**Problem Name:** Merge two sorted lists

**Topics:** Linked list, Recursion

**Companies:** Amazon, Microsoft, Facebook, Adobe, Apple, Google, Bloomberg, Oracle.

**Level:** Easy

**Language:** C++

**Problem Statement:** You are given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

**Input Format:**

First line contains integer n (size of first linked list).

Second line contain n space separated integer value.

Third line contain integer m (size of second linked).

Fourth line contain m space separated integer value.

**Output Format:** Print a sorted merged linked list.

**Constraints:**

* The number of nodes in both lists is in the range [0, 50].
* -100 <= Node.val <= 100
* Both list1 and list2 are sorted in **non-decreasing** order.

**Examples:**

**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Brute force Solution:**

**Explanation:**

* make a resultant linkedlist, then run a while loop to fill it with the smallest of current element from the two linkedlist and continue till one linkedlist reaches nullptr.\*
* fill the remaining elements from the other linkedlist in the resultantlinkedlist (as all elements are already sorted.
* copy the resultant array to first array and you are done.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

    int val;

    ListNode\* next;

};

void addNode(ListNode \*head, int data){

    ListNode \*ptr, \*aux;

    aux = new ListNode();

    aux->val = data;

    aux->next = NULL;

    ptr = head;

    while (ptr->next != NULL)

    {

        ptr = ptr->next;

    }

    ptr->next = aux;

}

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

    ListNode\* res = new ListNode();

    ListNode\* curr = res;

    while (l1 && l2)

    {

        if (l1->val < l2->val)

        {

            addNode(curr, l1->val);

            l1 = l1->next;

            curr = curr->next;

        }

        else

        {

            addNode(curr, l2->val);

            l2 = l2->next;

            curr = curr->next;

        }

    }

    while (l1)

    {

            addNode(curr, l1->val);

            l1 = l1->next;

            curr = curr->next;

    }

    while (l2)

    {

            addNode(curr, l2->val);

            l2 = l2->next;

            curr = curr->next;

    }

    return res->next;

}

void push(ListNode\*\* head\_ref, int new\_data)

{

    ListNode\* new\_node = new ListNode();

    new\_node->val = new\_data;

    new\_node->next = (\*head\_ref);

    (\*head\_ref) = new\_node;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

int main()

{

    ListNode\* res = NULL;

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    int n,m, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        push(&a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        push(&b, temp);

    }

    res = mergeTwoLists(a, b);

    printList(res);

    return 0;

}

**Time Complexity**: O(N+M)

**Space Complexity:** O(N+M)

**Optimized Solution:**

**Explanation:** Recursive approach:

* the base case: if one of the linkedlist if empty return the other
* so, what's actually happening is we select the node after checking smallest value of the two nodes from two linkedlists. And then start constructing the next part recursively.
* Other way to see it is, if we have a sorted linkedlist and two nodes to select from to add to the end of the sorted one, we choose the lower one and add the node to the sorted one and return it. That's exatly what we are doing, taking the smallest among the two nodes and constructing the final sorted linkedlist and retrning it.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

    int val;

    ListNode\* next;

};

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

       if (l1 == nullptr) return l2;

       if (l2 == nullptr) return l1;

        if (l1->val < l2->val)

        {

            l1->next = mergeTwoLists(l1->next, l2);

            return l1;

        }

        else

        {

            l2->next = mergeTwoLists(l1, l2->next);

            return l2;

        }

    }

void push(ListNode\*\* head\_ref, int new\_data)

{

    ListNode\* new\_node = new ListNode();

    new\_node->val = new\_data;

    new\_node->next = (\*head\_ref);

    (\*head\_ref) = new\_node;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

int main()

{

    ListNode\* res = NULL;

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    int n,m, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        push(&a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        push(&b, temp);

    }

    res = mergeTwoLists(a, b);

    printList(res);

    return 0;

}

**Time Complexity**: O(N+M)

**Space Complexity:** O(1)