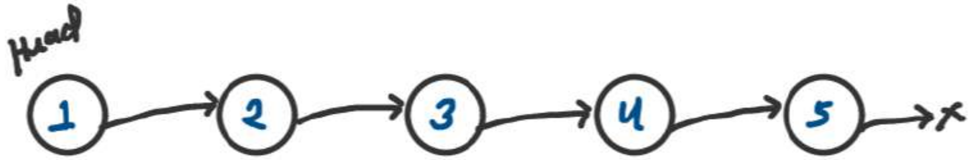
HW 08: Rotate List (Leetcode-61)

Ex



K=1 $5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow x$

K=2 $4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow x$

K=3 $3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow x$

K=4 $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow x$

K=5 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow x$

K=6

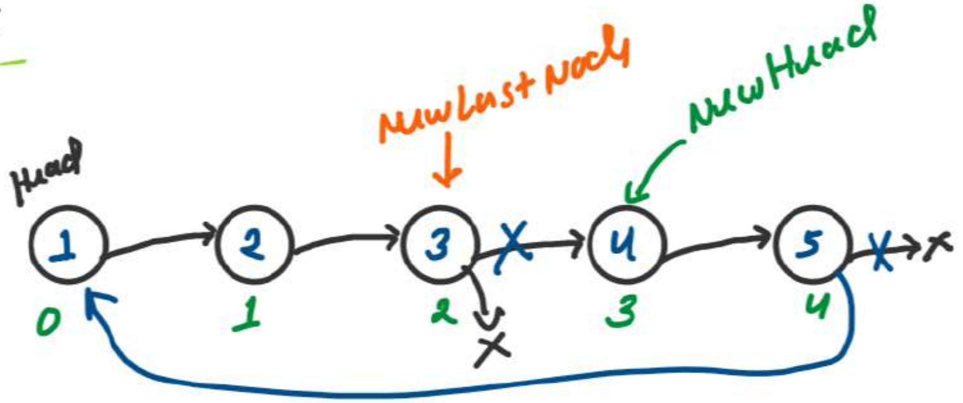
$5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow x$

K=7

$4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow x$

logic

Input



K=2

Step 1 Find length of List

Step 2 Find Actual Rotation of K

Step 3 Find newLastNode Position

Step1

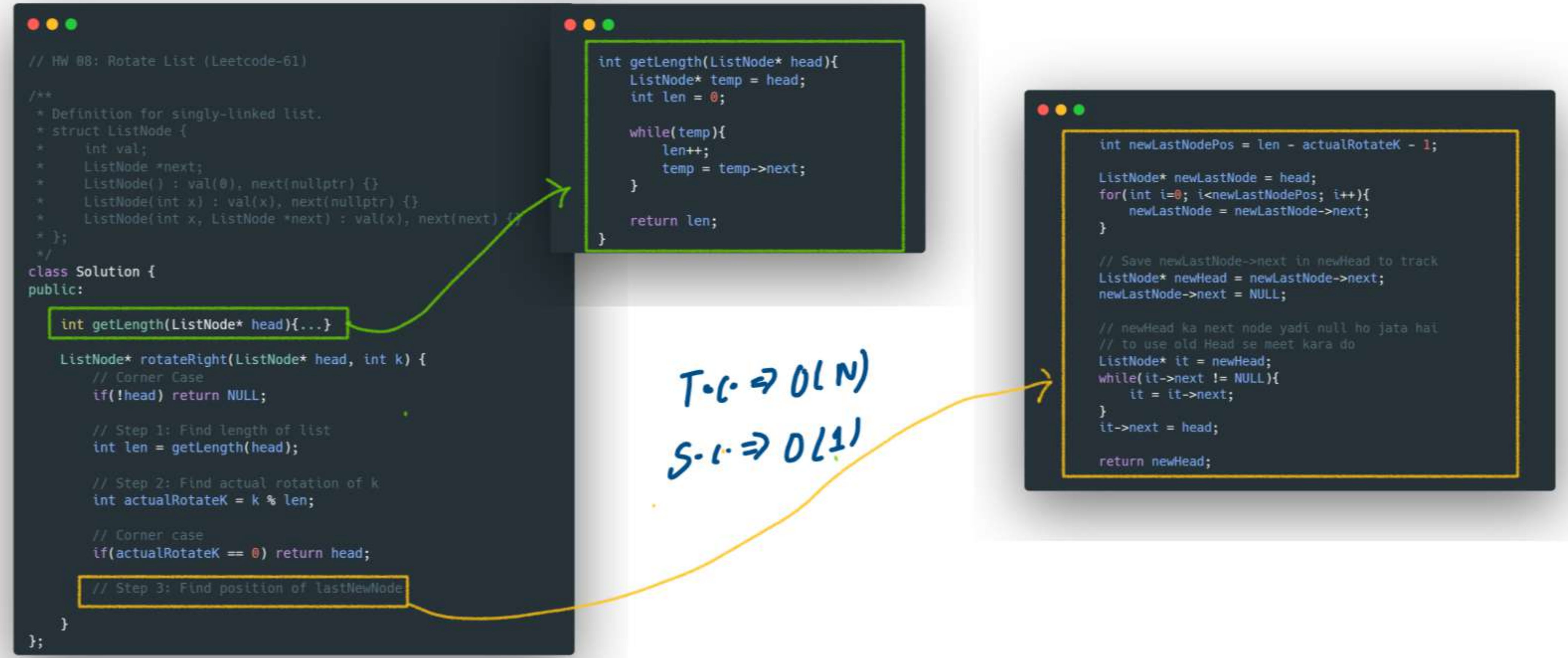
list length = 5

Step2

Actual Rotation = K % length
= 2 % 5
= 2

Step3

newLastNode posi = len - ActualRotation - 1
= 5 - 2 - 1
= 2



```cpp
// HW 08: Rotate List (Leetcode-61)

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:

   int getLength(ListNode* head){...}

   ListNode* rotateRight(ListNode* head, int k) {
       // Corner Case
       if(!head) return NULL;

       // Step 1: Find length of list
       int len = getLength(head);

       // Step 2: Find actual rotation of k
       int actualRotateK = k % len;

       // Corner case
       if(actualRotateK == 0) return head;

       // Step 3: Find position of lastNewNode

   }
};
```

```cpp
int getLength(ListNode* head){
    ListNode* temp = head;
    int len = 0;

    while(temp){
        len++;
        temp = temp->next;
    }

    return len;
}
```

```cpp
    int newLastNodePos = len - actualRotateK - 1;

    ListNode* newLastNode = head;
    for(int i=0; i<newLastNodePos; i++){
        newLastNode = newLastNode->next;
    }

    // Save newLastNode->next in newHead to track
    ListNode* newHead = newLastNode->next;
    newLastNode->next = NULL;

    // newHead ka next node yadi null ho jata hai
    // to use old Head se meet kara do
    ListNode* it = newHead;
    while(it->next != NULL){
        it = it->next;
    }
    it->next = head;

    return newHead;
```

T.C. ⇒ $O(N)$

S.C. ⇒ $O(1)$