Merge two sorted linked lists such that merged list is in reverse order

Given two linked lists sorted in increasing order. Merge them such a way that the result list is in decreasing order (reverse order).

Examples:

```
Input:  a: 5->10->15->40
        b: 2->3->20
Output: res: 40->20->15->10->5->3->2

Input:  a: NULL
        b: 2->3->20
Output: res: 20->3->2
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A **Simple Solution** is to do following.
1) Reverse first list 'a'.
2) Reverse second list 'b'.
3) Merge two reversed lists.

Another Simple Solution is first Merge both lists, then reverse the merged list.

Both of the above solutions require two traversals of linked list.

How to solve without reverse, O(1) auxiliary space (in-place) and only one traversal of both lists?
The idea is to follow merge style process. Initialize result list as empty. Traverse both lists from beginning to end. Compare current nodes of both lists and insert smaller of two at the beginning of the result list.

```
1) Initialize result list as empty: res = NULL.
2) Let 'a' and 'b' be heads first and second lists respectively.
3) While (a != NULL and b != NULL)
    a) Find the smaller of two (Current 'a' and 'b')
    b) Insert the smaller value node at the front of result.
    c) Move ahead in the list of smaller node.
4) If 'b' becomes NULL before 'a', insert all nodes of 'a'
   into result list at the beginning.
5) If 'a' becomes NULL before 'b', insert all nodes of 'a'
   into result list at the beginning.
```

Below is the implementation of above solution.

C/C++

```
/* Given two sorted non-empty linked lists. Merge them in
   such a way that the result list will be in reverse
   order. Reversing of linked list is not allowed. Also,
   extra space should be O(1) */
#include<iostream>
using namespace std;

/* Link list Node */
struct Node
{
    int key;
    struct Node* next;
};
```

```
// Given two non-empty linked lists 'a' and 'b'
Node* SortedMerge(Node *a, Node *b)
{
    // If both lists are empty
    if (a==NULL && b==NULL) return NULL;

    // Initialize head of resultant list
    Node *res = NULL;

    // Traverse both lists while both of then
    // have nodes.
    while (a != NULL && b != NULL)
    {
        // If a's current value is smaller or equal to
        // b's current value.
        if (a->key <= b->key)
        {
            // Store next of current Node in first list
            Node *temp = a->next;

            // Add 'a' at the front of resultant list
            a->next = res;
            res = a;

            // Move ahead in first list
            a = temp;
        }

        // If a's value is greater. Below steps are similar
        // to above (Only 'a' is replaced with 'b')
        else
        {
            Node *temp = b->next;
            b->next = res;
            res = b;
            b = temp;
        }
    }

    // If second list reached end, but first list has
    // nodes. Add remaining nodes of first list at the
    // front of result list
    while (a != NULL)
    {
        Node *temp = a->next;
        a->next = res;
        res = a;
        a = temp;
    }

    // If first list reached end, but second list has
    // node. Add remaining nodes of first list at the
    // front of result list
    while (b != NULL)
    {
        Node *temp = b->next;
        b->next = res;
        res = b;
        b = temp;
    }

    return res;
}

/* Function to print Nodes in a given linked list */
void printList(struct Node *Node)
{
    while (Node!=NULL)
    {
        cout << Node->key << " ";
        Node = Node->next;
    }
}

/* Utility function to create a new node with
   given key */
Node *newNode(int key)
{
    Node *temp = new Node;
    temp->key = key;
    temp->next = NULL;
    return temp;
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* res = NULL;

    /* Let us create two sorted linked lists to test
       the above functions. Created lists shall be
         a: 5->10->15
         b: 2->3->20  */
    Node *a = newNode(5);
    a->next = newNode(10);
    a->next->next = newNode(15);

    Node *b = newNode(2);
    b->next = newNode(3);
    b->next->next = newNode(20);

    cout << "List A before merge: \n";
    printList(a);

    cout << "\nList B before merge: \n";
    printList(b);

    /* merge 2 increasing order LLs in descresing order */
    res = SortedMerge(a, b);

    cout << "\nMerged Linked List is: \n";
    printList(res);

    return 0;
}
```

Java

```
// Java program to merge two sorted linked list such that merged
// list is in reverse order

// Linked List Class
class LinkedList {

    Node head;  // head of list
    static Node a, b;

    /* Node Class */
    static class Node {

        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    void printlist(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    Node sortedmerge(Node node1, Node node2) {

        // if both the nodes are null
        if (node1 == null && node2 == null) {
            return null;
        }

        // resultant node
        Node res = null;

        // if both of them have nodes present traverse them
        while (node1 != null && node2 != null) {

            // Now compare both nodes current data
            if (node1.data <= node2.data) {
                Node temp = node1.next;
                node1.next = res;
                res = node1;
                node1 = temp;
            } else {
                Node temp = node2.next;
                node2.next = res;
                res = node2;
                node2 = temp;
            }
        }

        // If second list reached end, but first list has
        // nodes. Add remaining nodes of first list at the
        // front of result list
        while (node1 != null) {
            Node temp = node1.next;
            node1.next = res;
            res = node1;
            node1 = temp;
        }

        // If first list reached end, but second list has
        // node. Add remaining nodes of first list at the
        // front of result list
        while (node2 != null) {
            Node temp = node2.next;
            node2.next = res;
            res = node2;
            node2 = temp;
        }

        return res;

    }

    public static void main(String[] args) {

        LinkedList list = new LinkedList();
        Node result = null;

        /*Let us create two sorted linked lists to test
         the above functions. Created lists shall be
         a: 5->10->15
         b: 2->3->20*/
        list.a = new Node(5);
        list.a.next = new Node(10);
        list.a.next.next = new Node(15);

        list.b = new Node(2);
        list.b.next = new Node(3);
        list.b.next.next = new Node(20);
        list.b.next.next = new Node(20);

        System.out.println("List a before merge :");
        list.printlist(a);
        System.out.println("");
        System.out.println("List b before merge :");
        list.printlist(b);

        // merge two sorted linkedlist in decreasing order
        result = list.sortedmerge(a, b);
        System.out.println("");
        System.out.println("Merged linked list : ");
        list.printlist(result);

    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
List A before merge:
5 10 15
List B before merge:
2 3 20
Merged Linked List is:
20 15 10 5 3 2
```

This solution traverses both lists only once, doesn’t require reverse and works in-place.