Linked List 2

Correct Output

Send Feedback

What does the following function do?

```
void fun(Node head)
{
 if(head == null)
   return;

  fun(head.next);
  System.out.print( head.data);

}
```

Options

Attempts le

This problem has only one correct answer

- Print all Nodes of Linked List
- Print all Nodes of Linked List in reverse order
- Print alternate Nodes of Linked list
- Print alternate Nodes of Linked list in reverse order

Hurray! Correct Answer

Solution Description

fun() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun() prints 5->4->3->2->1.

Correct Output

Send Feedback

What will be the correct output of the code, If the linked list is 1->2->3->4->5->6 and start stores the address of node 1.

```
void fun(Node start)
{
 if(start == null)
   return;
 System.out.printf( start.data);

 if(start.next != null )
   fun(start.next.next);
 System.out.printf(start.data);
}
```

Options

Attempts left:

This problem has only one correct answer

- 1 3 5
- 1 2 4
- 1 2 4 4 2 1
- 1 3 5 5 3 1
- Error

Hurray! Correct Answer

Solution Description

fun() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then skips the last node.

Delete node Recursively

Send Feedback

Given a singly linked list of integers and position 'i', delete the node present at the 'i-th' position in the linked list recursively.

Note :

Assume that the Indexing for the linked list always starts from 0.

No need to print the list, it has already been taken care. Only return the new head to the list.

input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'i'.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.
0 <= i < M

Time Limit: 1sec

Sample Input 1 :

```
1
3 4 5 2 6 1 9 -1
3
```

Sample Output 1 :

```
3 4 5 6 1 9
```

```
Sample Input 2 :

2
30 -1
0
10 20 30 50 60 -1
4

Sample Output 2 :

10 20 30 50

public class Solution {


   public static LinkedListNode<Integer>
deleteNodeRec(LinkedListNode<Integer> head, int pos) {
       //Your code goes here
       if(head.next==null){
           if(pos==0)
               return null;
           else
               return head;
       }
       if(pos==0){
           return head.next;
       }

       head.next = deleteNodeRec(head.next, pos-1);
       return head;
   }

}
```

```
Reverse LL (Recursive)
Send Feedback
Given a singly linked list of integers, reverse it using recursion and
return the head to the modified list. You have to do this in O(N) time
complexity where N is the size of the linked list.
Note :
```

No need to print the list, it has already been taken care. Only return the new head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^4
Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

```
1
1 2 3 4 5 6 7 8 -1
```

Sample Output 1 :

```
8 7 6 5 4 3 2 1
```

```
Sample Input 2 :

2
10 -1
10 20 30 40 50 -1

Sample Output 2 :

10
50 40 30 20 10
```

```java
public class Solution {

   public static LinkedListNode<Integer>
reverseLinkedListRec(LinkedListNode<Integer> head) {
       //Your code goes here
       if(head==null || head.next==null)
           return head;
       LinkedListNode<Integer> revHead = null;
       LinkedListNode<Integer> temp = null;
       revHead = reverseLinkedListRec(head.next);
       temp = revHead;
       while(temp.next!=null){
           temp = temp.next;
       }
       temp.next = head;
       head.next=null;
       return revHead;

   }

}
```

```
Reverse LL (Iterative)
Send Feedback
Given a singly linked list of integers, reverse it iteratively and return
the head to the modified list.
Note :
```

No need to print the list, it has already been taken care. Only return the new head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= N <= 10^4
Where N is the size of the singly linked list.

Time Limit: 1 sec

Sample Input 1 :

```
1
1 2 3 4 5 6 7 8 -1
```

Sample Output 1 :

```
8 7 6 5 4 3 2 1
```

```
Sample Input 2 :

2
10 -1
10 20 30 40 50 -1

Sample Output 2 :

10
50 40 30 20 10

public class Solution {

   public static LinkedListNode<Integer> reverse_I(LinkedListNode<Integer>
head){
       // if(head==null || head.next==null)
       //     return head;
       LinkedListNode<Integer> prev = null;
       LinkedListNode<Integer> curr = head;
       LinkedListNode<Integer> nextN = head;
       while(nextN!=null){
           nextN=nextN.next;
           curr.next=prev;
           prev=curr;
           curr=nextN;
       }
       return prev;
   }
}
```

```
Mid Point Linked List
Send Feedback
For a given singly linked list of integers, find and return the node
present at the middle of the list.
Note :

If the length of the singly linked list is even, then return the first
middle node.
```

Example: Consider, 10 -> 20 -> 30 -> 40 is the given list, then the nodes present at the middle with respective data values are, 20 and 30. We return the first node with data 20.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output Format :

For each test case/query, print the data value of the node at the middle of the given list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

```
1
1 2 3 4 5 -1
```

Sample Output 1 :

```
3

Sample Input 2 :

2
-1
1 2 3 4 -1

Sample Output 2 :

2



public class Solution {

   public static LinkedListNode<Integer> midPoint(LinkedListNode<Integer>
head) {
       //Your code goes here
       if(head==null || head.next==null)
           return head;
       LinkedListNode<Integer> fast = head;
       LinkedListNode<Integer> slow = head;
       while(fast.next!=null && fast.next.next!=null){
           slow = slow.next;
           fast = fast.next.next;
           // System.out.println("slow-->"+slow.data+"
fast-->"+fast.data);
       }
       return slow;
   }

}
```

Merge Two Sorted LL

Send Feedback

You have been given two sorted(in ascending order) singly linked lists of integers.

Write a function to merge them in such a way that the resulting singly linked list is also sorted(in ascending order) and return the **new** head to the list.

Note :

Try solving *this* in O(1) auxiliary space.

No need to print the list, it has already been taken care.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the first sorted singly linked list separated by a single space.

The second line of the input contains the elements of the second sorted singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output :

For each test case/query, print the resulting sorted singly linked list, separated by a single space.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t = 10^2
0 <= *N* <= 10 ^ 4
0 <= *M* <= 10 ^ 4
Where *N* and *M* denote the sizes of the singly linked lists.

Time Limit: 1sec

Sample Input 1 :

```
1
2 5 8 12 -1
3 6 9 -1

Sample Output 1 :

2 3 5 6 8 9 12

Sample Input 2 :

2
2 5 8 12 -1
3 6 9 -1
10 40 60 60 80 -1
10 20 30 40 50 60 90 100 -1

Sample Output 2 :

2 3 5 6 8 9 12
10 10 20 30 40 40 50 60 60 60 80 90 100

public class Solution {

   public static LinkedListNode<Integer>
mergeTwoSortedLinkedLists(LinkedListNode<Integer> head1,
LinkedListNode<Integer> head2) {
       //Your code goes here
       if(head1==null)
           return head2;
       if(head2==null)
           return head1;
       LinkedListNode<Integer> next1 = head1;
       LinkedListNode<Integer> next2 = head2;
       LinkedListNode<Integer> current = null;
       LinkedListNode<Integer> currentHead = null;
       if(next1.data<next2.data){
           current = next1;
           currentHead = current;
           next1 = next1.next;
       }
```

```
        else{
            current = next2;
            currentHead = current;
            next2 = next2.next;
        }

        while(next1!=null && next2!=null){
            if(next1.data<next2.data){
                current.next = next1;
                current = next1;
                next1 = next1.next;
            }
            else{
                current.next = next2;
                current = next2;
                next2 = next2.next;
            }
        }

        if(next1!=null){
            current.next = next1;
        }
        if(next2!=null){
            current.next = next2;
        }
        return currentHead;
    }

}
```

Merge Sort LL

Send Feedback

Given a singly linked list of integers, sort it using 'Merge Sort.'

Note :

No need to print the list, it has already been taken care. Only return the **new** head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the sorted singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

```
1
10 9 8 7 6 5 4 3 -1
```

Sample Output 1 :

```
3 4 5 6 7 8 9 10
```

Sample Output 2 :

```
2
-1
10 -5 9 90 5 67 1 89 -1
```

```
Sample Output 2 :

-5 1 5 9 10 67 89 90

public class Solution {

   public static LinkedListNode<Integer> merge(LinkedListNode<Integer>
head1, LinkedListNode<Integer> head2){
       LinkedListNode<Integer> current = null;
       LinkedListNode<Integer> currentHead = null;
       if(head1.data<head2.data){
           current = head1;
           currentHead = current;
           head1 = head1.next;
       }
       else{
           current = head2;
           currentHead = current;
           head2 = head2.next;
       }
       while(head1!=null && head2!=null){
           if(head1.data<head2.data){
               current.next = head1;
               current = current.next;
               head1 = head1.next;
           }
           else{
               current.next = head2;
               current = current.next;
               head2 = head2.next;
           }
       }
       if(head1!=null){
           current.next = head1;
       }
       else if(head2!=null){
           current.next = head2;
       }
       return currentHead;
```

```
    }

    public static LinkedListNode<Integer> mergeSort(LinkedListNode<Integer>
head) {
        if(head==null || head.next==null)
            return head;

        LinkedListNode<Integer> slow = head;
        LinkedListNode<Integer> fast = head;
        LinkedListNode<Integer> prev = head;

        while(fast.next!=null && fast.next.next!=null){
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }

        LinkedListNode<Integer> list1 = null;
        LinkedListNode<Integer> list2 = null;
        list2 = mergeSort(slow.next);
        slow.next = null;
        list1 = mergeSort(head);

        list1 = merge(list1,list2);
        return list1;
    }
}
```

Traversal

Send Feedback

In doubly linked lists, traversal can be done in ?

Options

This problem has only one correct answer

- Only forward direction
- Only reverse direction
- Both directions
- None of the above

Hurray! Correct Answer

Doubly LL

Send Feedback

Given an unsorted doubly Linked List, suppose you have references (or pointer) to its head and tail nodes, which of the following operation can be implemented in O(1) time ?

i) Insertion at the front of the linked list
ii) Insertion at the end of the linked list
iii) Deletion of the last node of the linked list
iv) Deletion of the front node of the linked list

Options

This problem has only one correct answer

- I and II
- I and III
- I,II and III
- I,II, III and IV

Hurray! Correct Answer

Circular LL

Send Feedback

Given an unsorted circular linked list, suppose you have reference (or pointer) to its head node only, which of the following operation can be implemented in O(1) time?

i) Insertion at the front of the linked list
ii) Insertion at the end of the linked list
iii) Deletion of the last node of the linked list
iv) Deletion of the front node of the linked list

Options

This problem has only one correct answer

- I and II
- I and III
- I,II and III
- None

Hurray! Correct Answer

Time Complexity

Send Feedback

What are the time complexities of finding 6th element from beginning and 6th element from end in a singly linked list? Let n be the number of nodes in linked list, you may assume that n > 7.

Options

This problem has only one correct answer

- (●) O(1) and O(n)
- ( ) O(n) and O(n)
- ( ) O(1) and O(1)
- ( ) O(n) and O(1)

Hurray! Correct Answer

Concat Linked List

Send Feedback

The concatenation of two lists is to be performed in O(1) time. Which of the following implementations of a list should be used?

Options

Attempts left: 1/2

This problem has only one correct answer

- ( ) Singly Linked List
- ( ) Doubly Linked List
- (●) Circular Doubly Linked List
- ( ) Circular Linked List

Hurray! Correct Answer

Solution Description

Singly-linked list cannot be answer because we cannot find last element of a singly linked list in O(1) time. Doubly linked list and Circular Linked List cannot also not be answer because of the same reason as singly linked list.

Circular Doubly LL

Send Feedback

Given an unsorted circular doubly linked list, suppose you have reference (or pointer) to its head node only, which of the following operation can be implemented in O(1) time?

i) Insertion at the front of the linked list
ii) Insertion at the end of the linked list
iii) Deletion of the last node of the linked list
iv) Deletion of the front node of the linked list

Options

This problem has only one correct answer

- ( ) I and II
- ( ) I,II and III
- (●) I,II,III and IV
- ( ) None

Hurray! Correct Answer

Find a node in LL (recursive)

Send Feedback

Given a singly linked list of integers and an integer n, find and return the index for the first occurrence of 'n' in the linked list. -1 otherwise.

Follow a recursive approach to solve this.
Note :

Assume that the Indexing for the linked list always starts from 0.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'n'.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

```
1
3 4 5 2 6 1 9 -1
```

```
100

Sample Output 1 :

-1

Sample Input 2 :

2
10 20010 30 400 600 -1
20010
100 200 300 400 9000 -34 -1
-34

Sample Output 2 :

1
5



public class Solution {

   public static int findNodeRec(LinkedListNode<Integer> head, int n) {
       //Your code goes here
       if(head==null)
           return -1;
       LinkedListNode<Integer> node = head;
       int index = 0;
       while(node.next!=null){
           if(n==node.data)
               return index;
           node = node.next;
           index++;
       }
       return -1;
   }

}
```

Even after Odd LinkedList

Send Feedback

For a given singly linked list of integers, arrange the elements such that all the even numbers are placed after the odd numbers. The relative order of the odd and even terms should remain unchanged.

Note :

No need to print the list, it has already been taken care. Only return the new head to the list.

Input format:

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format:

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2

0 <= M <= 10^5

Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

1

```
1 4 5 2 -1

Sample Output 1 :

1 5 4 2

Sample Input 2 :

2
1 11 3 6 8 0 9 -1
10 20 30 40 -1

Sample Output 2 :

1 11 3 9 6 8 0
10 20 30 40

public class Solution {

   public static LinkedListNode<Integer>
evenAfterOdd(LinkedListNode<Integer> head) {
       // Your code goes here
       if (head == null || head.next == null)
           return head;
       LinkedListNode<Integer> node = head;
       LinkedListNode<Integer> oddHead = null;
       LinkedListNode<Integer> oddTail = null;
       LinkedListNode<Integer> evenHead = null;
       LinkedListNode<Integer> evenTail = null;
       while (node!=null) {
           if(node.data%2!=0){
               if(oddHead==null){
                   oddHead = node;
                   oddTail = node;
                   node = node.next;
               }
               else{
                   oddTail.next = node;
                   oddTail = node;
                   node = node.next;
```

```
                }
            }
            else{
                if(evenHead==null){
                    evenHead = node;
                    evenTail = node;
                    node = node.next;
                }
                else{
                    evenTail.next = node;
                    evenTail = node;
                    node = node.next;
                }
            }
        }
        if(oddHead!=null){
            oddTail.next = evenHead;
            if(evenTail!=null)
                evenTail.next = null;
            return oddHead;
        }
        else{
            return evenHead;
        }
    }
}
```

Delete every *N* nodes

Send Feedback

You have been given a singly linked list of integers along with two integers, 'M,' and 'N.' *Traverse* the linked list such that you retain the 'M' nodes, then delete the next 'N' nodes. Continue the same until the end of the linked list.

To put it in other words, in the given linked list, you need to delete *N* nodes after every *M* nodes.

Note :

No need to print the list, it has already been taken care. Only return the **new** head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains two integer values 'M,' and 'N,' respectively. A single space will separate them.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= *P* <= 10^5
Where P is the size of the singly linked list.
0 <= *M* <= 10^5
0 <= *N* <= 10^5

Time Limit: 1sec

Sample Input 1 :

```
1
1 2 3 4 5 6 7 8 -1
2 2
```

Sample Output 1 :

```
1 2 5 6

Sample Input 2 :

2
10 20 30 40 50 60 -1
0 1
1 2 3 4 5 6 7 8 -1
2 3

Sample Output 2 :

1 2 6 7

Explanation of Sample Input 2 :

For the first query, we delete one node after every zero elements hence
removing all the items of the list. Therefore, nothing got printed.

For the second query, we delete three nodes after every two nodes,
resulting in the final list, 1 -> 2 -> 6 -> 7.

public class Solution {

   public static LinkedListNode<Integer>
skipMdeleteN(LinkedListNode<Integer> head, int M, int N) {
       //Your code goes here
       if(M==0)
           return null;
       LinkedListNode<Integer> node = head;
       LinkedListNode<Integer> temp = head;
       int countn=0, countm=1;
       while(node!=null){
           while(countm<M && node!=null){
               countm++;
               node = node.next;
           }
           countm=1;
           if(node==null)
```

```
                break;
            temp = node.next;
            while(countn<N && temp!=null){
                countn++;
                temp = temp.next;
            }
            node.next = temp;
            node = node.next;
            countn=0;
        }
        return head;
    }
}
```

Swap two *Nodes* of *LL*

Send Feedback

You have been given a singly linked list of integers along with two integers, 'i,' and 'j.' *Swap* the nodes that are present at the 'i-th' and 'j-th' positions.

Note :

Remember, the nodes themselves must be swapped and not the datas.

No need to print the list, it has already been taken care. Only return the **new** head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains two integer values 'i,' and 'j,' respectively. A single space will separate them.

Remember/consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

1 <= t <= 10^2
0 <= M <= 10^5
Where M is the size of the singly linked list.
0 <= i < M
0 <= j < M

Time Limit: 1sec

Sample Input 1 :

```
1
3 4 5 2 6 1 9 -1
3 4
```

Sample Output 1 :

```
3 4 5 6 2 1 9
```

Sample Input 2 :

```
2
10 20 30 40 -1
1 2
70 80 90 25 65 85 90 -1
0 6
```

Sample Output 2 :

```
10 30 20 40
90 80 90 25 65 85 70

public class Solution {

   public static LinkedListNode<Integer> swapNodes(LinkedListNode<Integer>
head, int i, int j) {
       //Your code goes here
       LinkedListNode<Integer> node = head;
       LinkedListNode<Integer> num1 = head;
       LinkedListNode<Integer> num2 = head;
       LinkedListNode<Integer> prev1 = null;
       LinkedListNode<Integer> prev2 = null;
       LinkedListNode<Integer> nextNode = null;
       int count=0;
       if(i==0 || j==0){
           if(i==0){
               while(count<j){
                   prev2=node;
                   node = node.next;
                   num2 = node;
                   count++;
               }
               nextNode = num1.next;
               prev2.next = num1;
               num1.next = num2.next;
               num2.next = nextNode;
           }
           else{
               while(count<i){
                   prev1=node;
                   node = node.next;
                   num1 = node;
                   count++;
               }
               nextNode = num2.next;
               prev1.next = num2;
               num2.next = num1.next;
               num1.next = nextNode;
           }
```

```
        }
        else if(i-j==1 || i-j==-1){
            if(i<j){
                while(count<i){
                    prev1 = node;
                    node = node.next;
                    num1 = node;
                    count++;
                }
                num2 = num1.next;
                prev1.next = num2;
                num1.next = num2.next;
                num2.next = num1;
            }
            else{
                while(count<j){
                    prev2 = node;
                    node = node.next;
                    num2 = node;
                    count++;
                }
                prev2.next = num1;
                num2.next = num1.next;
                num1.next = num2;
            }
        }
        else if((i==0 || j==0) && (i-j==1 || i-j==-1)){
            if(i==0){
                while(count<j){
                    prev2=node;
                    node = node.next;
                    num2 = node;
                    count++;
                }
                // nextNode = num1.next;
                prev2.next = num1;
                num1.next = num2.next;
                num2.next = nextNode;
            }
            else{
```

```
                while(count<i){
                    prev1=node;
                    node = node.next;
                    num1 = node;
                    count++;
                }
                nextNode = num2.next;
                prev1.next = num2;
                num2.next = num1.next;
                num1.next = nextNode;
            }
        }
        else{
            while(count<i){
                prev1=node;
                node = node.next;
                num1 = node;
                count++;
            }
            count = 0;
            node = head;
            while(count<j){
                prev2=node;
                node = node.next;
                num2 = node;
                count++;
            }
            LinkedListNode<Integer> temp = num1.next;
            num1.next = num2.next;
            prev2.next = num1;
            prev1.next = num2;
            num2.next = temp;
            return head;
        }
        return head;
    }
}
```

kReverse

Send Feedback

Given a singly linked list of integers, reverse the nodes of the linked list 'k' at a time and return its 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 be reversed as well.
Example :

Given this linked list: 1 -> 2 -> 3 -> 4 -> 5

For k = 2, you should return: 2 -> 1 -> 4 -> 3 -> 5

For k = 3, you should return: 3 -> 2 -> 1 -> 5 -> 4

Note :

No need to print the list, it has already been taken care. Only return the new head to the list.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains the elements of the singly linked list separated by a single space.

The second line of input contains a single integer depicting the value of 'k'.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the updated singly linked list.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= *M* <= 10^5
Where M is the size of the singly linked list.
0 <= k <= *M*

Time Limit: 1sec

Sample Input 1 :

```
1
1 2 3 4 5 6 7 8 9 10 -1
4
```

Sample Output 1 :

```
4 3 2 1 8 7 6 5 10 9
```

Sample Input 2 :

```
2
1 2 3 4 5 -1
0
10 20 30 40 -1
4
```

Sample Output 2 :

```
1 2 3 4 5
40 30 20 10
```

```
public class Solution {

   public static LinkedListNode<Integer> reverse(LinkedListNode<Integer>
head){
       if(head==null || head.next==null)
           return head;
```

```
        LinkedListNode<Integer> node=reverse(head.next);
        head.next.next = head;
        head.next = null;
        return node;
    }

    public static LinkedListNode<Integer> kReverse(LinkedListNode<Integer>
head, int k) {
        if(head==null || head.next==null)
            return head;
        LinkedListNode<Integer> tail = head;
        int count=1;
        while(count<k && tail.next!=null){
            tail=tail.next;
            count++;
        }
        LinkedListNode<Integer> nextHead = tail.next;
        tail.next = null;
        // System.out.println(tail.data);
        LinkedListNode<Integer> smallAnswer = reverse(head);
        head = smallAnswer;
        while(smallAnswer.next!=null){
            smallAnswer = smallAnswer.next;
        }
        // System.out.println(head.data);
        smallAnswer.next = kReverse(nextHead,k);
        return head;

    }

}
```

```
Bubble Sort (Iterative) LinkedList
Send Feedback
Given a singly linked list of integers, sort it using 'Bubble Sort.'
Note :

No need to print the list, it has already been taken care. Only return the
new head to the list.
```

Input format :

The first and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :

While specifying the list elements for input, -1 indicates the end of the singly linked list and hence, would never be a list element

Output format :

For each test case/query, print the elements of the sorted singly linked list.

Output for every test case will be printed in a seperate line.

Constraints :

0 <= *M* <= 10^3
Where M is the size of the singly linked list.

Time Limit: 1sec

Sample Input 1 :

10 9 8 7 6 5 4 3 -1

Sample Output 1 :

3 4 5 6 7 8 9 10

Sample Input 2 :

10 -5 9 90 5 67 1 89 -1

Sample Output 2 :

-5 1 5 9 10 67 89 90

```java
public class Solution {

   public static LinkedListNode<Integer>
bubbleSort(LinkedListNode<Integer> head) {

       if(head==null || head.next==null)
           return head;
       LinkedListNode<Integer> i = head;
       LinkedListNode<Integer> j = head;
       LinkedListNode<Integer> prev = null;
       LinkedListNode<Integer> temp = null;
       int count = 1;
       while(i.next!=null){
           i = i.next;
           count++;
       }
       while(count>1){
           j = head;
           prev = null;
           while(j.next!=null){
               if(j.data>j.next.data){
                   if(prev==null){
                       temp = j.next;
                       j.next = temp.next;
                       temp.next = j;
                       head = temp;
                   }
                   else{
                       temp = j.next;
                       prev.next = temp;
                       j.next = temp.next;
                       temp.next = j;
                   }
                   prev = temp;
               }
               else{
                   prev = j;
                   j=j.next;
               }
           }
```

```
            count--;

        }
        return head;

    }
}
```