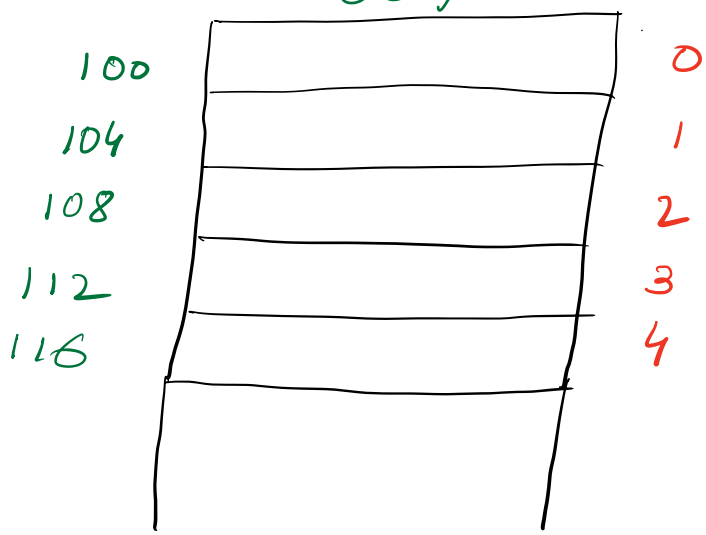
Most commonly used data structure? Array

Why? ar[i] ⇒ directly access.

O(1)

int ar[]

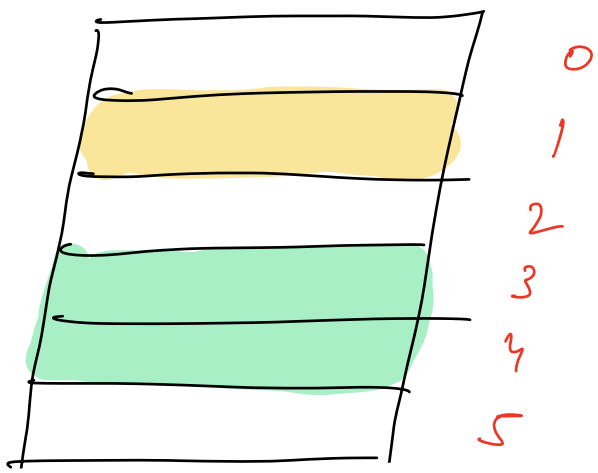


i^{th} index →

$100 + 4i$

Java specific

Same O(1) time for arraylist. And we can add elements dynamically

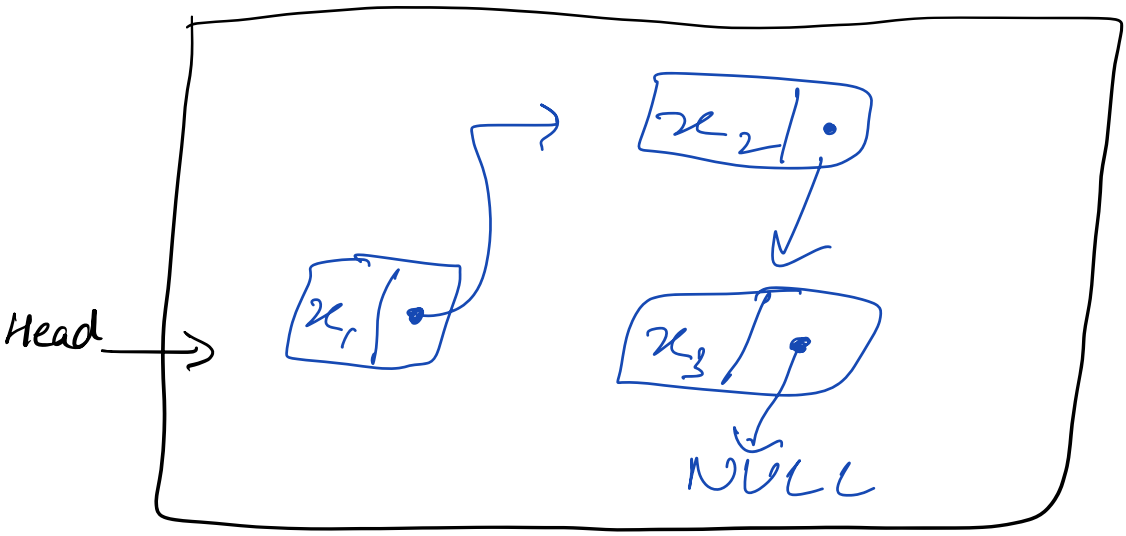


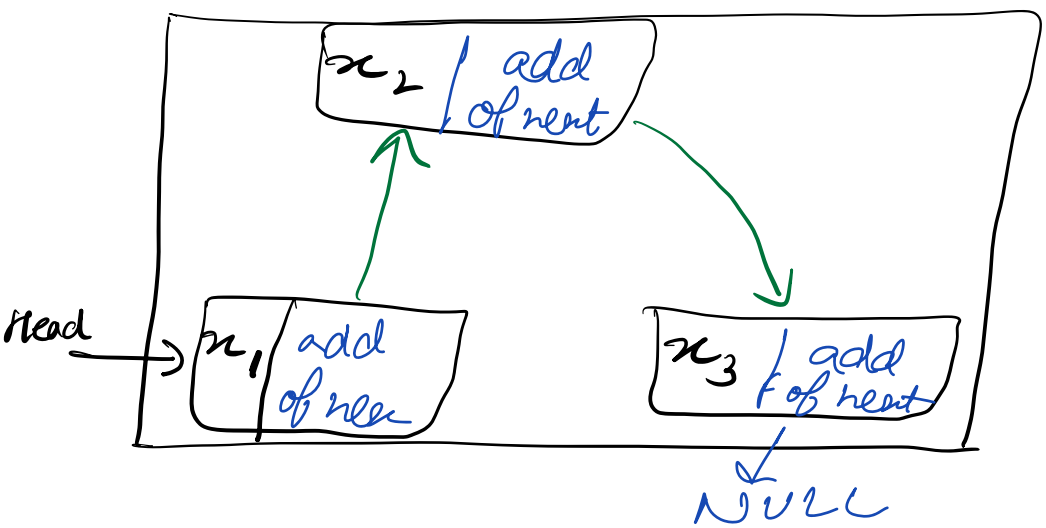
wasting space

To use memory properly and reduce space wastage → Linked list

However, we will have to compromise on O(1) access time.

How does it work ?





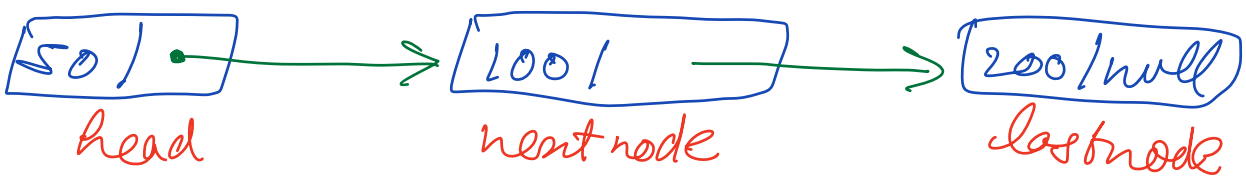
```
class Node {
    int data
    Node next

    Node (int x) {
        this.data = x
        this.next = null
    }
}
```

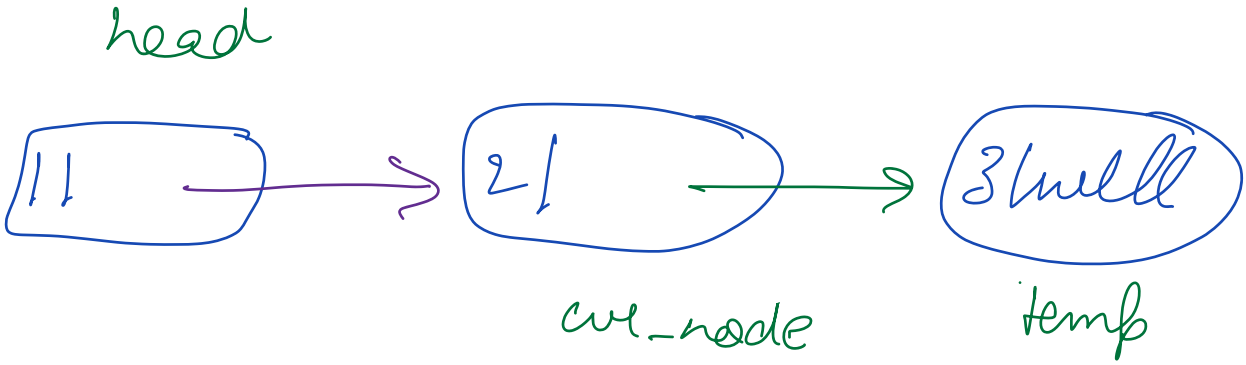
4B
8B
tot = 12B

```
Node head = new Node(50)
Node nextnode = new Node(100)
head.next = nextnode
Node lastnode = new Node(200)
nextnode.next = lastnode
```

Q. `head.next.next = lastnode`

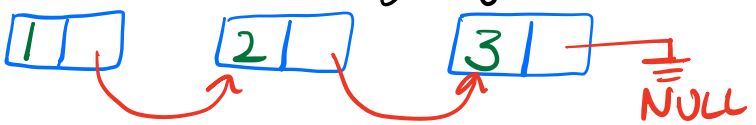


- Elastic Search uses Linked lists to create inverted indexes.

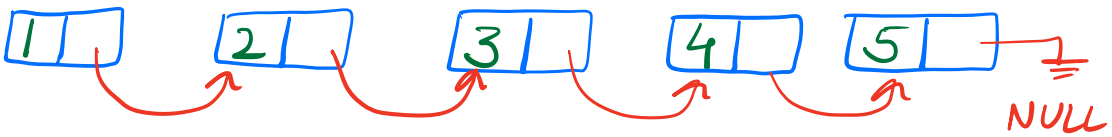

head
1
2
3 null
cur_node
temp

Q1 Create linked list of size N with values 1 to N

N=3



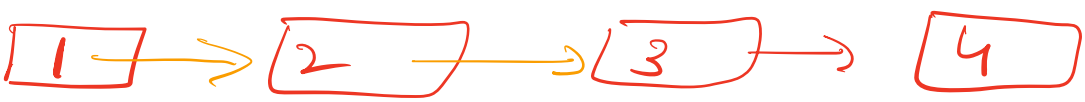
N=5



Return the starting node of the LL

```
Node createLL (int n) {
    Node head = new Node (1)
    Node temp = head        good practice
    for (i=2 ; i<=n ; i++) {
        Node cur_node = new Node (i)
        temp.next = cur_node
        temp = cur_node
    }
    return head
}
```

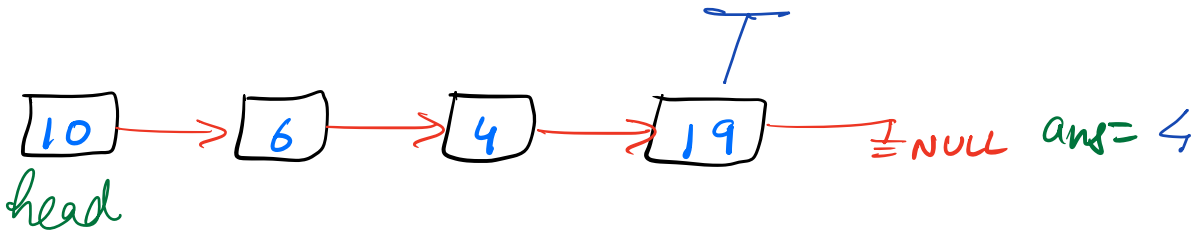
SC: O(n)
TC: O(n)



TC:

SC:

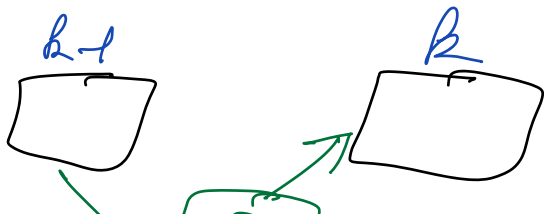
Q2 Find the size of given Linked List



C = 0 1 2 3 4

```
int size ( Node head ) {
    Node temp = head   // good practice
    int c = 0
    while ( temp != NULL ) {
        c++
        temp = temp.next
    }
    return c
}
```

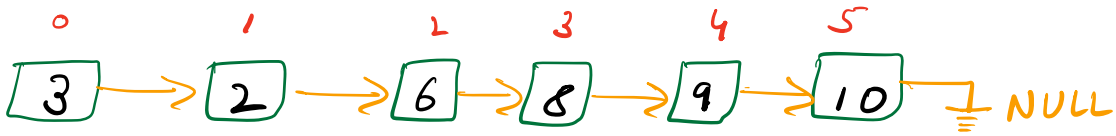
TC: O(N) SC: O(1)


k-1
k

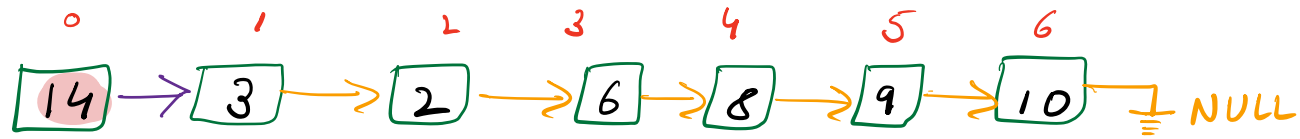
Y (20)

Q3 Insert a node of value x at the K^{th} pos of a Linked List

Ex:



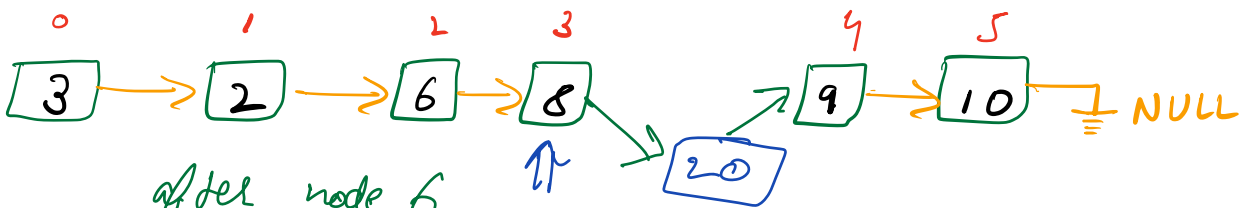
Case1 K=0 x = 14



new node

after node k-1

Case 2 K = 4 x = 20

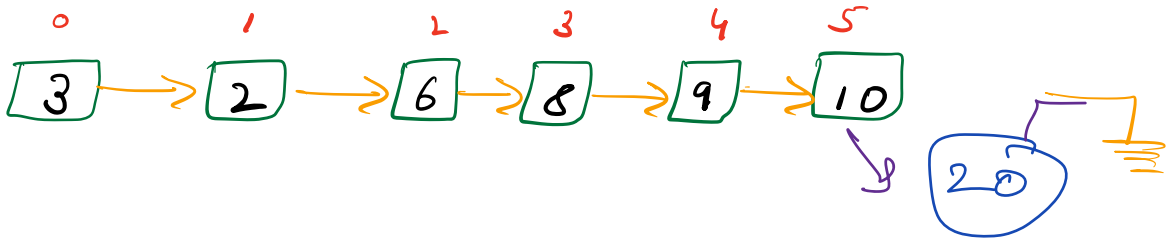


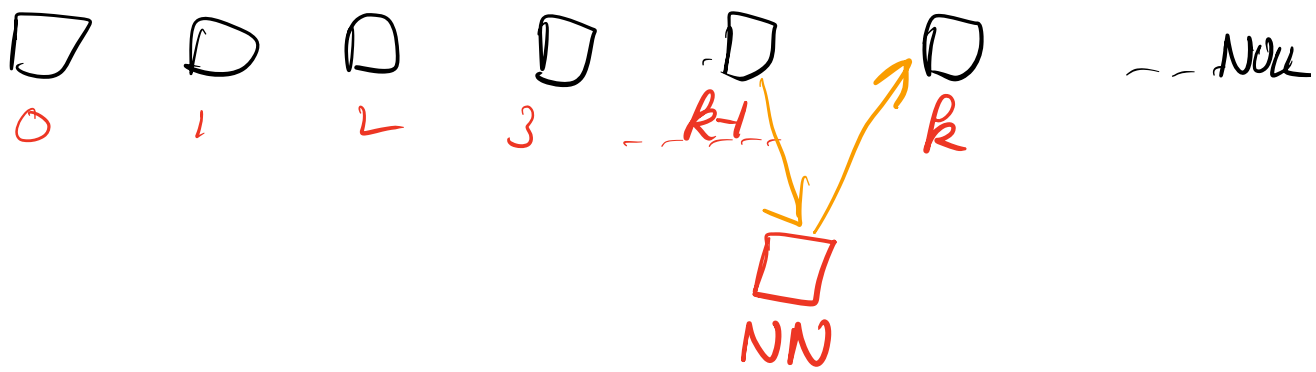
after node 6

Case 3 K = 7 x = 15

If (K > size of LL) impossible

Case 4 K = 6 x = 20



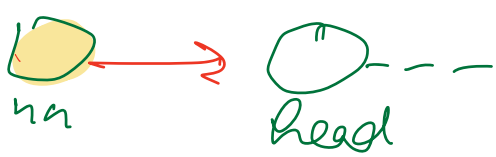


Node insertKpos(int k, int x, Node head){

if(K > size(head))
return head

Node nn = new Node(x)
Node temp = head. (good practice)

if(k==0){
nn.next = head
return nn



}

for(i=0; i<k-1; i++){
temp = temp.next
}

temp k-1

// Temp is $k-1^{th}$ node

Node k^{th} node = temp.next

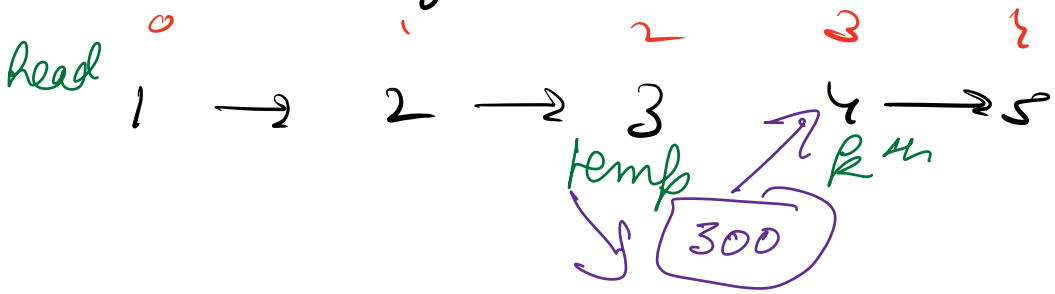
temp.next = nn

nn.next = k^{th} node

return head

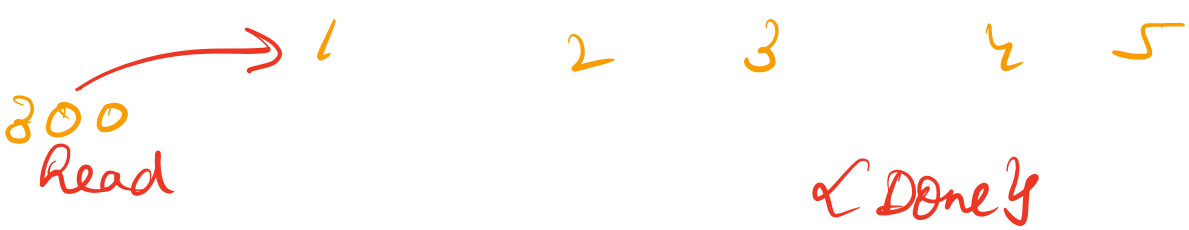
TC: O(N) SC: O(1)

Dry run



300 at pos 3

Done



Done

Q1 Print the rev LL

1 → 2 → 3 → 4 ans= 4 3 2 1

assumption ⇒ print the rev

```
void printRev( head ){
    if( head == null )
        return;
    printRev( head.next )
    print (head.data)
}
```

1 → 2 → 3 → 4

1
printRev(2)
print (1)

2
printRev(3)
print(2)

3
printRev(4)
print (3)

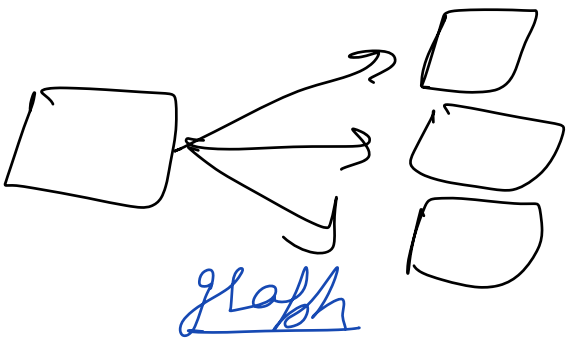
⇒ 4 3 2 1

4

buildRev(null, 1)

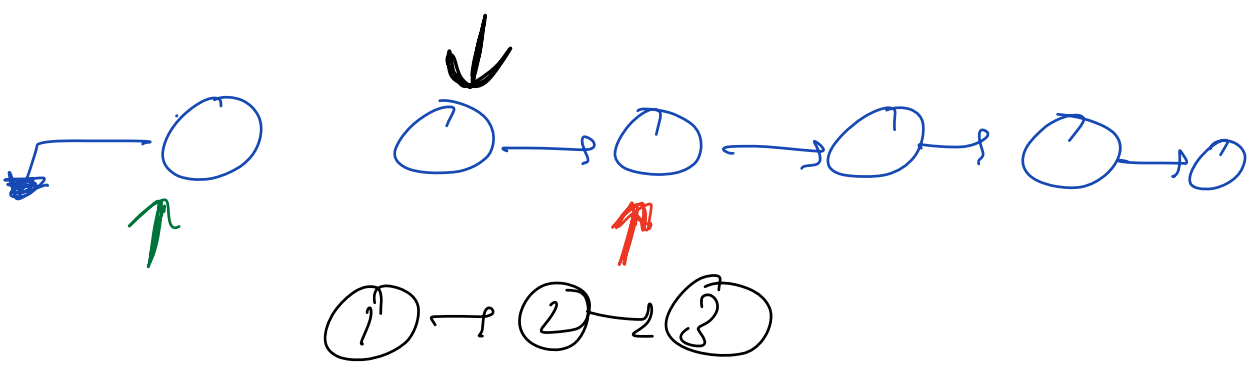
TC : O(N)

SC : O(N)



new

heap memory



[1, 2, 3] ⇒ [3, 2, 1]