**Problem Name:** Palindrome Linked List

**Topics:** Linked list, Two Pointers, Stack, Recursion

**Companies:** Facebook, Amazon, Microsoft, Apple, Bloomberg, Intuit, Google, VMware

**Level:** Easy

**Language:** C++

**Problem Statement**: Given the head of a singly linked list, return true if it is a palindrome.

**Input Format:**

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

Second line contain n space separated integer list values.

Ex:

5

1 2 3 4 5

**Output Format:** Boolean value 1 0r 0 indicating list is palindrome or not.

**Constraints:**

* The number of nodes in the list is in the range [1, 105].
* 0 <= Node.val <= 9

**Examples:**

**Input:** head = [1,2,2,1]

**Output:** true

**Brute force Solution:**

**Explanation:**   Copy each value from linked list to array then traverse array and linked list anti parallel checking each value, if found not same return false else true.

**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 isPalindrome(ListNode\* head){

    int arr[100000]={-1};

    ListNode\* ptr=head;

    int i=0;

    while(ptr){

        arr[i++]=ptr->val;

        ptr=ptr->next;

    }

    i--;

    ptr=head;

    while(ptr)

    {

        if(arr[i--]!=ptr->val)

        return false;

        ptr=ptr->next;

    }

    return true;

}

int main()

{

    ListNode\* a = NULL;

    int n, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cout<<isPalindrome(a)<<"\n";

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(N)

**Optimized Solution:**

**Explanation**: [Floyd's Cycle Detection Algorithm](https://en.wikipedia.org/wiki/Cycle_detection#Floyd's_tortoise_and_hare). With Floyd's, we'll travel through the linked list with two pointers, one of which is moving twice as fast as the other. When the fast pointer reaches the end of the list, the slow pointer must then be in the middle. With slow now at the middle, we can reverse the back half of the list with the help of another variable to contain a reference to the previous node (prev) and a three-way swap. Before we do this, however, we'll want to set prev.next = null, so that we break the reverse cycle and avoid an endless loop. Once the back half is properly reversed and slow is once again at the end of the list, we can now start fast back over again at the head and compare the two halves simultaneously, with no extra space required. If the two pointers ever disagree in value, we can return false, otherwise we can return true if both pointers reach the middle successfully.

**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 isPalindrome(ListNode\* head) {

    ListNode \*slow = head, \*fast = head, \*prev, \*temp;

    while (fast && fast->next)

        slow = slow->next, fast = fast->next->next;

    prev = slow, slow = slow->next, prev->next = NULL;

    while (slow)

        temp = slow->next, slow->next = prev, prev = slow, slow = temp;

    fast = head, slow = prev;

    while (slow)

        if (fast->val != slow->val) return false;

        else fast = fast->next, slow = slow->next;

    return true;

}

int main()

{

    ListNode\* a = NULL;

    int n, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cout<<isPalindrome(a)<<"\n";

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)