**Problem Name:** Intersection of Two Linked List

**Topics:** Linked list, Hash Table, Two Pointers

**Companies:** Facebook, Microsoft, Amazon, Apple, LinkedIn, Nvidia, tiktok.

**Level:** Easy

**Language:** C++

**Problem Statement**: Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

**Input Format:**

First line of the input contain integer n (size of list)

Second line contain n space separated integer list values.

Third line contain integer value m (size of list 2)

Fourth line contain m space separated integer list values.

Ex:

3

2 4 3

3

5 6 4

**Output Format:** Print linked list after adding both list.

**Constraints:**

* The number of nodes in each linked list is in the range [1, 100].
* 0 <= Node.val <= 9
* It is guaranteed that the list represents a number that does not have leading zeros.

**Examples:**

**Input:** l1 = [2,4,3], l2 = [5,6,4]

**Output:** [7,0,8]

**Explanation:** 342 + 465 = 807.

**Brute force Solution:**

**Explanation:** We have to Traverse Both Lists & add sum to new list. Create a dummy list. Sum is equivalent to val1 + val2 + carry from previous Operation. The resulting node will be sum%10. Carry is updated by sum/10 for next Operation. Check for two condition either list one is empty or list two is empty or not then check for carry to empty or not.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

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

    ListNode \*p = l1;

    ListNode \*q = l2;

    int carry = 0;

    ListNode \*dummy = new ListNode(0);

    dummy->next = NULL;

    ListNode \*res = dummy;

    while(p!=NULL && q!=NULL){

        int data = p->val + q->val + carry;

        ListNode \*t = new ListNode(0);

        t->next = NULL;

        t->val = data%10;

        carry = data/10;

        res->next = t;

        res = t;

        p = p->next;

        q = q->next;

    }

    while(p!=NULL){

        int data = p->val + carry;

        ListNode \*t = new ListNode(0);

        t->next = NULL;

        t->val = data%10;

        carry = data/10;

        res->next = t;

        res = t;

        p = p->next;

    }

    while(q!=NULL){

        int data =  q->val + carry;

        ListNode \*t = new ListNode(0);

        t->next = NULL;

        t->val = data%10;

        carry = data/10;

        res->next = t;

        res = t;

        q = q->next;

    }

    if(carry!=0){

        ListNode \*t = new ListNode(0);

        t->next = NULL;

        t->val = carry;

        res->next = t;

        res = t;

    }

    return dummy->next;

}

int main() {

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    ListNode\* res = NULL;

    int n, m, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        insertNode(b, temp);

    }

    res = addTwoNumbers(a, b);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(N)

**Optimized Solution:**

Explanation: We have to Traverse Both Lists & add sum to new list. Sum is equivalent to val1 + val2 + carry from previous Operation. The resulting node will be sum%10. Carry is updated by sum/10 for next Operation.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

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

    ListNode \*ptr = new ListNode(0);

    ListNode \*temp = ptr;

    int c = 0;

    while (l1 != NULL ||  l2 != NULL || c){

        int sum = 0;

        if(l1 != NULL){

            sum += l1->val;

            l1 = l1 -> next;

        }

        if(l2 != NULL){

            sum += l2->val;

            l2 = l2 -> next;

        }

        sum += c;

        c = sum/10;

        ListNode \*node = new ListNode(sum%10);

        temp -> next = node;

        temp = temp -> next;

    }

    return ptr -> next;

}

int main() {

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    ListNode\* res = NULL;

    int n, m, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        insertNode(b, temp);

    }

    res = addTwoNumbers(a, b);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)