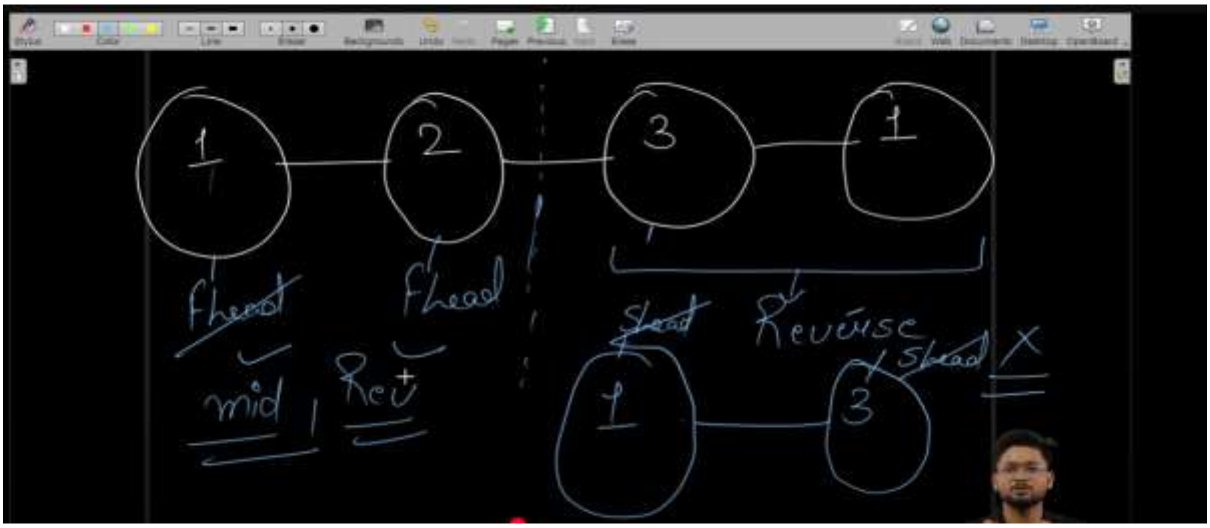


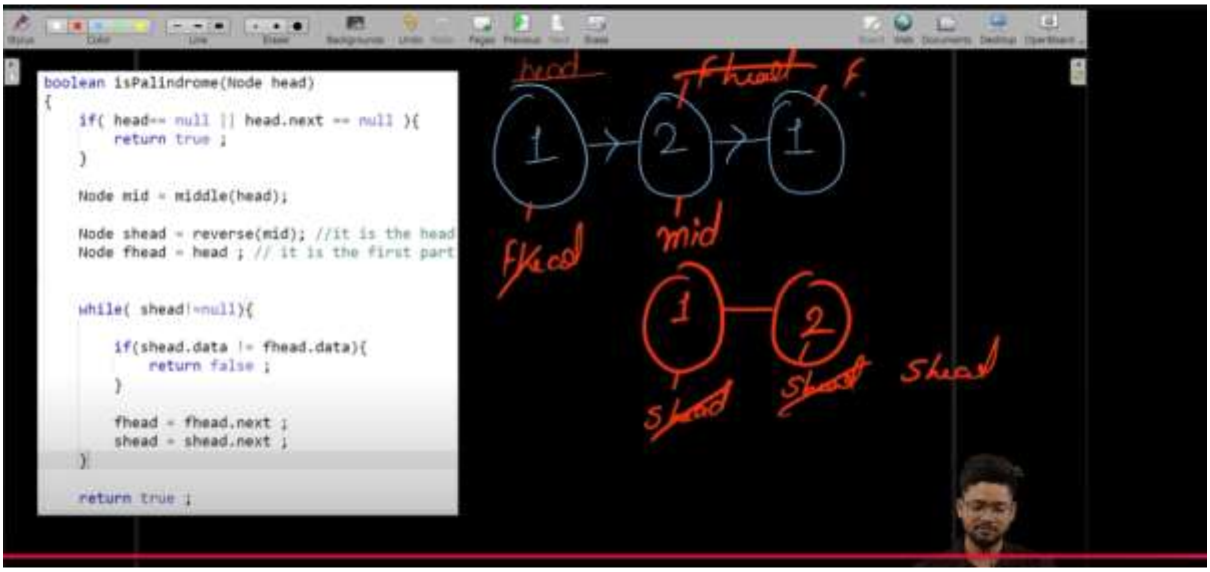
```
if( head == null){
    return -1 ;
}

Node curr = head ;

while(curr!=null && n>0 ){
    curr = curr.next ;
    n--;
}

if( curr == null && n>0){
    return -1 ;
}

Node ans = head ;

while( curr!= null){
    curr = curr.next ;
    ans = ans.next ;
}

return ans.data ;
```

4. Check if linked list is palindrome

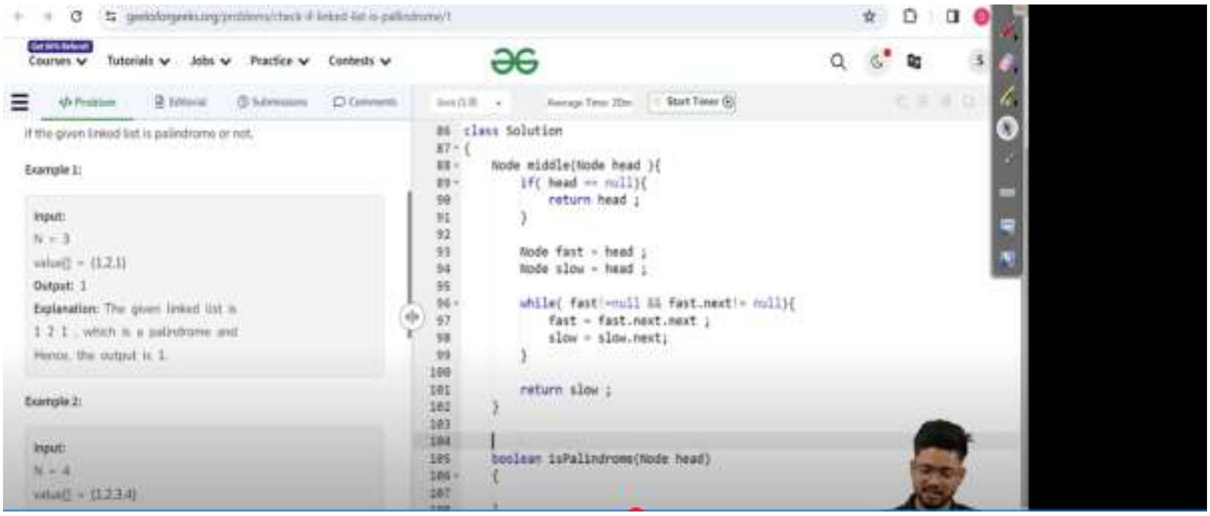








```
boolean isPalindrome(Node head)
{
    if( head== null || head.next == null ){
        return true ;
    }

    Node mid = middle(head);

    Node shead = reverse(mid); //it is the head
    Node fhead = head ; // it is the first part


    while( shead!=null){

        if(shead.data != fhead.data){
            return false ;
        }

        fhead = fhead.next ;
        shead = shead.next ;
    }

    return true ;
```



If the given linked list is palindrome or not.

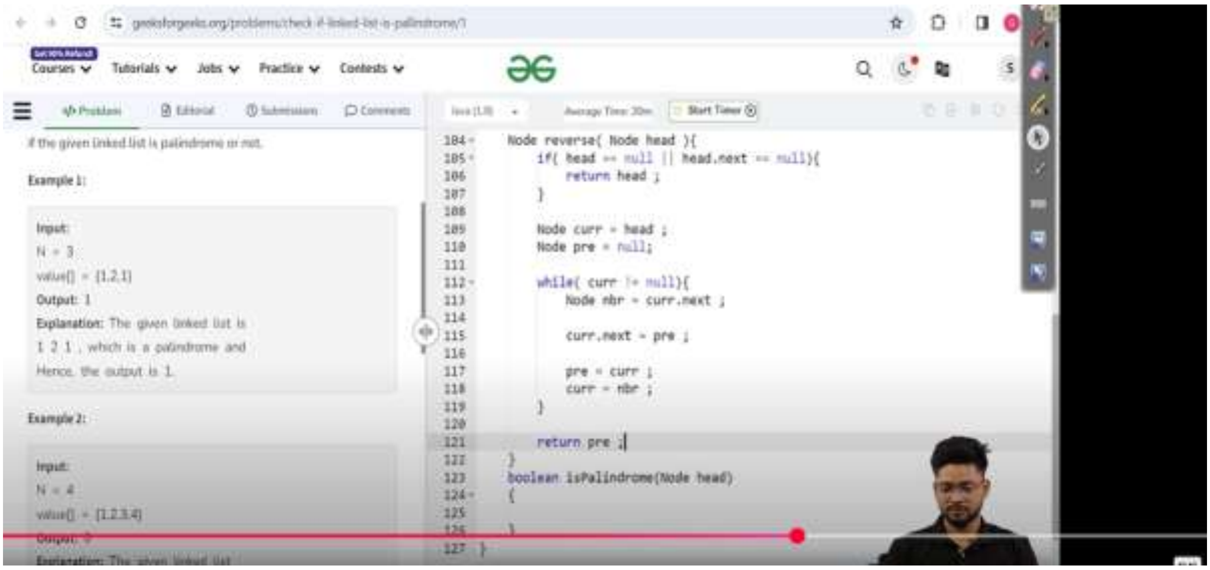
Example 1:

Input:
N = 3
value[] = {1,2,1}
Output: 1
Explanation: The given linked list is 1 2 1 , which is a palindrome and Hence, the output is 1.

Example 2:

Input:
N = 4
value[] = {1,2,3,4}

```
class Solution
{
    Node middle(Node head ){
        if( head == null){
            return head ;
        }

        Node fast = head ;
        Node slow = head ;

        while( fast!=null && fast.next!= null){
            fast = fast.next.next ;
            slow = slow.next;
        }

        return slow ;
    }


    boolean isPalindrome(Node head)
    {
```



If the given linked list is palindrome or not.

Example 1:

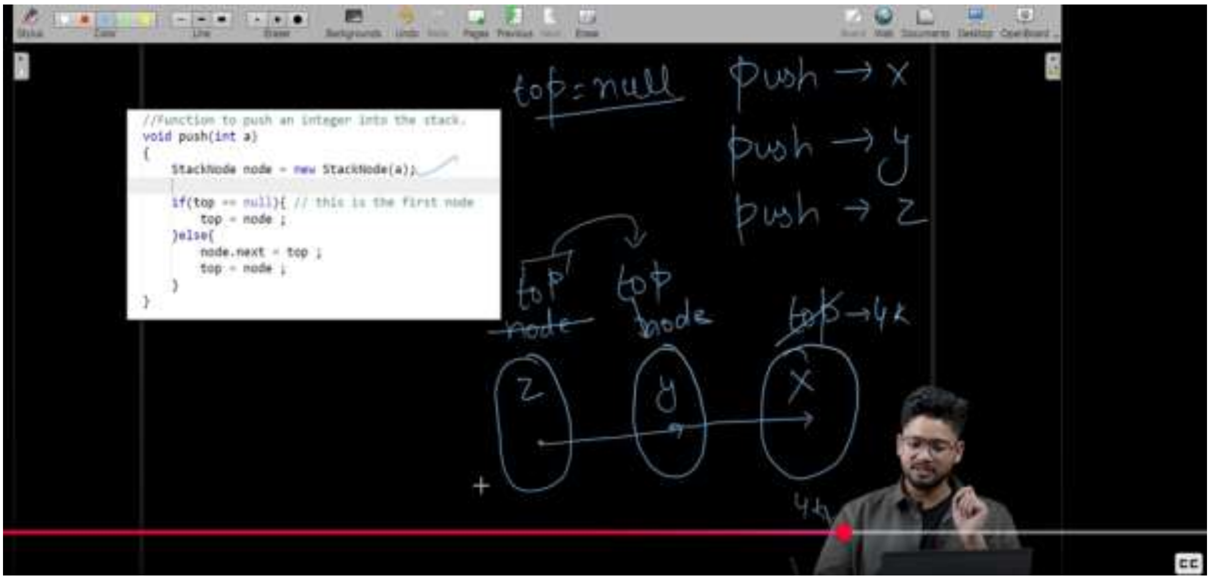
Input:
N = 3
value[] = {1,2,1}
Output: 1
Explanation: The given linked list is 1 2 1 , which is a palindrome and Hence, the output is 1.

Example 2:

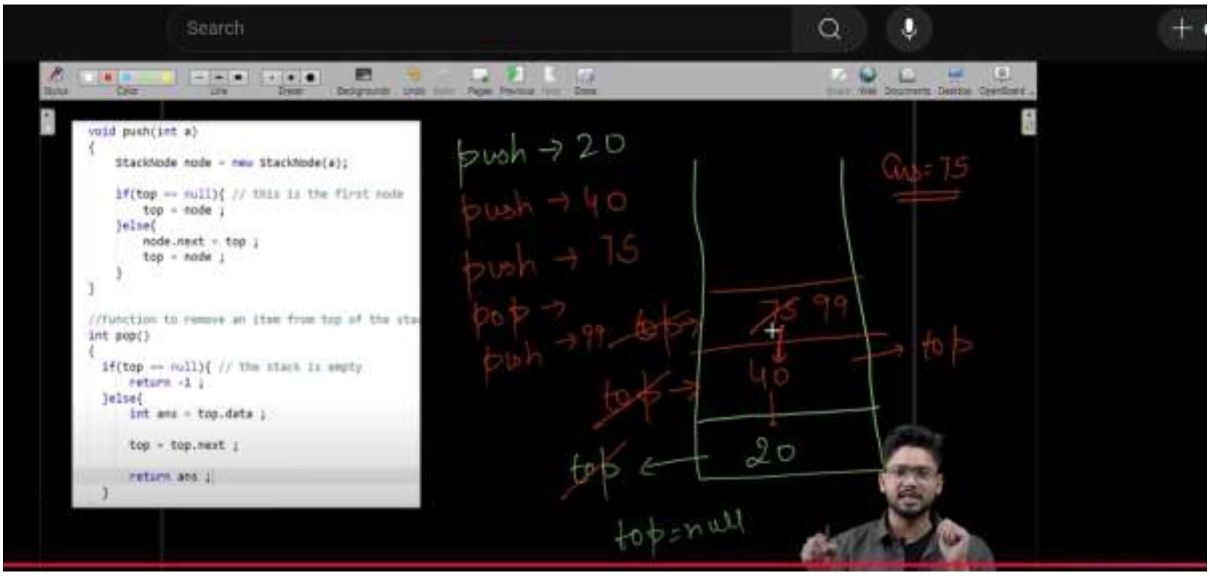
Input:
N = 4
value[] = {1,2,3,4}
Output: 0
Explanation: The given linked list

```
    Node reverse( Node head ){
        if( head == null || head.next == null){
            return head ;
        }

        Node curr = head ;
        Node pre = null;

        while( curr != null){
            Node nbr = curr.next ;

            curr.next = pre ;

            pre = curr ;
            curr = nbr ;
        }

        return pre ;
    }
    boolean isPalindrome(Node head)
    {

    }
}
```

5. Implimenet stack using Linked list

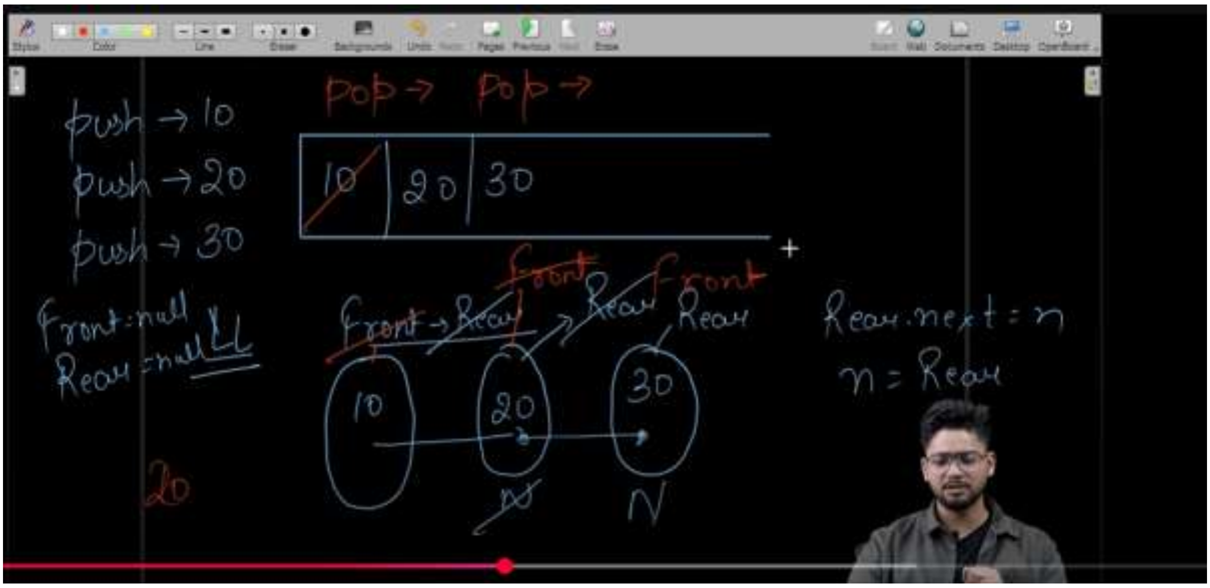


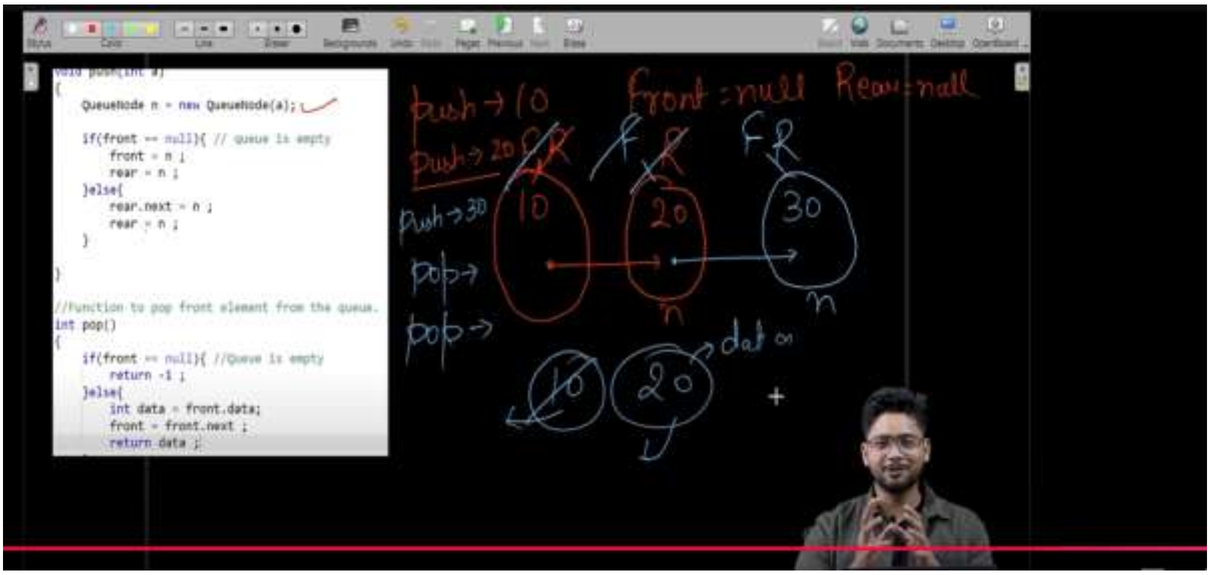
```
//Function to push an integer into the stack.
void push(int a)
{
    StackNode node = new StackNode(a);

    if(top == null){ // this is the first node
        top = node ;
    }else{
        node.next = top ;
        top = node ;
    }
}
```



```
void push(int a)
{
    StackNode node = new StackNode(a);

    if(top == null){ // this is the first node
        top = node ;
    }else{
        node.next = top ;
        top = node ;
    }
}

//Function to remove an item from top of the sta
int pop()
{
  if(top == null){ // the stack is empty
      return -1 ;
  }else{
      int ans = top.data ;

      top = top.next ;

      return ans ;
  }
```

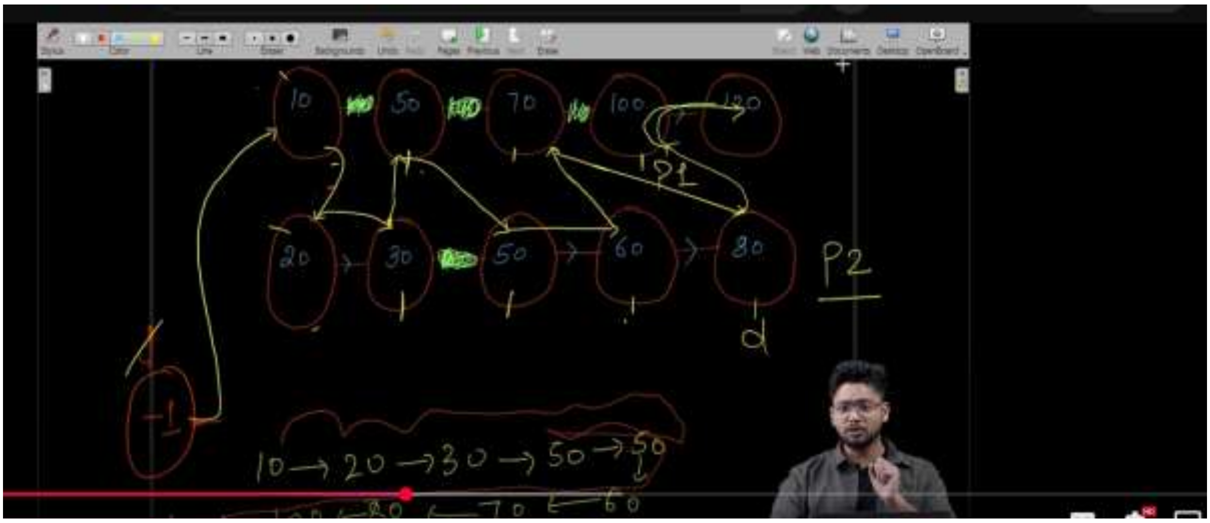
6.Implimenet Queue using LinkedList

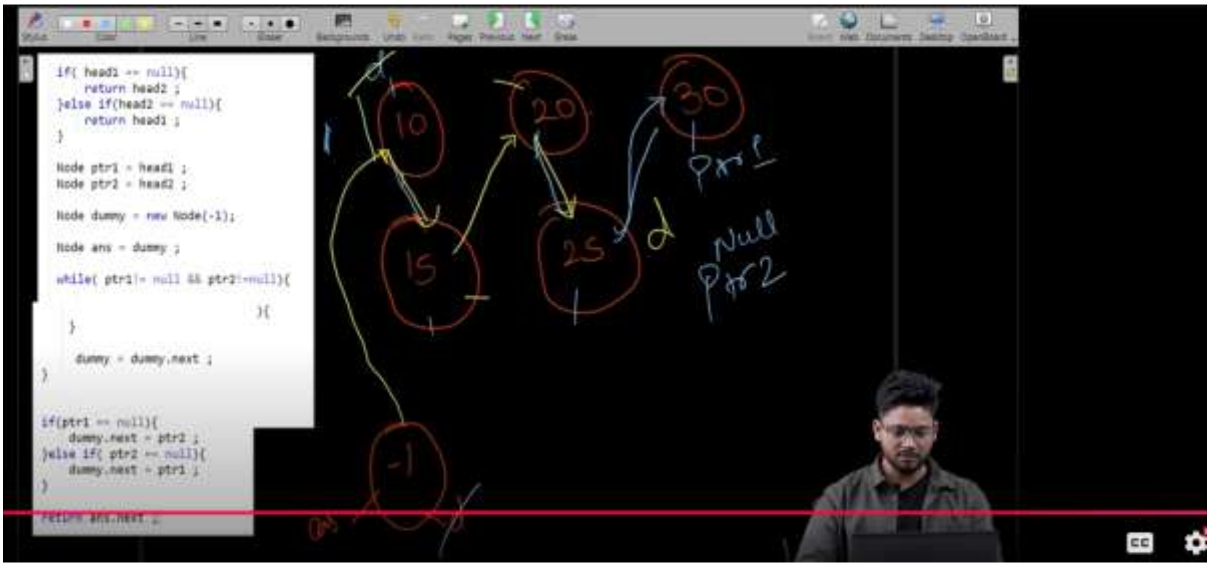




```
void push(int a)
{
    QueueNode n = new QueueNode(a);

    if(front == null){ // queue is empty
        front = n ;
        rear = n ;
    }else{
        rear.next = n ;
        rear = n ;
    }

}

//Function to pop front element from the queue.
int pop()
{
    if(front == null){ //Queue is empty
        return -1 ;
    }else{
        int data = front.data;
        front = front.next ;
        return data ;
```

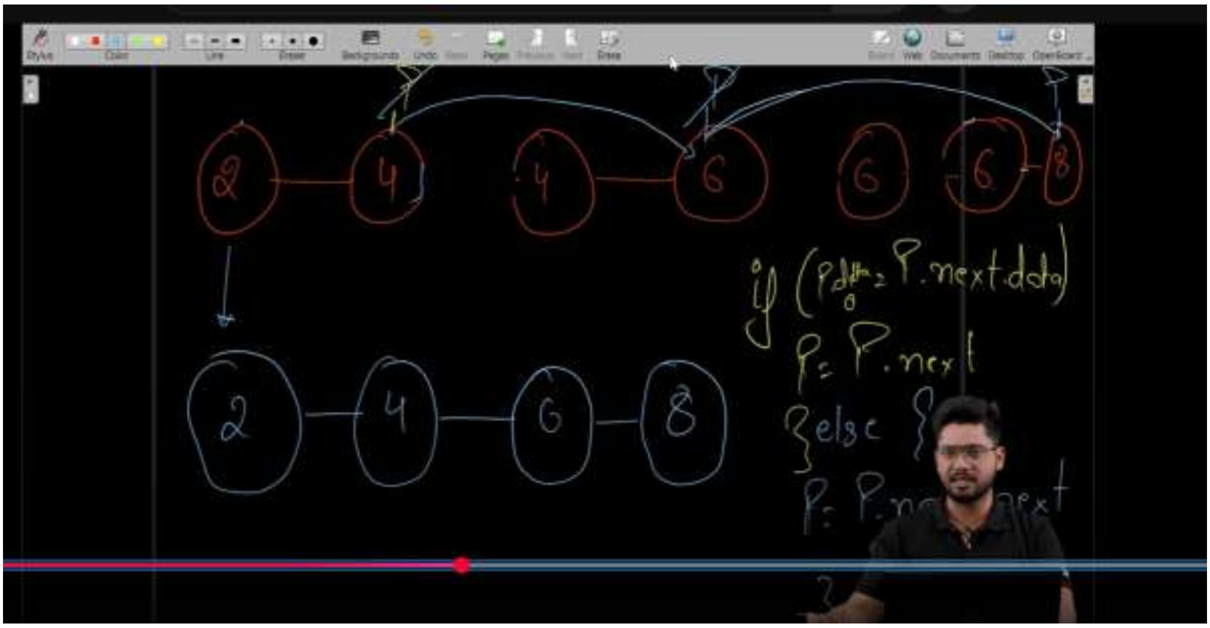
7.merge Two Sorted LinkedList

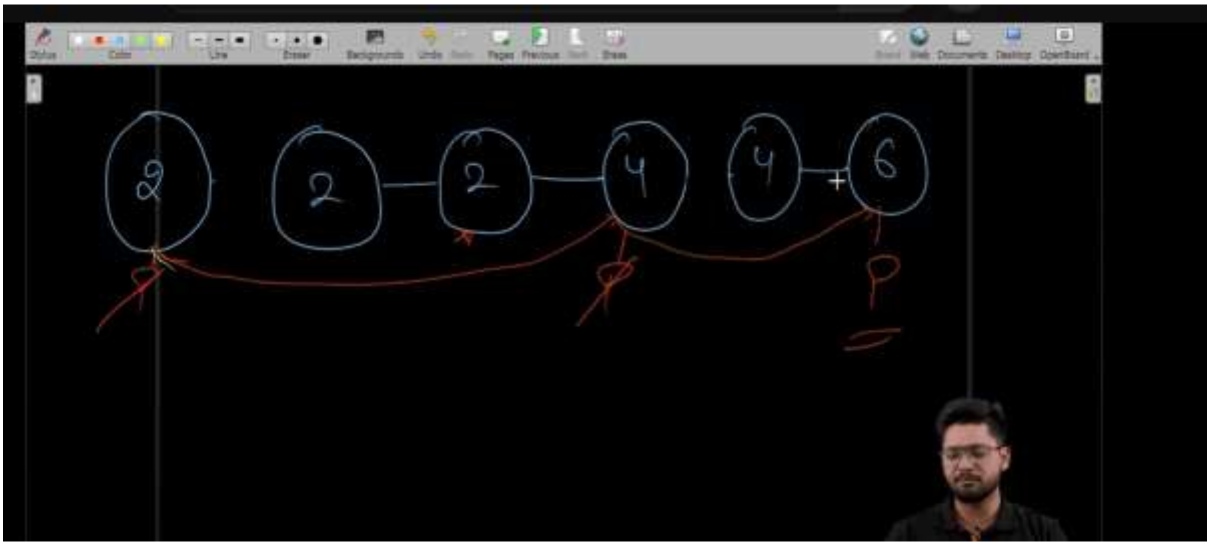


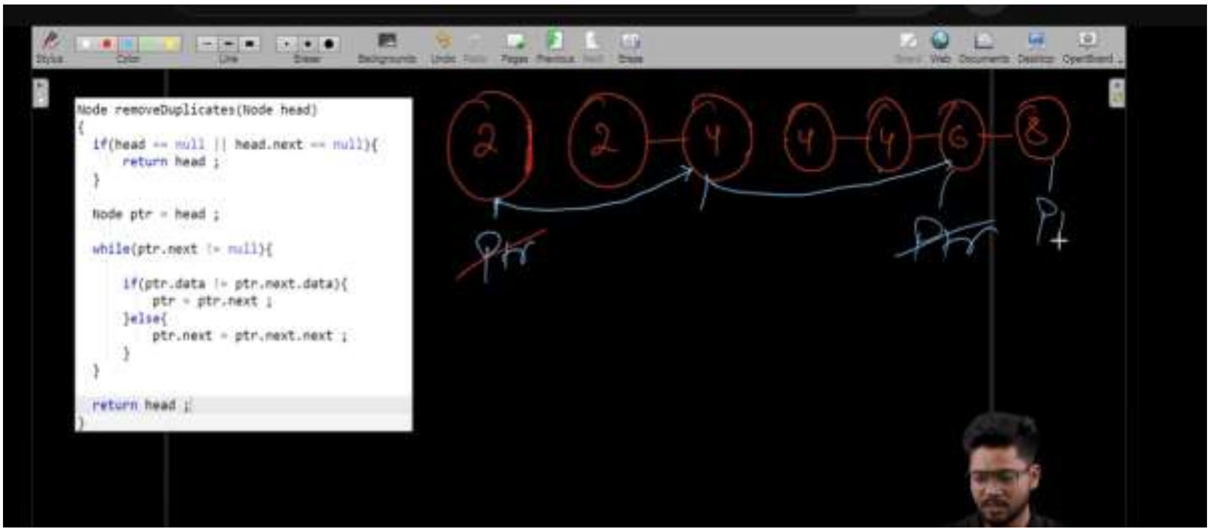


```
if( head1 == null){
    return head2 ;
}else if(head2 == null){
    return head1 ;
}

Node ptr1 = head1 ;
Node ptr2 = head2 ;

Node dummy = new Node(-1);

Node ans = dummy ;

while( ptr1!= null && ptr2!=null){

                                    ){
  }

   dummy = dummy.next ;
}

if(ptr1 == null){
    dummy.next = ptr2 ;
}else if( ptr2 == null){
    dummy.next = ptr1 ;
}

return ans.next ;
```

8.Remove duplicate element from sorted linked list

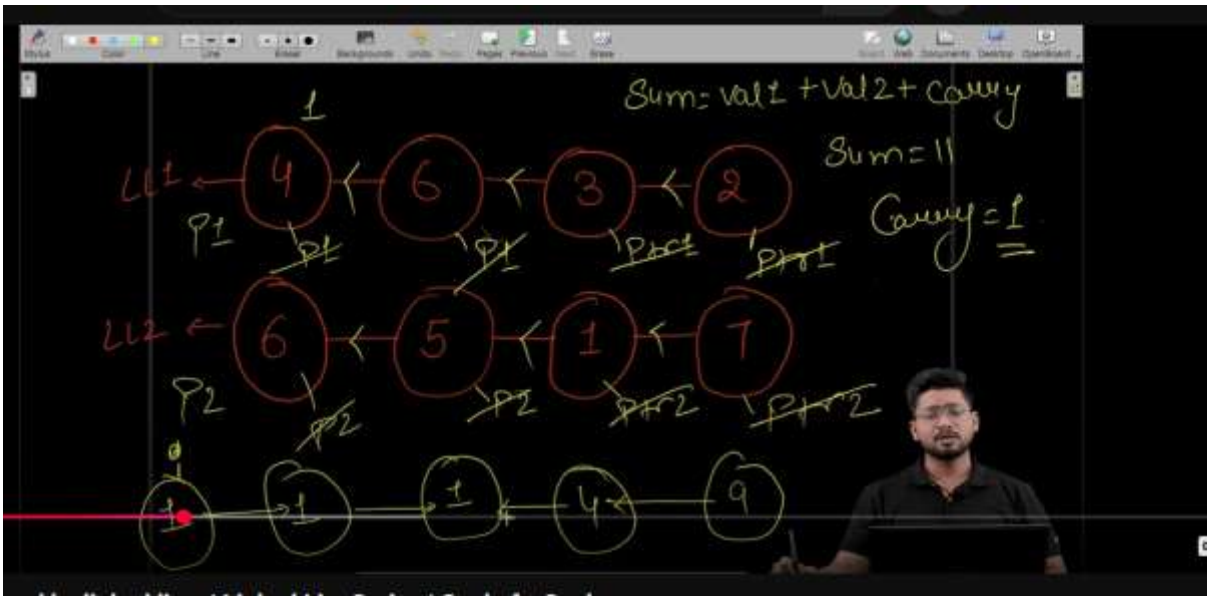


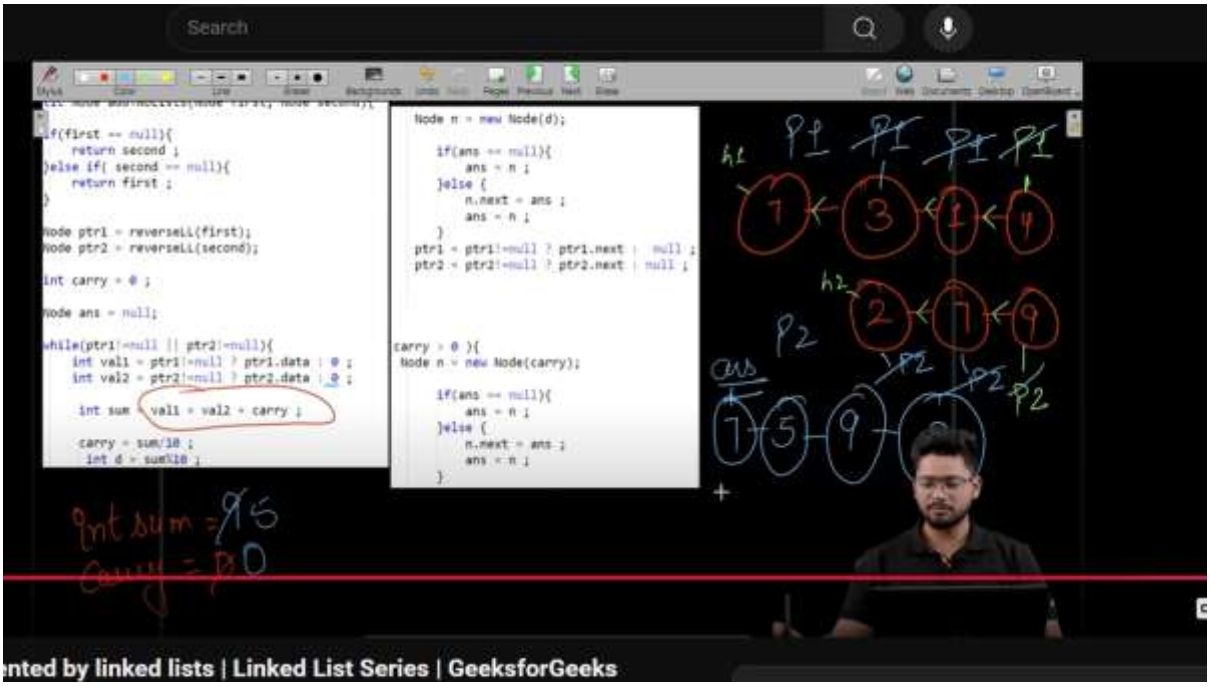




```
Node removeDuplicates(Node head)
{
  if(head == null || head.next == null){
      return head ;
  }

  Node ptr = head ;

  while(ptr.next != null){

      if(ptr.data != ptr.next.data){
          ptr = ptr.next ;
      }else{
          ptr.next = ptr.next.next ;
      }
  }

  return head ;
}
```

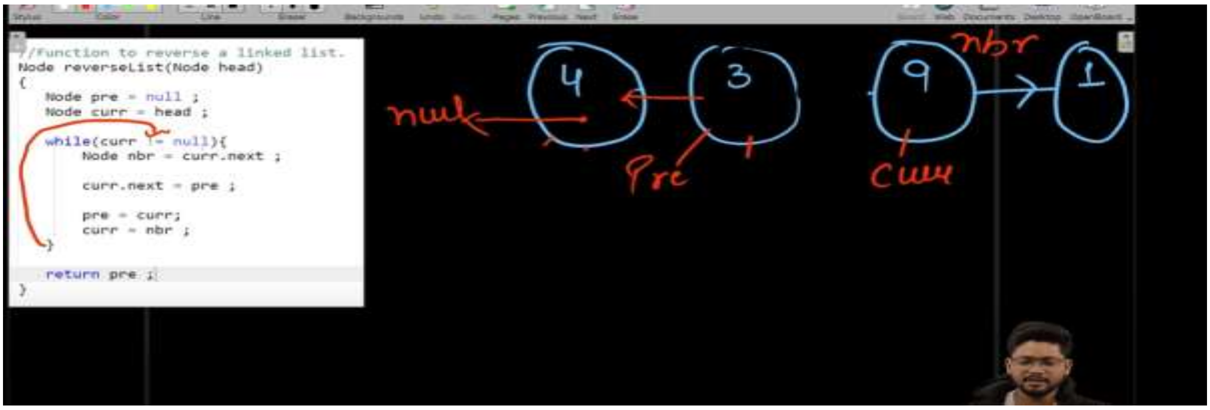
9.Add Two Number represented by linked list



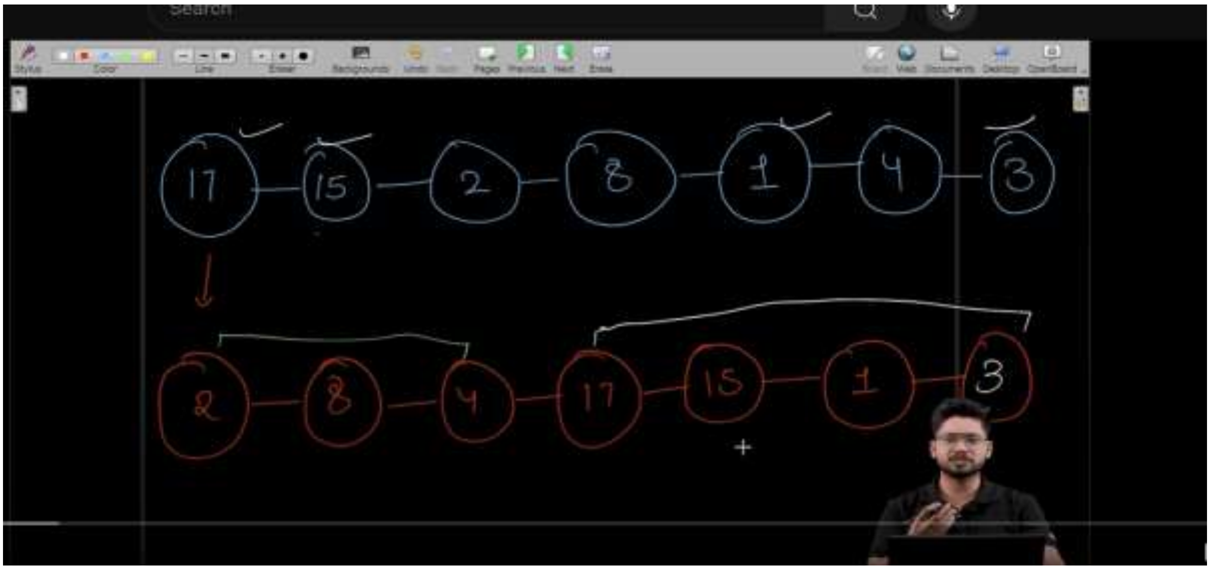

```
if(first == null){
    return second ;
}else if( second == null){
    return first ;
}

Node ptr1 = reverseLL(first);
Node ptr2 = reverseLL(second);

int carry = 0 ;

Node ans = null;

while(ptr1!=null || ptr2!=null){
    int val1 = ptr1!=null ? ptr1.data : 0 ;
    int val2 = ptr2!=null ? ptr2.data : 0 ;

     int sum = val1 + val2 + carry ;

     carry = sum/10 ;
      int d = sum%10 ;
```

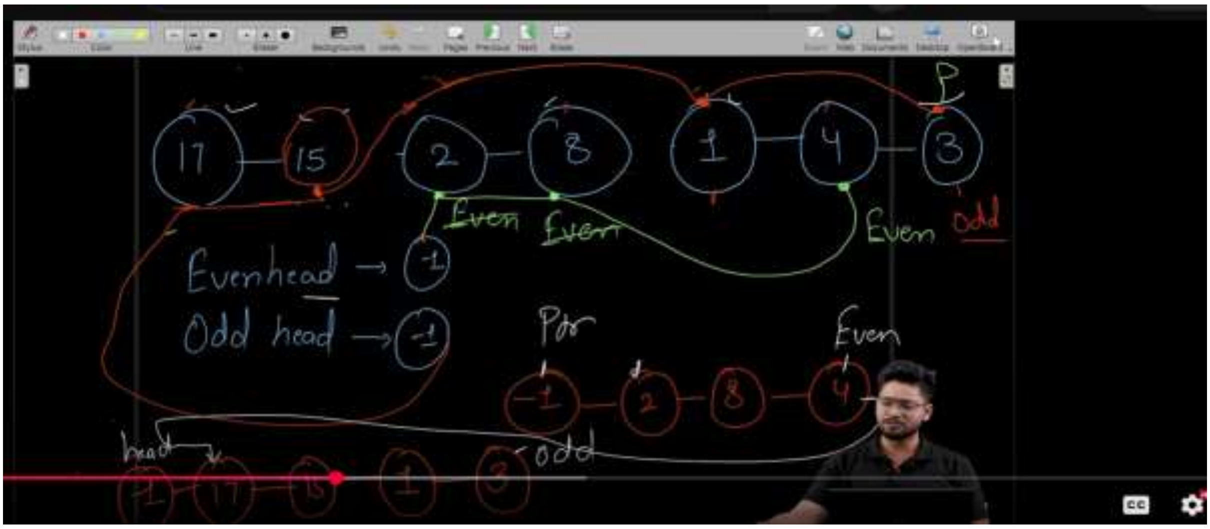
```
   Node n = new Node(d);

      if(ans == null){
          ans = n ;
      }else {
          n.next = ans ;
          ans = n ;
      }
   ptr1 = ptr1!=null ? ptr1.next :  null ;
   ptr2 = ptr2!=null ? ptr2.next : null ;



carry > 0 ){
 Node n = new Node(carry);

      if(ans == null){
          ans = n ;
      }else {
          n.next = ans ;
          ans = n ;
      }
```

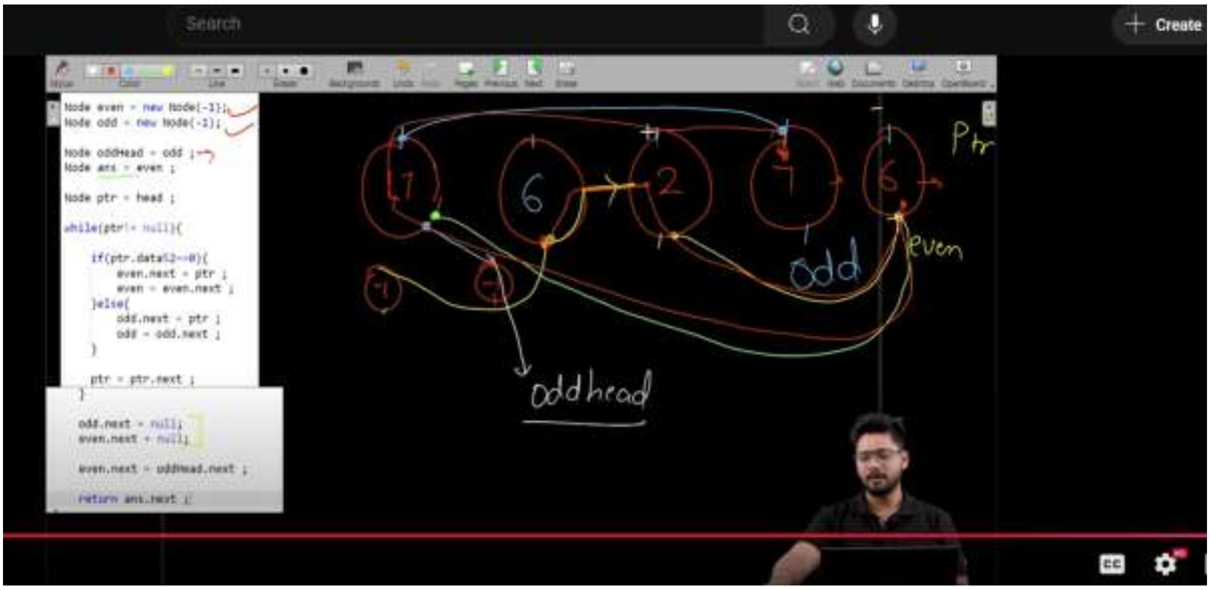
nted by linked lists | Linked List Series | GeeksforGeeks


```
//Function to reverse a linked list.
Node reverseList(Node head)
{
    Node pre = null ;
    Node curr = head ;

    while(curr != null){
        Node nbr = curr.next ;

        curr.next = pre ;

        pre = curr;
        curr = nbr ;
    }

    return pre ;
}
```

10.Segregate even ans odd in linked list

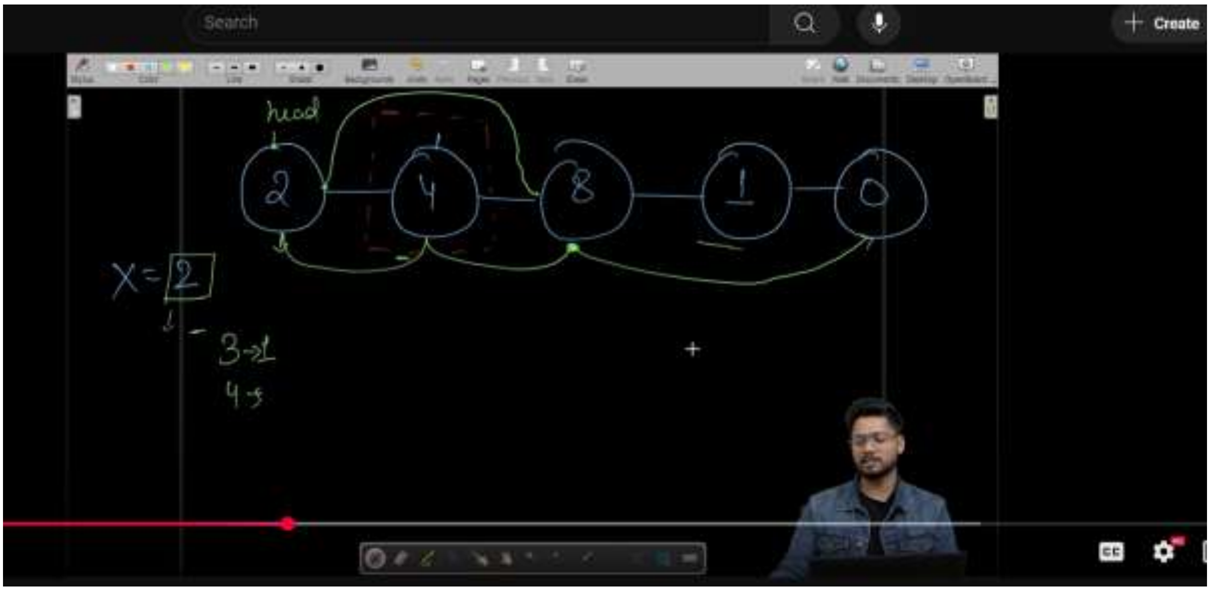


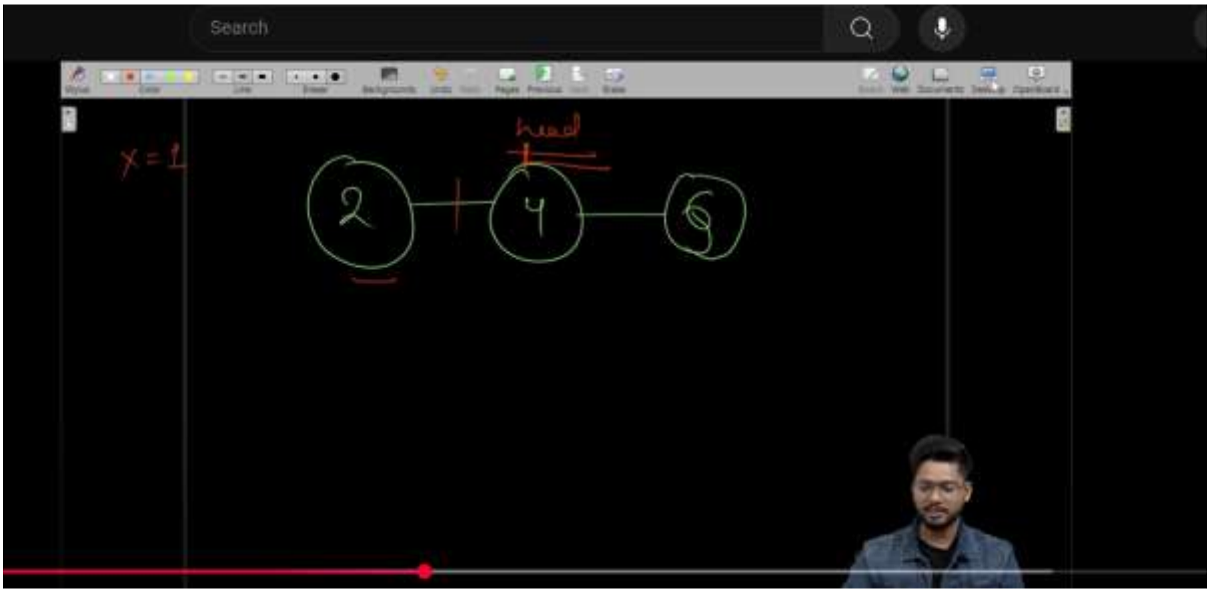


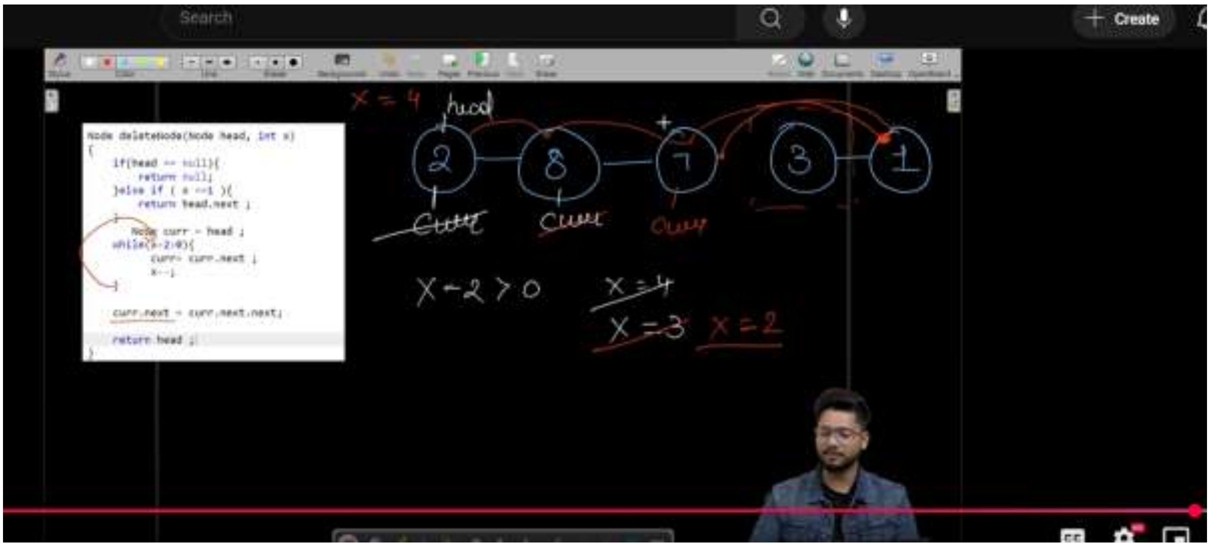


```
Node even = new Node(-1);
Node odd = new Node(-1);

Node oddHead = odd ;
Node ans = even ;

Node ptr = head ;

while(ptr!= null){

    if(ptr.data%2==0){
        even.next = ptr ;
        even = even.next ;
    }else{
        odd.next = ptr ;
        odd = odd.next ;
    }

    ptr = ptr.next ;
  }

  odd.next = null;
  even.next = null;

  even.next = oddHead.next ;

  return ans.next ;
```

11.Delete a node in single linked list

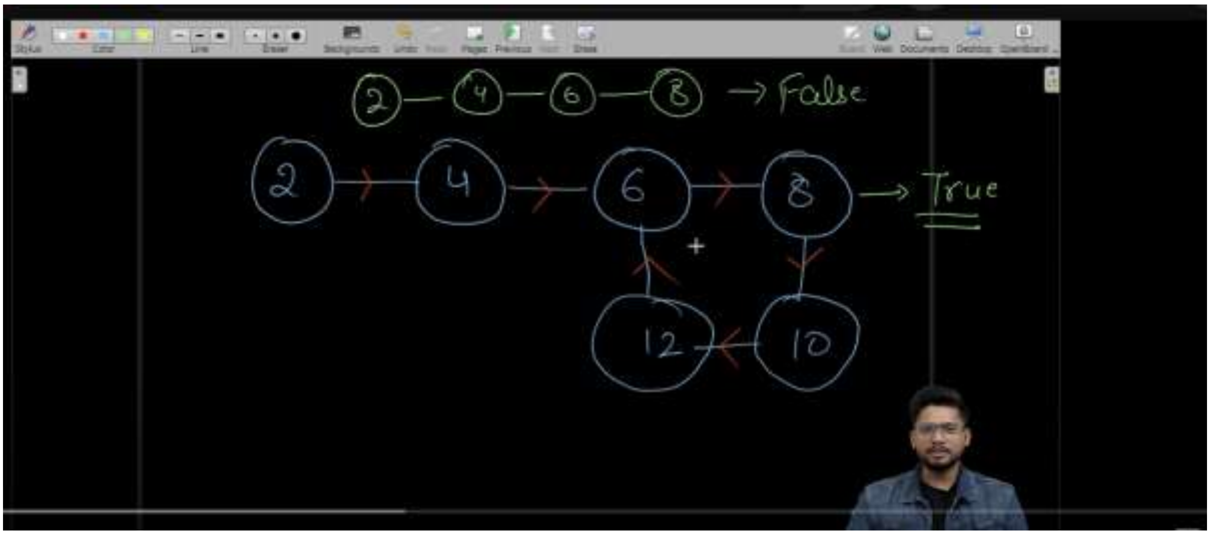


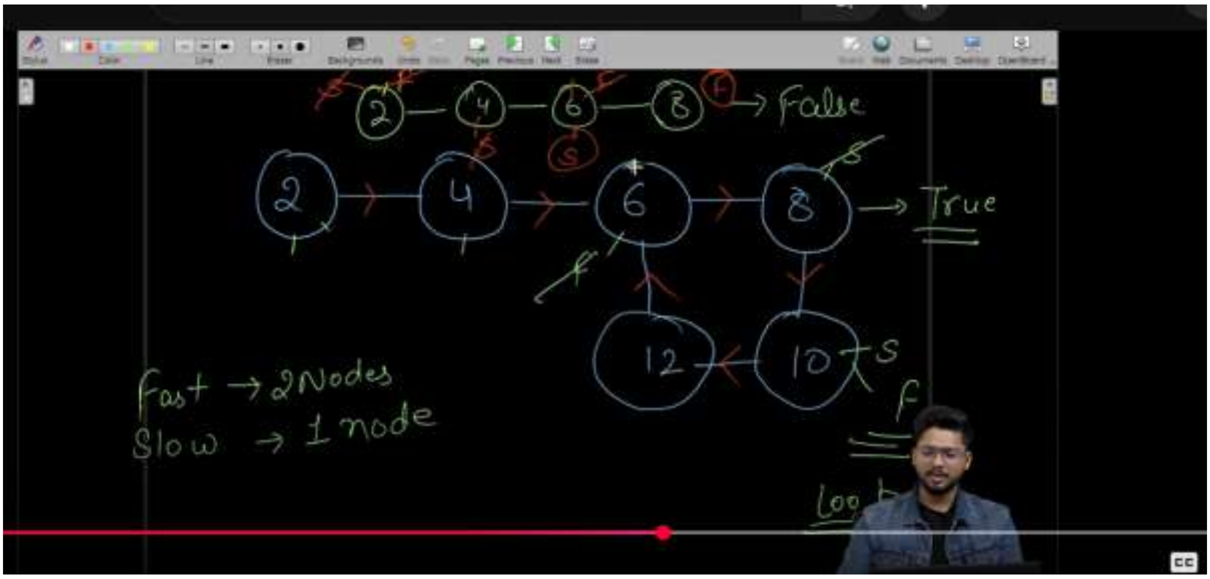


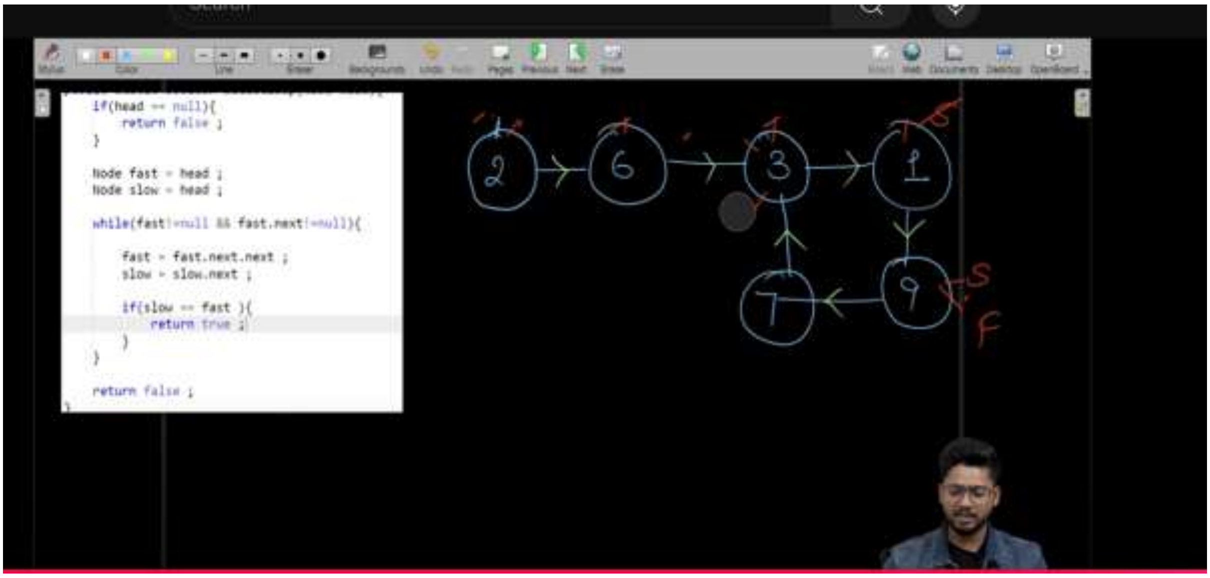


```
Node deleteNode(Node head, int x)
{
    if(head == null){
        return null;
    }else if ( x ==1 ){
        return head.next ;
    }
        Node curr = head ;
    while(x-2>0){
        curr= curr.next ;
        x--;
    }

    curr.next = curr.next.next;

    return head ;
}
```

12. Detect loop in linked list

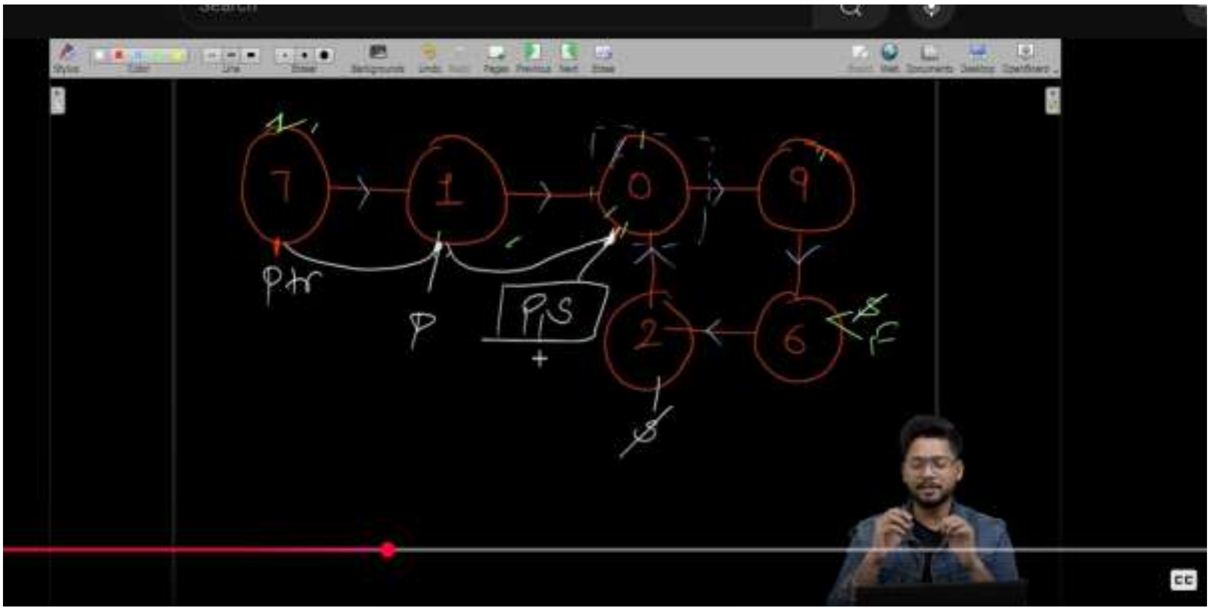


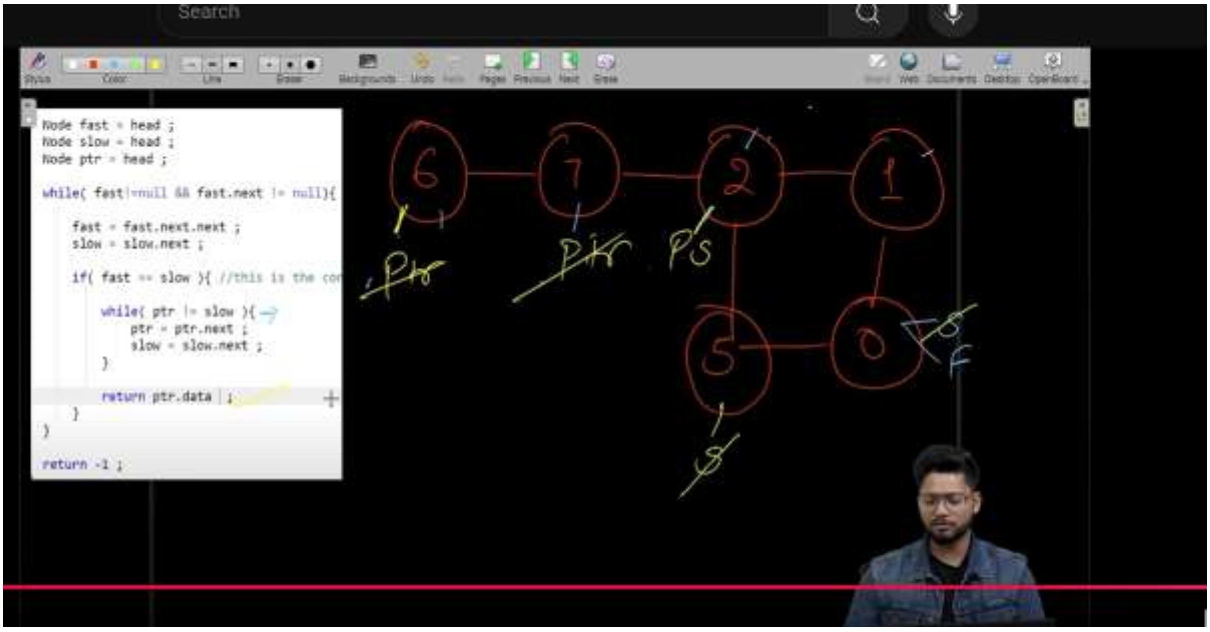




```
if(head == null){
    return false ;
}

Node fast = head ;
Node slow = head ;

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

    fast = fast.next.next ;
    slow = slow.next ;

    if(slow == fast ){
        return true ;
    }
}

return false ;
}
```

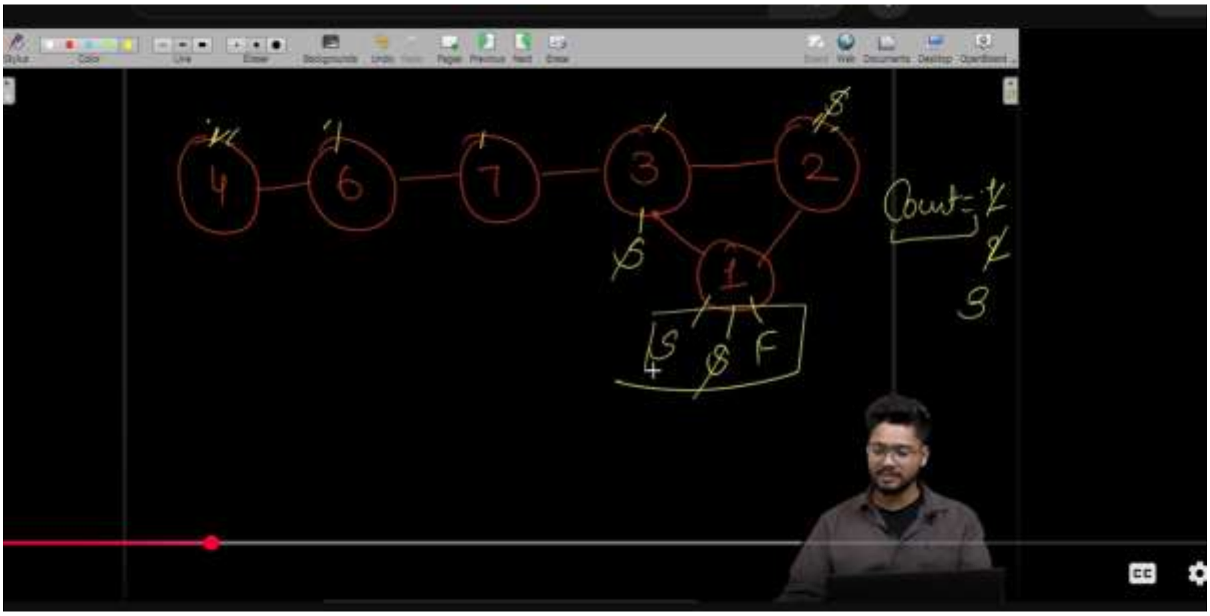
13.find the first node of loop in linked list

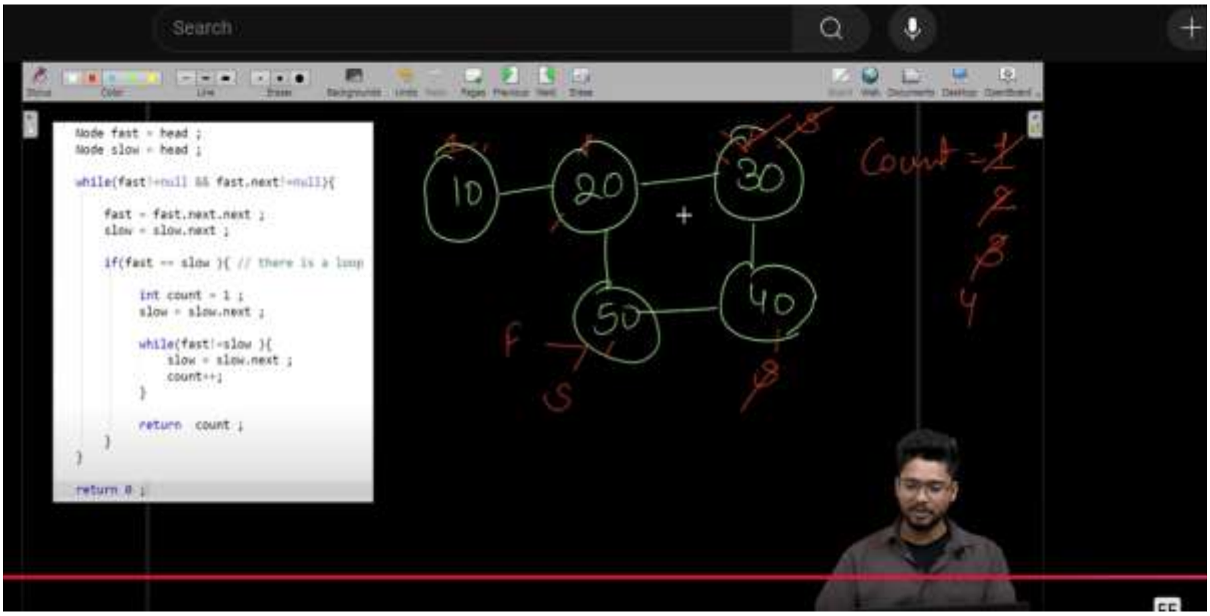




```
Node fast = head ;
Node slow = head ;
Node ptr = head ;

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

    fast = fast.next.next ;
    slow = slow.next ;

    if( fast == slow ){ //this is the co

        while( ptr != slow ){
            ptr = ptr.next ;
            slow = slow.next ;
        }

        return ptr.data ;
    }
}

return -1 ;
```

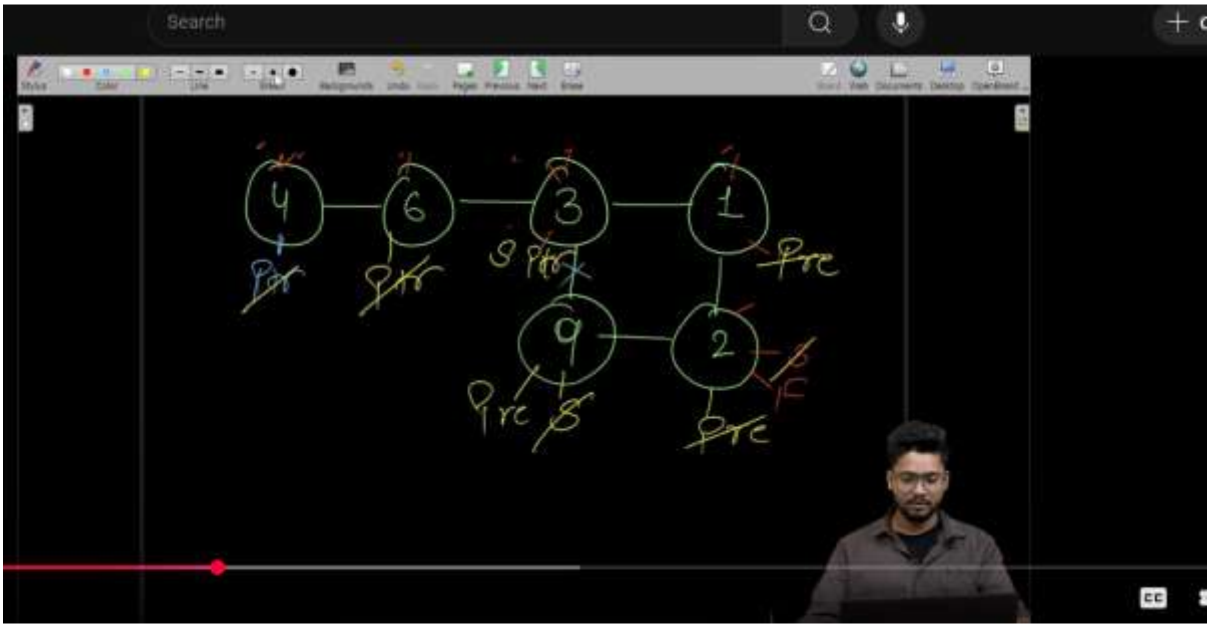
14 .Find the length of the loop in linked list

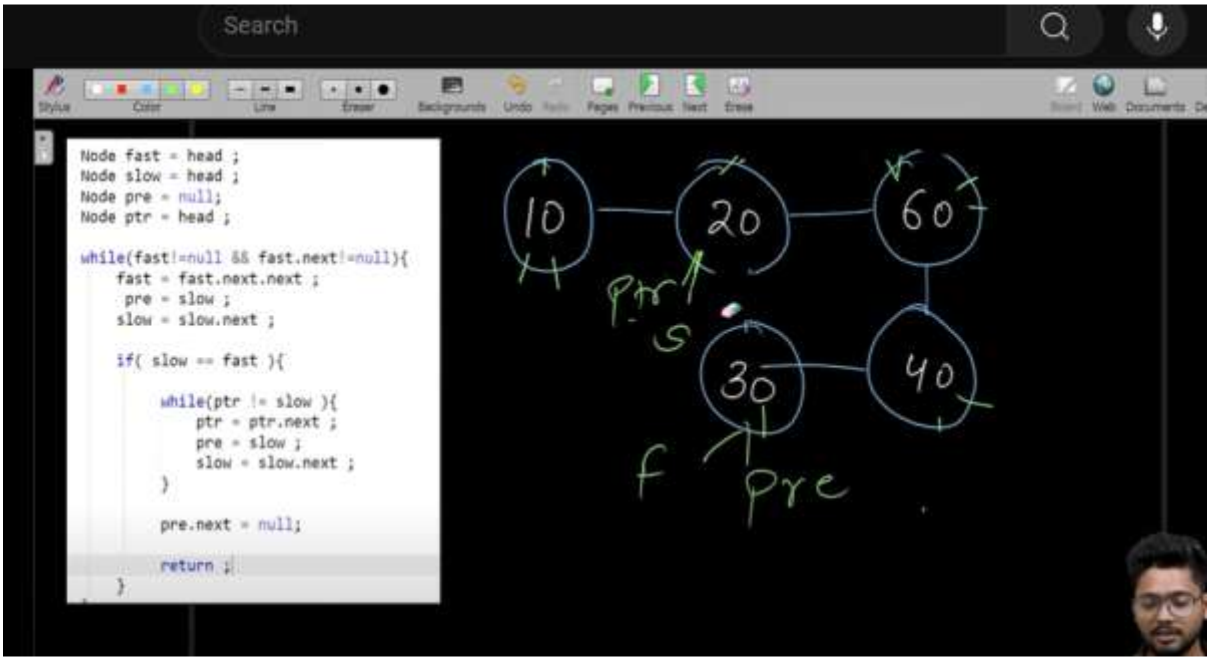




```
Node fast = head ;
Node slow = head ;

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

    fast = fast.next.next ;
    slow = slow.next ;

    if(fast == slow ){ // there is a loop

        int count = 1 ;
        slow = slow.next ;

        while(fast!=slow ){
            slow = slow.next ;
            count++;
        }

        return  count ;
    }
}

return 0 ;
```

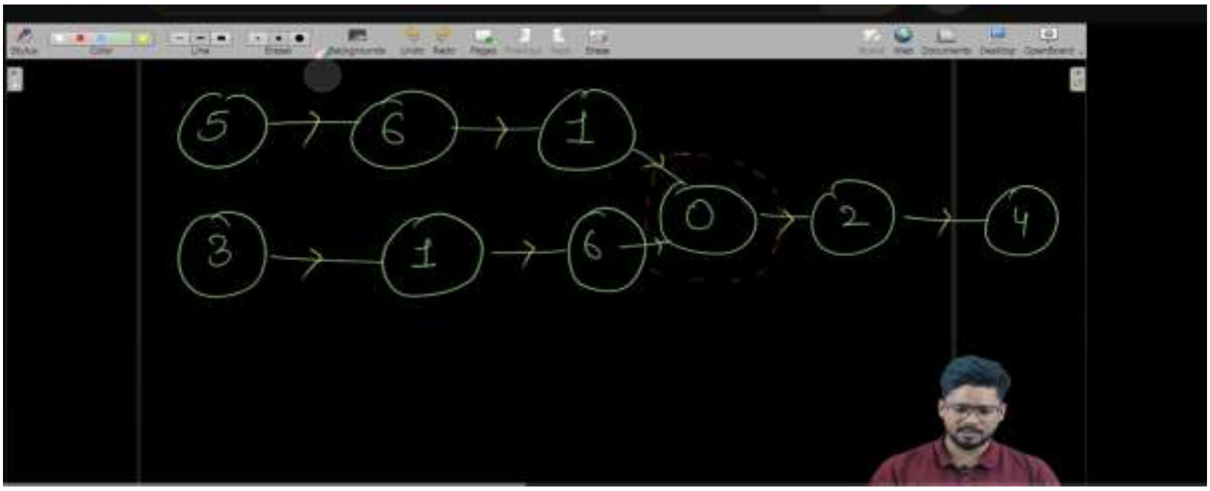
15. Remove loop in linked list

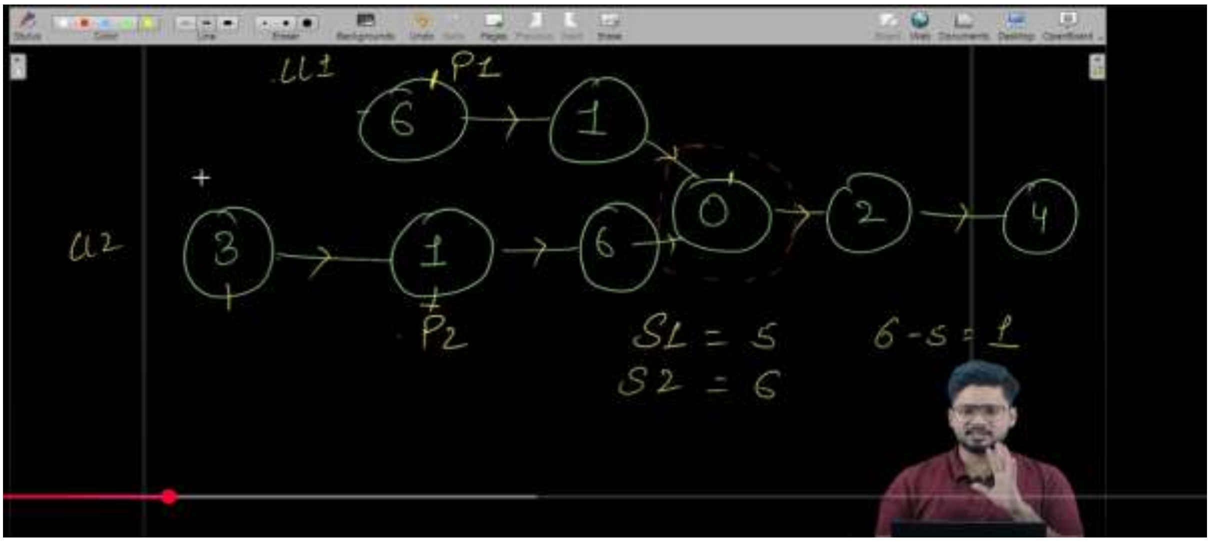


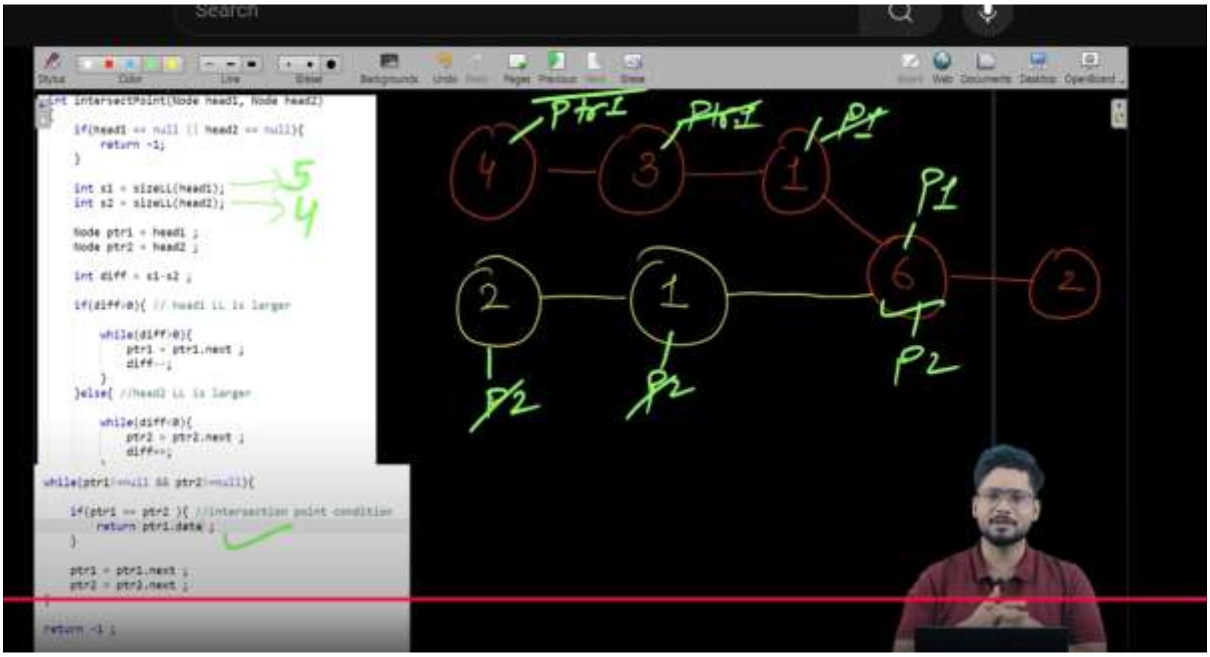


```
Node fast = head ;
Node slow = head ;
Node pre = null;
Node ptr = head ;

while(fast!=null && fast.next!=null){
    fast = fast.next.next ;
     pre = slow ;
    slow = slow.next ;

    if( slow == fast ){

        while(ptr != slow ){
            ptr = ptr.next ;
            pre = slow ;
            slow = slow.next ;
        }

        pre.next = null;

        return ;
    }
```

16.Intesection point in Y shaped linked list

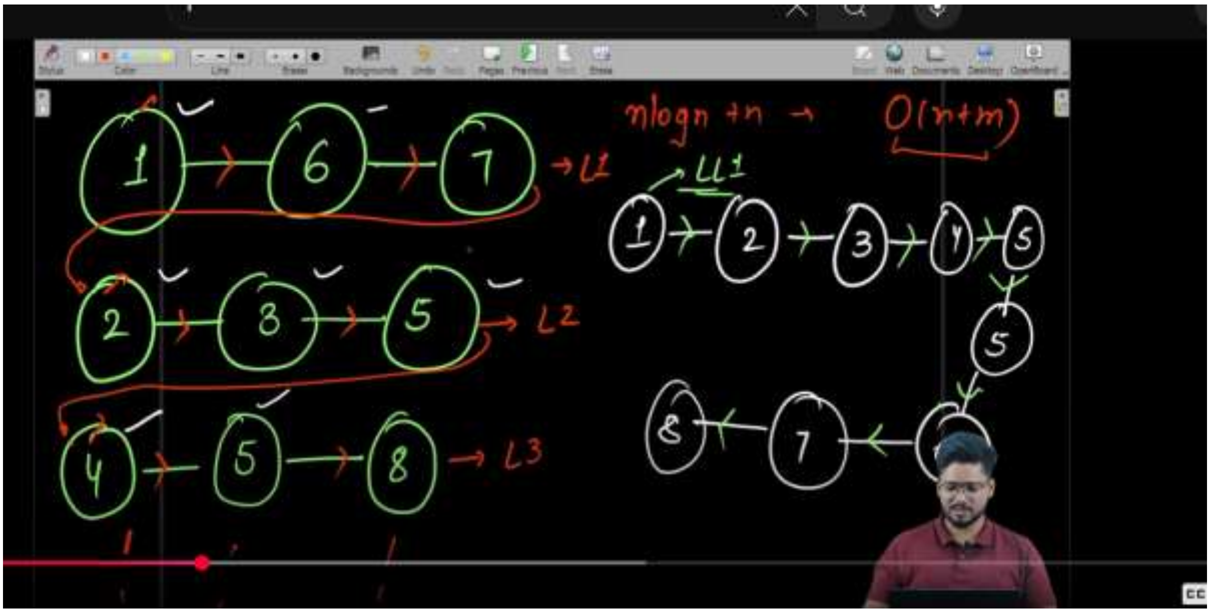




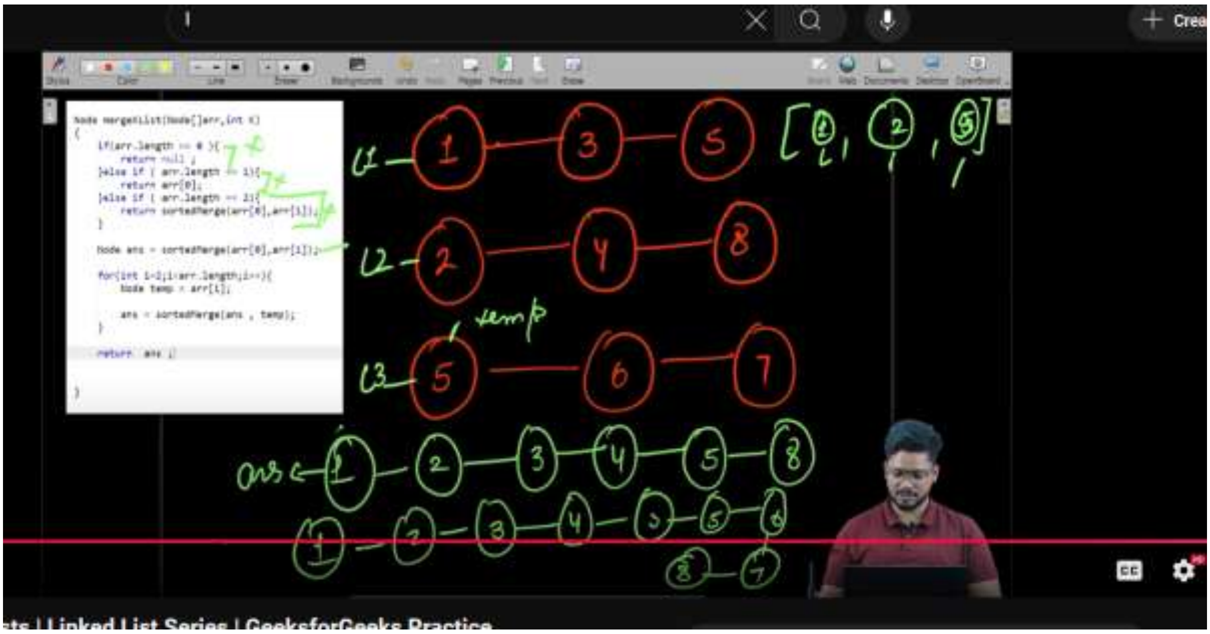


```
int intersectPoint(Node head1, Node head2)
{
    if(head1 == null || head2 == null){
        return -1;
    }

    int s1 = sizeLL(head1);
    int s2 = sizeLL(head2);

    Node ptr1 = head1 ;
    Node ptr2 = head2 ;

    int diff = s1-s2 ;

    if(diff>0){ // head1 LL is larger

        while(diff>0){
            ptr1 = ptr1.next ;
            diff--;
        }
    }else{ //head2 LL is larger

        while(diff<0){
            ptr2 = ptr2.next ;
            diff++;
        }
while(ptr1!=null && ptr2!=null){

    if(ptr1 == ptr2 ){ //intersection point condition
        return ptr1.data ;
    }

    ptr1 = ptr1.next ;
    ptr2 = ptr2.next ;
}

return -1 ;
```

17.Merge k Sorted Linked List

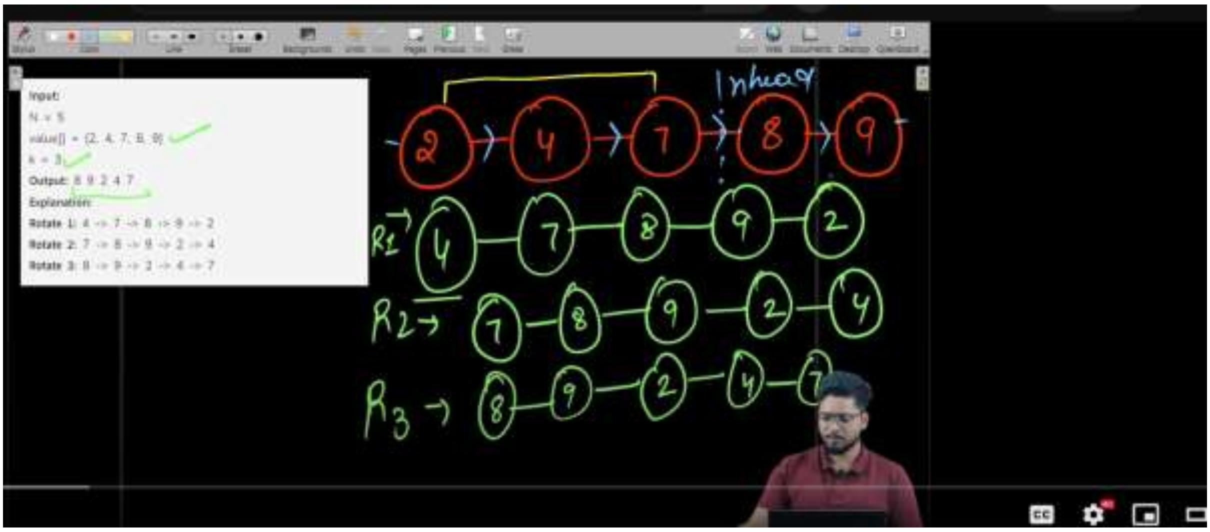


Approach : *In this,first we merge the first two arr into linked list and then for upcoming arr we use for loop from i=2,and most important is to use the exact code for (merge two sorted linked list) ans we call that function(sortedMerge) when converted arr to linked node.*

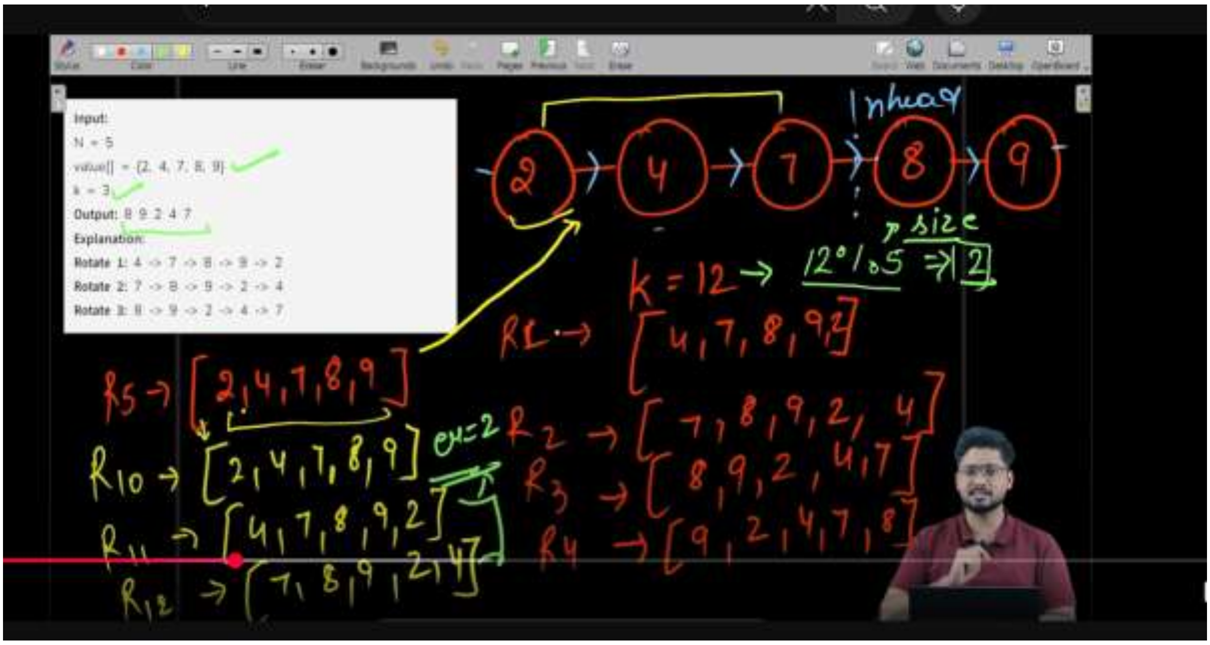


```
Node mergeKList(Node[]arr,int K)
{
    if(arr.length == 0 ){
        return null ;
    }else if ( arr.length == 1){
        return arr[0];
    }else if ( arr.length == 2){
        return sortedMerge(arr[0],arr[1]);
    }

    Node ans = sortedMerge(arr[0],arr[1]);

    for(int i=2;i<arr.length;i++){
        Node temp = arr[i];

        ans = sortedMerge(ans , temp);
    }

    return  ans ;

}
```

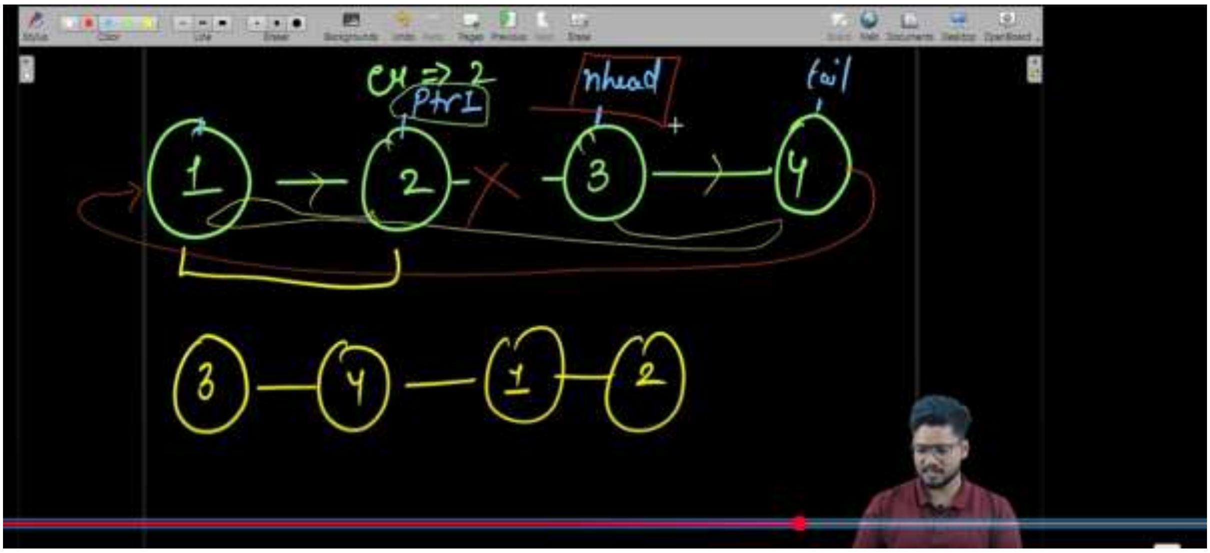
18.Rotate a Linked List

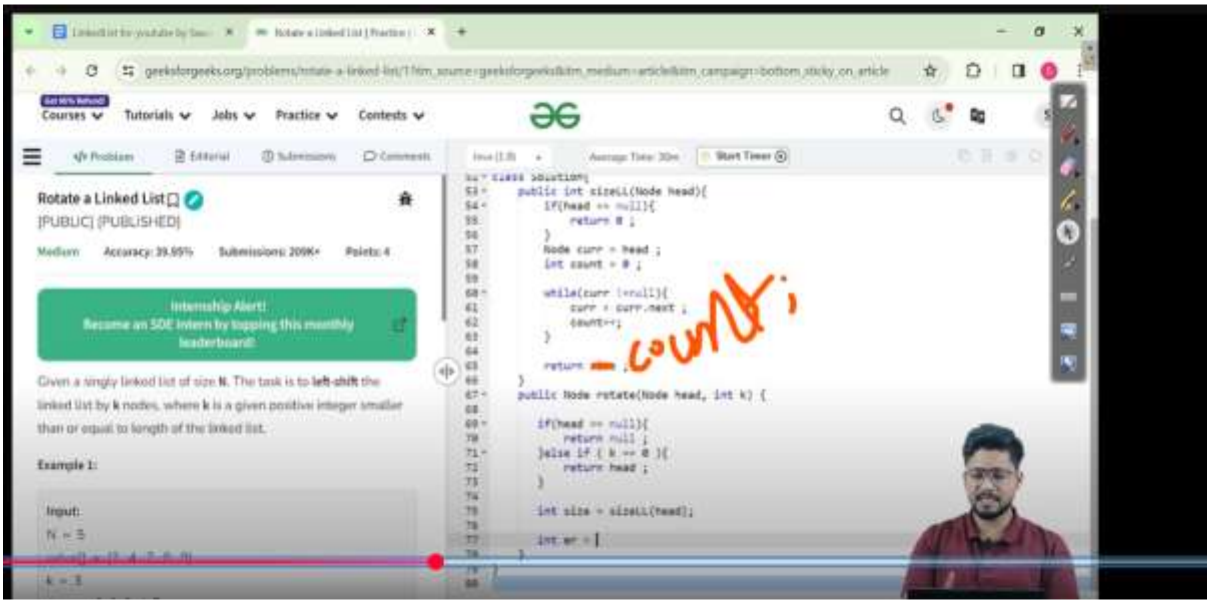


Input:
N = 5
value[] = {2, 4, 7, 8, 9}
k = 3
Output: 8 9 2 4 7
Explanation:
Rotate 1: 4 -> 7 -> 8 -> 9 -> 2
Rotate 2: 7 -> 8 -> 9 -> 2 -> 4
Rotate 3: 8 -> 9 -> 2 -> 4 -> 7



Input:
N = 5
value[] = {2, 4, 7, 8, 9}
k = 3
Output: 8 9 2 4 7
Explanation:
Rotate 1: 4 -> 7 -> 8 -> 9 -> 2
Rotate 2: 7 -> 8 -> 9 -> 2 -> 4
Rotate 3: 8 -> 9 -> 2 -> 4 -> 7





geeksforgeeks.org/problems/rotate-a-linked-list/1?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=bottom_sticky_on_article

Rotate a Linked List

[PUBLIC] [PUBLISHED]

Medium   Accuracy: 39.95%   Submissions: 209K+   Points: 4

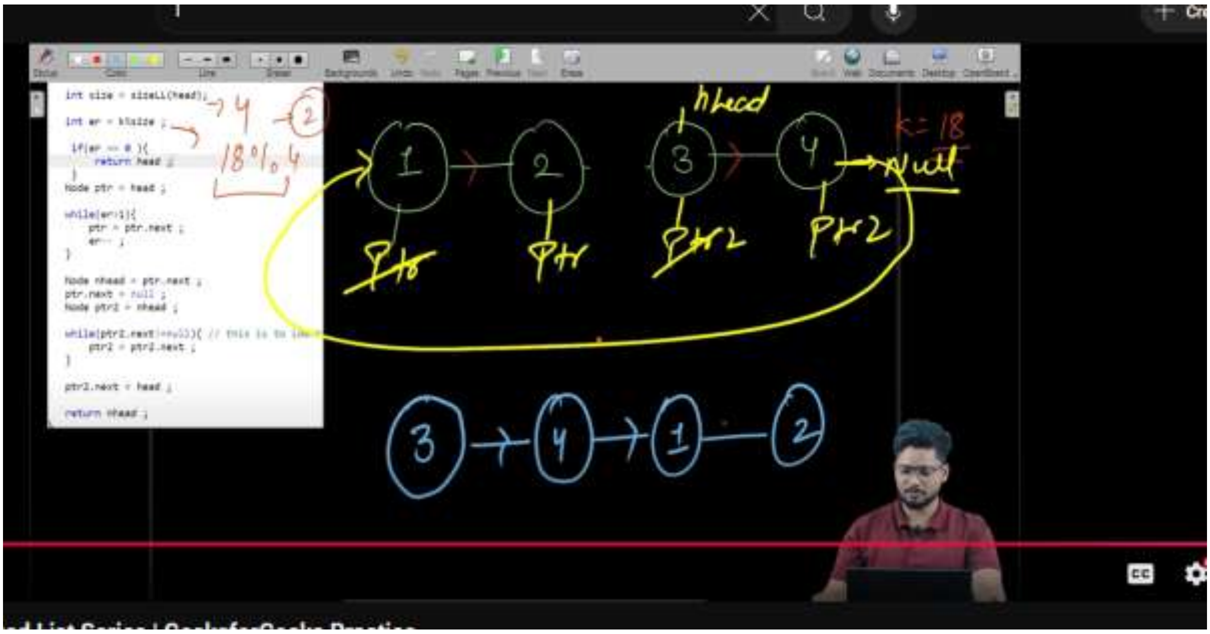
Internship Alert!
Become an SDE intern by topping this monthly leaderboard!

Given a singly linked list of size **N**. The task is to **left-shift** the linked list by **k** nodes, where **k** is a given positive integer smaller than or equal to length of the linked list.

Example 1:

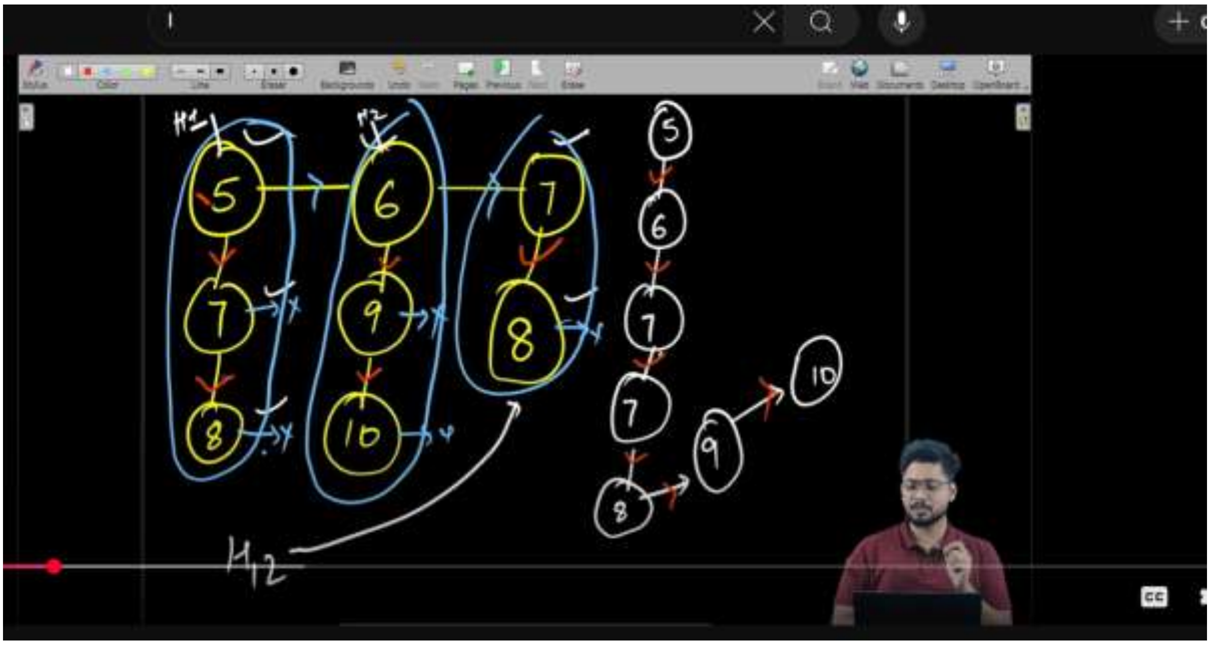
Input:
N = 5
value[] = {2, 4, 7, 8, 9}
k = 3

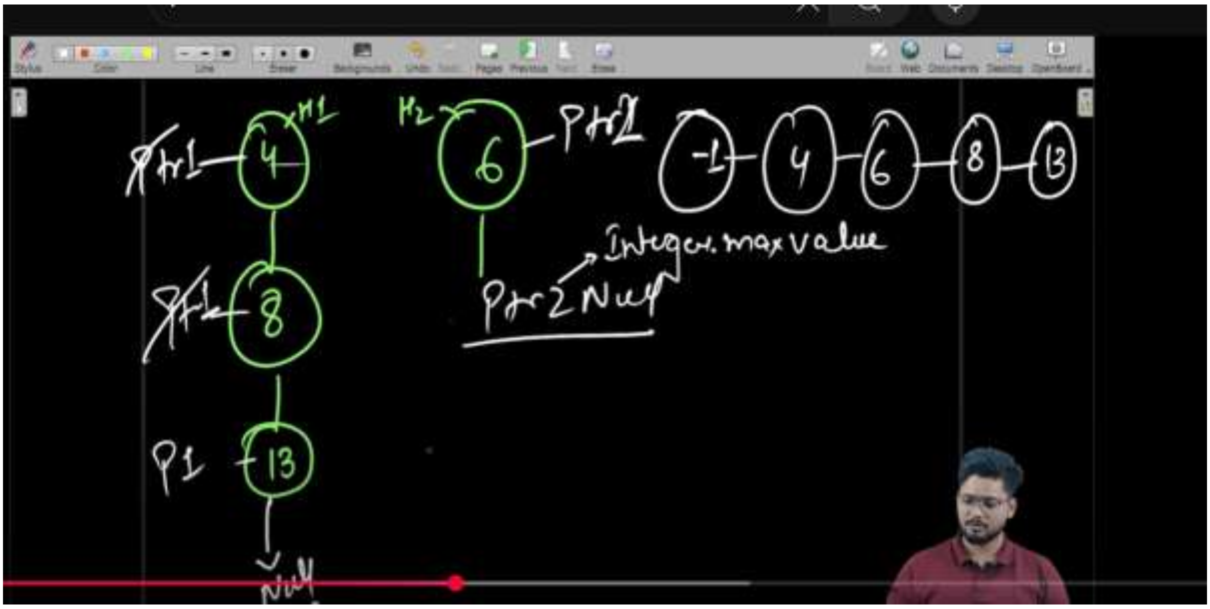
```java
class Solution{
    public int sizeLL(Node head){
        if(head == null){
            return 0 ;
        }
        Node curr = head ;
        int count = 0 ;

        while(curr !=null){
            curr = curr.next ;
            count++;
        }

        return     ;
    }
    public Node rotate(Node head, int k) {

       if(head == null){
           return null ;
       }else if ( k == 0 ){
           return head ;
       }

       int size = sizeLL(head);

       int er = 
    }
}
```

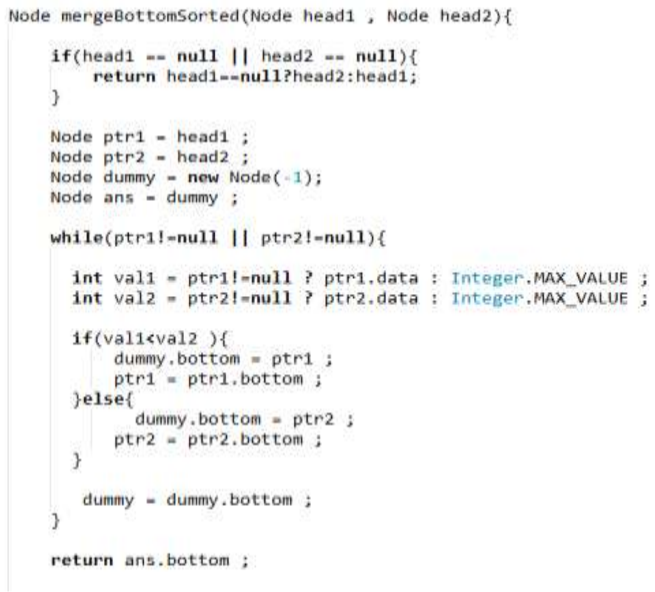


```java
int size = sizeLL(head);

int er = k%size ;

 if(er == 0 ){
     return head ;
 }
Node ptr = head ;

while(er>1){
    ptr = ptr.next ;
    er-- ;
}

Node nhead = ptr.next ;
ptr.next = null ;
Node ptr2 = nhead ;

while(ptr2.next!=null)){ // this is to [illegible]
    ptr2 = ptr2.next ;
}

ptr2.next = head ;

return nhead ;
```

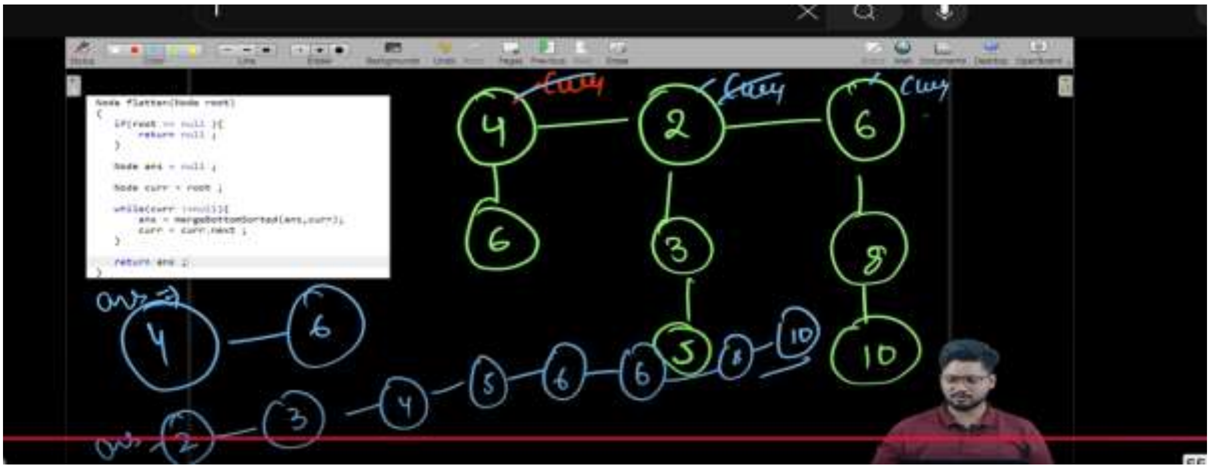
19.Flattening a Linked List





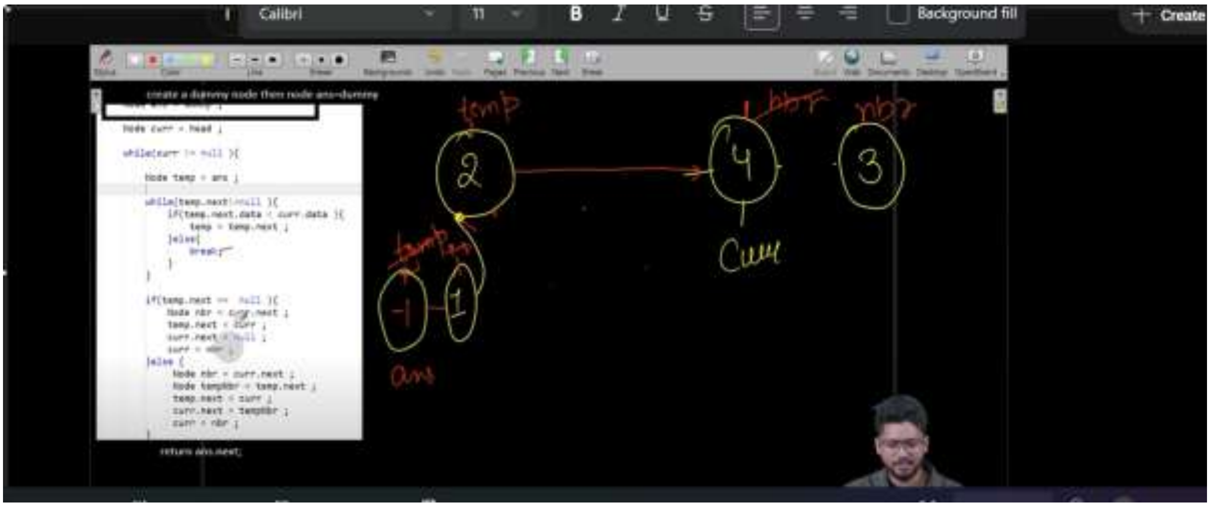


```
Node mergeBottomSorted(Node head1 , Node head2){

    if(head1 == null || head2 == null){
        return head1==null?head2:head1;
    }

    Node ptr1 = head1 ;
    Node ptr2 = head2 ;
    Node dummy = new Node(-1);
    Node ans = dummy ;

    while(ptr1!=null || ptr2!=null){

      int val1 = ptr1!=null ? ptr1.data : Integer.MAX_VALUE ;
      int val2 = ptr2!=null ? ptr2.data : Integer.MAX_VALUE ;

      if(val1<val2 ){
          dummy.bottom = ptr1 ;
          ptr1 = ptr1.bottom ;
      }else{
            dummy.bottom = ptr2 ;
          ptr2 = ptr2.bottom ;
      }

       dummy = dummy.bottom ;
    }

    return ans.bottom ;
```



```
Node flatten(Node root)
{
    if(root == null ){
        return null ;
    }

    Node ans = null ;

    Node curr = root ;

    while(curr !=null){
        ans = mergeBottomSorted(ans,curr);
        curr = curr.next ;
    }

    return ans ;
}
```

20.Insertion sort for Single linked list

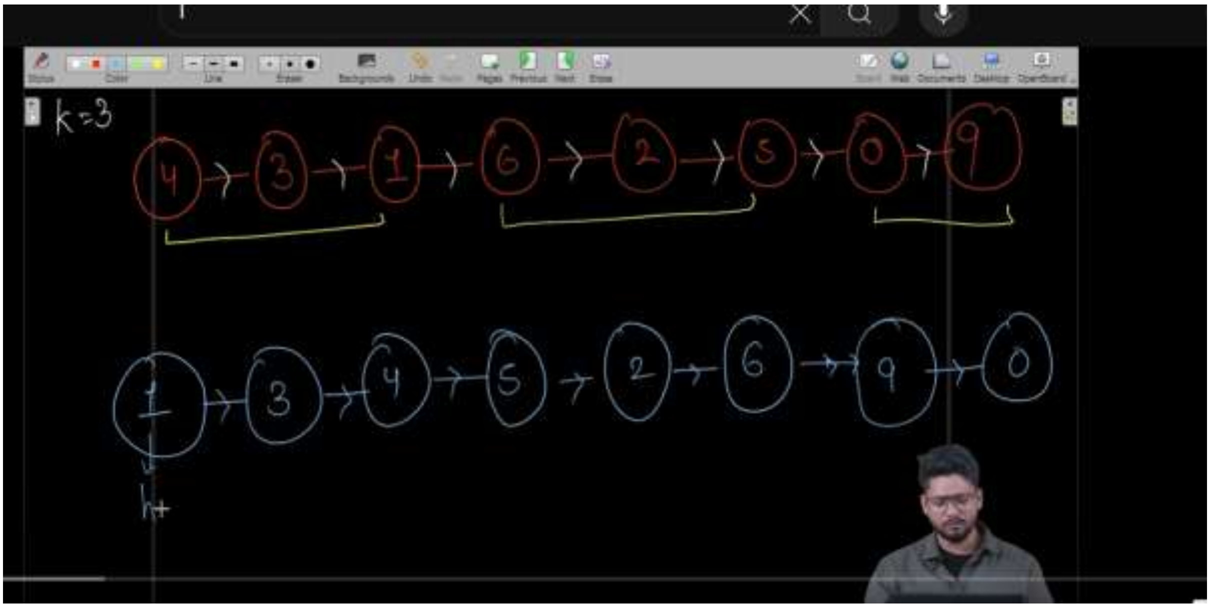


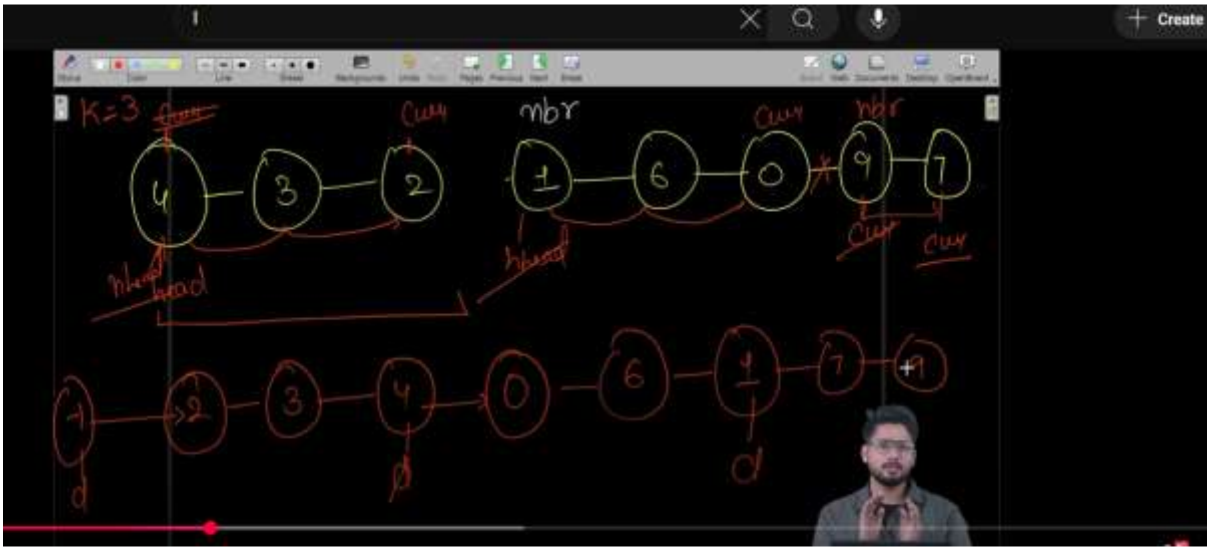
create a dummy node then node ans=dummy

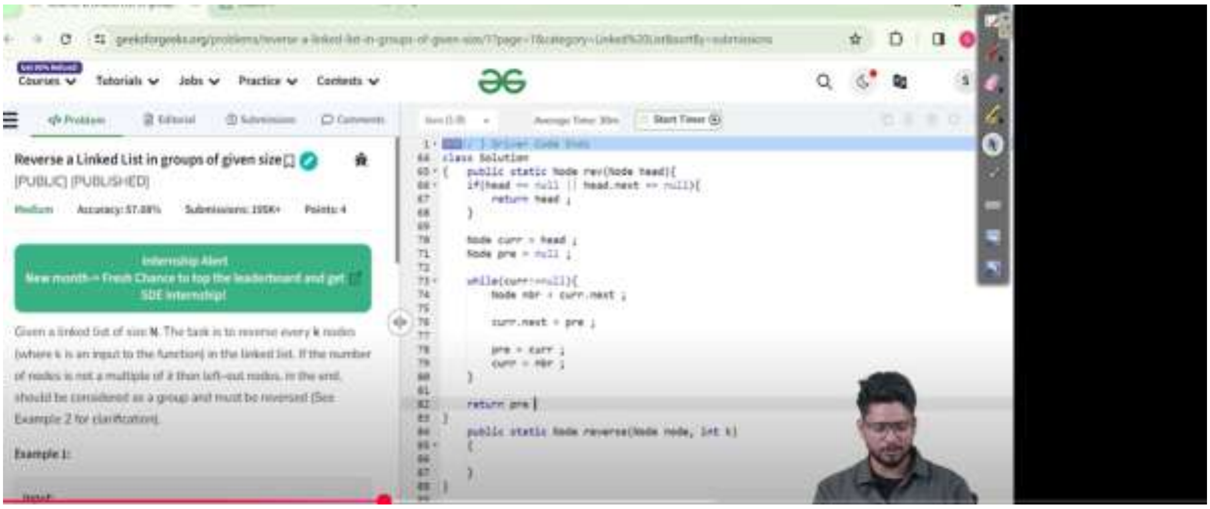
```
Node curr = head ;

while(curr != null ){

    Node temp = ans ;

    while(temp.next!=null ){
        if(temp.next.data < curr.data ){
            temp = temp.next ;
        }else{
            break;
        }
    }

    if(temp.next ==  null ){
        Node nbr = curr.next ;
        temp.next = curr ;
        curr.next = null ;
        curr = nbr ;
    }else {
         Node nbr = curr.next ;
         Node tempNbr = temp.next ;
         temp.next = curr ;
         curr.next = tempNbr ;
         curr = nbr ;
    }
```

return ans.next;

21.Reverse a linked list in group size







geeksforgeeks.org/problems/reverse-a-linked-list-in-groups-of-given-size/1?page=1&category=Linked%20List&sortBy=submissions
Courses
Tutorials
Jobs
Practice
Contests
Problem
Editorial
Submissions
Comments
Average Time: 30m
Start Timer

Reverse a Linked List in groups of given size

[PUBLIC] [PUBLISHED]

Medium Accuracy: 57.08% Submissions: 195K+ Points: 4

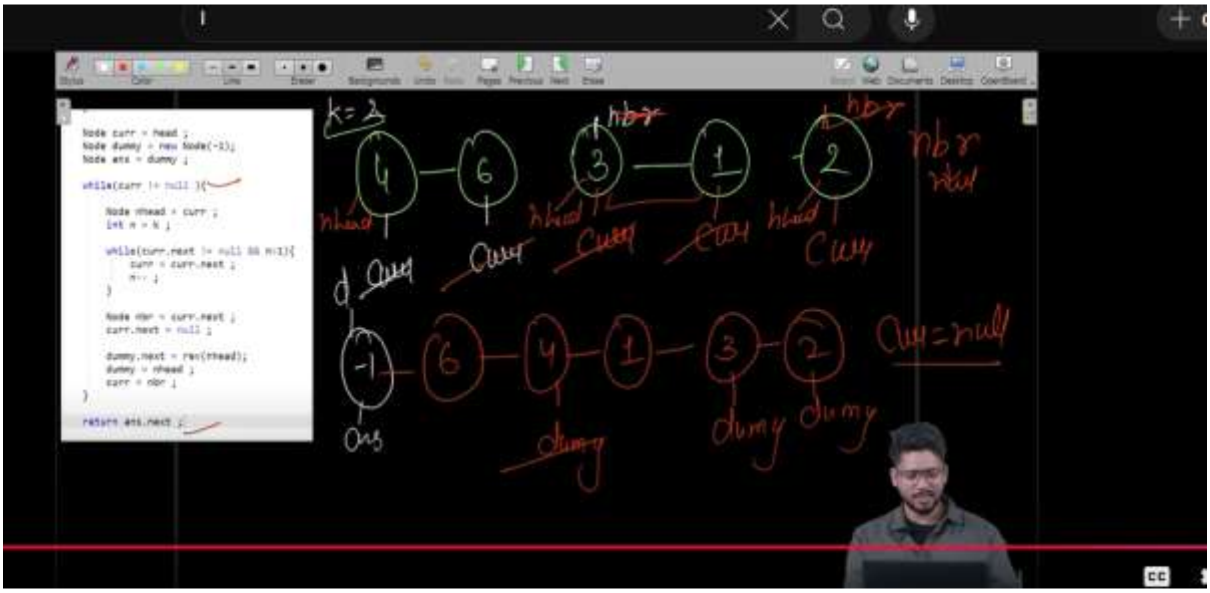
Internship Alert
New month→ Fresh Chance to top the leaderboard and get SDE internship!

Given a linked list of size **N**. The task is to reverse every **k** nodes (where k is an input to the function) in the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should be considered as a group and must be reversed (See Example 2 for clarification).

Example 1:

Input:

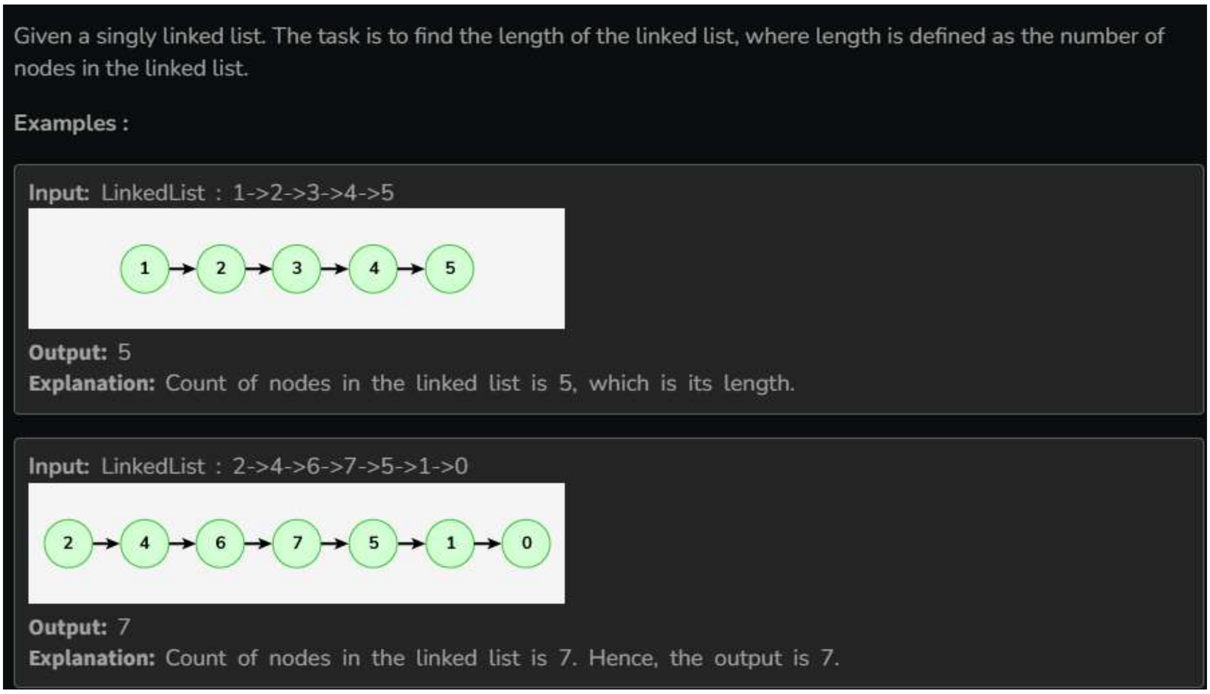
```java
// } Driver Code Ends
class Solution
{   public static Node rev(Node head){
    if(head == null || head.next == null){
        return head ;
    }

    Node curr = head ;
    Node pre = null ;

    while(curr!=null){
        Node nbr = curr.next ;

        curr.next = pre ;

        pre = curr ;
        curr = nbr ;
    }

    return pre
}
    public static Node reverse(Node node, int k)
    {

    }
}
```



```java
Node curr = head ;
Node dummy = new Node(-1);
Node ans = dummy ;

while(curr != null ){

    Node nhead = curr ;
    int n = k ;

    while(curr.next != null && n>1){
        curr = curr.next ;
        n-- ;
    }

    Node nbr = curr.next ;
    curr.next = null ;

    dummy.next = rev(nhead);
    dummy = nhead ;
    curr = nbr ;
}

return ans.next ;
```

Problem Set

22.Count Linked List Nodes



Given a singly linked list. The task is to find the length of the linked list, where length is defined as the number of nodes in the linked list.

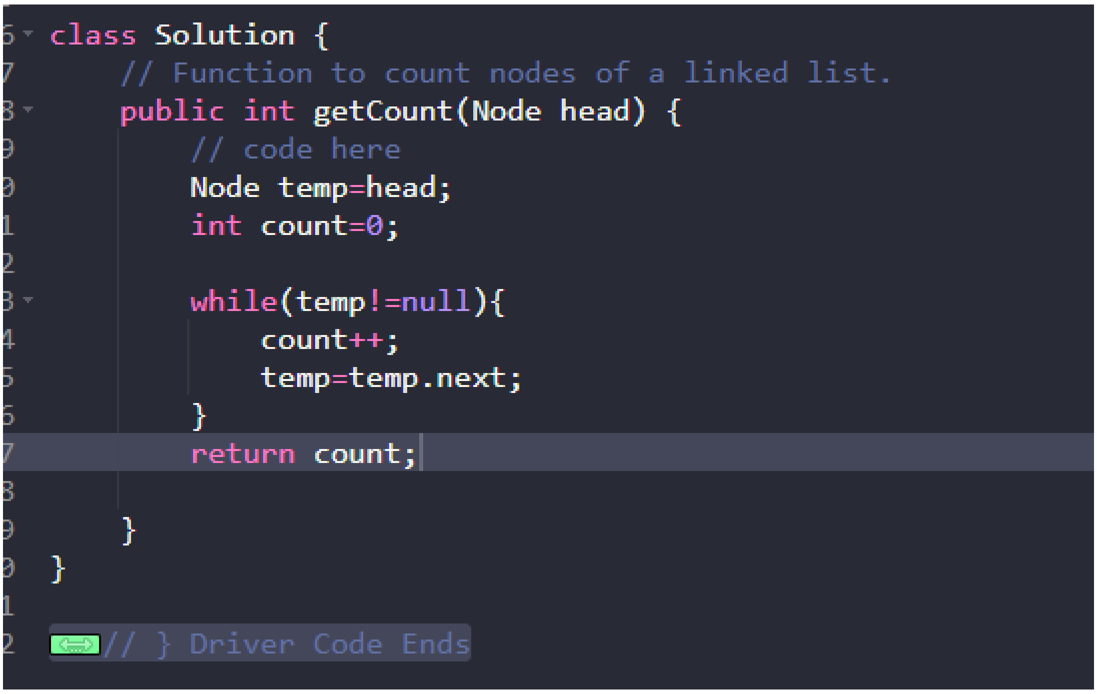
Examples :

Input: LinkedList : 1->2->3->4->5

Output: 5
Explanation: Count of nodes in the linked list is 5, which is its length.

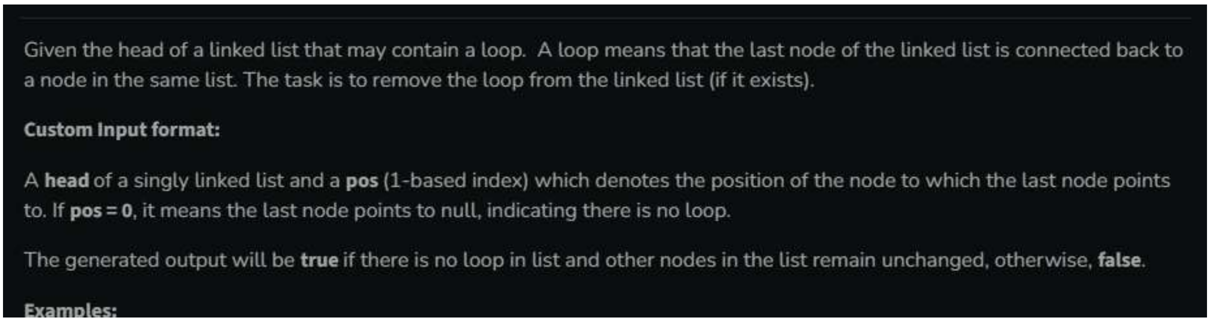
Input: LinkedList : 2->4->6->7->5->1->0

Output: 7
Explanation: Count of nodes in the linked list is 7. Hence, the output is 7.



```java
class Solution {
    // Function to count nodes of a linked list.
    public int getCount(Node head) {
        // code here
        Node temp=head;
        int count=0;

        while(temp!=null){
            count++;
            temp=temp.next;
        }
        return count;

    }
}

// } Driver Code Ends
```

23.Remove loop in LinkedList



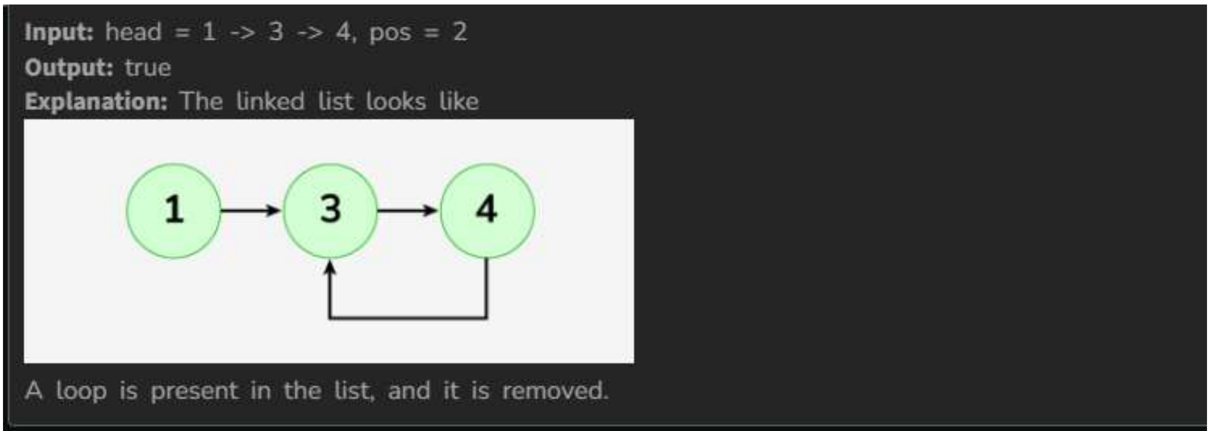
Given the head of a linked list that may contain a loop. A loop means that the last node of the linked list is connected back to a node in the same list. The task is to remove the loop from the linked list (if it exists).

Custom Input format:

A **head** of a singly linked list and a **pos** (1-based index) which denotes the position of the node to which the last node points to. If **pos = 0**, it means the last node points to null, indicating there is no loop.

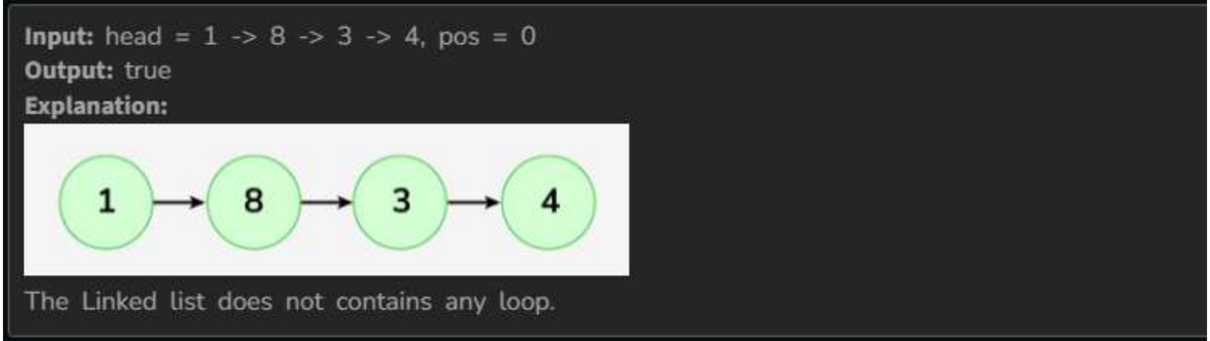
The generated output will be **true** if there is no loop in list and other nodes in the list remain unchanged, otherwise, **false**.

Examples:



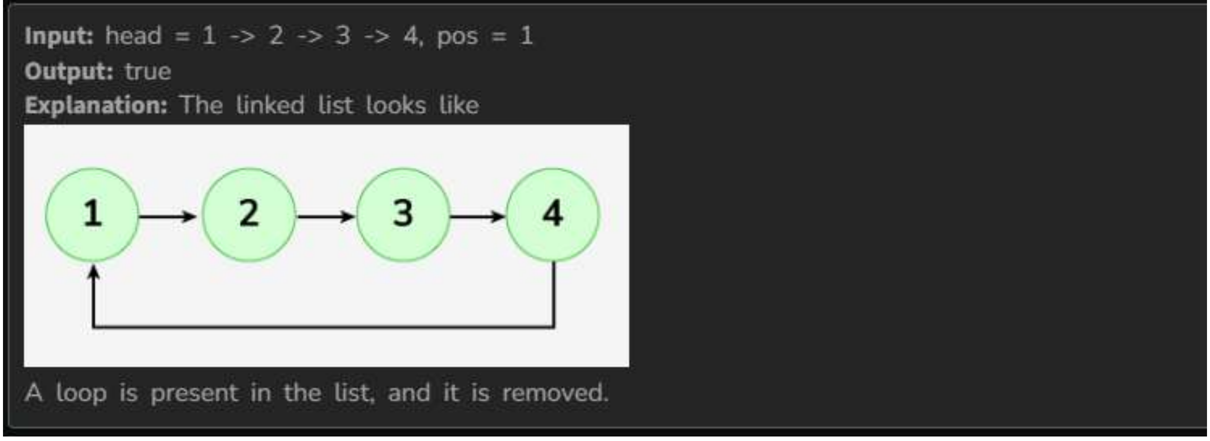
Input: head = 1 -> 3 -> 4, pos = 2
Output: true
Explanation: The linked list looks like

A loop is present in the list, and it is removed.



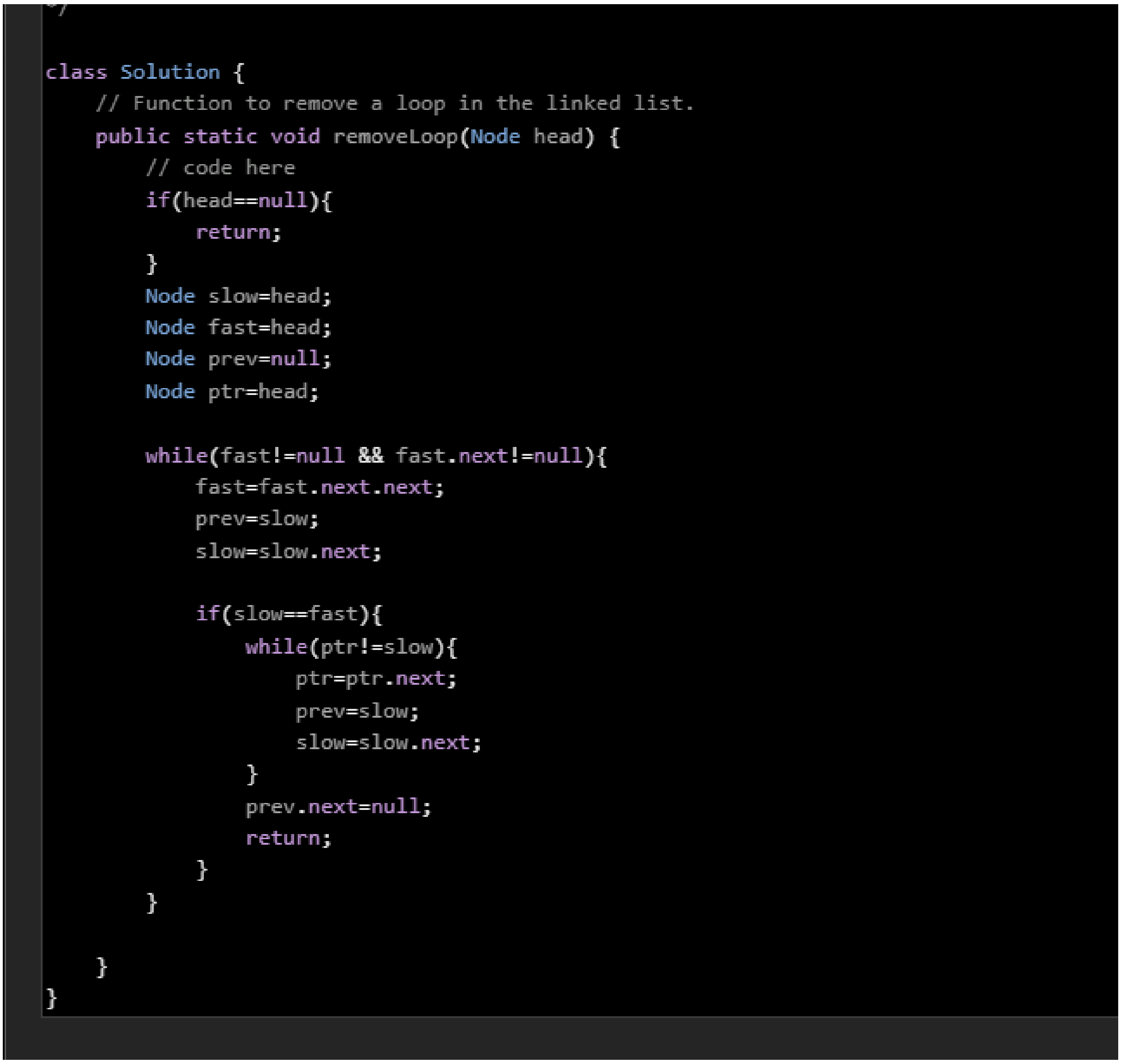
Input: head = 1 -> 8 -> 3 -> 4, pos = 0
Output: true
Explanation:

The Linked list does not contains any loop.



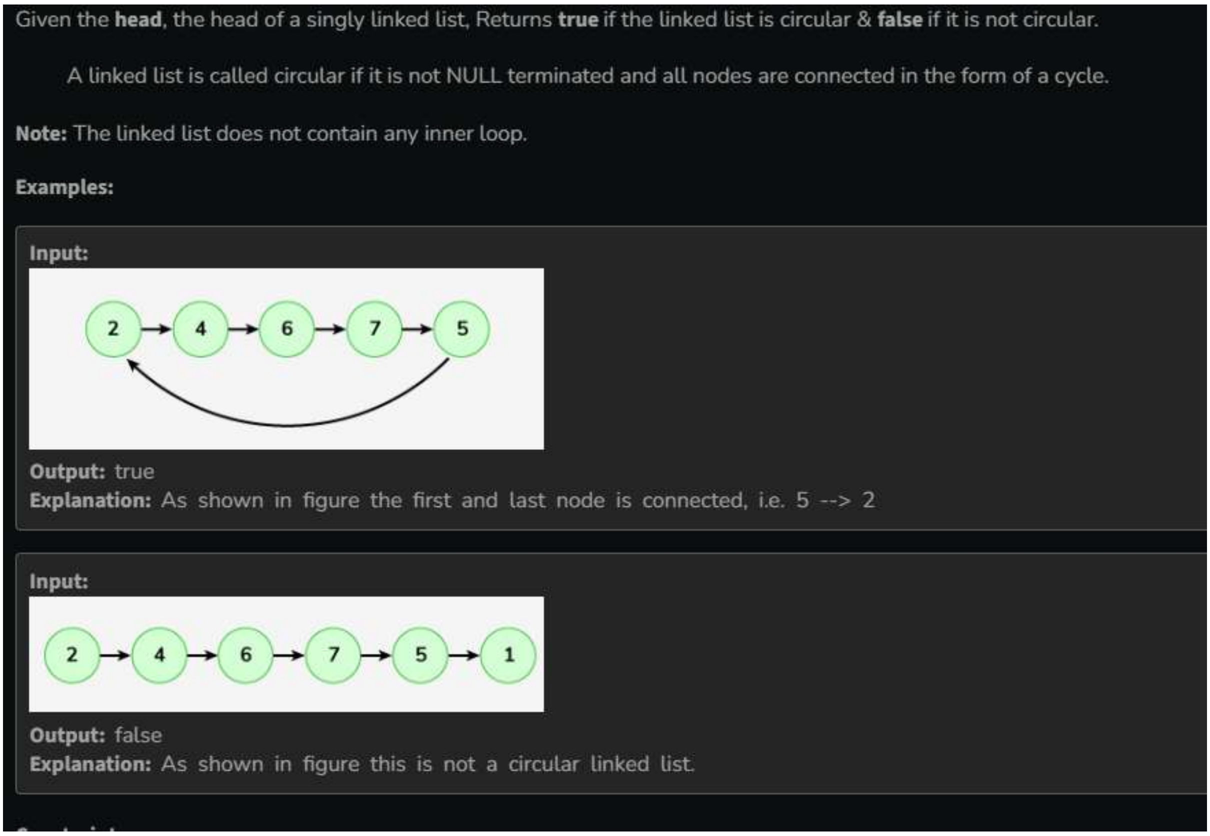
Input: head = 1 -> 2 -> 3 -> 4, pos = 1
Output: true
Explanation: The linked list looks like

A loop is present in the list, and it is removed.



```java
*/

class Solution {
    // Function to remove a loop in the linked list.
    public static void removeLoop(Node head) {
        // code here
        if(head==null){
            return;
        }
        Node slow=head;
        Node fast=head;
        Node prev=null;
        Node ptr=head;

        while(fast!=null && fast.next!=null){
            fast=fast.next.next;
            prev=slow;
            slow=slow.next;

            if(slow==fast){
                while(ptr!=slow){
                    ptr=ptr.next;
                    prev=slow;
                    slow=slow.next;
                }
                prev.next=null;
                return;
            }
        }

    }
}
```

24.check if the Circular Linked List



Given the **head**, the head of a singly linked list, Returns **true** if the linked list is circular & **false** if it is not circular.

A linked list is called circular if it is not NULL terminated and all nodes are connected in the form of a cycle.

Note: The linked list does not contain any inner loop.

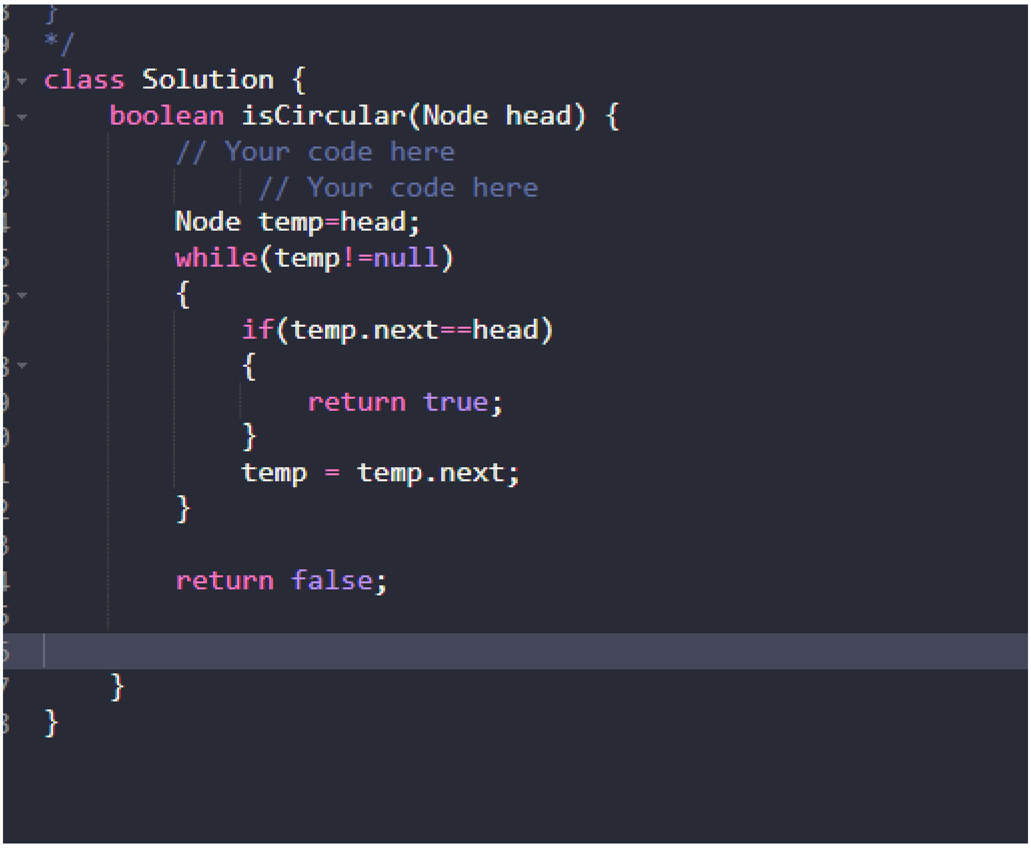
Examples:

Input:

Output: true
Explanation: As shown in figure the first and last node is connected, i.e. 5 --> 2

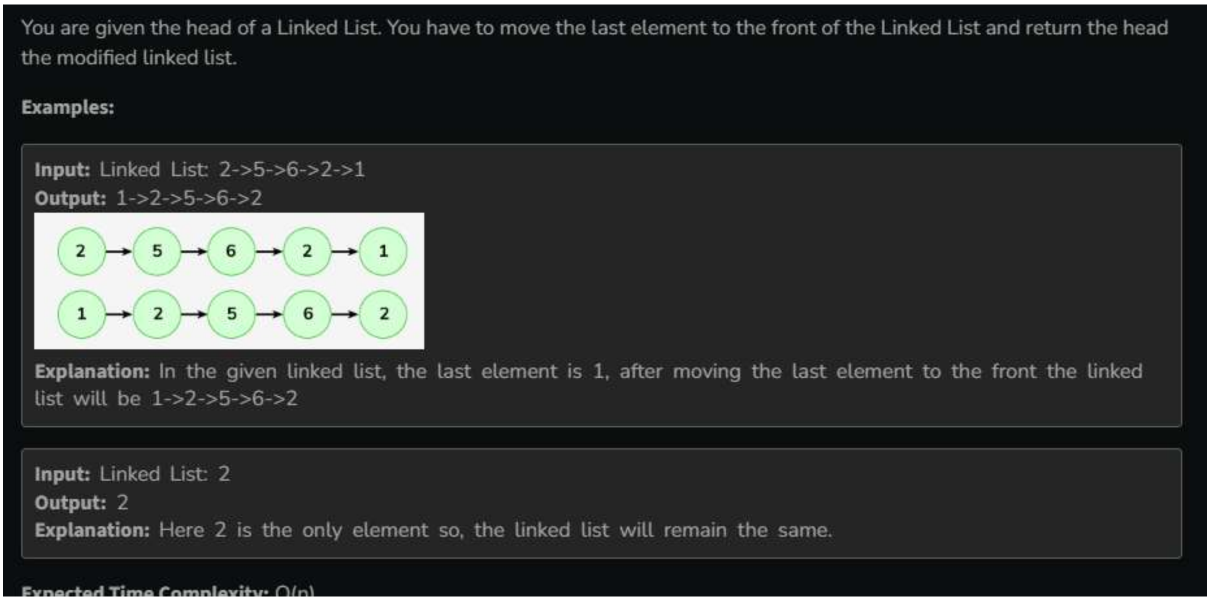
Input:

Output: false
Explanation: As shown in figure this is not a circular linked list.



```
    }
    */
class Solution {
    boolean isCircular(Node head) {
        // Your code here
            // Your code here
        Node temp=head;
        while(temp!=null)
        {
            if(temp.next==head)
            {
                return true;
            }
            temp = temp.next;
        }

        return false;


    }
}
```

25.Move last Element to front of a linked list



You are given the head of a Linked List. You have to move the last element to the front of the Linked List and return the head the modified linked list.

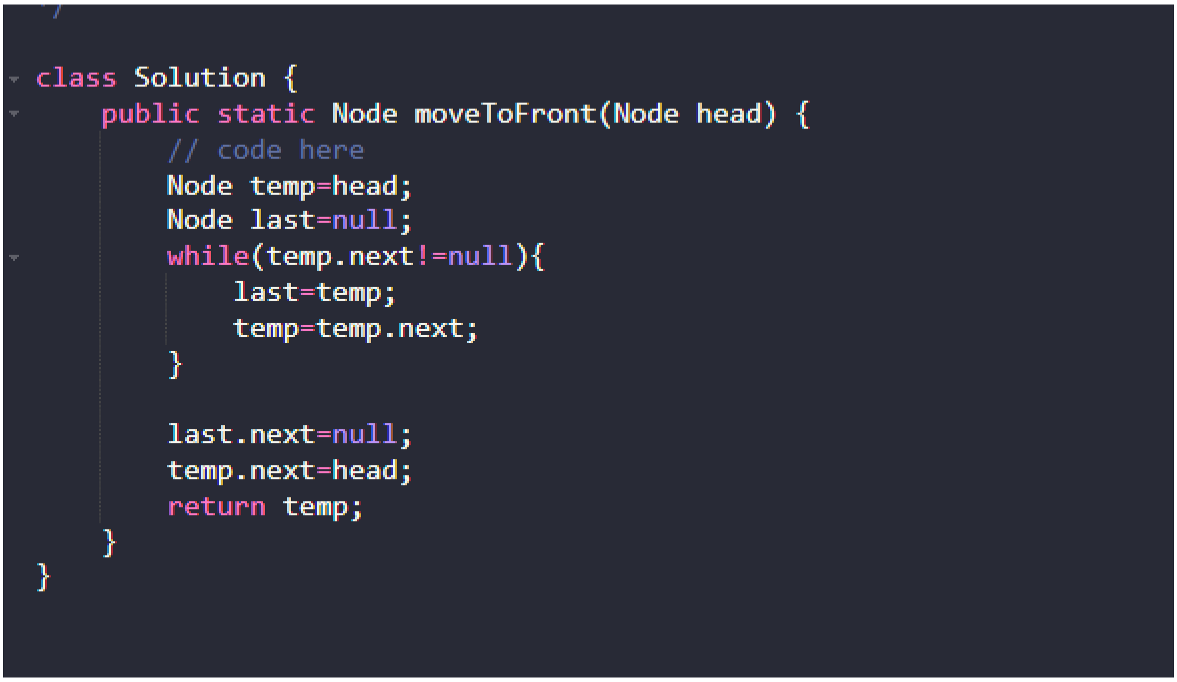
Examples:

Input: Linked List: 2->5->6->2->1
Output: 1->2->5->6->2

Explanation: In the given linked list, the last element is 1, after moving the last element to the front the linked list will be 1->2->5->6->2

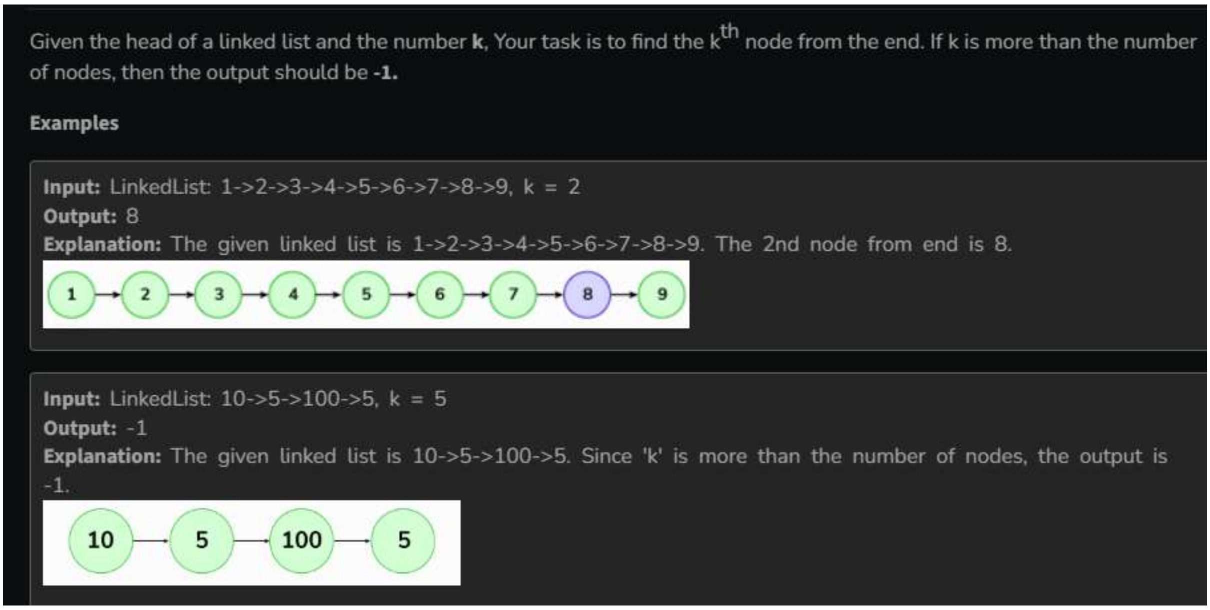
Input: Linked List: 2
Output: 2
Explanation: Here 2 is the only element so, the linked list will remain the same.

Expected Time Complexity: O(n)



```
 */

class Solution {
    public static Node moveToFront(Node head) {
        // code here
        Node temp=head;
        Node last=null;
        while(temp.next!=null){
            last=temp;
            temp=temp.next;
        }

        last.next=null;
        temp.next=head;
        return temp;
    }
}
```

26.Kth from End of Linked List

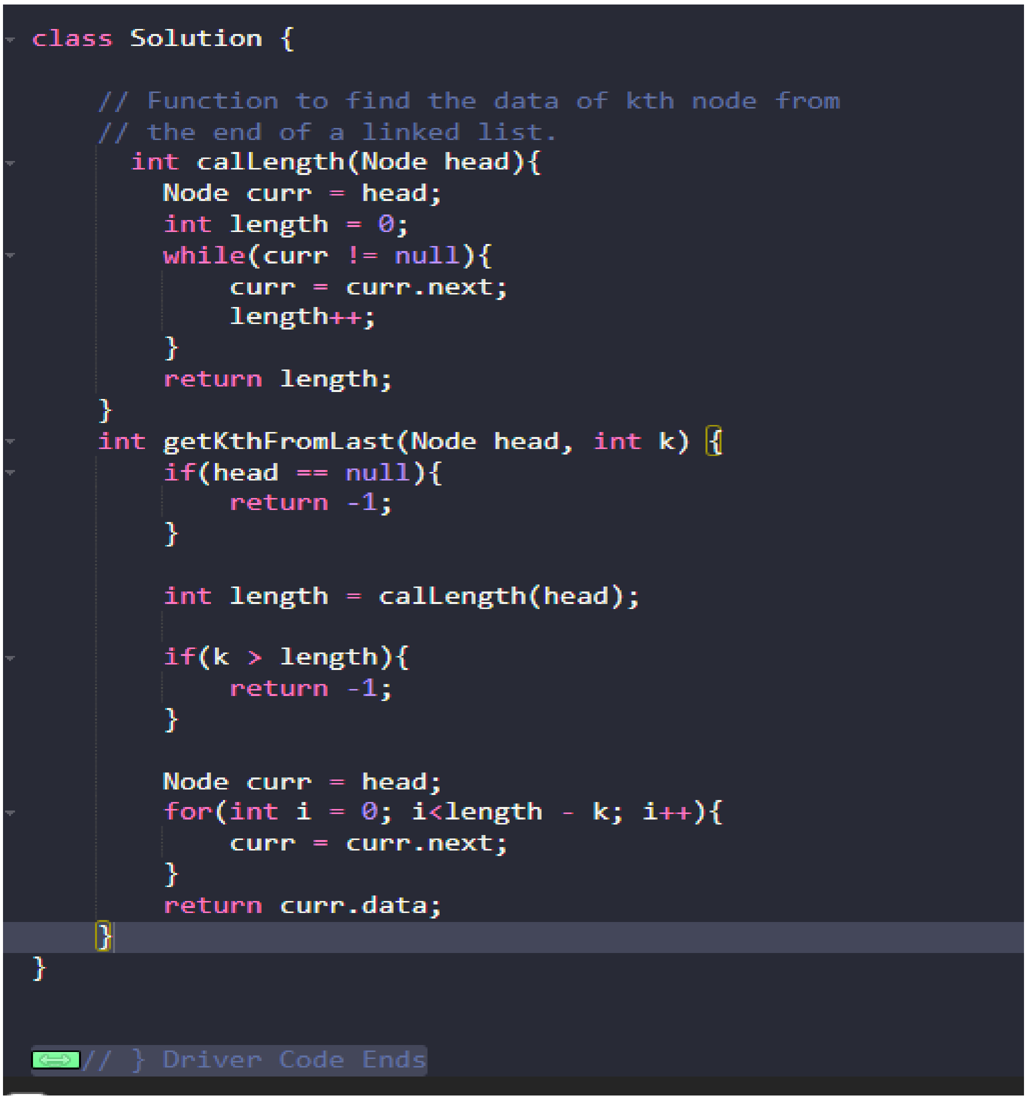


Given the head of a linked list and the number **k**, Your task is to find the kth node from the end. If k is more than the number of nodes, then the output should be **-1.**

Examples

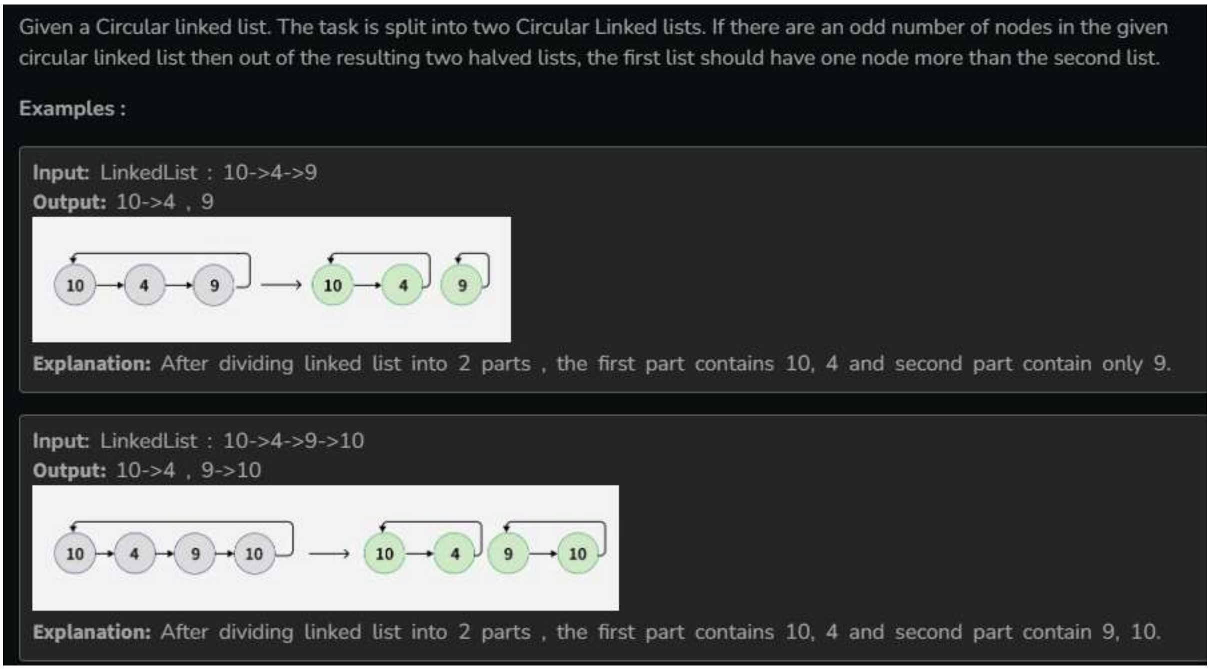
Input: LinkedList: 1->2->3->4->5->6->7->8->9, k = 2
Output: 8
Explanation: The given linked list is 1->2->3->4->5->6->7->8->9. The 2nd node from end is 8.

Input: LinkedList: 10->5->100->5, k = 5
Output: -1
Explanation: The given linked list is 10->5->100->5. Since 'k' is more than the number of nodes, the output is -1.



```
class Solution {

    // Function to find the data of kth node from
    // the end of a linked list.
      int calLength(Node head){
        Node curr = head;
        int length = 0;
        while(curr != null){
            curr = curr.next;
            length++;
        }
        return length;
    }
    int getKthFromLast(Node head, int k) {
        if(head == null){
            return -1;
        }

        int length = calLength(head);

        if(k > length){
            return -1;
        }

        Node curr = head;
        for(int i = 0; i<length - k; i++){
            curr = curr.next;
        }
        return curr.data;
    }
}


// } Driver Code Ends
```

27.Split a Linked List into two halves



Given a Circular linked list. The task is split into two Circular Linked lists. If there are an odd number of nodes in the given circular linked list then out of the resulting two halved lists, the first list should have one node more than the second list.

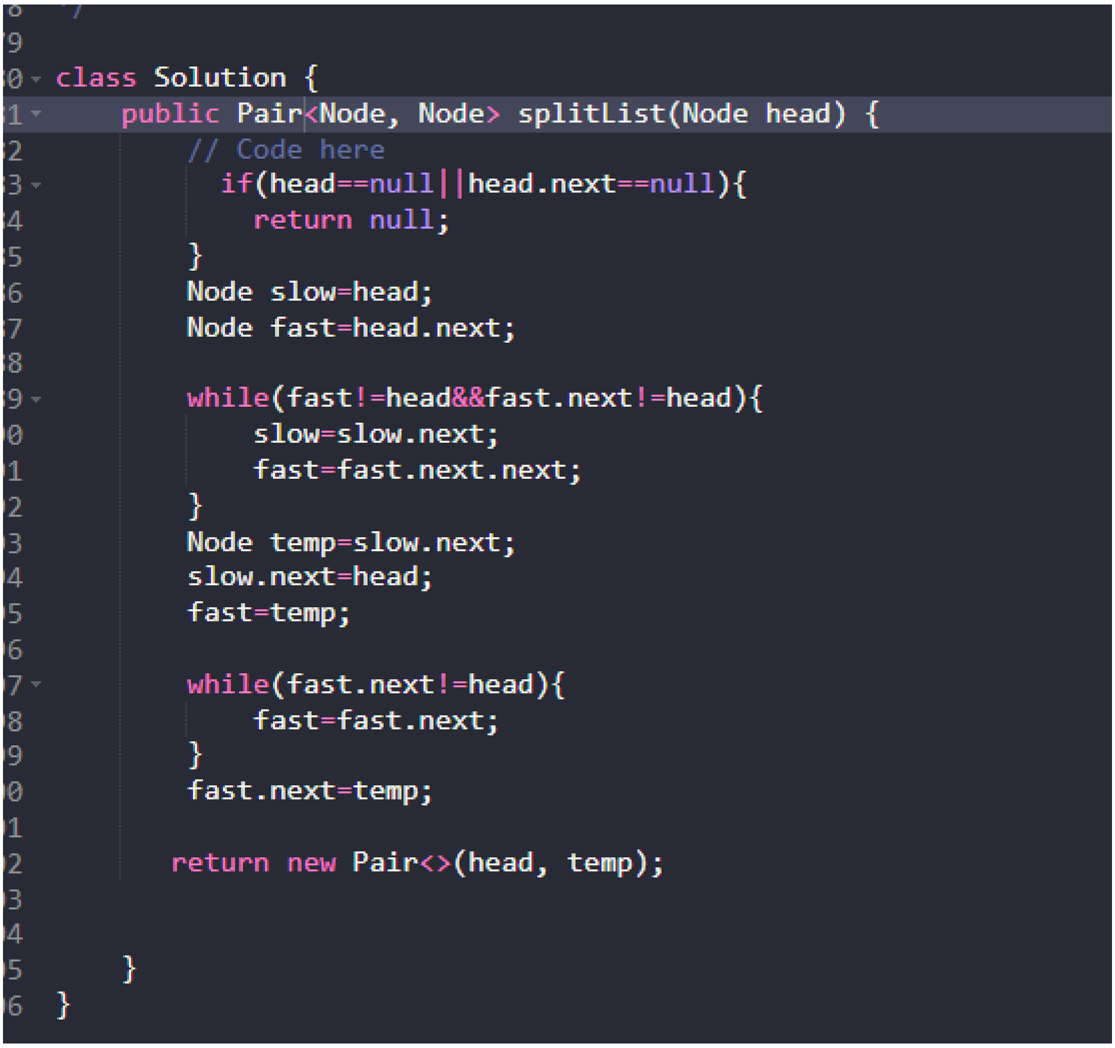
Examples :

Input: LinkedList : 10->4->9
Output: 10->4 , 9

Explanation: After dividing linked list into 2 parts , the first part contains 10, 4 and second part contain only 9.

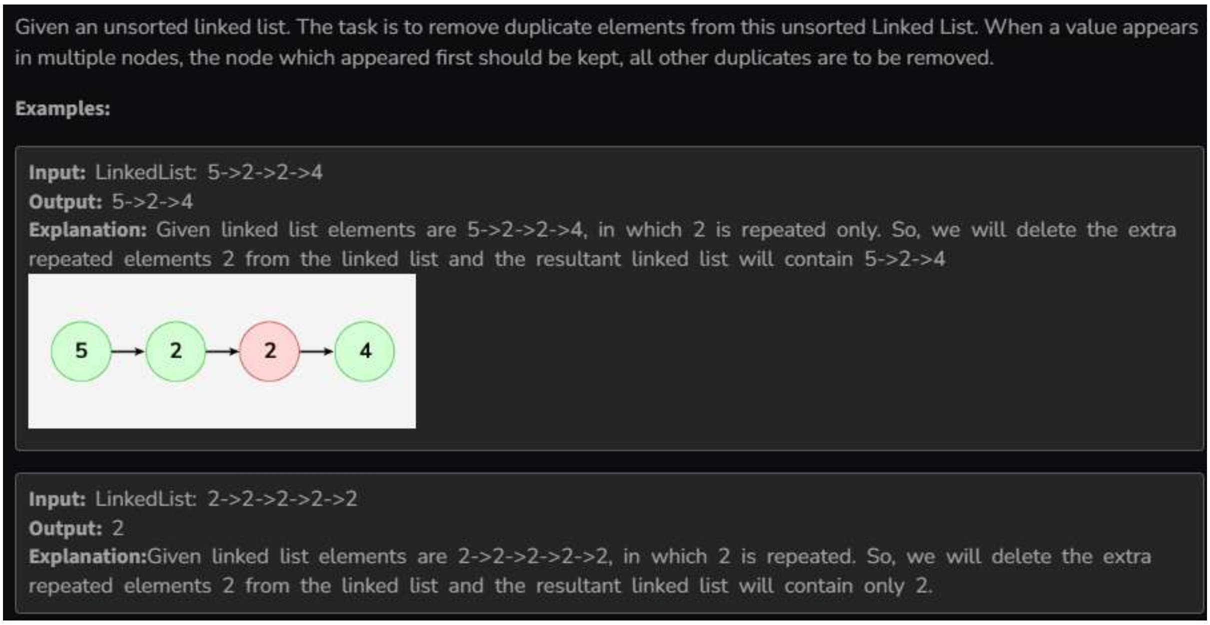
Input: LinkedList : 10->4->9->10
Output: 10->4 , 9->10

Explanation: After dividing linked list into 2 parts , the first part contains 10, 4 and second part contain 9, 10.



```
 */

class Solution {
    public Pair<Node, Node> splitList(Node head) {
        // Code here
          if(head==null||head.next==null){
            return null;
        }
        Node slow=head;
        Node fast=head.next;

        while(fast!=head&&fast.next!=head){
            slow=slow.next;
            fast=fast.next.next;
        }
        Node temp=slow.next;
        slow.next=head;
        fast=temp;

        while(fast.next!=head){
            fast=fast.next;
        }
        fast.next=temp;

       return new Pair<>(head, temp);


    }
}
```

28.Remove duplicate from unsorted Linked list

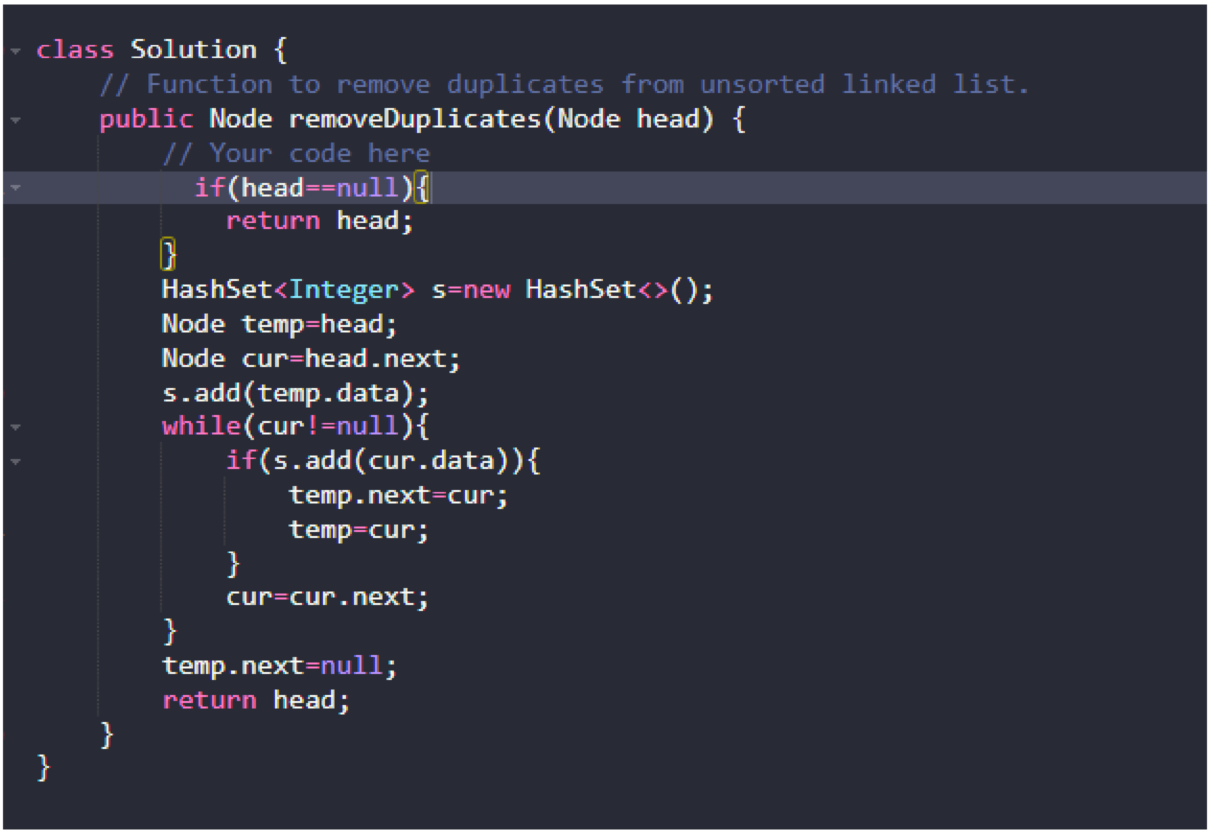


Given an unsorted linked list. The task is to remove duplicate elements from this unsorted Linked List. When a value appears in multiple nodes, the node which appeared first should be kept, all other duplicates are to be removed.

Examples:

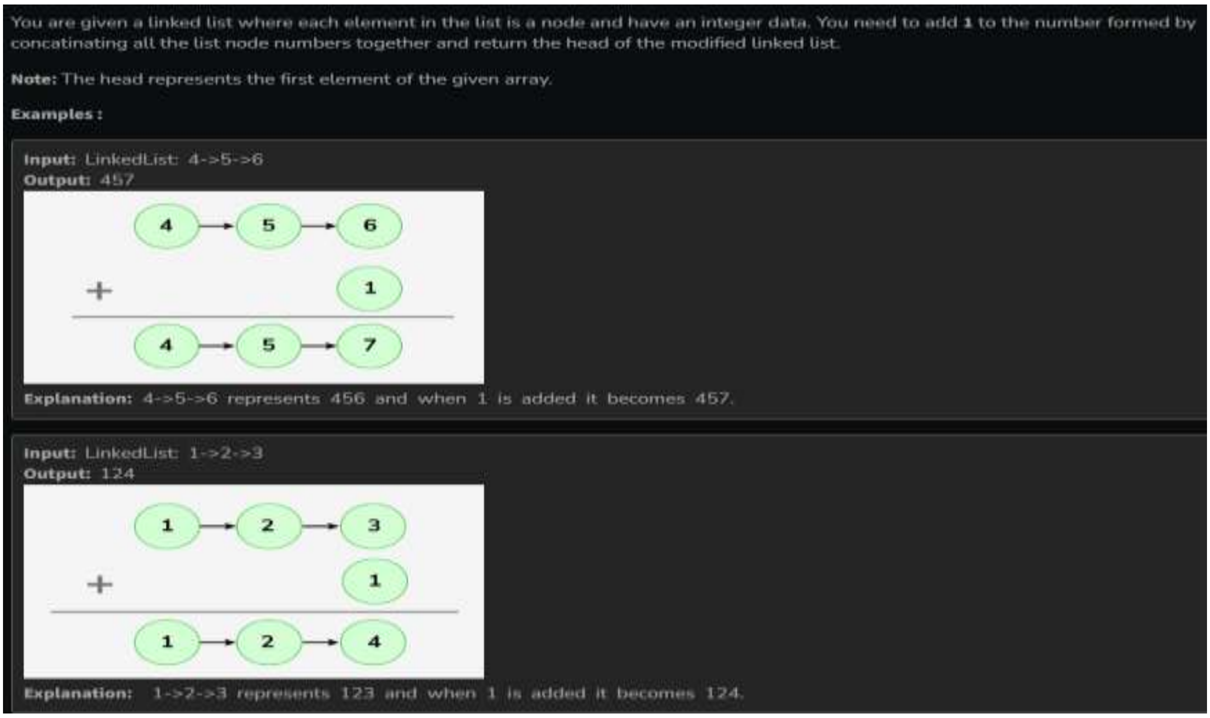
Input: LinkedList: 5->2->2->4
Output: 5->2->4
Explanation: Given linked list elements are 5->2->2->4, in which 2 is repeated only. So, we will delete the extra repeated elements 2 from the linked list and the resultant linked list will contain 5->2->4

Input: LinkedList: 2->2->2->2->2
Output: 2
Explanation: Given linked list elements are 2->2->2->2->2, in which 2 is repeated. So, we will delete the extra repeated elements 2 from the linked list and the resultant linked list will contain only 2.



```
class Solution {
    // Function to remove duplicates from unsorted linked list.
    public Node removeDuplicates(Node head) {
        // Your code here
          if(head==null){
            return head;
        }
        HashSet<Integer> s=new HashSet<>();
        Node temp=head;
        Node cur=head.next;
        s.add(temp.data);
        while(cur!=null){
            if(s.add(cur.data)){
                temp.next=cur;
                temp=cur;
            }
            cur=cur.next;
        }
        temp.next=null;
        return head;
    }
}
```

29.Add 1 to a Linked List number



You are given a linked list where each element in the list is a node and have an integer data. You need to add **1** to the number formed by concatinating all the list node numbers together and return the head of the modified linked list.

Note: The head represents the first element of the given array.

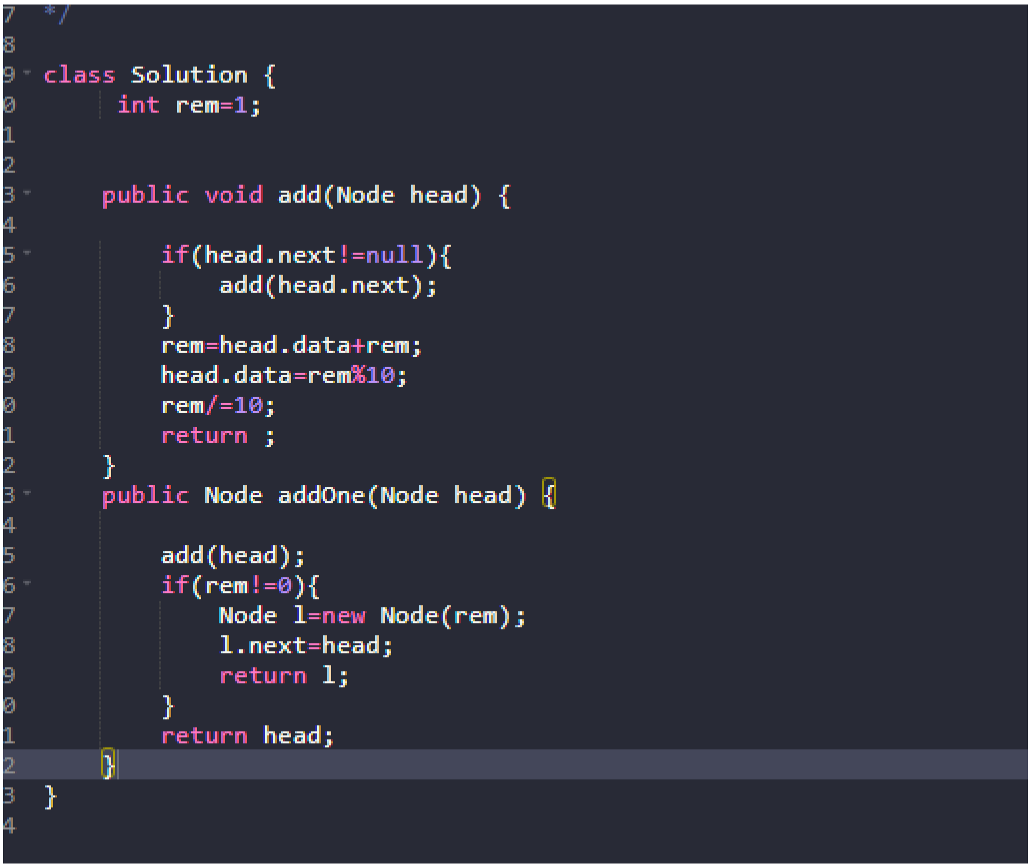
Examples :

Input: LinkedList: 4->5->6
Output: 457

Explanation: 4->5->6 represents 456 and when 1 is added it becomes 457.

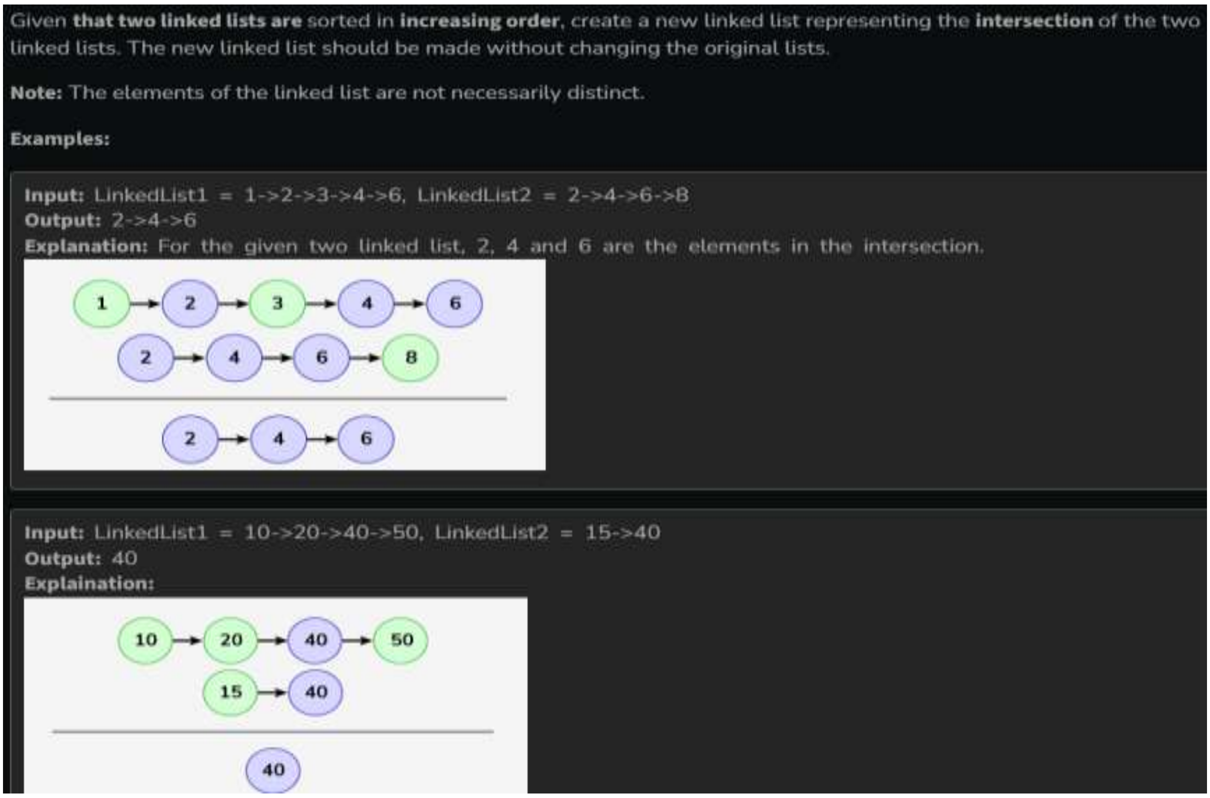
Input: LinkedList: 1->2->3
Output: 124

Explanation: 1->2->3 represents 123 and when 1 is added it becomes 124.



```java
 */

class Solution {
    int rem=1;


    public void add(Node head) {

        if(head.next!=null){
            add(head.next);
        }
        rem=head.data+rem;
        head.data=rem%10;
        rem/=10;
        return ;
    }
    public Node addOne(Node head) {

        add(head);
        if(rem!=0){
            Node l=new Node(rem);
            l.next=head;
            return l;
        }
        return head;
    }
}
```

30.Intersection of sorted Linked list



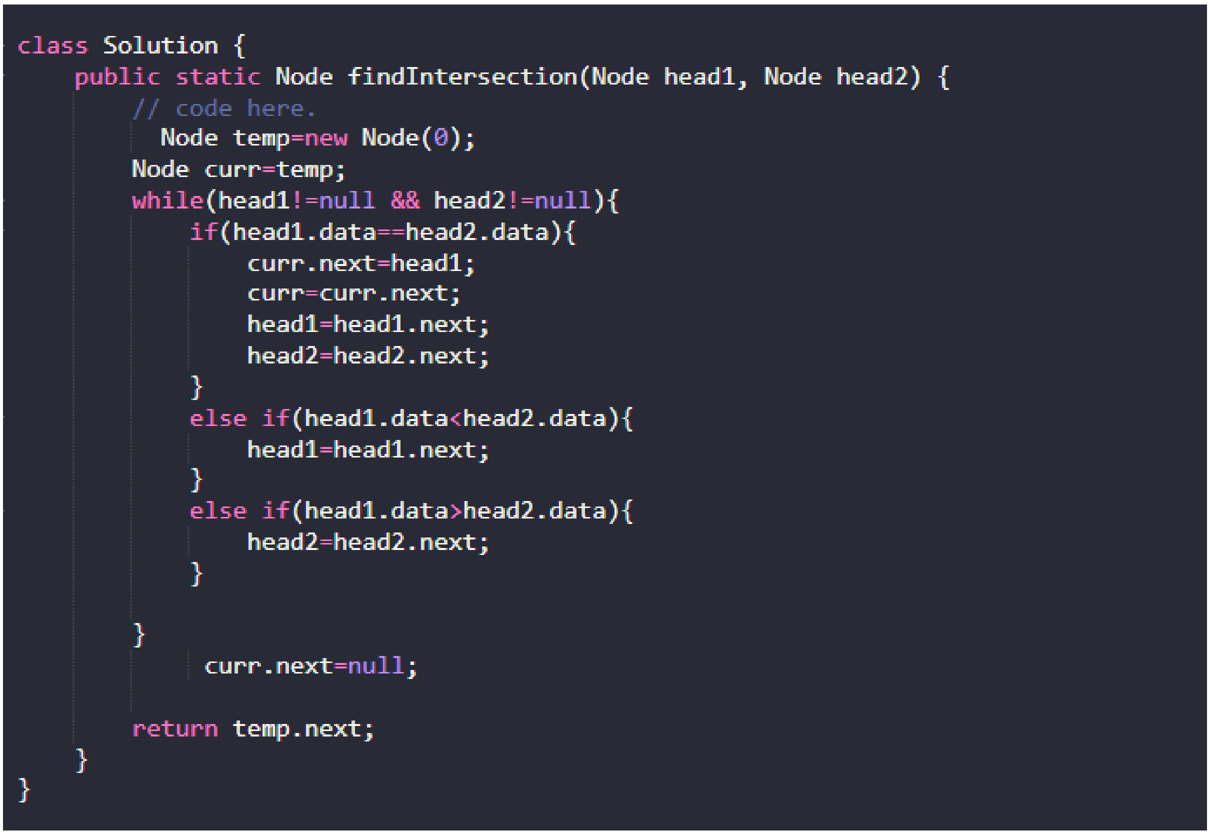
Given **that two linked lists are** sorted in **increasing order**, create a new linked list representing the **intersection** of the two linked lists. The new linked list should be made without changing the original lists.

Note: The elements of the linked list are not necessarily distinct.

Examples:

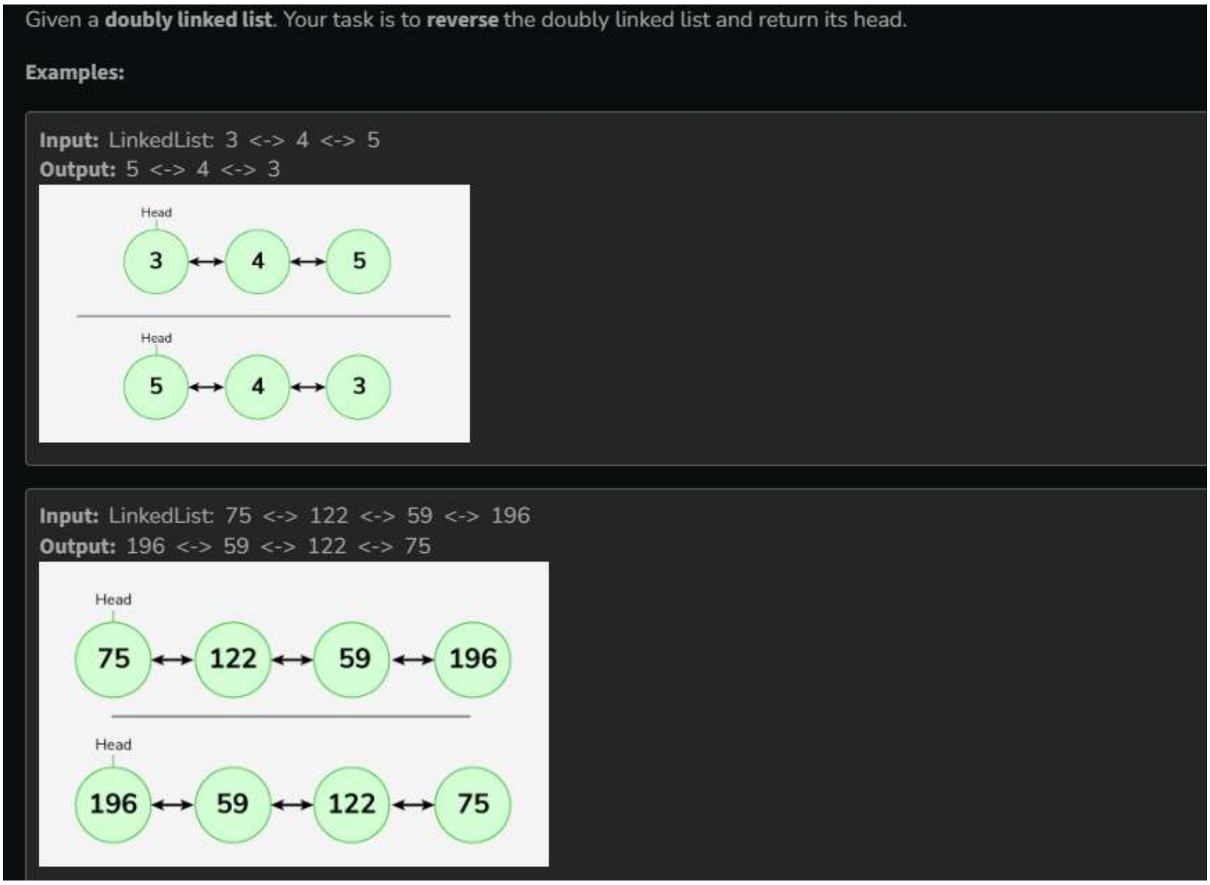
Input: LinkedList1 = 1->2->3->4->6, LinkedList2 = 2->4->6->8
Output: 2->4->6
Explanation: For the given two linked list, 2, 4 and 6 are the elements in the intersection.

Input: LinkedList1 = 10->20->40->50, LinkedList2 = 15->40
Output: 40
Explaination:



```
class Solution {
    public static Node findIntersection(Node head1, Node head2) {
        // code here.
          Node temp=new Node(0);
        Node curr=temp;
        while(head1!=null && head2!=null){
            if(head1.data==head2.data){
                curr.next=head1;
                curr=curr.next;
                head1=head1.next;
                head2=head2.next;
            }
            else if(head1.data<head2.data){
                head1=head1.next;
            }
            else if(head1.data>head2.data){
                head2=head2.next;
            }

        }
             curr.next=null;

        return temp.next;
    }
}
```

31.Reverse a Double Linked list

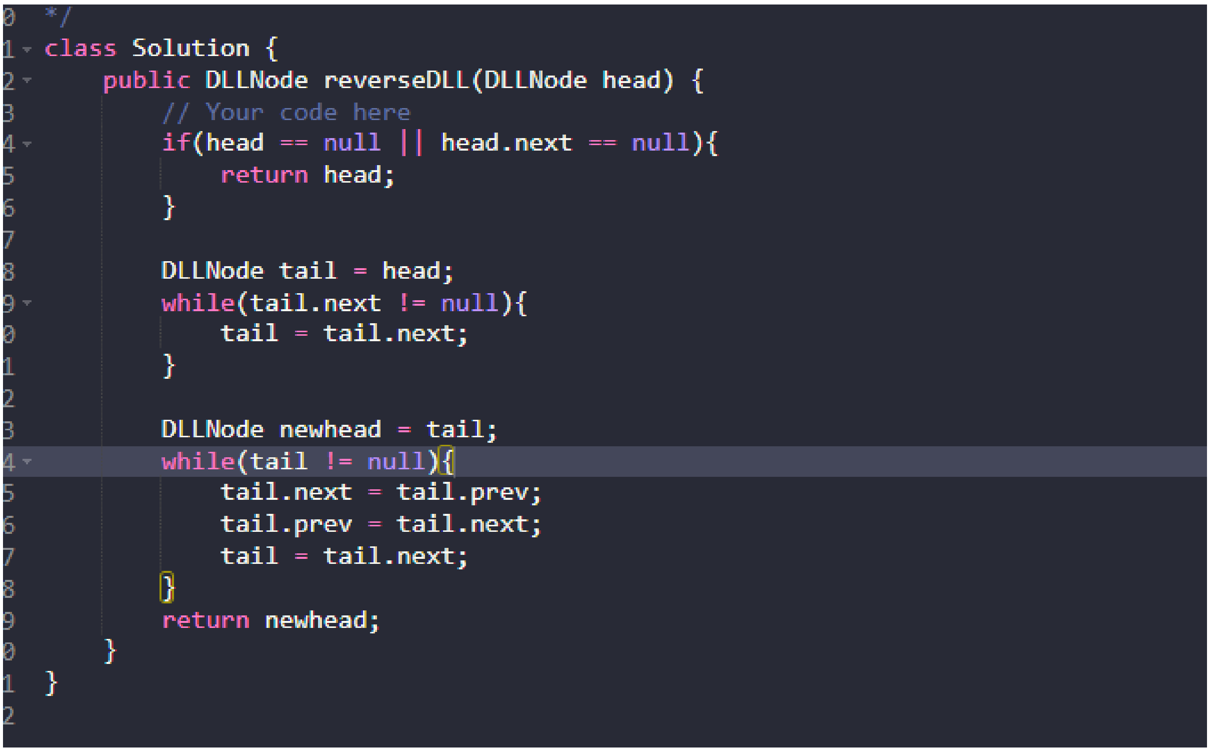


Given a **doubly linked list**. Your task is to **reverse** the doubly linked list and return its head.

Examples:

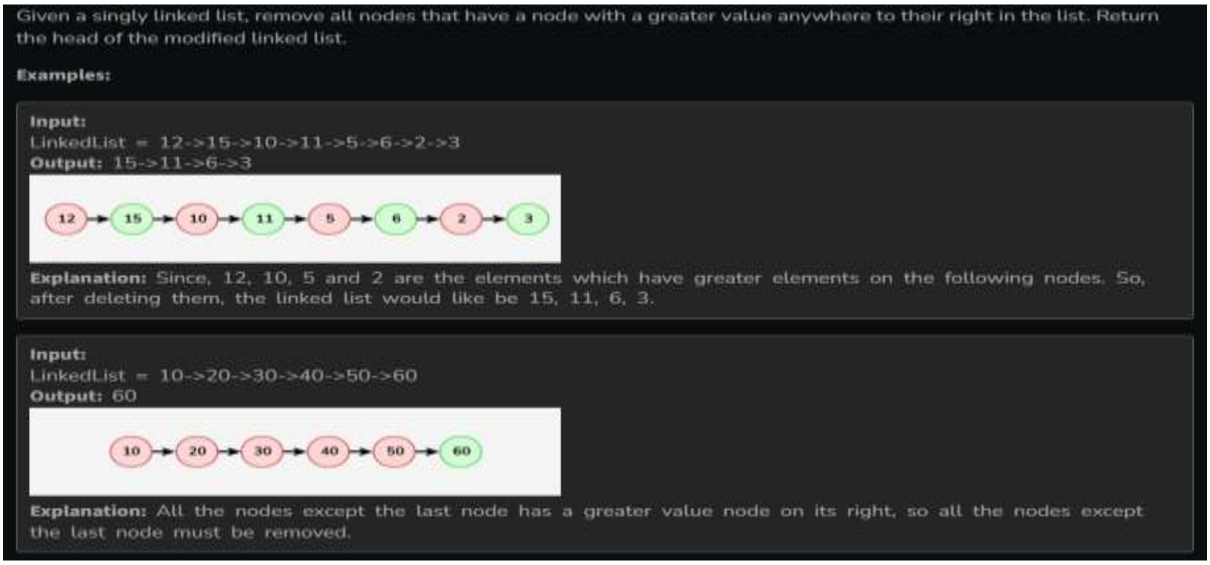
Input: LinkedList: 3 <-> 4 <-> 5
Output: 5 <-> 4 <-> 3

Input: LinkedList: 75 <-> 122 <-> 59 <-> 196
Output: 196 <-> 59 <-> 122 <-> 75



```java
 */
class Solution {
    public DLLNode reverseDLL(DLLNode head) {
        // Your code here
        if(head == null || head.next == null){
            return head;
        }

        DLLNode tail = head;
        while(tail.next != null){
            tail = tail.next;
        }

        DLLNode newhead = tail;
        while(tail != null){
            tail.next = tail.prev;
            tail.prev = tail.next;
            tail = tail.next;
        }
        return newhead;
    }
}
```

32.Delete node having greater value on right



Given a singly linked list, remove all nodes that have a node with a greater value anywhere to their right in the list. Return the head of the modified linked list.

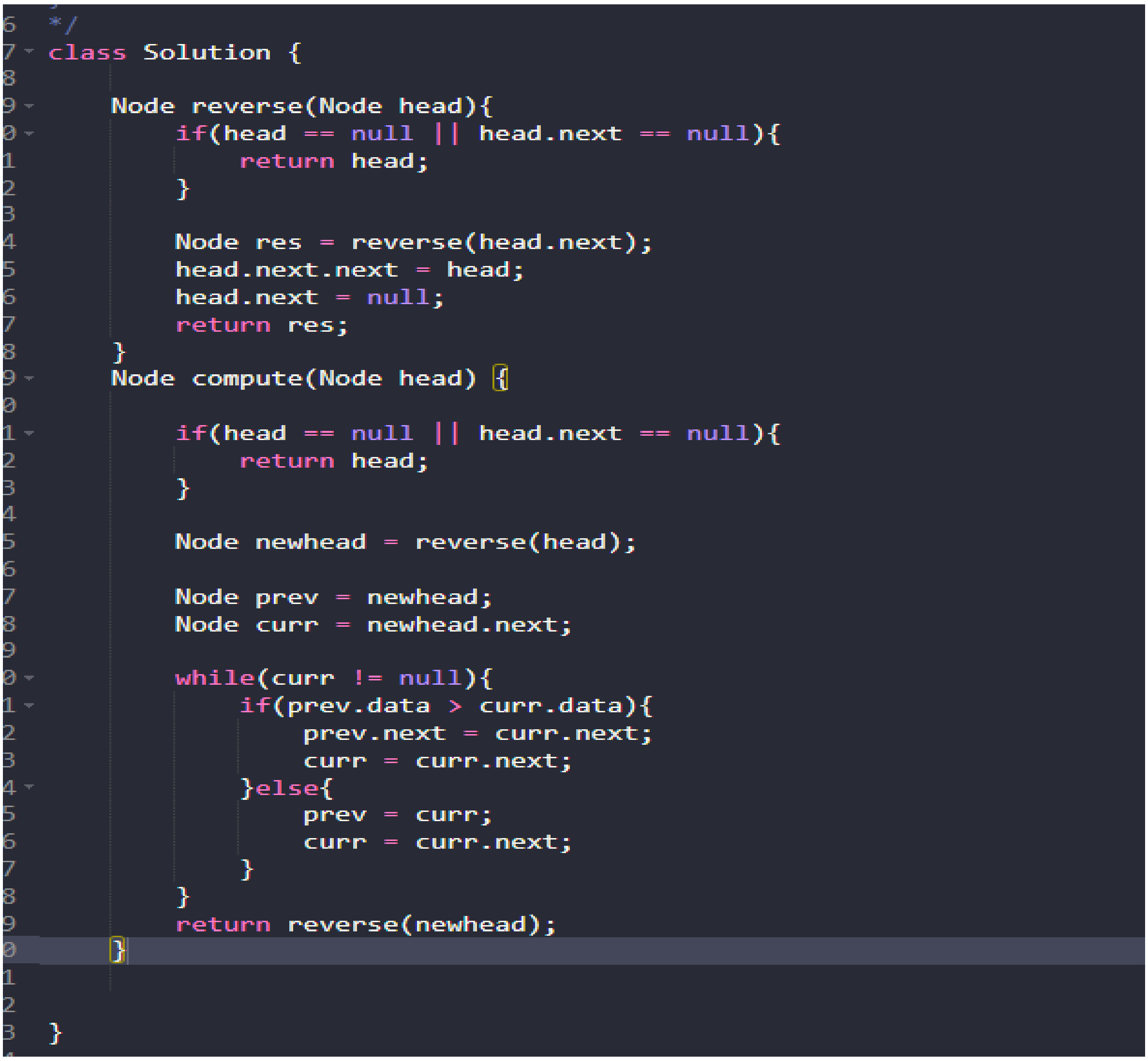
Examples:

Input:
LinkedList = 12->15->10->11->5->6->2->3
Output: 15->11->6->3

Explanation: Since, 12, 10, 5 and 2 are the elements which have greater elements on the following nodes. So, after deleting them, the linked list would like be 15, 11, 6, 3.

Input:
LinkedList = 10->20->30->40->50->60
Output: 60

Explanation: All the nodes except the last node has a greater value node on its right, so all the nodes except the last node must be removed.



```
 */
class Solution {

    Node reverse(Node head){
        if(head == null || head.next == null){
            return head;
        }

        Node res = reverse(head.next);
        head.next.next = head;
        head.next = null;
        return res;
    }
    Node compute(Node head) {

        if(head == null || head.next == null){
            return head;
        }

        Node newhead = reverse(head);

        Node prev = newhead;
        Node curr = newhead.next;

        while(curr != null){
            if(prev.data > curr.data){
                prev.next = curr.next;
                curr = curr.next;
            }else{
                prev = curr;
                curr = curr.next;
            }
        }
        return reverse(newhead);
    }


}
```