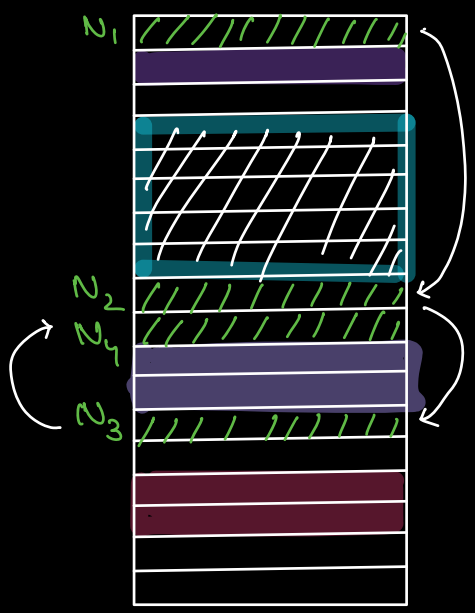
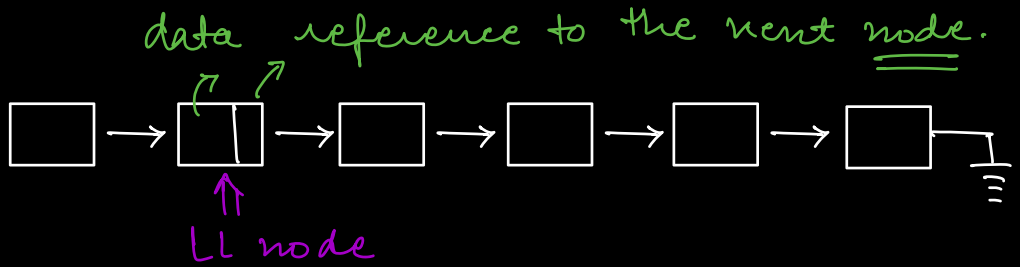
array ⇒ Continous memory locations.

⇒ O(1) random access.

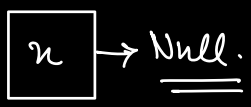


* Worst case TC in dynamic array ⇒ O(N)
* Amortized TC of insertion ⇒ O(1) in dynamic array.

Linked list

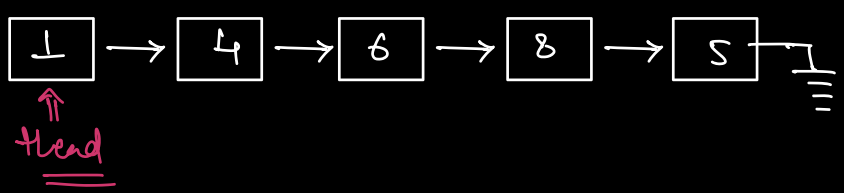


```
Class Node {
    int data;
    Node nent;
    Node(int n) {
        data = n;
        nent = Null;
    }
}
```

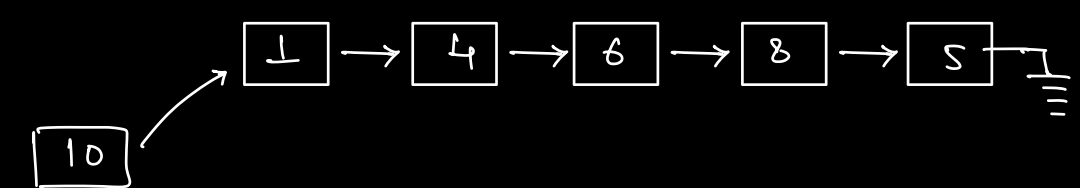


Insertion in L.L

- front
- k^{th} position
- end



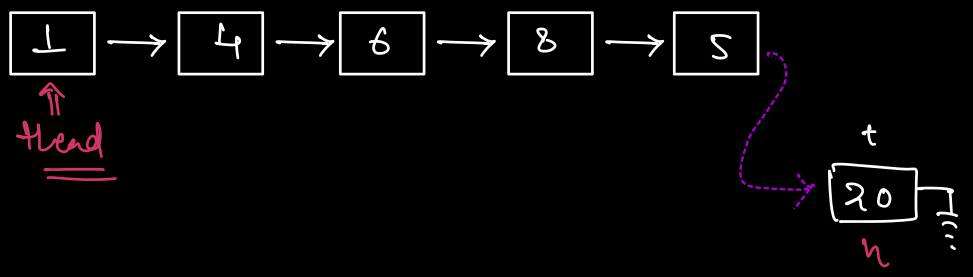
① Insert at head.




```
Node n = new Node(10)
n.nent = head
head = n;
```

⇒ O(1)

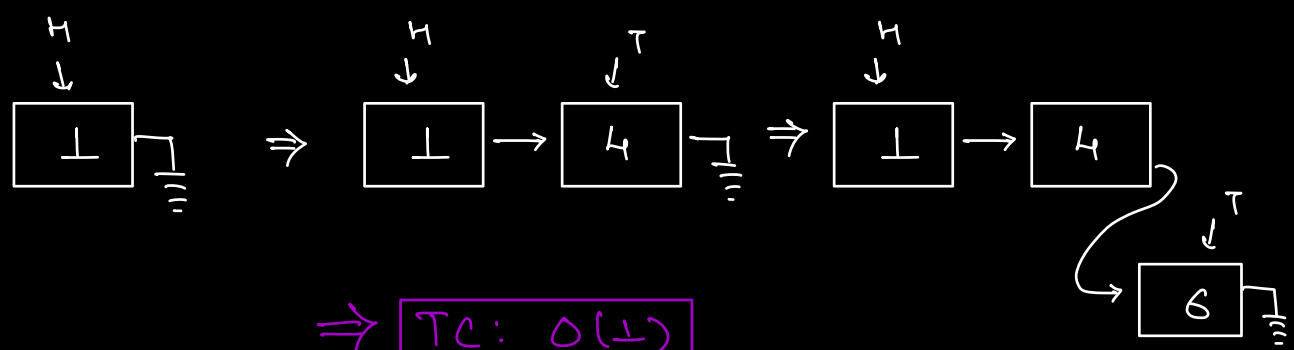
② Insert at End




```
Node n = new Node(20);
tail = head;
while (tail.next != Null) {
        tail = tail.next;
}
tail.next = n;
tail = n
```

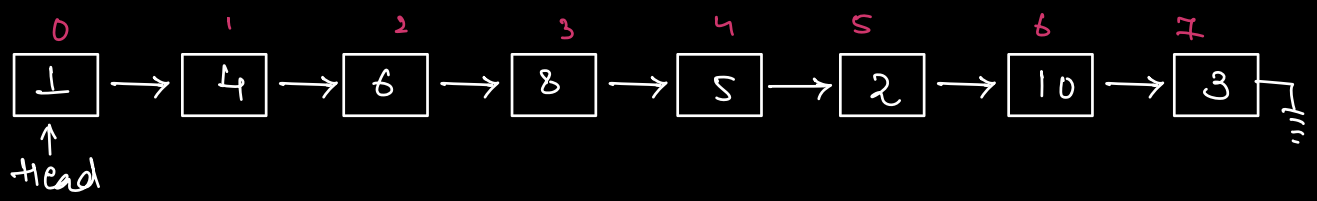
TC: O(N)

TC ⇒ O(1) { by maintaining the tail reference }

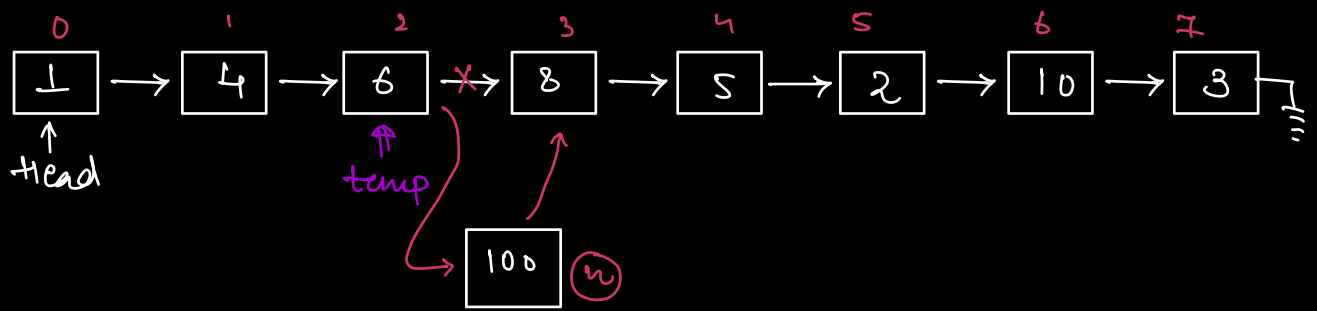


⇒ TC: O(1)

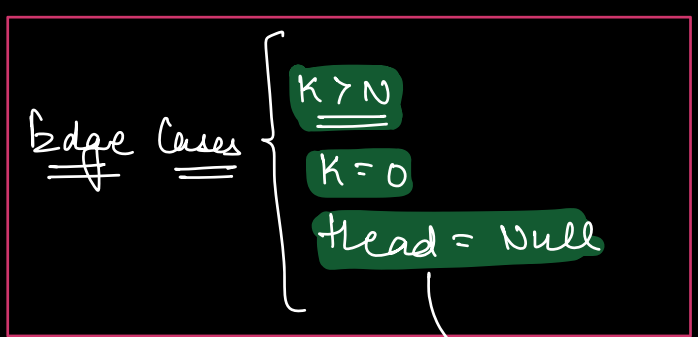
③ Insert at K^{th} Position



$K = 3 \Rightarrow data = 100$

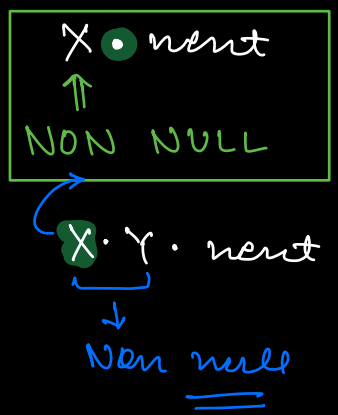



```
n.next = temp.next;
temp.next = n
```

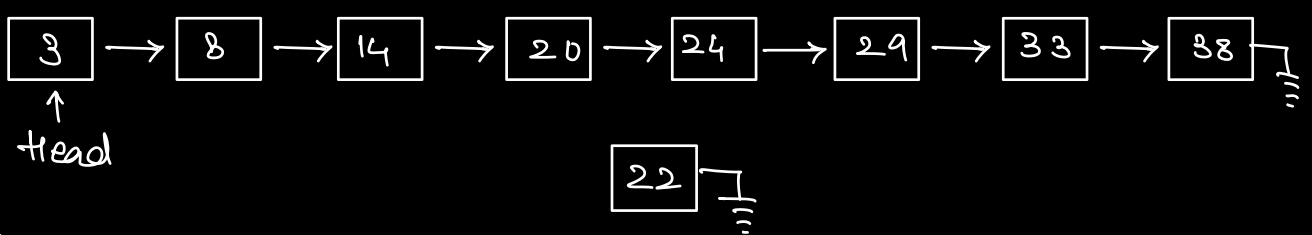


Edge Cases :

1. head = null;
2. N = 1 / 2 / 3
3. Problem specific



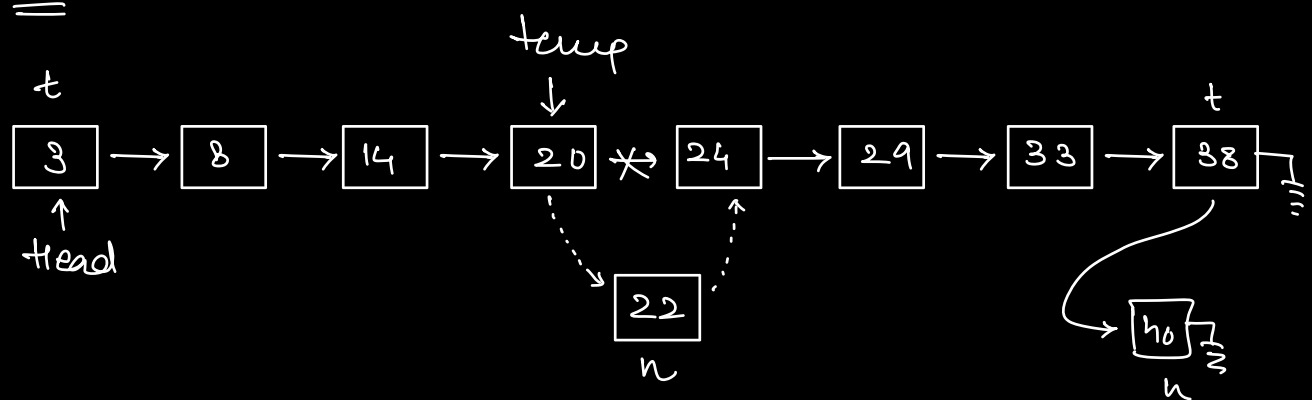
Q. Given a L.L sorted in ascending order Insert a value at its correct position in sorted order.



K = 22

⇒ Find the last node with data < K.

K = 2



head of new LL

Node

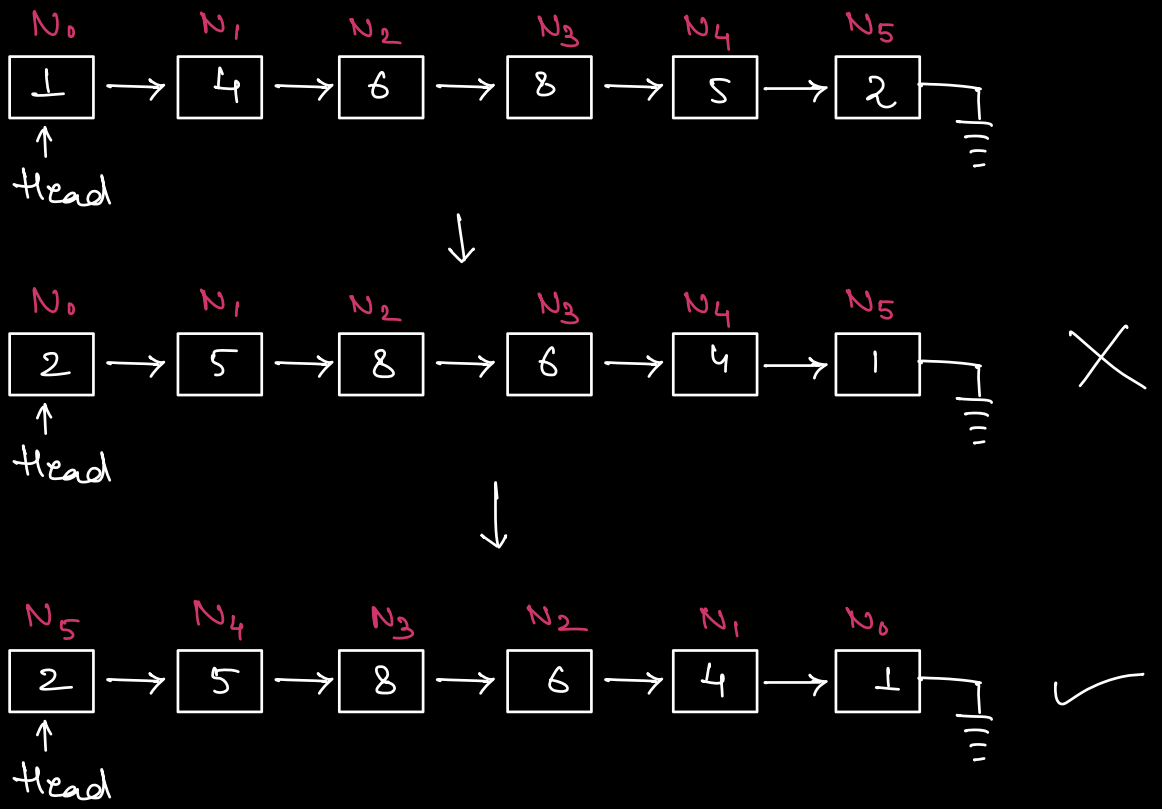
```
insertInSortedOrder(head, k){
Node n = new Node(k);
if (head == null){ // LL is Empty
        return n;
}
if (k <= head.data) {
        //insert at front;
        n.next = head;
        return n;
}
temp = head;
while (temp.next != null &&
                temp.next.data < k) {
        temp = temp.next;
}
n.next = temp.next;
temp.next = n;
return head;
}
```

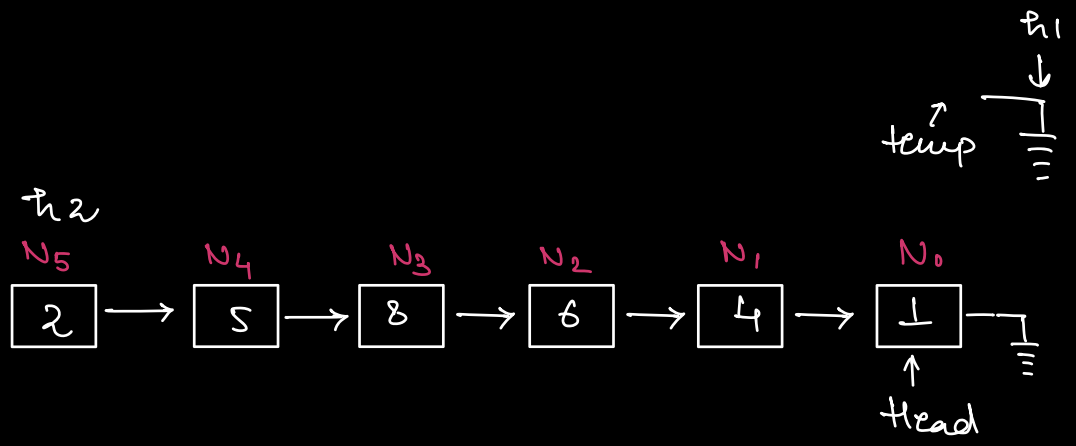
TC: O(N) / SC: O(1)

Q. Reverse the given L.L

(Expected S.C : O(1))

* Changing the value of nodes isn't allowed.

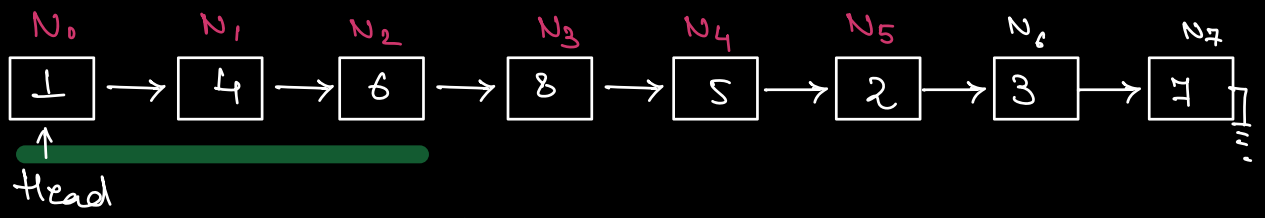




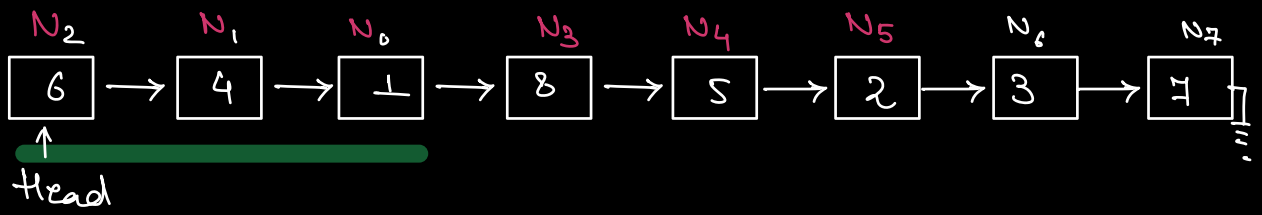

```
Node reverse (head) {
    if (head == NULL) return null;
    h1 = head , t = h1 , h2 = NULL;
    while (h1 != NULL) {
        temp = h1;
        h1 = h1.next;
        t.next = h2;
        h2 = temp;
    }
    return h2;
}
```

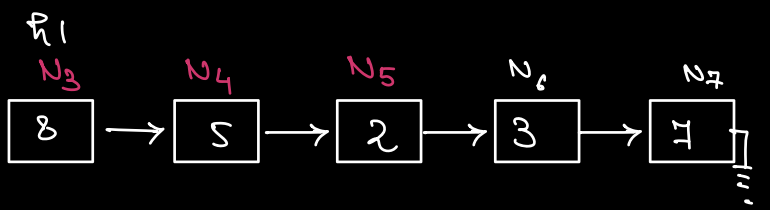
TC : O(N)
SC : O(1)

Q: Given a L.L, Reverse the first K nodes of the L.L.

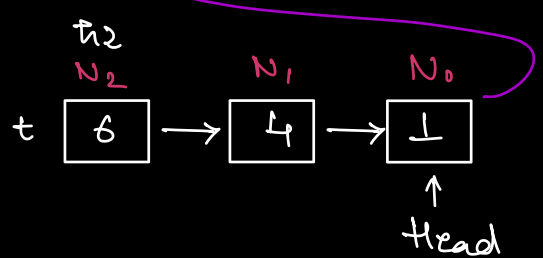


K=3





K=3 2 1 0



head.next = h1;

```
Node reverseFirstKNodes(head, K) {
    if (K==0 || head == NULL) return head;
    h1 = head, t = h1, h2 = NULL;
    while (K>0 && h1 != NULL) {
        temp = h1;
        h1 = h1.next;
        t.next = h2;
        h2 = temp;
        K--;
    }
    head.next = h1;
    return h2;
}
```

TC: O(N) / SC: O(1)

Q. Reverse L.L in group of K nodes.

Google: Reverse K Groups.