Don't wait for the right time.

Create it.

→ Why linked-linked ?

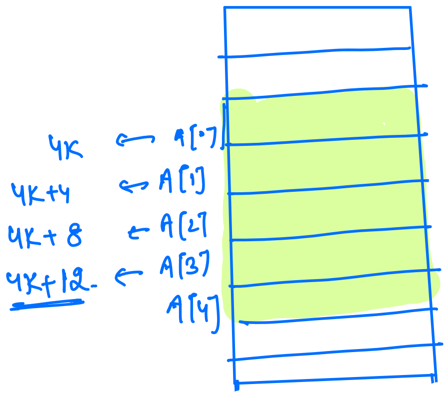
→ Insertion in linked list
  - → At start
  - → At end
  - → At K^th^- index

→ Deletion in linked list (#idea)
  - → At start
  - → At end
  - → At K^th^- Index.

Arrays.

arr[5].

{ T.C to access any random element of array = O(1) }



arr[3] ?

⇓

contiguous space for 3 integers is not available in this given situation.

✗ Not allowed.

(base address) + (idx * 4) ⇐

[→ arr[N].
→ No. of elements ≤ N.]

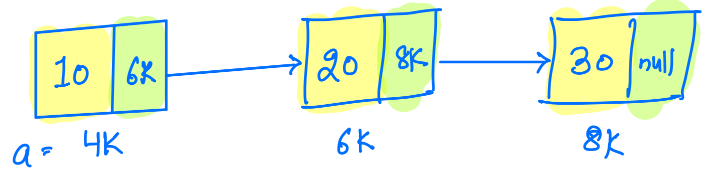
Arrays.

	At start.	At end.	K^{th} index.
Insertion	O(N)	O(1)	O(N)
Deletion	O(N)	O(1)	O(N)

{ shifting }

LinkedList-

```
class Node {
    int val;
    Node next;
    Node (x) {
        val = x
        next = null
    }
}
```




```
Node a = new Node(10);
a.next = new Node(20);
a.next.next = new Node(30);
a.next.next.next = new Node(40);
```

```
Node a = new Node(10);
Node b = new Node(20);
a.next = b;
Node c = new Node(30);
b.next = c.
```

Music Player

* Favourites.

→ Song-1
→ Song-2
|
|
→ Song 50

* Search engines.

iph → [iphone 14, iphone 13]
—, —, —

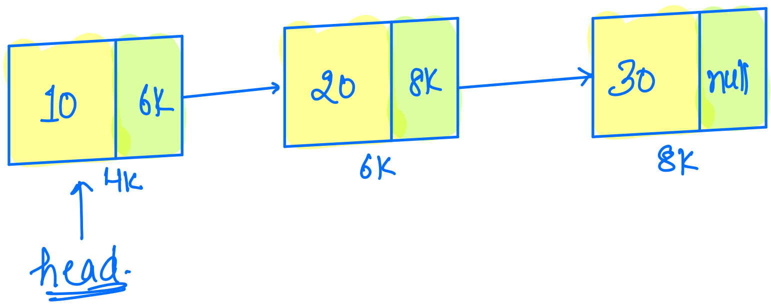
[D.S → inverted index.]

→ Appendix.
→ Glossary.

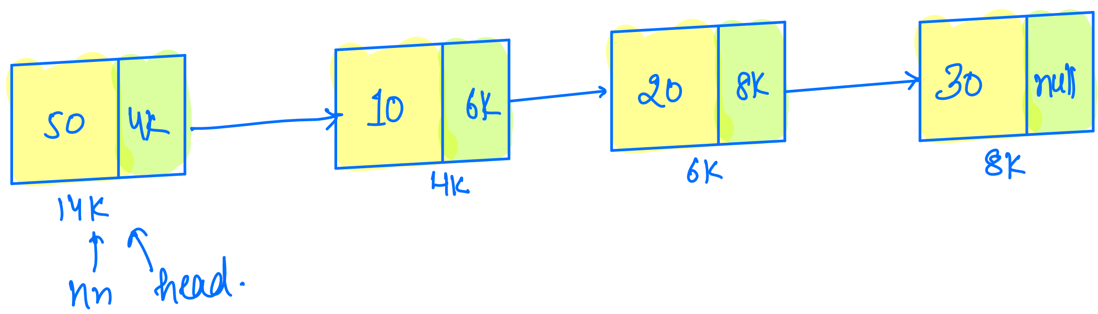
galaxy → [—, —, —, —,]

Insertion At Start.

Node head
⇓
reference of first node.



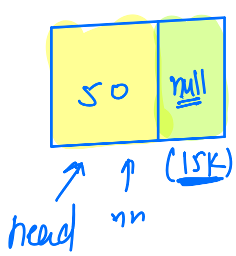
→ insert 50 at start



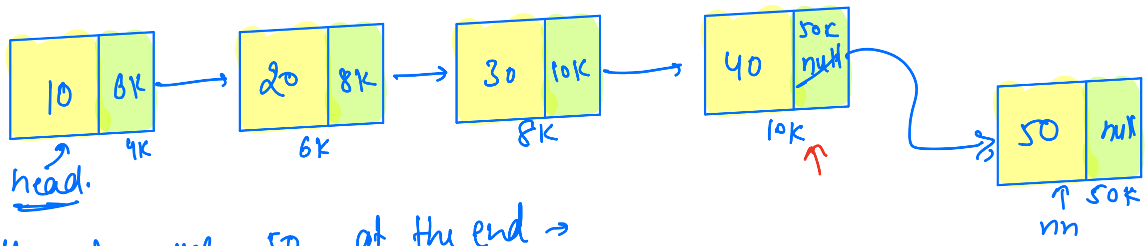
[head → object reference
↳ pointer.]

```
Node. insertAtStart (head, val){
    Node nn = new Node(val);
    nn.next = head;
    head = nn;
    return head;
}
```

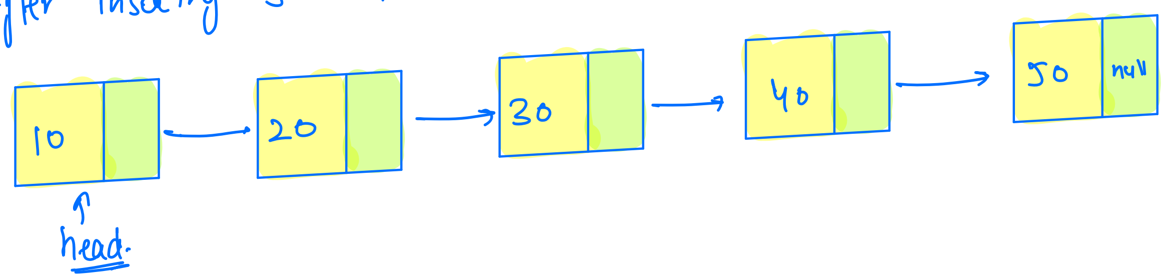
T.C → O(1)
S.C → O(1)



Insertion At End.



After inserting 50 at the end →



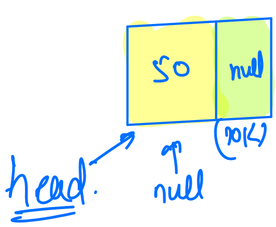

```
Node insertAtEnd ( head, val ) {
        Node nn = new Node(50)
    if ( head == null) { head = nn }

    else {
            Node temp = head;          [#shallow copy]
          while ( temp . next != null){
              temp = temp.next
          }

            temp . next = nn;
    }
    return head;
}
```

null.next
⇓
{ NULL Pointer Exception }

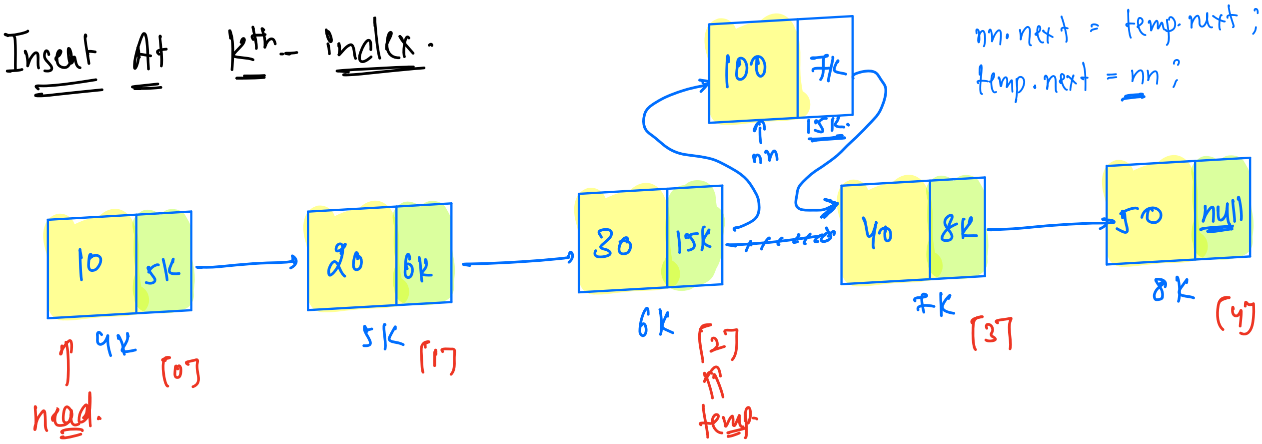
$\boxed{\begin{array}{l} T.C \to O(N) \\ S.C \to O(1) \end{array}}$



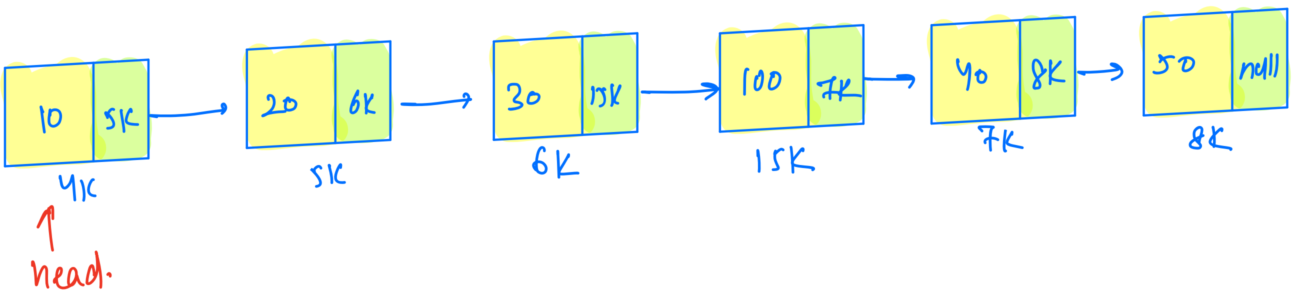
{ → By using tail pointer → T.C → O(1) }

Insert At k^{th} index.

nn.next = temp.next;
temp.next = nn;



→ insert 100 at idx 3.



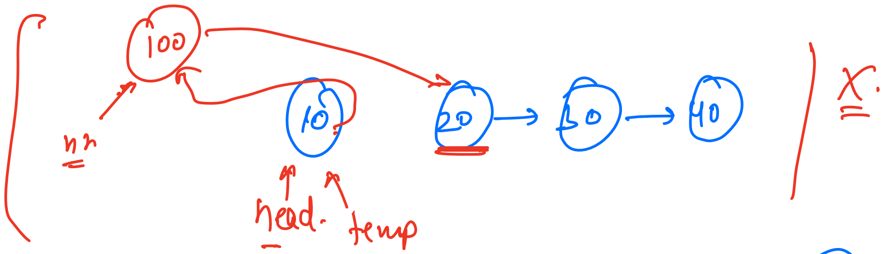

```
Node insertAtK ( head, val , K) {
    Node nn = new Node ( val);
    if ( K = = 0) {
        return insertAtStart ( head, val);
    }
    else {
        Node temp = head;
        for( i = 1 ; i < K ; i++){
            temp = temp.next
        }
        nn.next = temp.next;
        temp.next = nn;
        return head;
    }
}
```

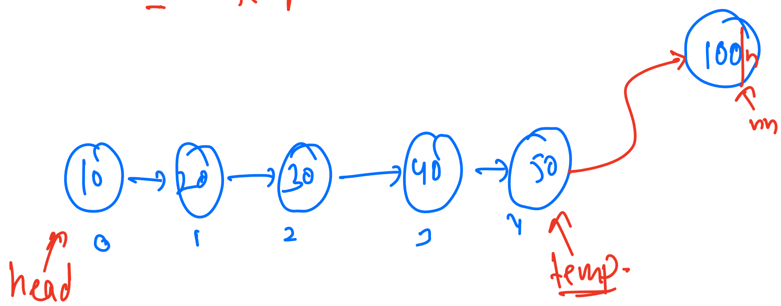
updating temp (k-1) times.

$$\boxed{\begin{array}{l} T.C \rightarrow O(N) \\ S.C \rightarrow O(1) \end{array}}$$

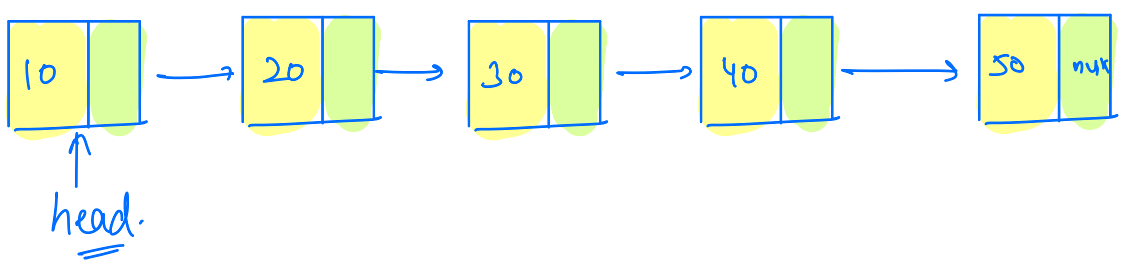
valid index.
{K → 0 to N}

① K = 0.
② K = N





Deletion



① Delete At First

head = head.next

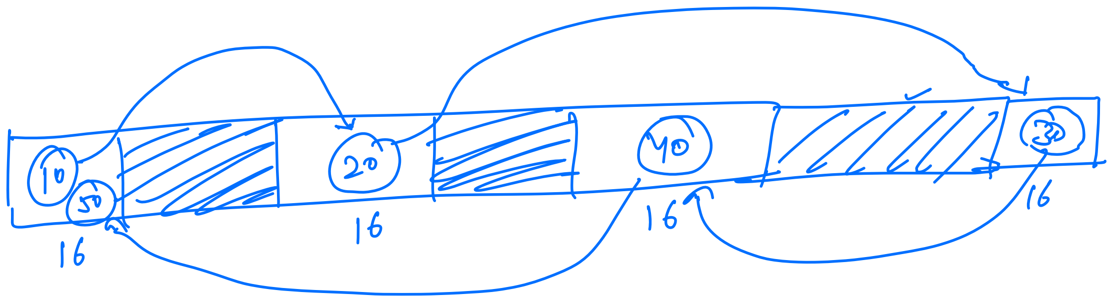
② Delete At Last

→ Traverse upto second last node

→ temp.next = null

③ Delete At Kth idx.

(# todo)



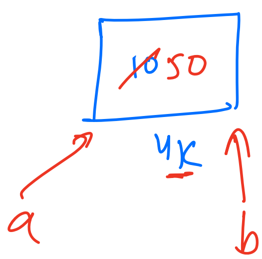
arr[5].

64 bytes → but contiguous 20 bytes is not available.

Shallow.

Node a = new Node(10);

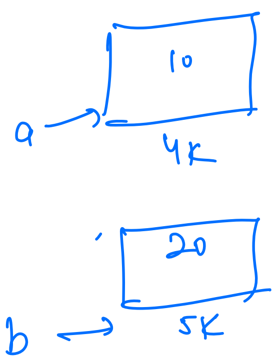
Node b = a



{b.val = 50}

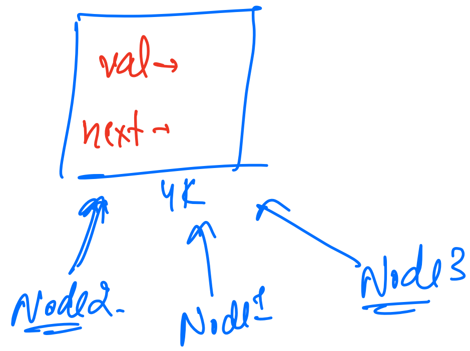
print(a.val) →

deep.



$$\left\{\begin{array}{lll} 1 \text{ to } K-1 & \Rightarrow & K-1 \\ 0 \text{ to } K-2 & \Rightarrow & K-1 \end{array}\right\}$$

{ Node 1 = node 2
Node 3 = node 2 }



→ Keep in mind.

if you change attribute of any one of these nodes. n1, n2, n3.

⇒ that change will be reflected for all the three nodes.

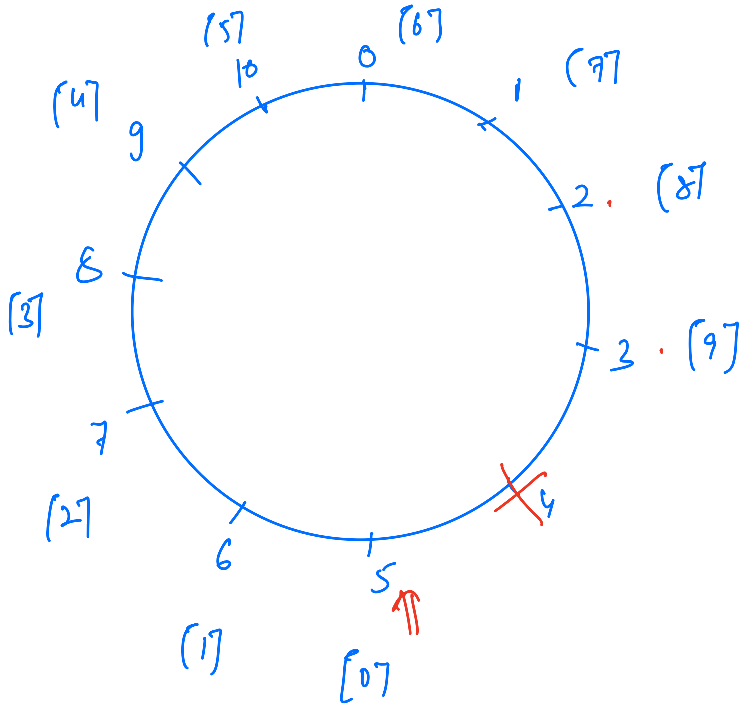
W.A ✓

Errors. ✗

Josephus → n people, every k^{th} man is killed.

$n = 11$

$K = 4.$



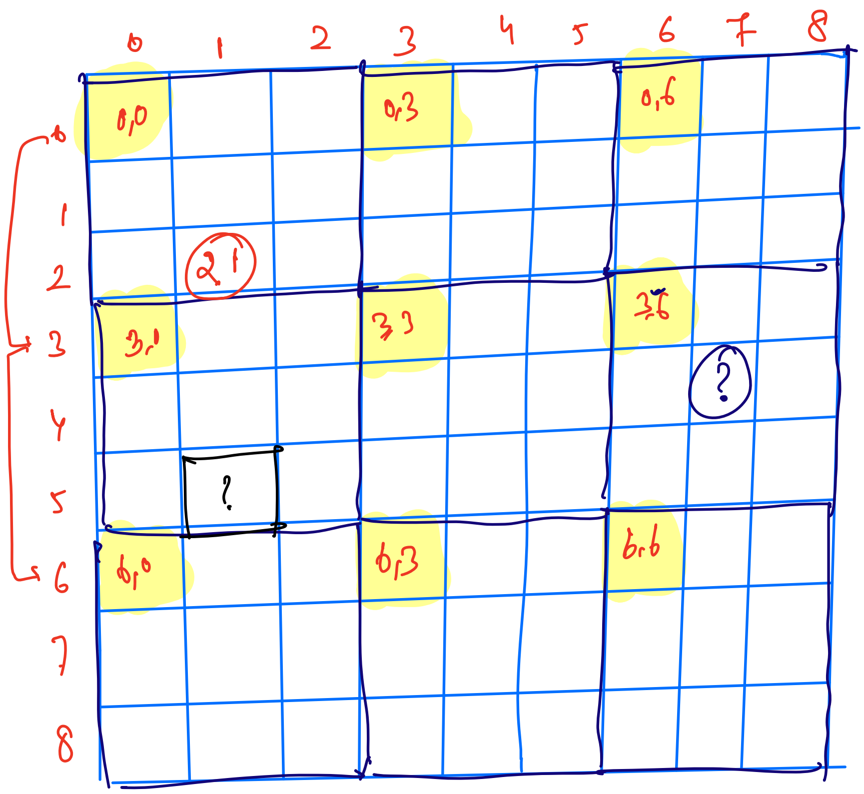
$$fun(n, k)$$

$$\Downarrow$$

$$fun(n-1, k) \rightarrow i$$

$$\left[fun(n-1, k) + (k+1)\right] \% \ n.$$

[
* → understand the sub-problem clearly.
* → try to map the correct indexing
]



[$(i,j) \rightarrow$ top-left corner of 3*3 grid.]

$4,7 \rightarrow (4 - 4\%3), (7 - 7\%3)$

$\Rightarrow (4-1), (7-1)$

$\Rightarrow 3, 6.$

$5,1 \rightarrow (5 - 5\%3), (1 - 1\%3)$

$\Rightarrow (3, 0)$

$$(i,j) \rightarrow (i - i\%3), (j - j\%3)$$

$$(i,j) \rightarrow \left(\frac{i}{3}\right)*3, \left(\frac{j}{3}\right)*3$$