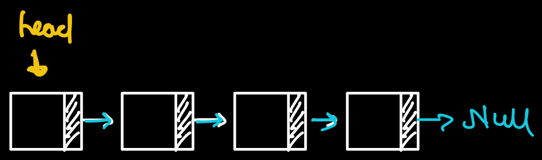
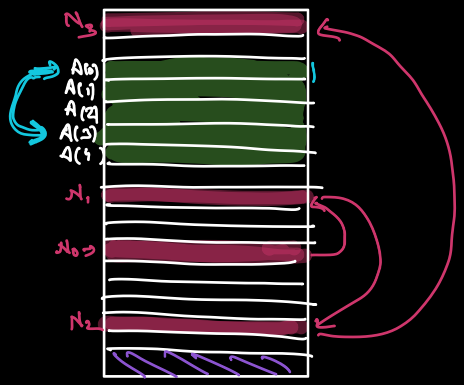
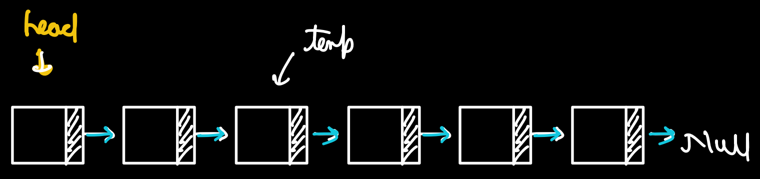




```
Class Node {

    int data;
    Node next;
    Public Node (int a) {
        this.data = a;
        this.next = null;
    }
}
```

Q Given a LL. Return the length.



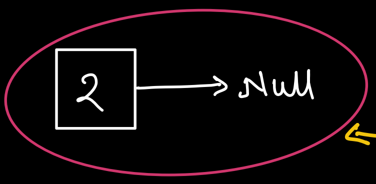
c = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

```
int size( Node head) {

    Node temp = head;
    int count = 0;
    While ( temp != Null) {
        Count++;
        temp = temp.next;
    }

    ret cout;
}
```

temp = new Node(2);

Ref/Name

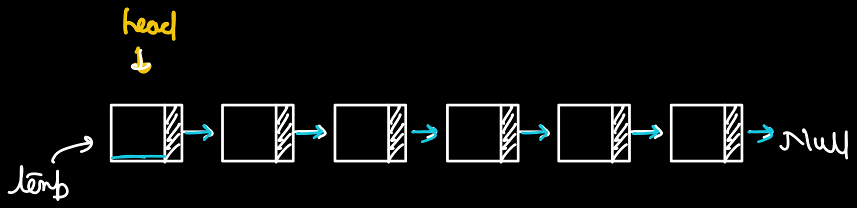
Allocates Memory



Dangeling pointer

Garbage collection

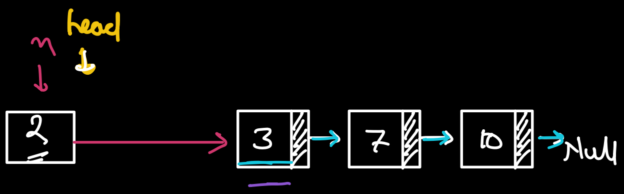
temp = head;



Insert values

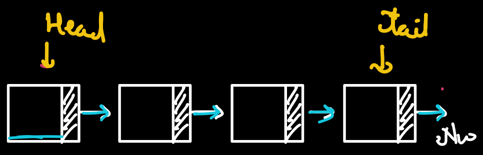
at start of a LL

TC: O(1)



```
Node n = new Node (x);
n.next = head;
head = n;
```

at end of LL



TC: O(N)

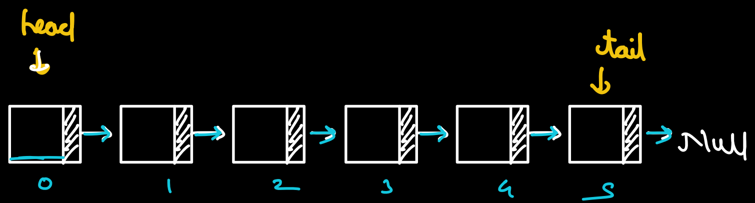
↓

O(1) (using tail)

Insert at Kth Pos

Iterate till (K-1)

K = 0
K = 1
K > 1



TC : O(N) ⇒ (Tail -1)

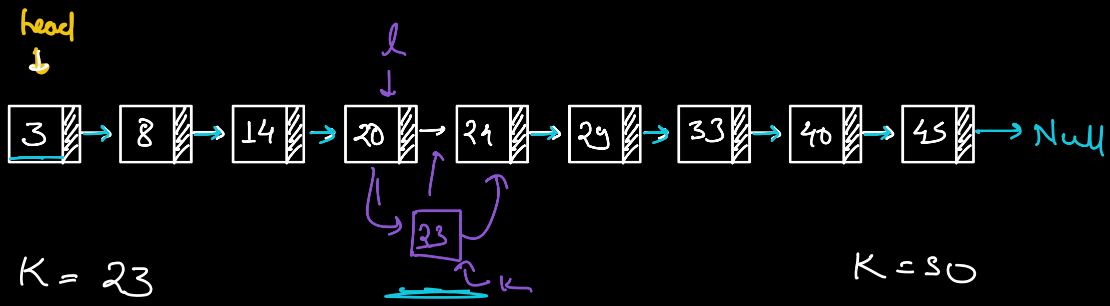
Edge Cases

- Head is Null
- LL has only 1/2/3

Insertion / Deletion

→ at/from start
→ at/from end.

Q Given a linked list sorted in ASC order.
Insert a val in correct position in sorted order.



K = 23

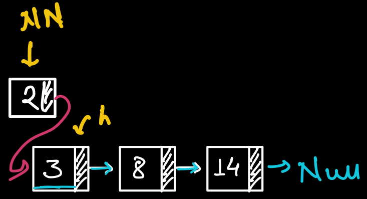
K = 50

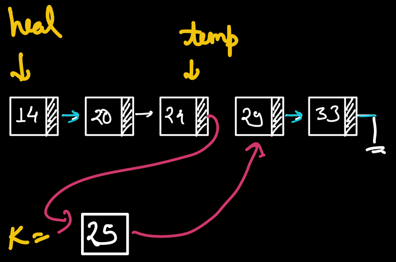
Find the last node with a val smaller than K
↳ l ⇒ temp

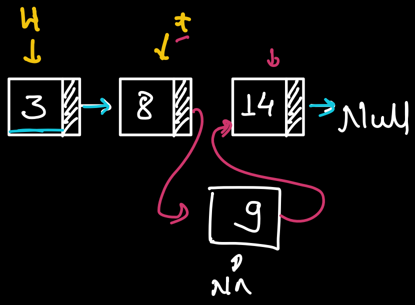
→ K.next = l.next
l.next = K

$$\begin{tikzcd}\text{Node} \arrow[d] \\ \text{New head}\end{tikzcd}$$

```
insert In SortedOrder ( Node head, K) {

    Node newNode = new Node (K);

    // head is Null
    if( head == Null) {
            ret newNode;
    }
    // if the node is being inserted at start .
    if ( K <= head.val) {
        newNode.next = head;
        ret newNode;
    }

    Node temp = head;

    While ( temp.next != Null && temp.next.val < K){
          temp = temp.next;

    }

        newNode . next = temp.next;
        temp .next = newNode
        ret head;
}
```





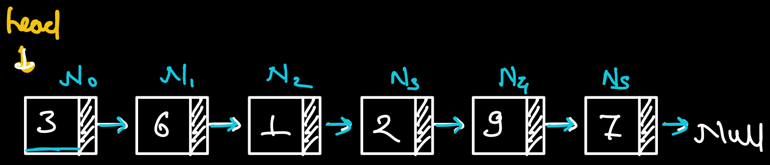


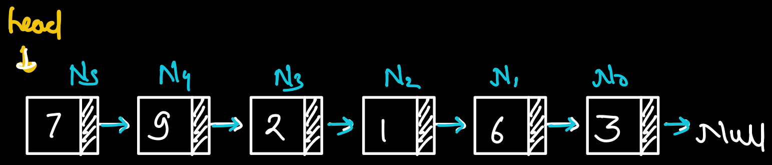
Q Reverse the linked list.

- In place ⇒ without using any extra space

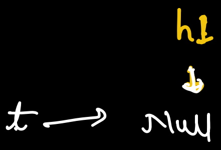
SC : O(1)

- Changing the value of a node is not allowed.

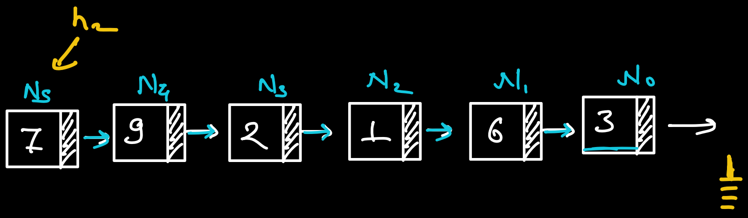




N.next;
N.val

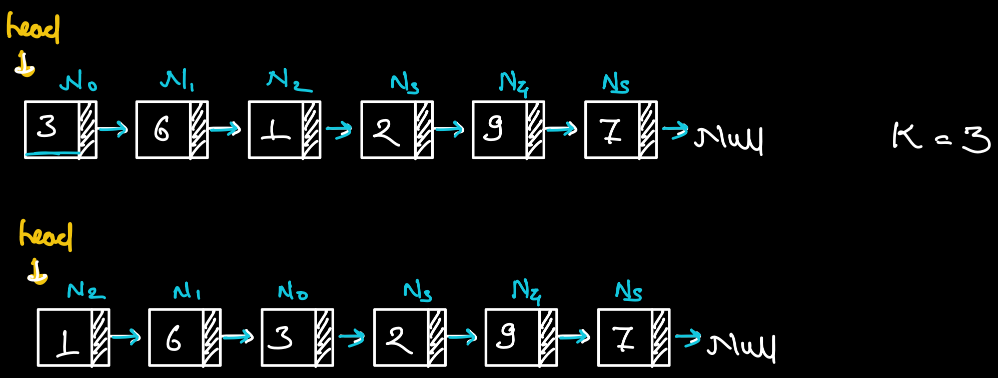


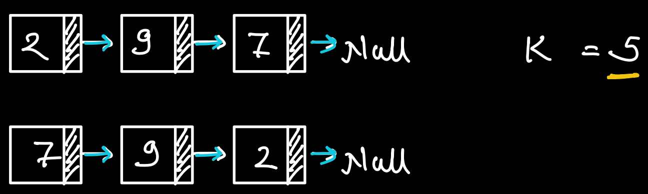
```
while( h1 != Null) {
    t = h1;
    h1 = h1.next;
    t.next = h2
    h2 = t
}
ret h2;
```

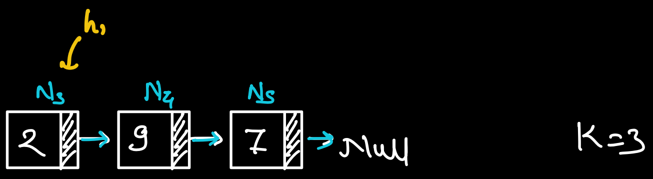


```
Node reverse ( Node head) {
    Node h2 = null , h1 = head , t = h1 ;
    While ( h1 != Null) {
        t = h1;
        h1 = h1. next;
        t. next = h2;
        h2 = t;
    }
    ret h2;
}
```

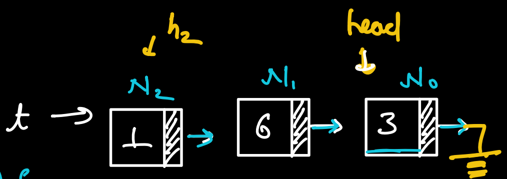
Q Given a LL. Reverse the first K nodes.







K=3



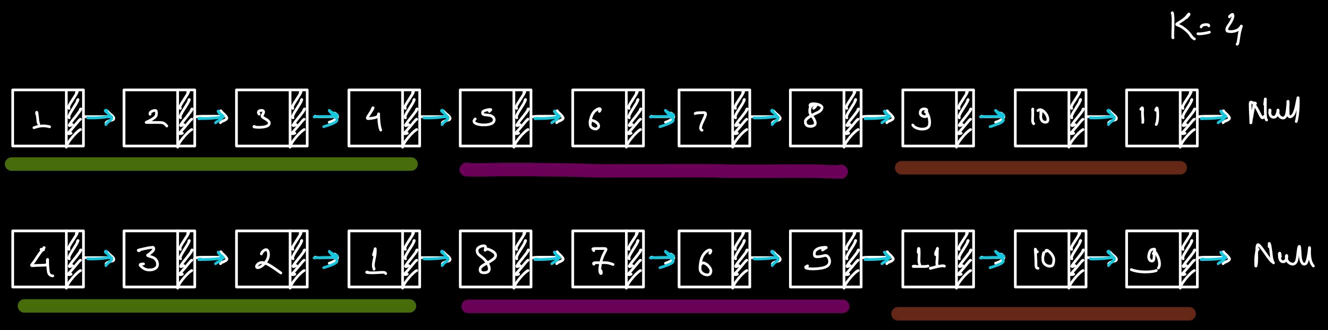
```
Node reverseFirstK( head, K) {
    if (K==0 || head==Null) ret head;
    h2 = Null,  h1 = head ;
    while( h1 != null && K > 0) {
        t = h1,
        h1 = h1.next;
        t.next = h2;
        h2 = t;
        K--;
    }
    head.next = h1;
    ret h2;
}
```

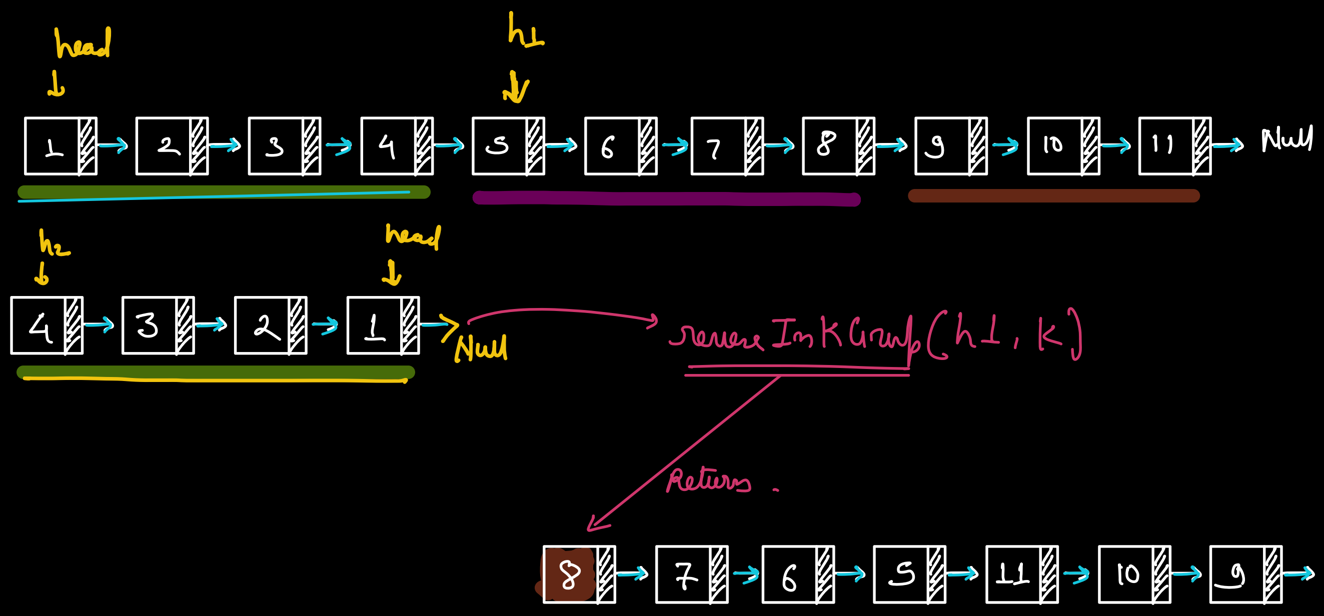
If K=0 ?

→ ret Null

Google Reverse in K Groups

Given a LL, Reverse all sub-lists of size K





```
Node    reverseInKGroups (Node head, K) {
        //Assumption : reverseInKGroup (node, K) will reverse
                       all groups of size K in the list starting
                       from node. & returns the new head.
            if( K <= 1  ||  head == null)
                                  ret head;
            Count = K;
        //Reverse the 1st K nodes
                 h2 = Null,  h1 = head ;
                 while( h1 != null && K > 0){
                      t = h1,
                      h1 = h1.next;
                      t.next = h2;
                      h2 = t;
                      K--;
                 }
            head.next = reverseInKGroup(h1, count);
            ret h2;
}
```

TC : O(N)

SC : O(N) [Size of rec stack = N/K

$K = 2, \rightarrow N/2 \approx O(N)$