**Problem Name:** Reverse nodes in k-groups

**Topics:** Linked list, Recursion

**Companies:** Capital One,Amazon, Microsoft, Facebook, Google, Bloomberg, Snapchat, Oracle, Zoom

**Level:** Hard

**Language:** C++

**Problem Statement:** Given the head of a linked list, reverse the nodes of the list k at a time, and return *the modified list*.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

**Input Format:**

First line contain integer n (size of linked list).

Second line contain n space separated integer values.

Third line contain value of k.

Ex.

5

1 2 3 4 5

2

**Output Format:** Print a linked list after reversing the nodes k at a time.

Ex.

2 1 4 3 5

**Constraints:**

* The number of nodes in the list is n.
* 1 <= k <= n <= 5000
* 0 <= Node.val <= 1000

**Examples:**

**Input:** head = [1,2,3,4,5], k = 2

**Output:** [2,1,4,3,5]

**Brute force Solution:**

**Explanation:**

Idea is to traverse the list with k groups in mind. For every group found, we separate and reverse the group and insert it back in the following order:

prev\_node--> reversed --->next\_node

**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;

}

ListNode\* reverseList(ListNode\* head) {

    ListNode \*prev = nullptr, \*nextptr = nullptr;

    while(head) {

        nextptr = head->next;

        head->next = prev;

        prev = head;

        head = nextptr;

    }

    return prev;

}

ListNode\* reverseKGroup(ListNode\* head, int k) {

    ListNode \*dummy = new ListNode(-1), \*tail = dummy;

    ListNode \*nextptr = nullptr;

    tail->next = head;

    while(head) {

        int i = 1;

        ListNode \*first\_node = head;

        while(head->next && i < k) {

            head = head->next;

            ++i;

        }

        nextptr = head->next;

        head->next = nullptr;

        if(i == k) {

            tail->next = reverseList(first\_node);

            first\_node->next = nextptr;

            tail = first\_node;

        }

        head = nextptr;

    }

    head = dummy->next;

    delete dummy;

    return head;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

int main()

{

    ListNode\* res = NULL;

    ListNode\* a = NULL;

    int n, k, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>k;

    res = reverseKGroup(a, k);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)

**Recursive Solution:**

**Explanation:** 1) The first step is to check whether the Head is NULL or Not, if its NULL then we can directly return NULL,

2) If the Head is not NULL, then we need to check the length of Linked List starting from current Head.

3) If it is not a multiple of K(Less than K) , then there is no need to reverse it and hence we can directly return head,

4) Else if its a multiple of K, then we have to reverse the K elements starting from current Head,

5) We will follow the same steps for the rest of the elements Recursively.

**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;

}

ListNode\* reverseKGroup(ListNode\* head, int k)

{

    if(!head)

        return NULL;

    ListNode \*KSizeChecker = head;

    for(int i=0;i<k;i++)

    {

        if(KSizeChecker==NULL)

            return head;

        KSizeChecker = KSizeChecker->next;

    }

    int cnt=0;

    ListNode \*cur=head,\*prev=NULL,\*next=NULL;

    while(cur and cnt<k)

    {

        next=cur->next;

        cur->next=prev;

        prev=cur;

        cur=next;

        cnt++;

    }

    if(next)

        head->next=reverseKGroup(next,k);

    return prev;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

int main()

{

    ListNode\* res = NULL;

    ListNode\* a = NULL;

    int n, k, temp;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>k;

    res = reverseKGroup(a, k);

    printList(res);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)