Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes

Given two sorted linked lists, construct a linked list that contains maximum sum path from start to end. The result list may contain nodes from both input lists. When constructing the result list, we may switch to the other input list only at the point of intersection (which mean the two node with the same value in the lists). You are allowed to use O(1) extra space.

```
Input:
List1 =  1->3->30->90->120->240->511
List2 =  0->3->12->32->90->125->240->249

Output: Following is maximum sum linked list out of two input lists
list =  1->3->12->32->90->125->240->511
we switch at 3 and 240 to get above maximum sum linked list
```

We strongly recommend to minimize the browser and try this yourself first.

The idea here in the below solution is to adjust next pointers after common nodes.
1. Start with head of both linked lists and find first common node. Use merging technique of sorted linked list for that.
2. Keep track of sum of the elements too while doing this and set head of result list based on greater sum till first common node.
3. After this till the current pointers of both lists don't become NULL we need to adjust the next of prev pointers based on greater sum.

This way it can be done in-place with constant extra space.
Time complexity of the below solution is O(n).

C++

```
// C++ program to construct the maximum sum linked
// list out of two given sorted lists
#include<iostream>
using namespace std;

//A linked list node
struct Node
{
    int data; //data belong to that node
    Node *next; //next pointer
};

// Push the data to the head of the linked list
void push(Node **head, int data)
{
    //Alocation memory to the new node
    Node *newnode = new Node;

    //Assigning data to the new node
    newnode->data = data;

    //Adjusting next pointer of the new node
    newnode->next = *head;

    //New node becomes the head of the list
    *head = newnode;
}

// Method that adjusts the pointers and prints the final list
void finalMaxSumList(Node *a, Node *b)
{
    Node *result = NULL;

    // Assigning pre and cur to the head of the
    // linked list
    // linked list.
    Node *pre1 = a, *curr1 = a;
    Node *pre2 = b, *curr2 = b;

    // Till either of the current pointers is not
    // NULL execute the loop
    while (curr1 != NULL || curr2 != NULL)
    {
        // Keeping 2 local variables at the start of every
        // loop run to keep track of the sum between pre
        // and cur pointer elements.
        int sum1 = 0, sum2 = 0;

        // Calculating sum by traversing the nodes of linked
        // list as the merging of two linked list.  The loop
        // stops at a common node
        while (curr1!=NULL && curr2!=NULL && curr1->data!=curr2->data)
        {
            if (curr1->data < curr2->data)
            {
                sum1 += curr1->data;
                curr1 = curr1->next;
            }
            else // (curr2->data < curr1->data)
            {
                sum2 += curr2->data;
                curr2 = curr2->next;
            }
        }

        // If either of current pointers becomes NULL
        // carry on the sum calculation for other one.
        if (curr1 == NULL)
        {
            while (curr2 != NULL)
            {
                sum2 += curr2->data;
                curr2 = curr2->next;
            }
        }
        if (curr2 == NULL)
        {
            while (curr1 != NULL)
            {
                sum1 += curr1->data;
                curr1 = curr1->next;
            }
        }

        // First time adjustment of resultant head based on
        // the maximum sum.
        if (pre1 == a && pre2 == b)
            result = (sum1 > sum2)? pre1 : pre2;

        // If pre1 and pre2 don't contain the head pointers of
        // lists adjust the next pointers of previous pointers.
        else
        {
            if (sum1 > sum2)
                pre2->next = pre1->next;
            else
                pre1->next = pre2->next;
        }

        // Adjusting previous pointers
        pre1 = curr1, pre2 = curr2;

        // If curr1 is not NULL move to the next.
        if (curr1)
            curr1 = curr1->next;
        // If curr2 is not NULL move to the next.
        if (curr2)
            curr2 = curr2->next;
    }
    }

    // Print the resultant list.
    while (result != NULL)
    {
        cout << result->data << " ";
        result = result->next;
    }
}

//Main driver program
int main()
{
    //Linked List 1 : 1->3->30->90->110->120->NULL
    //Linked List 2 : 0->3->12->32->90->100->120->130->NULL
    Node *head1 = NULL, *head2 = NULL;
    push(&head1, 120);
    push(&head1, 110);
    push(&head1, 90);
    push(&head1, 30);
    push(&head1, 3);
    push(&head1, 1);

    push(&head2, 130);
    push(&head2, 120);
    push(&head2, 100);
    push(&head2, 90);
    push(&head2, 32);
    push(&head2, 12);
    push(&head2, 3);
    push(&head2, 0);

    finalMaxSumList(head1, head2);
    return 0;
}
```

Java

```
// Java program to construct a Maximum Sum Linked List out of
// two Sorted Linked Lists having some Common nodes
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    // Method to adjust pointers and print final list
    void finalMaxSumList(Node a, Node b)
    {
        Node result = null;

        /* assigning pre and cur to head
           of the linked list */
        Node pre1 = a, curr1 = a;
        Node pre2 = b, curr2 = b;

        /* Till either of current pointers is not null
           execute the loop */
        while (curr1 != null || curr2 != null)
        {
            // Keeping 2 local variables at the start of every
            // Keeping 2 local variables at the start of every
            // loop run to keep track of the sum between pre
            // and cur reference elements.
            int sum1 = 0, sum2 = 0;

            // Calculating sum by traversing the nodes of linked
            // list as the merging of two linked list.  The loop
            // stops at a common node
            while (curr1 != null && curr2 != null &&
                   curr1.data != curr2.data)
            {

                if (curr1.data<curr2.data)
                {
                    sum1 += curr1.data;
                    curr1 = curr1.next;
                }
                else
                {
                    sum2 += curr2.data;
                    curr2 = curr2.next;
                }
            }

            // If either of current pointers becomes null
            // carry on the sum calculation for other one.
            if (curr1 == null)
            {
                while (curr2 != null)
                {
                    sum2 += curr2.data;
                    curr2 = curr2.next;
                }
            }
            if (curr2 == null)
            {
                while(curr1 != null)
                {
                    sum1 += curr1.data;
                    curr1 = curr1.next;
                }
            }

            // First time adjustment of resultant head based on
            // the maximum sum.
            if (pre1 == a && pre2 == b)
                result = (sum1 > sum2) ? pre1 : pre2;

            // If pre1 and pre2 don't contain the head refernces of
            // lists adjust the next pointers of previous pointers.
            else
            {
                if (sum1 > sum2)
                    pre2.next = pre1.next;
                else
                    pre1.next = pre2.next;
            }

            // Adjusting previous pointers
            pre1 = curr1;
            pre2 = curr2;

            // If curr1 is not NULL move to the next.
            if (curr1 != null)
                curr1 = curr1.next;

            // If curr2 is not NULL move to the next.
            if (curr2 != null)
                curr2 = curr2.next;
        }

        while (result != null)
        {
            System.out.print(result.data + " ");
            result = result.next;
        }
        System.out.println();
    }

    /*  Inserts a node at start of linked list */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }


    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();

        //Linked List 1 : 1->3->30->90->110->120->NULL
        //Linked List 2 : 0->3->12->32->90->100->120->130->NULL

        llist1.push(120);
        llist1.push(110);
        llist1.push(90);
        llist1.push(30);
        llist1.push(3);
        llist1.push(1);

        llist2.push(130);
        llist2.push(120);
        llist2.push(100);
        llist2.push(90);
        llist2.push(32);
        llist2.push(12);
        llist2.push(3);
        llist2.push(0);

        llist1.finalMaxSumList(llist1.head, llist2.head);
    }
} /* This code is contributed by Rajat Mishra */
```

Python

```
# Python program to construct a Maximum Sum Linked List out of
# two Sorted Linked Lists having some Common nodes
class LinkedList(object):
    def __init__(self):
     # head of list
         self.head = None

    # Linked list Node
    class Node(object):
        def __init__(self, d):
            self.data = d
            self.next = None

    # Method to adjust pointers and print final list
    def finalMaxSumList(self, a, b):
        result = None
        # assigning pre and cur to head
        # of the linked list
        pre1 = a
        curr1 = a
        pre2 = b
        curr2 = b
        # Till either of current pointers is not null
        # execute the loop
        while curr1 != None or curr2 != None:
            # Keeping 2 local variables at the start of every
            # loop run to keep track of the sum between pre
            # and cur reference elements.
            sum1 = 0
            sum2 = 0
            # Calculating sum by traversing the nodes of linked
            # list as the merging of two linked list.  The loop
            # stops at a common node
            while curr1 != None and curr2 != None and curr1.data != curr2.data:
                if curr1.data < curr2.data:
                    sum1 += curr1.data
                    curr1 = curr1.next
                else:
                    sum2 += curr2.data
                    curr2 = curr2.next
            # If either of current pointers becomes null
            # carry on the sum calculation for other one.
            if curr1 == None:
                while curr2 != None:
                    sum2 += curr2.data
                    curr2 = curr2.next
            if curr2 == None:
                while curr1 != None:
                    sum1 += curr1.data
                    curr1 = curr1.next
            # First time adjustment of resultant head based on
            # the maximum sum.
            if pre1 == a and pre2 == b:
                result = pre1 if (sum1 > sum2) else pre2
            else:
                # If pre1 and pre2 don't contain the head refernces of
                # lists adjust the next pointers of previous pointers.
                if sum1 > sum2:
                    pre2.next = pre1.next
                else:
                    pre1.next = pre2.next
            # Adjusting previous pointers
            pre1 = curr1
            pre2 = curr2
            # If curr1 is not NULL move to the next.
            if curr1 != None:
                curr1 = curr1.next
            # If curr2 is not NULL move to the next.
            if curr2 != None:
                curr2 = curr2.next

        while result != None:
            print str(result.data),
            result = result.next
        print ''

    # Utility functions
    # Inserts a new Node at front of the list.
    def push(self, new_data):
        # 1 & 2: Allocate the Node &
        # Put in the data
        new_node = self.Node(new_data)
        # 3. Make next of new Node as head
        new_node.next = self.head
        # 4. Move the head to point to new Node
        self.head = new_node

# Driver program
llist1 = LinkedList()
llist2 = LinkedList()
```

```
# Linked List 1 : 1->3->30->90->110->120->NULL
# Linked List 2 : 0->3->12->32->90->100->120->130->NULL

llist1.push(120)
llist1.push(110)
llist1.push(90)
llist1.push(30)
llist1.push(3)
llist1.push(1)

llist2.push(130)
llist2.push(120)
llist2.push(100)
llist2.push(90)
llist2.push(32)
llist2.push(12)
llist2.push(3)
llist2.push(0)

llist1.finalMaxSumList(llist1.head, llist2.head)

# This code is contributed by BHAVYA JAIN
```

Output:

```
1 3 12 32 90 110 120 130
```

Time complexity = O(n) where n is the length of bigger linked list
Auxiliary space = O(1)
However a problem in this solution is that the original lists are changed.

Exercise
1. Try this problem when auxiliary space is not a constraint.
2. Try this problem when we don’t modify the actual list and create the resultant list.