```
public class Striver_LinkedlList {
   //*****************************Linked List-
I************************************************************************** */
   //*****************************Reverse a
LinkedList********************************************************************* */
   class Solution {
      public ListNode reverseList(ListNode head) {
         ListNode curr=head;
         ListNode prev=null;
         while(curr!=null){
            ListNode temp=curr.next;
            curr.next=prev;
            prev=curr;
            curr=temp;
         }
         return prev;
      }
   }
   //*****************************Middle of a
LinkedList********************************************************************* */
   class Solution {
      public ListNode middleNode(ListNode head) {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null && fast.next!=null){
           slow=slow.next;
           fast=fast.next.next;
        }
         return slow;
      }
   }
   //*****************************Merge Two
LinkedLists******************************************************************** */
   class Solution {
      public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
         ListNode temp1=list1;
         ListNode temp2=list2;
         ListNode temp4=new ListNode(0);
         ListNode temp3=temp4;
         while(true){
            if(temp1==null){
               temp3.next=temp2;
               break;
            }
            if(temp2==null){
               temp3.next=temp1;
               break;
            }
            if(temp1.val<=temp2.val){
```

```
                temp3.next=temp1;
                temp1=temp1.next;
            }
            else{
                temp3.next=temp2;
                temp2=temp2.next;
            }
            temp3=temp3.next;
        }
        return temp4.next;
    }
}
//*************************Remove Nth Node from the End******************************************************* */

//*************************Add Two Numbers************************************************************************ */
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy=new ListNode();
        ListNode temp=dummy;
        int sum=0,carry=0;
        while(l1!=null|| l2!=null || carry==1){
            if(l1!=null){
                sum+=l1.val;
                l1=l1.next;
            }
            if(l2!=null){
                sum+=l2.val;
                l2=l2.next;
            }
            sum+=carry;
            carry=sum/10;
            ListNode h=new ListNode(sum%10);
            temp.next=h;
            temp=h;
            sum=0;
        }
        return dummy.next;
    }
}
//*************************Delete a Node************************************************************************** */
class Solution {
    public void deleteNode(ListNode node) {
        if (node == null) return;
        if (node.next != null) {
            int nextValue = node.next.val;
            node.next = node.next.next;
```

```
            node.val = nextValue;
          }
        }
    }


    //**************************Linked List-
II************************************************************************ */
    //*************************Find intersection point of Y
LinkedList ************************************/

      //*************************Detect a cycle in Linked
List*****************************************************/
        class Solution{
        static boolean cycleDetect(Node head) {
        if(head==null)
        return false;
        Node fast=head;
        Node slow=head;

        while(fast.next!=null&&fast.next.next!=null){
            fast=fast.next.next;
            slow=slow.next;
            if(fast==slow)
            return true;
        }
        return false;
    }
    }
    //*************************Reverse a LinkedList in groups of size
k************************************/

    //*************************Check if a LinkedList is palindrome or
not**********************************/
    class Solution {
        public boolean isPalindrome(ListNode head) {
         ListNode slow=head;
         ListNode slowprev=head;

         ListNode fast=head;
         while(fast!=null&&fast.next!=null){
             slowprev=slow;
             slow=slow.next;
             fast=fast.next.next;
         }
         ListNode revHead=reverse(slowprev);
         ListNode startF=head;
         ListNode tailB=revHead;
         while(startF!=null){
```

```
            if(startF.val!=tailB.val){
                return false;
            }
            else
            {
                startF=startF.next;
                tailB=tailB.next;
            }
        }
        return true;
    }
    public ListNode reverse(ListNode head){
        ListNode prev=null;
        ListNode nextN=head;
        ListNode curr=head;
        while(curr!=null){
            nextN=curr.next;
            curr.next=prev;
            prev=curr;
            curr=nextN;
        }
        return prev;
    }
}
//*************************Find the starting point of the Loop of LinkedList*****************************/
class Solution {
    public ListNode detectCycle(ListNode head) {
      ListNode slow=head;
      ListNode fast=head;
      int flag=0;
      while(fast!=null && fast.next!=null){
          slow=slow.next;
          fast=fast.next.next;
          if(slow==fast){
              flag=1;
              break;
          }
      }
      if(flag==0){
          return null;
      }
      ListNode first=head;
      ListNode second=slow;
      while(first!=second){
          first=first.next;
          second=second.next;
      }
      return first;
```

```
        }
    }
    //*************************Flattening of a
LinkedList************************************************** */
    class GfG
    {
        class Node
    {
        int data;
        Node next;
        Node bottom;

        Node(int d)
        {
            data = d;
            next = null;
            bottom = null;
        }
    }
        Node flatten(Node root){
                if (root == null || root.next == null)
                    return root;
                root.next = flatten(root.next);
                root = mergeTwoLists(root, root.next);

                // return the root
                // it will be in turn merged with its left
                return root;
        }
         Node mergeTwoLists(Node a, Node b) {

            Node temp = new Node(0);
            Node res = temp;

            while(a != null && b != null) {
                if(a.data < b.data) {
                    temp.bottom = a;
                    temp = temp.bottom;
                    a = a.bottom;
                }
                else {
                    temp.bottom = b;
                    temp = temp.bottom;
                    b = b.bottom;
                }
            }

            if(a != null) temp.bottom = a;
            else temp.bottom = b;
```

```
            return res.bottom;
        }
    }
}
```