In-place Merge two linked lists without changing links of first list

Given two sorted singly linked lists having n and m elements each, merge them using constant space. First n smallest elements in both the lists should become part of first list and rest elements should be part of second list. Sorted order should be maintained. We are not allowed to change pointers of first linked list.

For example,

```
Input:
First List: 2->4->7->8->10
Second List: 1->3->12

Output:
First List: 1->2->3->4->7
Second List: 8->10->12
```

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

The problem becomes very simple if we're allowed to change pointers of first linked list. If we are allowed to change links, we can simply do something like merge of merge-sort algorithm. We assign first n smallest elements to the first linked list where n is the number of elements in first linked list and the rest to second linked list. We can achieve this in O(m + n) time and O(1) space, but this solution violates the requirement that we can't change links of first list.

The problem becomes a little tricky as we're not allowed to change pointers in first linked list. The idea is something similar to this post but as we are given singly linked list, we can't proceed backwards with the last element of LL2.

The idea is for each element of LL1, we compare it with first element of LL2. If LL1 has a greater element than first element of LL2, then we swap the two elements involved. To keep LL2 sorted, we need to place first element of LL2 at its correct position. We can find mismatch by traversing LL2 once and correcting the pointers.

Below is C++ implementation of this idea.

```
// Program to merge two sorted linked lists without
// using any extra space and without changing links
// of first list
#include <bits/stdc++.h>
using namespace std;

 /* Structure for a linked list node */
struct node
{
    int data;
    struct node *next;
};

/* Given a reference (pointer to pointer) to the head
  of a list and an int, push a new node on the front
  of the list. */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}
```

```
// Function to merge two sorted linked lists
// LL1 and LL2 without using any extra space.
void mergeLists(struct node *a, struct node * &b)
{
    // run till either one of a or b runs out
    while (a && b)
    {
        // for each element of LL1,
        // compare it with first element of LL2.
        if (a->data > b->data)
        {
            // swap the two elements involved
            // if LL1 has a greater element
            swap(a->data, b->data);

            struct node *temp = b;

            // To keep LL2 sorted, place first
            // element of LL2 at its correct place
            if (b->next && b->data > b->next->data)
            {
                b = b->next;
                struct node *ptr= b, *prev = NULL;

                // find mismatch by traversing the
                // second linked list once
                while (ptr && ptr->data < temp->data)
                {
                    prev = ptr;
                    ptr = ptr -> next;
                }

                // correct the pointers
                prev->next = temp;
                temp->next = ptr;
            }
        }

        // move LL1 pointer to next element
        a = a->next;
    }
}

// Code to print the linked link
void printList(struct node *head)
{
    while (head)
    {
        cout << head->data << "->" ;
        head = head->next;
    }
    cout << "NULL" << endl;
}

// Driver code
int main()
{
    struct node *a = NULL;
    push(&a, 10);
    push(&a, 8);
    push(&a, 7);
    push(&a, 4);
    push(&a, 2);

    struct node *b = NULL;
    push(&b, 12);
    push(&b, 3);
    push(&b, 1);

    mergeLists(a, b);

    cout << "First List: ";
    printList(a);

    cout << "Second List: ";
    printList(b);

    return 0;
}
```

Output :

```
First List: 1->2->3->4->7->NULL
Second List: 8->10->12->NULL
```

Time Complexity : O(mn)