Assignment Questions 14

Given a linked list of N nodes such that it may contain a loop.
A loop here means that the last node of the link list is connected to the node at position X(1-based index).
If the link list does not have any loop, X=0.
Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.
Example 1:
Input:
N = 3
value[] = {1,3,4}
X = 2
Output:1
Explanation:The link list looks like

```
1 -> 3 -> 4
     ^    |
     |____|
```

A loop is present. If you remove it
successfully, the answer will be 1.
Example 2:
Input:
N = 4
value[] = {1,8,3,4}
X = 0
Output:1
Explanation:The Linked list does not
contains any loop.
Example 3:
Input:
N = 4
value[] = {1,2,3,4}
X = 1
Output:1
Explanation:The link list looks like

```
1 -> 2 -> 3 -> 4
^              |
|______________|
```

A loop is present.
If you remove it successfully,
the answer will be 1.

code:-

```
class Solution
{
    public static void removeLoop(Node head){
        Node fast = head.next;
        Node slow = head;
        while( fast!=slow )
        {
            if( fast==null || fast.next==null )
            return;
            fast = fast.next.next;
            slow = slow.next;
        }
        int size = 1;
        fast = fast.next;
        while( fast!=slow )
        {
            fast = fast.next;
            size+=1;
        }
        slow = head;
        fast = head;
        for(int i=0; i<size-1; i++)
            fast = fast.next;
        while( fast.next != slow )
        {
            fast = fast.next;
            slow = slow.next;
        }
        fast.next = null;
    }
}
```

Question 2
A number N is represented in Linked List such that each digit corresponds to a node in linked list. You ne ed to add 1 to it.
Example 1:
Input:
LinkedList: 4->5->6
Output:457
Example 2:
Input:
LinkedList: 1->2->3
Output:124

code:-
```
class Solution
{
    public:

    Node* reverse(Node *head)
    {
        Node *prev = NULL;
        Node *current = head;
        Node *next;

        while(current != NULL)
        {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }

    Node* addOne(Node *head)
    {
        head = reverse(head);
        Node* current = head;
        int carry = 1;

        while(carry)
        {
            current->data += 1;

            if(current->data < 10)
                return reverse(head);

            else
                current->data = 0;

            if(current->next == NULL)
                break;

            else
                current = current->next;
        }
        current->next = new Node(1);
        return reverse(head);
    }
};
```

Question 3

Given a Linked List of size N, where every node represents a sub-linked-list and contains two pointers:(i) a next pointer to the next node,(ii) a bottom pointer to a linked list where this node is head.Each of the sub-linked-list is in sorted order.Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order. Note: The flattened list will be printed using the bottom pointer instead of next pointer.

Example 1:

Input:

```
5 -> 10 -> 19 -> 28
|    |     |     |
7    20    22   35
|          |     |
8          50    40
|                |
30               45
```

Output: 5-> 7-> 8- > 10 -> 19-> 20->
22-> 28-> 30-> 35-> 40-> 45-> 50.

Explanation:

The resultant linked lists has every
node in a single level.(Note:| represents the bottom pointer.)

Example 2:

Input:

```
5 -> 10 -> 19 -> 28
|          |
7          22
|          |
8          50
|
30
```

Output: 5->7->8->10->19->22->28->30->50

Explanation:
The resultant linked lists has every
node in a single level.

(Note:| represents the bottom pointer.)

code:-
```
class LinkedList {
 Node head; // head of list
 class Node {
  int data;
  Node right, down;
  Node(int data)
  {
   this.data = data;
   right = null;
   down = null;
  }
 }
 Node merge(Node a, Node b)
 {
  if (a == null)
   return b;
  if (b == null)
   return a;
  Node result;
  if (a.data < b.data) {
   result = a;
   result.down = merge(a.down, b);
  }
  else {
   result = b;
   result.down = merge(a, b.down);
  }
  result.right = null;
  return result;
 }

 Node flatten(Node root)
 {
  if (root == null || root.right == null)
   return root;

  // recur for list on right
  root.right = flatten(root.right);

  // now merge
  root = merge(root, root.right);
  return root;
 }
 Node push(Node head_ref, int data)
 {
  Node new_node = new Node(data);
  new_node.down = head_ref;
  head_ref = new_node;
  return head_ref;
 }

 void printList()
 {
  Node temp = head;
  while (temp != null) {
   System.out.print(temp.data + " ");
   temp = temp.down;
  }
  System.out.println();
 }

 public static void main(String args[])
 {
  LinkedList L = new LinkedList();
  L.head = L.push(L.head, 30);
  L.head = L.push(L.head, 8);
  L.head = L.push(L.head, 7);
  L.head = L.push(L.head, 5);
  L.head.right = L.push(L.head.right, 20);
  L.head.right = L.push(L.head.right, 10);
  L.head.right.right = L.push(L.head.right.right, 50);
  L.head.right.right = L.push(L.head.right.right, 22);
  L.head.right.right = L.push(L.head.right.right, 19);
  L.head.right.right.right= L.push(L.head.right.right.right, 45);
  L.head.right.right.right= L.push(L.head.right.right.right, 40);
  L.head.right.right.right= L.push(L.head.right.right.right, 35);
  L.head.right.right.right= L.push(L.head.right.right.right, 20);
  L.head = L.flatten(L.head);
  L.printList();
 }
}
```

Question 4.
You are given a special linked list with N nodes where each node has a next pointer pointing to its next no de. You are also given M random pointers, where you will be given M number of pairs denoting two nodes a and b i.e. a->arb = b (arb is pointer to random node).
Construct a copy of the given list. The copy should consist of exactly N new nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copie d list represent the same list state. None of the pointers in the new list should point to nodes in the original list.
For example, if there are two nodes X and Y in the original list, where X.arb--> Y, then for the correspondi ng two nodes x and y in the copied list, x.arb --> y.
Return the head of the copied linked list.

Example 1:
Input:
N = 4, M = 2
value = {1,2,3,4}
pairs = {{1,2},{2,4}}

Output:1
Explanation:In this test case, there are 4 nodes in linked list. Among these 4 nodes, 2 nodes have arbitrary pointer set, rest two nodes have arbitrary pointer as NULL. Second line tells us the value of four nodes. The third line gives the information about arbitrary pointers.
The first node arbitrary pointer is set to node 2. The second node arbitrary pointer is set to node 4.
Example 2:
Input:
N = 4, M = 2
value[] = {1,3,5,9}
pairs[] = {{1,1},{3,4}}
Output:1
Explanation:In the given testcase , applying the method as stated in the above example, the output will be 1.

code:-

```
class Node {
int val;
Node next;
Node arbit;
Node(int x)
{
 this.val = x;
 this.next = null;
 this.arbit = null;
}
}
class GFG {

static Node cloneLinkedList(Node head)
{
 HashMap<Node, Node> mp = new HashMap<>();
 Node temp, nhead;
 temp = head;
 nhead = new Node(temp.val);
 mp.put(temp, nhead);
 while (temp.next != null) {
 nhead.next = new Node(temp.next.val);
 temp = temp.next;
 nhead = nhead.next;
 mp.put(temp, nhead);
 }
 temp = head;
 while (temp != null) {
 mp.get(temp).arbit = mp.get(temp.arbit);
 temp = temp.next;
 }
 return mp.get(head);
}

static void printList(Node head)
{
 System.out.print(head.val + "(" + head.arbit.val+ ")");
 head = head.next;
 while (head != null) {
 System.out.print(" -> " + head.val + "("+ head.arbit.val + ")");
 head = head.next;
 }
 System.out.println();
}

public static void main(String[] args)
{
 Node head = new Node(1);
 head.next = new Node(2);
 head.next.next = new Node(3);
 head.next.next.next = new Node(4);
 head.next.next.next.next = new Node(5);
 head.arbit = head.next.next;
 head.next.arbit = head;
 head.next.next.arbit = head.next.next.next.next;
 head.next.next.next.arbit = head.next.next;
 head.next.next.next.next.arbit = head.next;
 System.out.println("The original linked list:");
 printList(head);
 Node sol = cloneLinkedList(head);
 System.out.println("The cloned linked list:");
 printList(sol);
}
}
```

Question 5.
Given the ‘head‘ of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.
The first node is considered odd, and the second node is even, and so on.
Note that the relative order inside both the even and odd groups should remain as it was in the input.
You must solve the problem in ‘O(1)‘ extra space complexity and ‘O(n)‘ time complexity.

Example 1:
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
Example 2:
Input: head = [2,1,3,5,6,4,7]
Output: [2,3,6,7,1,5,4]

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 oddEvenList(ListNode head) {
    ListNode es=null,ee=null,os=null,oe = null;
    // es->even start  ee->even end  os->odd start  oe->odd end
    int count=0;
    ListNode curr = head;
    while(curr!=null){
            count++;
            if(count%2 != 0){  if(os==null) {os = oe = curr;}
            else  {oe.next = curr;  oe = oe.next; }
    }
    else{
        if(es==null){
        es = ee = curr;}
        else{
        ee.next = curr;
        ee = ee.next;
    }}
    curr = curr.next;
    }
    if(os==null || es==null)   return head;
    oe.next = es;
    ee.next = null;
    return os;
    }
}
```

Question 6.
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 pos
itive integer smaller than or equal to length of the linked list.
Example 1:
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
Example 2:
Input:
N = 8
value[] = {1, 2, 3, 4, 5, 6, 7, 8}
k = 4
Output:5 6 7 8 1 2 3 4

code:-
```
class LinkedList {
 Node head; // head of list
 class Node {
```

```
 int data;
 Node next;
 Node(int d)
 {
  data = d;
  next = null;
 }
}
void rotate(int k)
{
 if (k == 0)
  return;
 Node current = head;
 int count = 1;
 while (count < k && current != null) {
  current = current.next;
  count++;
 }
 if (current == null)
  return;
 Node kthNode = current;
 while (current.next != null)
  current = current.next;

 current.next = head;
 head = kthNode.next;
 kthNode.next = null;
}
void push(int new_data)
{
 Node new_node = new Node(new_data);
 new_node.next = head;
 head = new_node;
}

void printList()
{
 Node temp = head;
 while (temp != null) {
  System.out.print(temp.data + " ");
  temp = temp.next;
 }
 System.out.println();
}
public static void main(String args[])
{
 LinkedList llist = new LinkedList();
 for (int i = 60; i >= 10; i -= 10)
  llist.push(i);

 System.out.println("Given list");
 llist.printList();
 llist.rotate(4);
 System.out.println("Rotated Linked List");
 llist.printList();
 }
}
```

Question 7.
You are given the ‘head‘ of a linked list with ‘n‘ nodes.
For each node in the list, find the value of the next greater node**. That is, for each node, find the value of the first node that is next to it and has a strictly larger value than it.
Return an integer array ‘answer‘ where ‘answer[i]‘ is the value of the next greater node of the ‘ith‘ node (** 1-indexed**). If the ‘ith‘ node does not have a next greater node, set ‘answer[i] = 0‘.

Example 1:
Input: head = [2,1,5]
Output: [5,5,0]
Example 2:
Input: head = [2,7,4,3,5]
Output: [7,0,5,5,0]

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 int[] nextLargerNodes(ListNode head) {
        ArrayList<Integer> l = new ArrayList<>();
        int n=0;
        while(head!=null){
            n++;
            l.add(head.val);
            head=head.next;
        }

        int[] ans = new int[n];
        Stack<Integer> s = new Stack<>();
        for(int i=l.size()-1;i>=0;i--){
            while(!s.isEmpty() && s.peek()<=l.get(i)){
                s.pop();
            }
            if(s.isEmpty()) ans[i]=0;
            else ans[i]=s.peek();
            s.push(l.get(i));
        }

        return ans;

    }
}
```

Question 8.

Given the ‘head‘ of a linked list, we repeatedly delete consecutive sequences of nodes that sum to ‘0‘ until there are no such sequences.
After doing so, return the head of the final linked list. You may return any such answer.
(Note that in the examples below, all sequences are serializations of ‘ListNode‘ objects.)

Example 1:
Input: head = [1,2,-3,3,1]
Output: [3,1]
Note: The answer [1,2,1] would also be accepted.
Example 2:
Input: head = [1,2,3,-3,4]
Output: [1,2,4]
Example 3:
Input: head = [1,2,3,-3,-2]
Output: [1]

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 removeZeroSumSublists(ListNode head) {
        int sum = 0;
    ListNode dm = new ListNode(0);
    dm.next = head;

    Map<Integer, ListNode> mp = new HashMap<>();
    mp.put(0, dm);


    for (ListNode i = dm; i != null; i = i.next) {
        sum += i.val;
        mp.put(sum, i);
    }



    sum = 0;
    for (ListNode i = dm; i != null; i = i.next) {
        sum += i.val;
        i.next = mp.get( sum ).next;
    }

    return dm.next;
    }
}
```