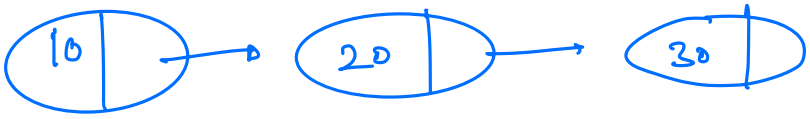
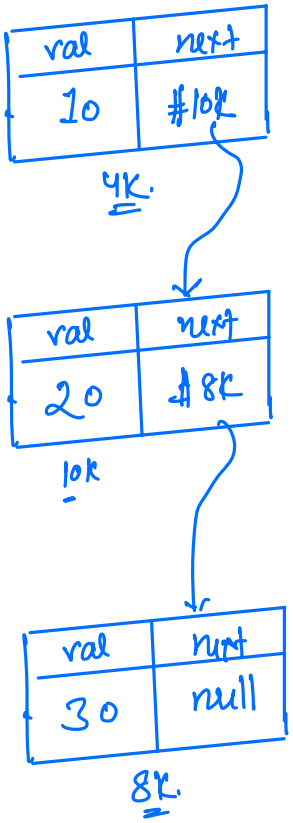
Linkedlist

- data structure used to store linear information.
- → chain of nodes.



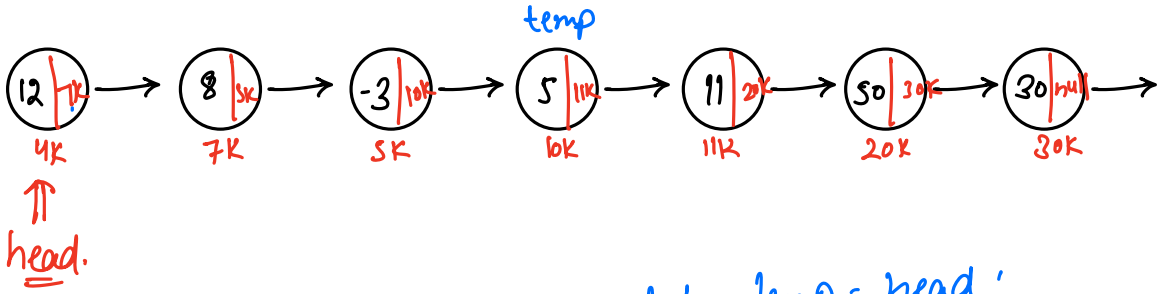

```
a = new Node(10);
b = new Node(20);
a.next = b
```




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

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

Q) Search for a given element K in the linked list.



K=5.

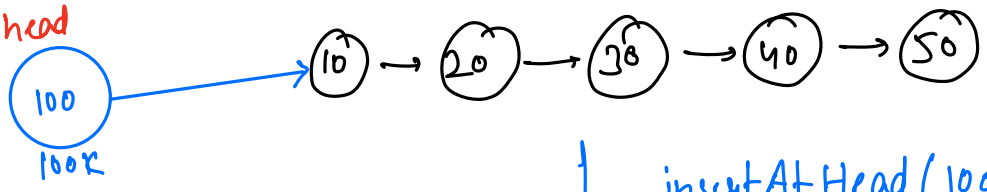
Never ever leave head to traverse the linked list.

```
Node temp = head;
while( temp != NULL){
    if(temp.val == K){
        return true;
    }
    temp = temp.next;
}
return false;
```

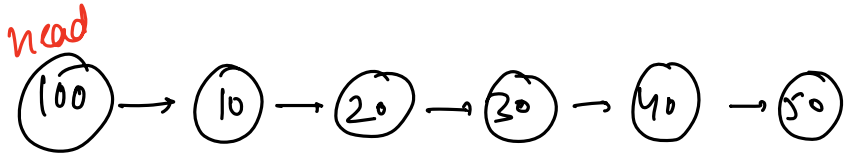
(T.C → O(N), S.C → O(1))

Insertion in LinkedList

- At Head.



insertAtHead(100);

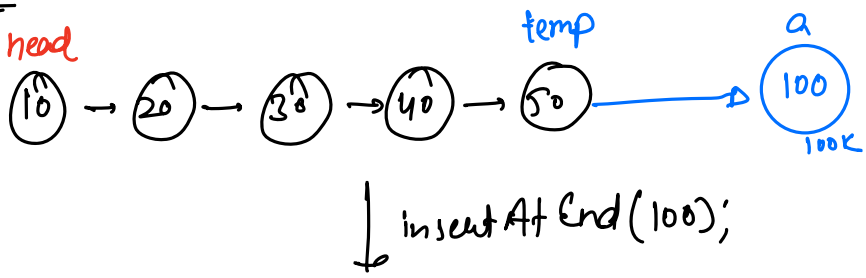


pseudo-code:

```
a = new Node(val);
a.next = head;
head = a;
```

T.C → O(1)
S.C → O(1)

- At End



code:-

```
a = new Node(val);
temp = head;
if(temp == NULL){ head = a, return }
while( temp.next != NULL){
        temp = temp.next;
}
temp.next = a;
```

head → NULL
↓
Null pointer exception

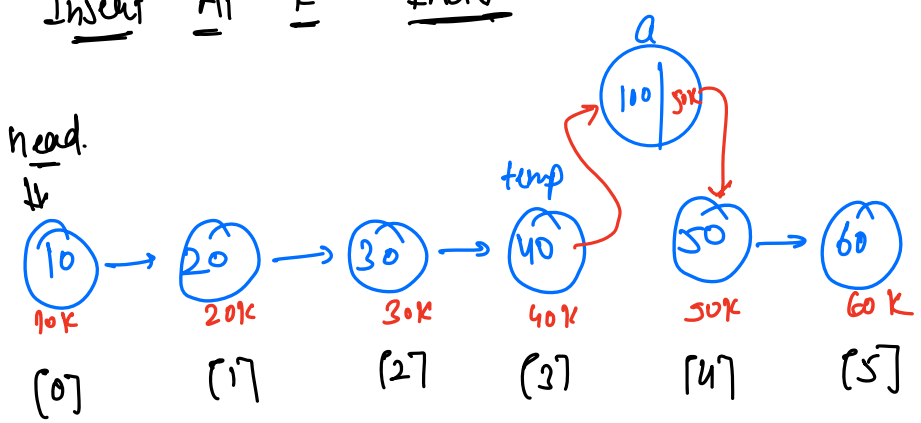
T.C → O(N)
S.C → O(1)

tail pointer

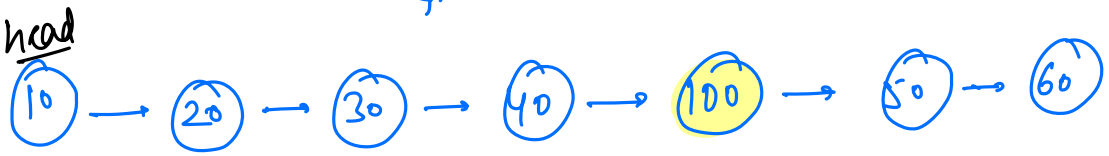
O(1)
O(1)

- Insert At k^{th} Index

a.next = temp.next
temp.next = a



insert(100, 4)



Code

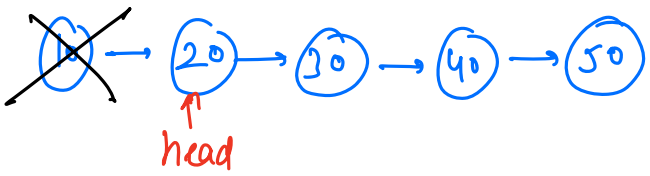
```
if(k==0 || head==NULL){ insertAtHead(val), return }

temp = head;

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

Node a = new Node(val);

a.next = temp.next;

temp.next = a;
```

} updating temp k-1 times

T.C → O(k)
S.C → O(1)

Deletion.

- **At Head.**



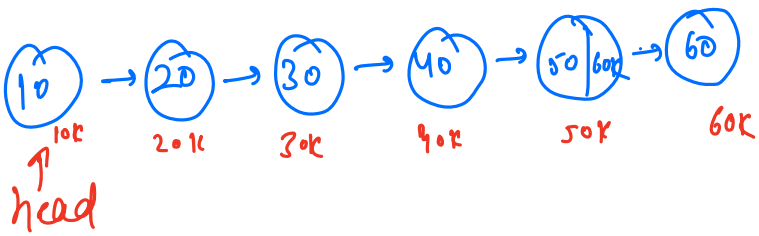
T.C → O(1)
S.C → O(1)

#Code.

```
if (head == NULL) { "List is empty", return}
head = head.next;
```

- **At End.**

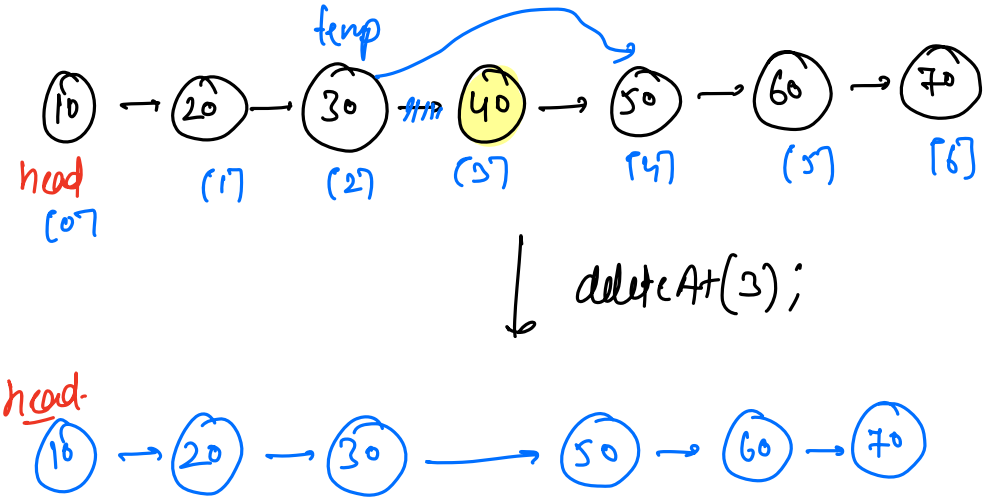
temp.next.next → NULL




```
Node temp = head;
if ( temp == NULL) { print ("List is empty"), return }
if ( temp.next == NULL){ head = NULL, return }
while ( temp.next.next != NULL){
        temp = temp.next;
}
temp.next = NULL;
```

T.C → O(N)
S.C → O(1)

- At k^{th} index.

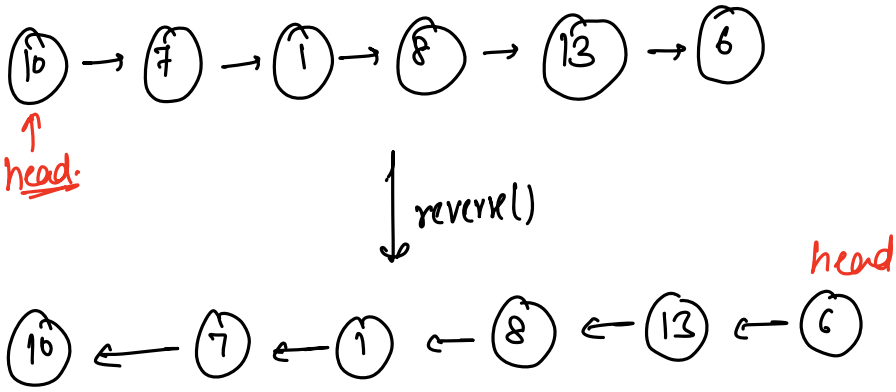



```
Node temp = head;

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

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

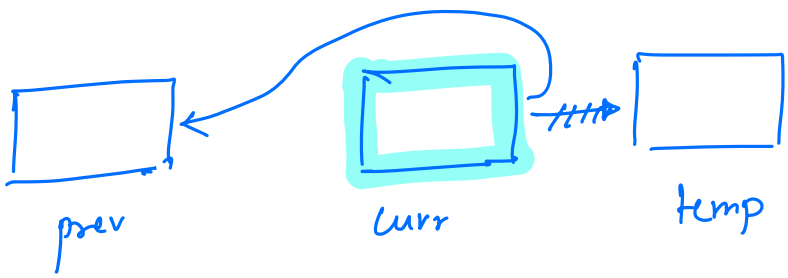
→ handle the edge-cases.

T.C → O(k)
S.C → O(1)

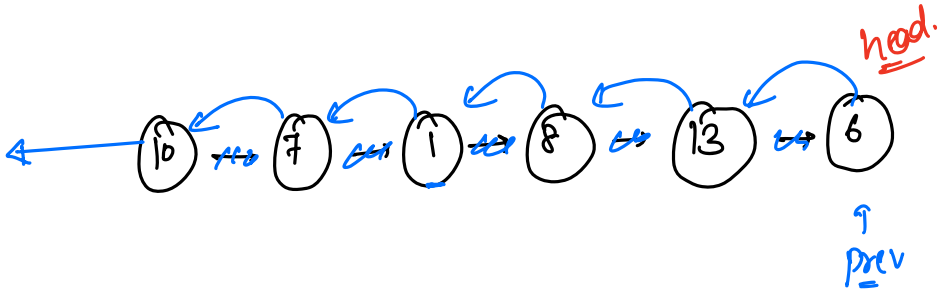
Reverse a linked-list



Idea.



curr.next = prev



[temp → &K]

temp = curr.next

curr.next = prev

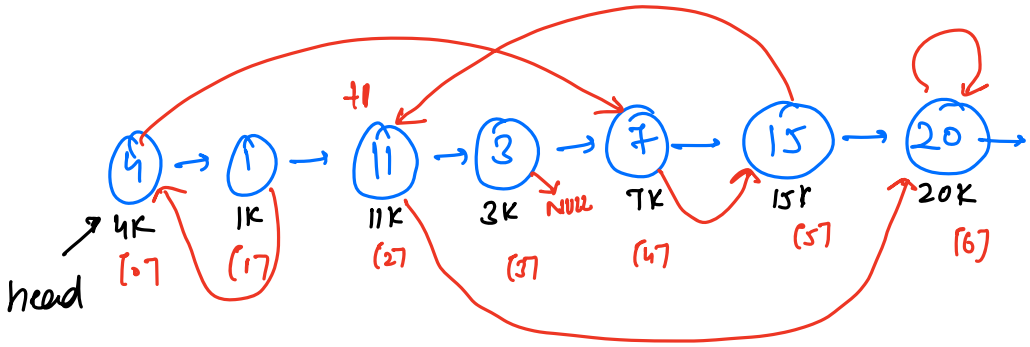
{ prev = curr
curr = temp; }

#code

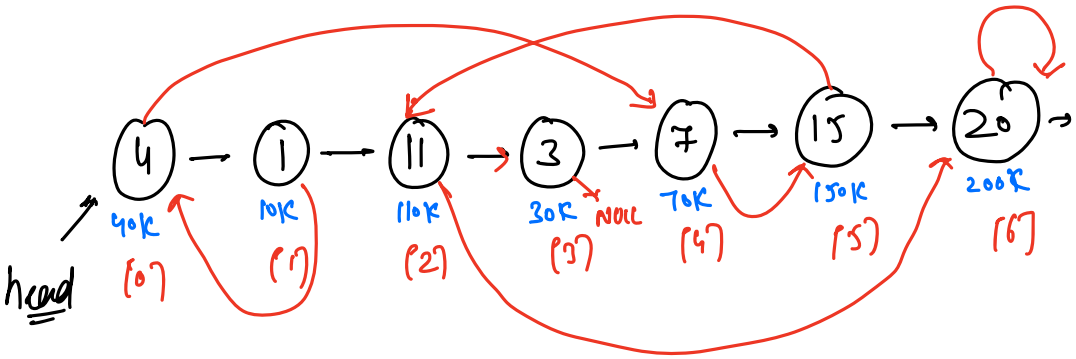
```
prev → NULL,  curr = head

while( curr != NULL ){
        Node temp = curr.next;
        curr.next = prev
            prev = curr;
            curr = temp;
}
head = prev;
```

T.C → O(N)
S.C → O(1)

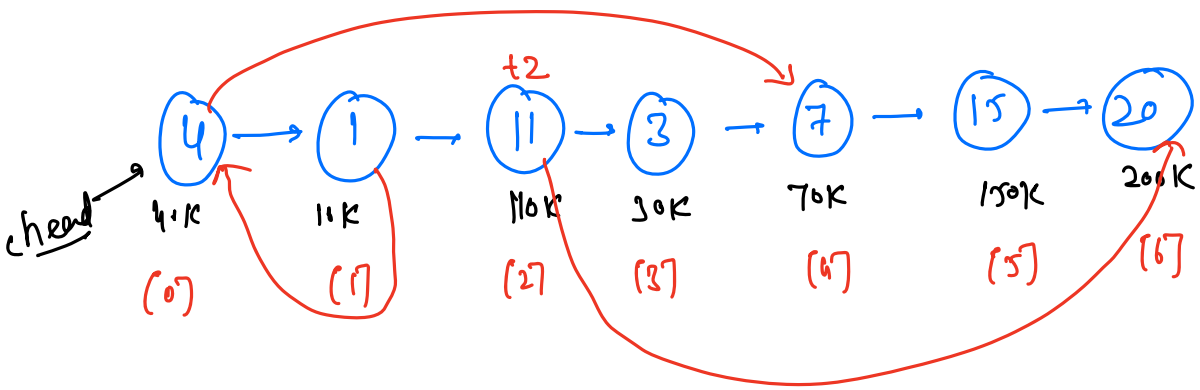
Clone A Linked-list With Random Pointer




```
Node {
    int val;
    Node next;
    Node random;
}
```



idea-1. Clone the linked with next reference only.

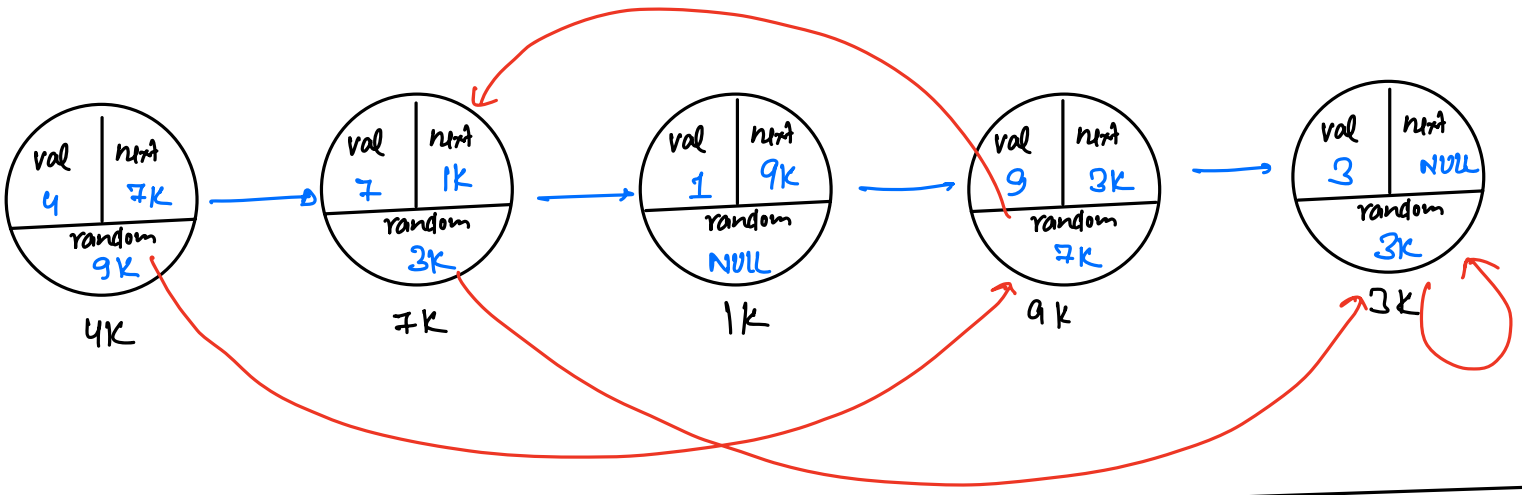



```
oldrandom = t1.random;
idx = 0 , temp = head
while (temp != oldrandom) {
        temp = temp.next;
        idx++;
}
```

T.C → $O(N^2)$

⇓

T.L.E.



Idea:- for every node in l.l, store the address of corresponding cloned node.

O.n	C.n
4K	40K
1K	10K
11K	110K
3K	30K
7K	70K
15K	150K
20K	200K

```
t1 = head,  t2 = chead;

while( t1 != NULL){
    random = t1.random;
    t2.random = map[random];
    t1 = t1.next;
    t2 = t2.next;
}
```

[T.C → O(N)
S.C → O(N)]

⇓

{T.C → O(N)
S.C → O(1)} [think]

how to prepare hashmap

```
Hashmap < Node, Node> map;

Node temp = head;
while (temp != NULL) {
    Node cn = new Node(temp.val);
    map.insert(temp, cn);
    temp = temp.next;
}
```

Syllabus for contest 3 → Searching, Two Pointers, Hashing, strings.

{ 27th Sept → 29th → 2 Oct → 4th October }
X X

[Doubt session → This Saturday]