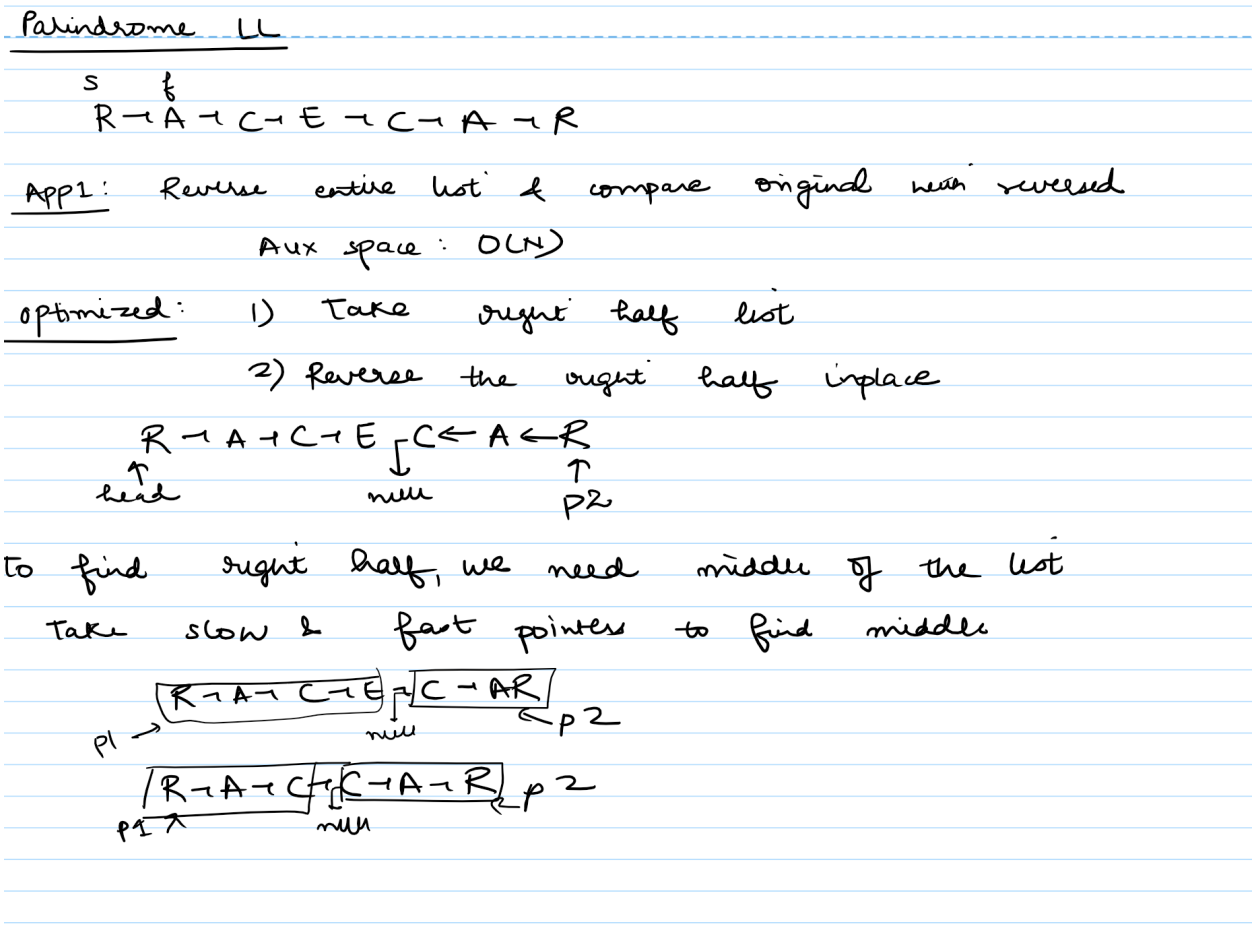
A1 https://leetcode.com/problems/palindrome-linked-list/description/

R->A->C->E->C->A->R
R->A->C->C->A->R



Reverse Half

The idea is to reverse the right half of the original list and then compare half elements. The difficulties lie in:

- Handle even and odd cases.
- How to get the right half.

// even
1 -> 2 -> 3 -> 4
left half: 1 -> 2
right half: 3 -> 4

// odd
1 -> 2 -> 3 -> 4 -> 5
left half: 1 -> 2

right half: 4 -> 5

To get the right half, we have two ways: Traversal based on the length, Slow-fast pointers. Here we use the slow-fast pointers to find the middle point.

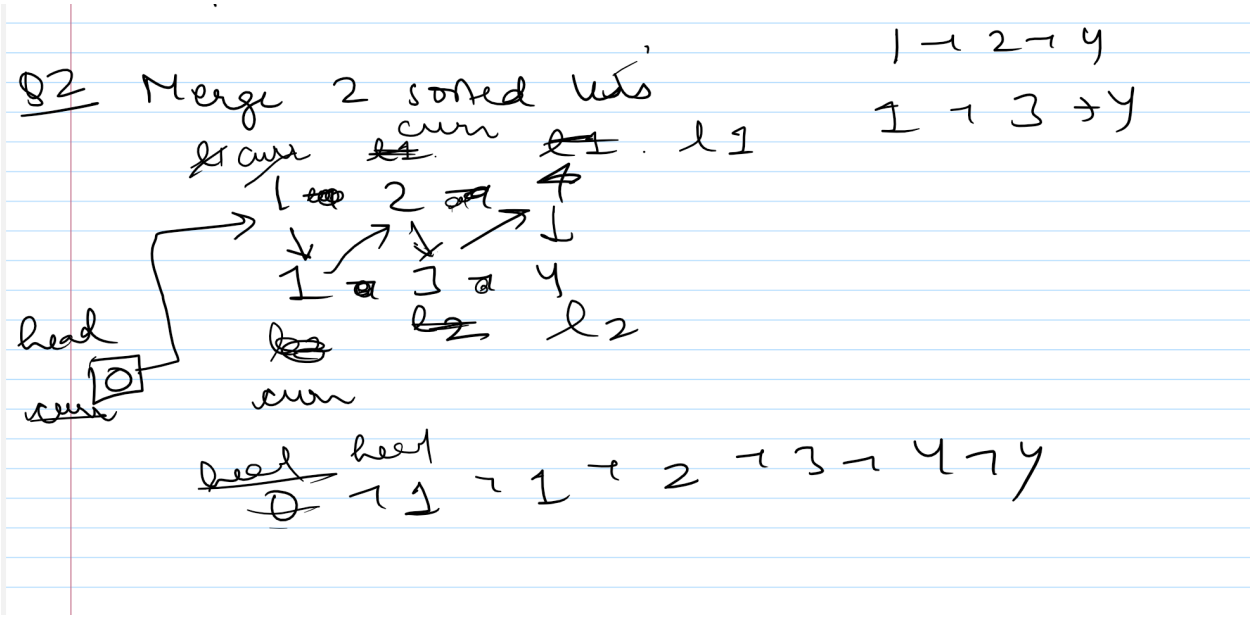
In the examples above, which nodes do we want as heads of the right half? 3 (even case) and 4 (odd case).

```
public boolean isPalindrome(ListNode head) {
  ListNode p1 = head; // for left half
  ListNode p2 = reverse(getRightHalf(head)); // for right half
  while (p2 != null && p1 != null) {
    if (p1.val != p2.val) return false;
    p1 = p1.next;
    p2 = p2.next;
  }
  return true;
}


// iteration
private ListNode reverse(ListNode head) {
  if (head == null || head.next == null) {
    return head;
  }
  ListNode p = head;
  ListNode prev = null;
  while (p != null) {
    ListNode nextTemp = p.next;
    p.next = prev;
    prev = p;
    p = nextTemp;
  }
  return prev;
}
private ListNode getRightHalf(ListNode head) {
  if (head == null) return null;
  ListNode slow = head;
  ListNode fast = head.next;  // left-leaning
  while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
  }
```

```
  return slow.next; // important!
}
```

Time: O(N)
Space: O(1) if not using recursion.

A2 https://leetcode.com/problems/merge-two-sorted-lists/description/



Iterative

1. Introduce a new node called head, which will show the head of our result list.
2. Introduce another variable called curr that which will help us iterate over the list and by doing this we won't lose the head.
3. Iterate over list1 and list2.
4. curr.next shows which value is smaller in list1 and list2.
5. Stop the iteration when list1 or list2 is null (no more element to process in them)
6. Add rest of the values from the not null list to our result array.
7. Return head.next since head is the node we generated and head.next is the start of the linked list where values from list1 and list2 begin.

```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if(list1 == null) {
        return list2;
    }

    if(list2 == null) {
        return list1;
```

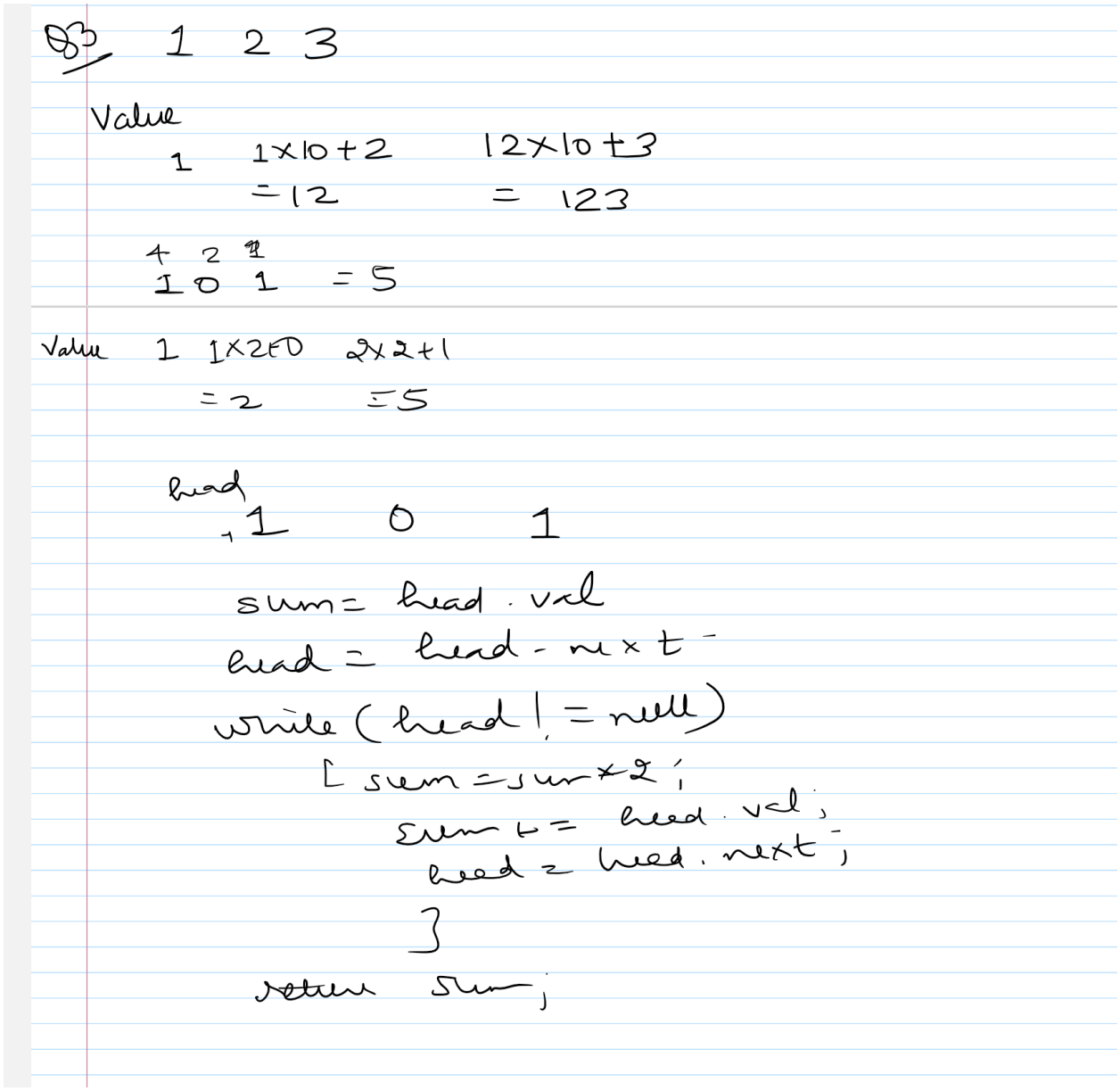
```
    }

    //Create a new list for the result list
    ListNode head = new ListNode(0);
    //We want to return the head so we should make it stay where it is
    //So introduce a new variable which will show head
    ListNode curr = head;

    while(list1!= null && list2 != null) {
        if(list1.val < list2.val) {
            curr.next = list1;
            list1 = list1.next;
        }else {
            curr.next = list2;
            list2 = list2.next;
        }
        curr = curr.next;
    }

    //Which one is not null add all of its left values.
    if(list1 != null) {
        curr.next = list1;
    } else {
        curr.next = list2;
    }
    return head.next;

  }
```

Time Complexity: O(N+M). Where N is the size of list1 and M is the size of list2. Since we iterated over list1 and list2 one pass it is O(N+M).

Space Complexity: O(1) because we only introduced two new nodes(head and curr). We used already defined nodes in list1 and list2 and just changed their next pointers.

A3

https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer/description/



Using the number 123 as an example, then from basic Math, we know this:

- 1 is in the hundreds column so it's value is 100
- 2 is in the tens column so it's value is 20
- 3 is in the ones column so it's value is just 3

Let's forget about binary for a second and focus on decimal counting since that's what humans use. So, if we were to convert the string "123" to integer 123, we'll go through each character one at a time.

When we start with the first character, 1, we don't really know it's value is yet. i.e whether it's in the millions or hundreds or thousands etc. But, we do know that if we encounter a character again, then it's value has to go up by x10.

So if we get 1, we assume it's just 1. Then we encounter 2. At this point, we realize that the previous was is no longer just a 1 but (1 x 10). So we do that arithmetic and then add the current value. This is how is looks like in tabular form.

Loop	Character	Operation	Result
1	'1'	1	1
2	'2'	(1x10) + 2	12
3	'3'	(12*10) + 3	123

Note: Operation is always the previous multiplied by the counting system. In this this, we're doing decimal so x10. If we were doing hex, it'll x16. Binary will be x 2.

Using this logic, let's come back to binary. It's exactly the same except for instead of multiplying by 10 when we encounter the next digit, we multiply by 2. So If we have a LinkeList `1 -> 0 -> 1` and apply the same logic, this is how it'll be

Loop	Character	Operation	Result
1	'1'	1	1
2	'0'	(1x2) + 0	2
3	'1'	(2*2) + 1	5

```
class Solution {
    public int getDecimalValue(ListNode head) {
        if( head == null)
        return 0;

       int sum = head.val;
       head = head.next;

        while (head != null){
            sum *= 2;
            sum += head.val;
```

```
            head = head.next;
        }
        return sum;
    }
}
```

Complexity Analysis

- Time complexity:
- $O(N)$.
- Space complexity:
- $O(1)$.

A4 https://leetcode.com/problems/intersection-of-two-linked-lists/description/

1st approach : hashing

We hash the node address of linked list 1

Once completed, iterate over list 2, if any address same, we return that

TC :O(N+M)

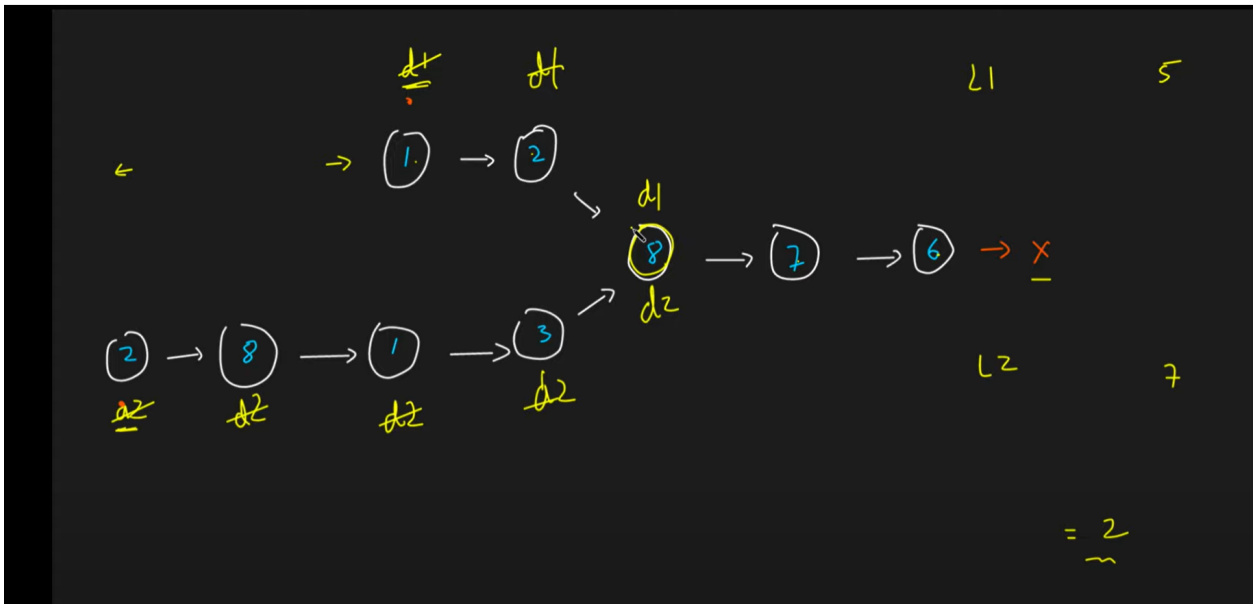
SC: O(M)

Optimal Approach : 1) We take a dummy node in both the linked lists and move them simultaneously till both move to the end. Find length of both the lists.

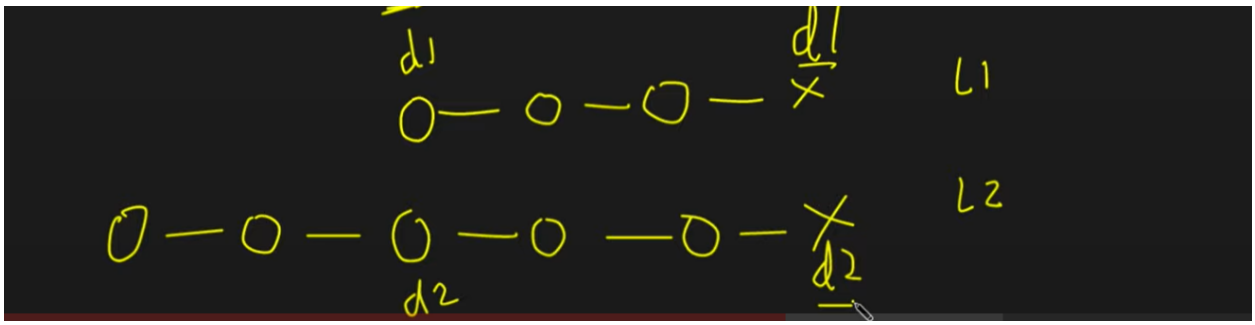
2) Find difference between 2 lists.

3) Move longer list by the difference.

4) After that simultaneously move them till they reach common node.



If there was no intersection, ultimately both dummy will reach null.



TC : O(N) (to find length) + O(N - M) (in moving) + O(M) (will move max to length of smaller LL)

Optimal 2 : 1) Take 2 dummy nodes and move them simultaneously in each iteration.

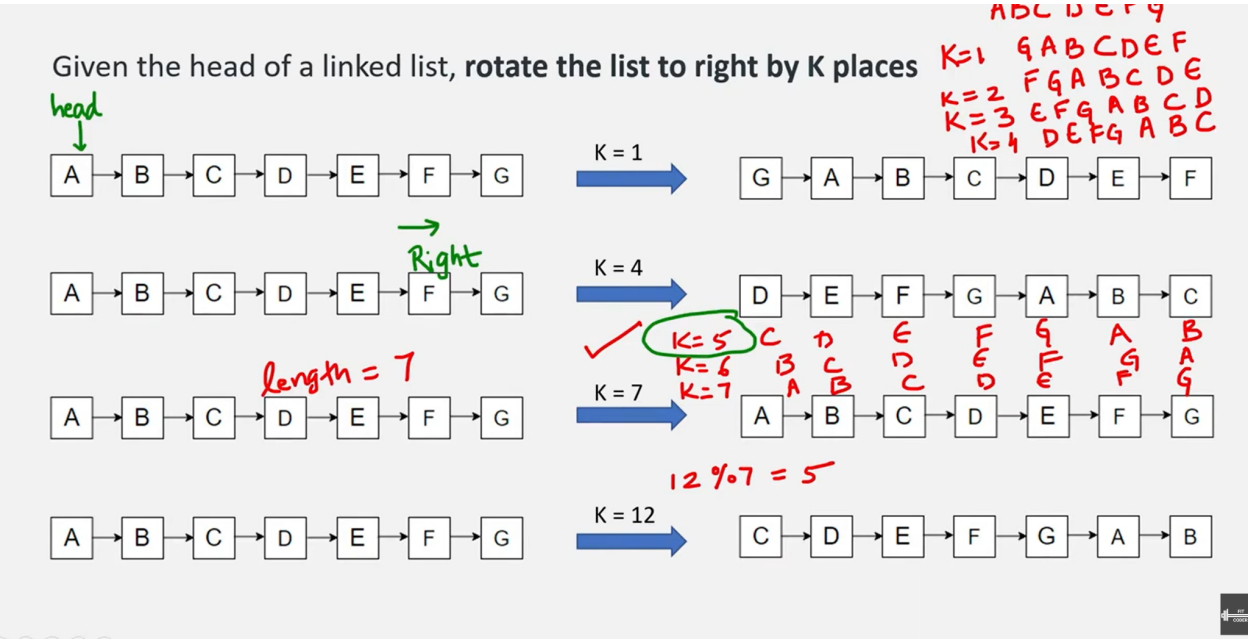
2) Moment any of the nodes reaches null, move it to the head of other list.

Why this works?
Because when once node becomes null, the only distance left for other to cover is the difference, till that time it covers the distance, we move the other pointer also, so it also covers the difference. Ultimately they will be aligned.

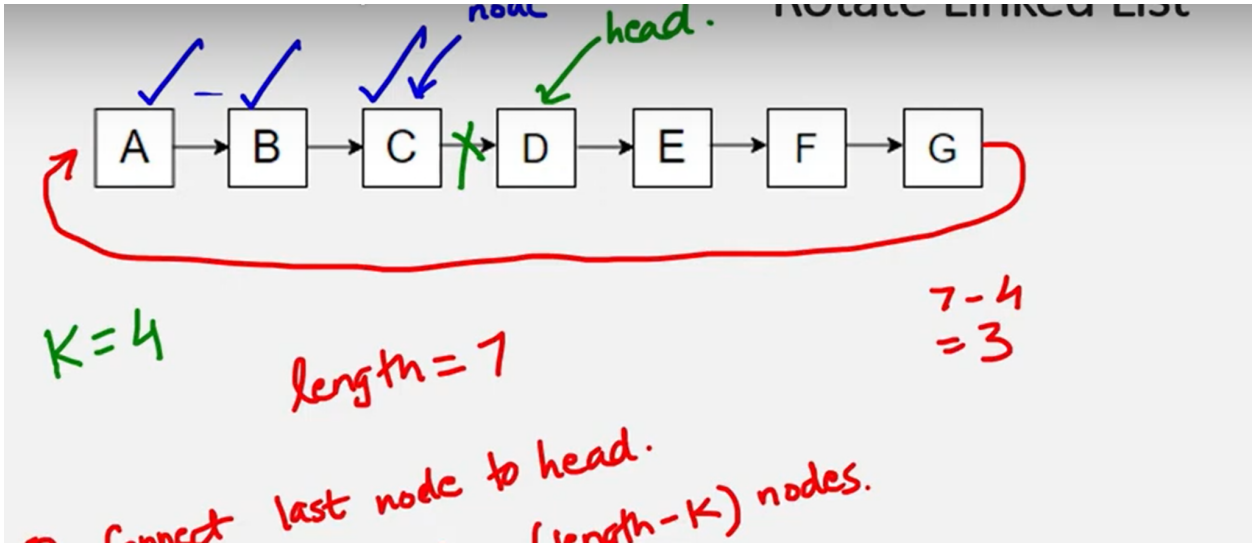
```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
```

```
    //boundary check
    if(headA == null || headB == null) return null;

    ListNode a = headA;
    ListNode b = headB;

    //if a & b have different len, then we will stop the loop after second iteration
    while( a != b){
            //for the end of first iteration, we just reset the pointer to the head of another linkedlist
        a = a == null? headB : a.next;
        b = b == null? headA : b.next;
    }

    return a;
}
```

A5 https://leetcode.com/problems/rotate-list/description/



1) In rotation, last node will come to first node. Connect last node to the first node so that we can form a cycle. Once we find the new head, we will break this cycle.
2) Last node will be (length-k) nodes from current last node.
3) Head node will be next of new last node
4) Make next of new last node as null




```
class Solution {

    public ListNode rotateRight(ListNode head, int k) {

        if(head == null || k == 0) {

        return head;

    }

    ListNode p = head;

    int len = 1;

    while(p.next != null) {

        p = p.next;

        len++;

    }
```

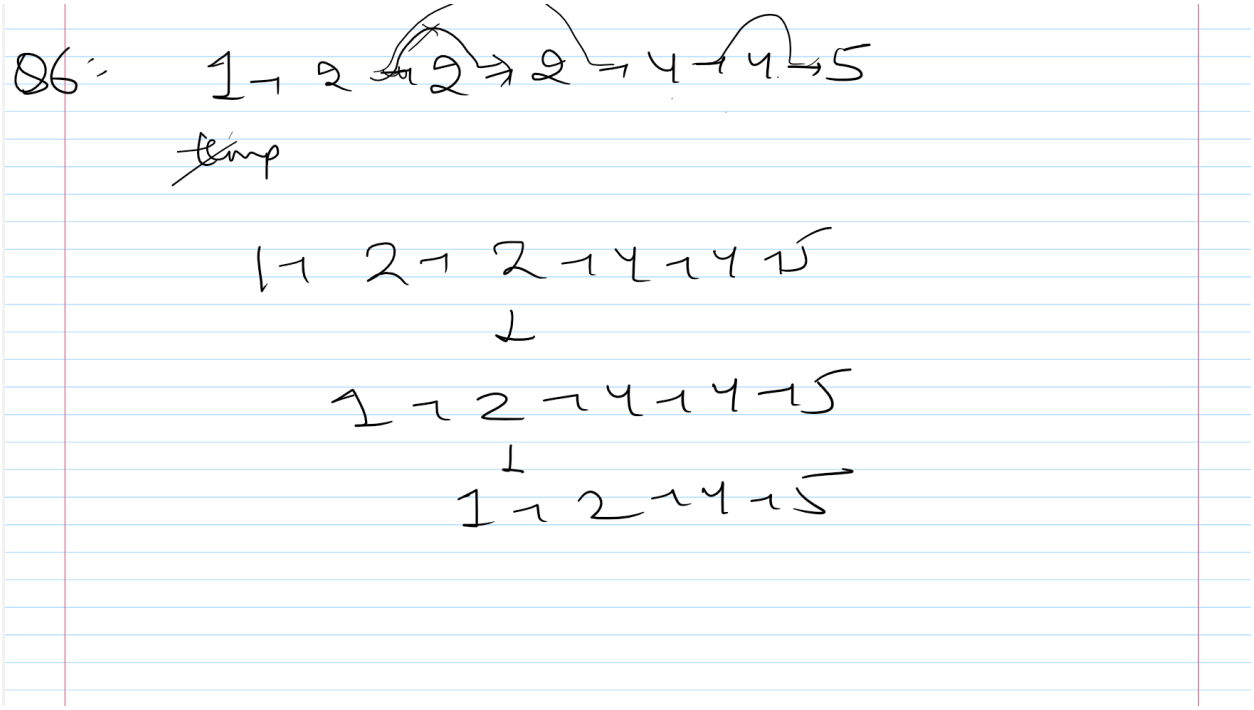
```
        // after this loop p will point to the last node
         p.next = head;


         k %= len;
        for(int i = 0; i < len - k; i++) {
            p = p.next;
        }


        // head will be the next node of current last node
         head = p.next;
         // next node of current last node will be the new head
        p.next = null;
        return head;
        }


}
```

TC :O(N)

SC : O(1)

A6

https://practice.geeksforgeeks.org/problems/remove-duplicate-element-from-sorted-linked-list/1



- Traverse the list from the head node.
- While traversing, compare each node with its next node.
- If the data of the next node is the same as the current node
    - Then delete the next node.
    - Before we delete a node, we need to store the next pointer of the node.
- Else, move forward the current.

```
Example : Let the linked list be 1->2->2->2->4->4->5.


At first current will be at 1,

     current's next is 2 i.e. current->data != 
current->next->data, move forward current.

Now current points to 2,

     current next is also 2 so we delete current's next (i.e. 
2), linked list becomes 1->2->2->4->4->5

     again we see current's next is still 2, so we delete it 
making the linked list 1->2->4->4->5

     now current's next becomes 4, so move forward current.
```

Now current points to 4,

its next is also 4 so we delete current's next making the linked list 1->2->4->5

now current's next becomes 5, so move forward current

current points to 5 and current's next is NULL so loop is terminated and we get the linked list with all

duplicate element removed

So our final linked list = 1->2->4->5.

```
class GfG
{
    //Function to remove duplicates from sorted linked list.
    Node removeDuplicates(Node head)
    {
           Node temp=head,prev=null;
           if(head==null)
           {
               return null;
           }
           if(head!=null && head.next==null)
           {
               return head;
           }
           while(temp!=null)
           {

               if( temp.next!=null &&temp.data==temp.next.data)
               {
                  temp.next=temp.next.next;
               }
               else
               {
                   temp=temp.next;
               }

           }
           return head;
    }
}
```