**Problem Name:** Linked list Component

**Topics:** Linked list, Hash table

**Companies:** Google

**Level:** Medium

**Language:** C++

**Problem Statement**: You are given the head of a linked list containing unique integer values and an integer array nums that is a subset of the linked list values.

Return *the number of connected components in*nums*where two values are connected if they appear****consecutively****in the linked list*.

**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 vector nums)

Fourth line contain m space separated integer nums values.

Ex:

4

0 1 2 3

3

0 1 3

**Output Format:** Print integer value returned from function.

**Constraints:**

* The number of nodes in the linked list is n.
* 1 <= n <= 104
* 0 <= Node.val < n
* All the values Node.val are **unique**.
* 1 <= nums.length <= n
* 0 <= nums[i] < n
* All the values of nums are **unique**.

**Examples:**

**Input:** head = [0,1,2,3], nums = [0,1,3]

**Output:** 2

**Explanation:** 0 and 1 are connected, so [0, 1] and [3] are the two connected components.

**Brute force Solution:**

**Explanation:** apply binary seach to figure out whether every element of the given linked list belongs to a connected component. Problem can be solved by checking whether an element in the list is present in the array G, if so we add it to a connected component. If not, we check whether there is already a connected component formed and increment the count.

**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;

    }

}

bool search(vector<int> &nums,int val1){

    int start = 0, end = nums.size()-1,mid;

    while(start<=end){

        mid = start + (end-start)/2;

        if(nums[mid] == val1){

            return true;

        }

        else if(nums[mid]>val1){

            end = mid-1;

        }

        else{

            start = mid+1;

        }

    }

    return false;

}

int numComponents(ListNode\* head, vector<int>& nums) {

    ListNode \*temp = head;

    bool flag = false;

    int count = 0;

    sort(nums.begin(),nums.end());

    while(temp->next != NULL){

        bool found = search(nums,temp->val);

        if(!found && flag){

            count++;

            flag = false;

        }

        else if(found){

            flag = true;

        }

        else;

        temp = temp->next;

    }

    bool found = search(nums,temp->val);

    if(!found && flag){

        count++;

        flag = false;

    }

    else if(found){

        flag = true;

        count++;

    }

    else;

    return count;

}

int main() {

    ListNode\* a = NULL;

    int n, m, temp, res;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    vector<int> b(m);

    for(int i=0; i<m; i++){

        cin>>b[i];

    }

    res = numComponents(a, b);

    cout<<res<<"\n";

    return 0;

}

**Time Complexity**: O(nlogn)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:** Our intuition behind this problem is to put the **nums** vector into a hashmap, and then interate over the linked-list and find if any node is in components (side by side) with another, which must be present in the **nums** vector!

At every position we're taking a **streak** variable which updates and if the value of the Node was not in nums, then we check if the **streak** is greater than 0, if yes we increment the **count(final answer)**, and make the present **streak** back to 0.  
Finally, we return the **count(ans)**.

**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;

    }

}

int numComponents(ListNode\* head, vector<int>& nums) {

    ListNode \*curr = head;

    unordered\_map<int, bool> mpp;

    for(int i=0; i<nums.size(); i++)

        mpp[nums[i]] = true;

    int count = 0, streak = 0;

    while(curr) {

        if(mpp[curr->val]==true) {

            streak++;

        } else {

            if(streak>0) {

                count++;

            }

            streak = 0;

        }

        curr = curr->next;

    }

    if(streak>0) {

        count++;

    }

    return count;

}

int main() {

    ListNode\* a = NULL;

    int n, m, temp, res;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    vector<int> b(m);

    for(int i=0; i<m; i++){

        cin>>b[i];

    }

    res = numComponents(a, b);

    cout<<res<<"\n";

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(N)