kReverse

Problem Level: Hard

Problem Description:

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.

Sample Input 1:

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

Sample Output 1:

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

Approach to be followed:

Initially, we still need to think to break down the big problem into smaller ones.

In this problem, we will reverse the first set of k nodes, and then send the left over linked list into recursion, so that the next set of k nodes can be reversed. This will be done until the point there are no nodes left to reverse. Thus, if we can do that reversal in the current function call, and then somehow directly obtain a reversed result for the subsequent linked list, and connect them correctly, the problem should be solved. One edge case we need to keep in mind is that, we might reach a position, where we have to reverse k nodes, but the linked list (or left over linked list) has less than k nodes.

Steps:

1. Base case: If k is equal to 0 or 1, return the linked list as it is (as if k=0 or 1, then there is no reversal required).
2. Recursively reverse the list beginning from **(k+1)th** node.

3. To tackle the edge case discussed earlier, we maintain a **count** variable.
4. Reverse the **k** nodes while **count** is less than **k**.
5. Reverse the **k** nodes starting from current head by implementing the logic of reversing a linked list.
6. Connect the new reversed sub-list to the post result.

Pseudo Code:

```
Function kReverse(head, k) :
     if k equals 0 or k equals 1 :
          return head

     current = head
     fwd = null
     prev = null

     count = 0

     # Reverse first k nodes of linked list
     while (count < k) and (current is not None) :
          fwd = current.next
          current.next = prev
          prev = current
          current = fwd
          count = count + 1

     if fwd is not null:
          head.next = kReverse(fwd, k)

     return prev
```

Time Complexity: O(N), where **N** is the number of nodes in the linked list. As we are traversing the list only once thus the time complexity will be O(N).

Approach 2 : Using Stacks

Another way of doing this problem is by using Stacks. We can use a stack to store the linked list nodes. Initially, Push the first k elements of the linked list in stack. Pop the elements from the stack and maintain the data of the last popped node. Point the next pointer of the previous node to the top element of stack. Repeat the process, until NULL is reached.

Time Complexity: O(N) where N is the number of nodes in the linked list.