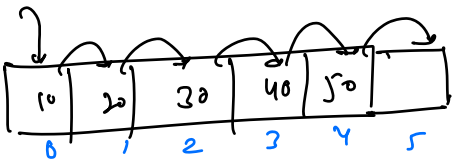
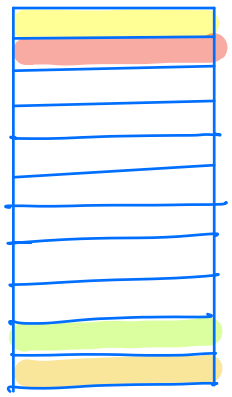
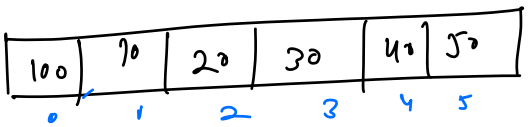
Arrays → contiguous memory location ⇒ random access time
⇓
$O(1)$

Linked-list →

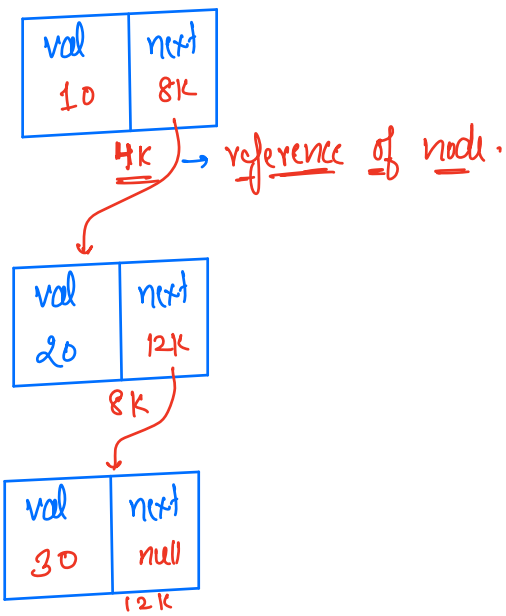






insertion / deletion

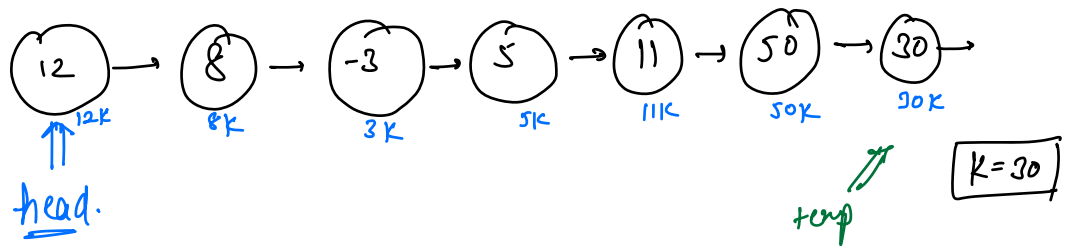
```
Node a = new Node(10);
Node b = new Node(20);
a.next = b;
```




```
class Node {
    int val;
    Node next;

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

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

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

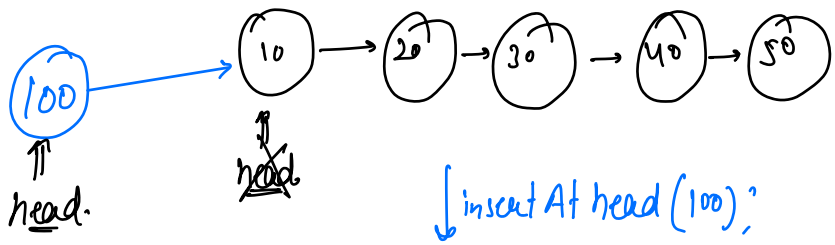


① Never ever leave head to traverse the linked-list.

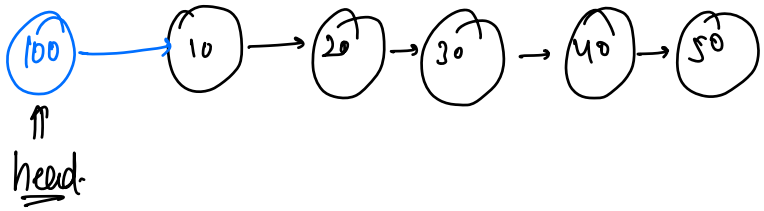
```
if (head == null) {return false}
Node temp = head;
while ( temp != null ){
    if (temp.val == K) {return true}
    temp = temp.next
}
return false;
```

Insertion in L.L.

- At head.



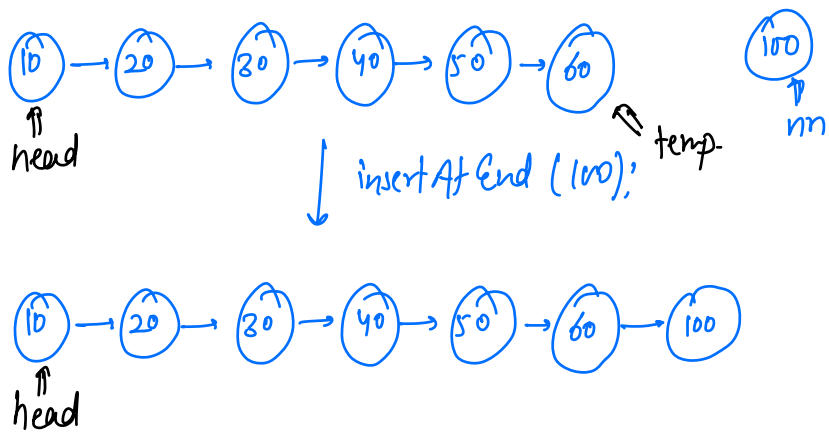
insertAtHead(100);



pseudo-code.

```
Node nn = new Node (100);
nn.next = head;
head = nn
```

• At the end

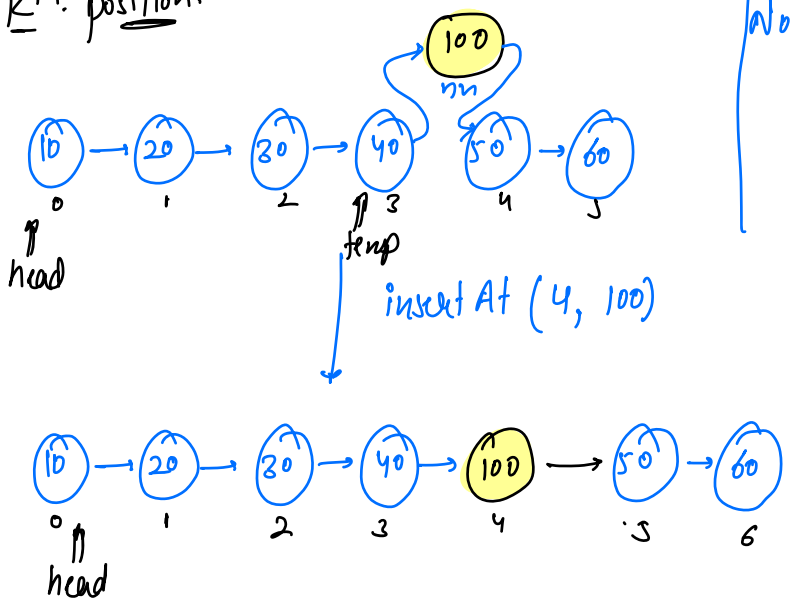


pseudo-code.

```
Node nn = new Node (x);
Node temp = head;
if( temp == NULL) { head = nn , return }
while (temp.next != null){
        temp = temp.next;
}
temp.next = nn;
```

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

Insert At K^{th}. position.



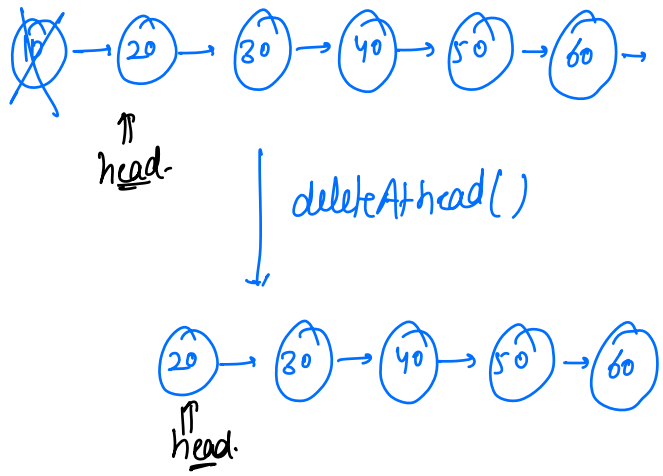

```
Node temp2 = temp.next
temp.next = nn
nn.next = temp2
```

pseudo-code-

```
Node nn = new Node(x);
if ( head == null) { head = nn, return }

Node temp = head
//update it K-1 times.
for ( i = 1 ; i ≤ K-1 ; i++) {
        temp = temp.next
}

nn.next = temp.next   //creating right connection
temp.next = nn        // creating left connection.
```

Deletion

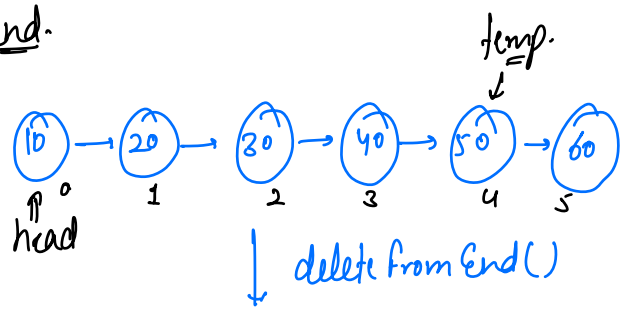
- At head.



// pseudo-code.

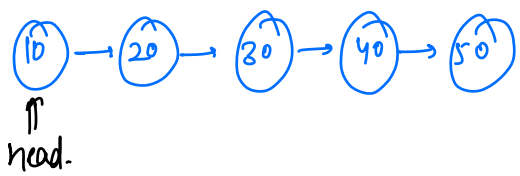
```
if (head == null) { "List is empty");
head = head.next
```

- At End.



temp.next.next = null

last node



pseudo-code.

```
Node temp = head;

while( temp.next.next != null){
    temp = temp.next
}

temp.next = null;
```

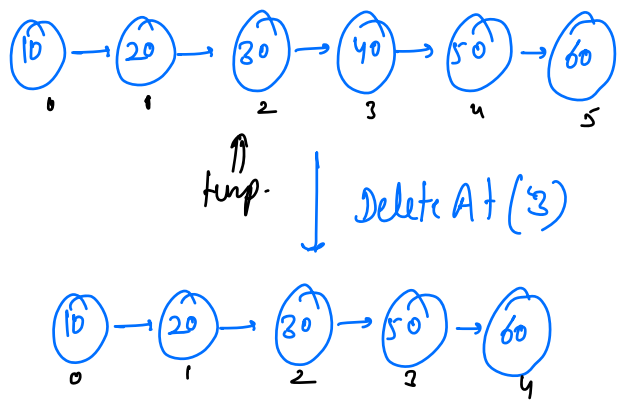
① length < 2.

handle these two edge-cases on your own.

length = 1,

length = 0;

- **Deletion At k^{th} Index**



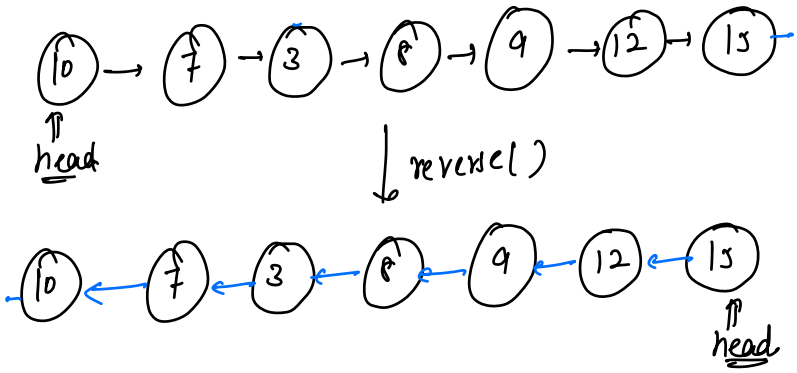
pseudo-code.

```
Node temp = head.

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

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

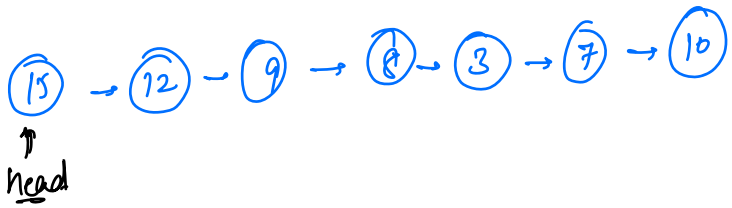
Think & handle the edge-cases (# todo).

{Break → 10:53 → 11:00 pm}

Reverse the linked-list.

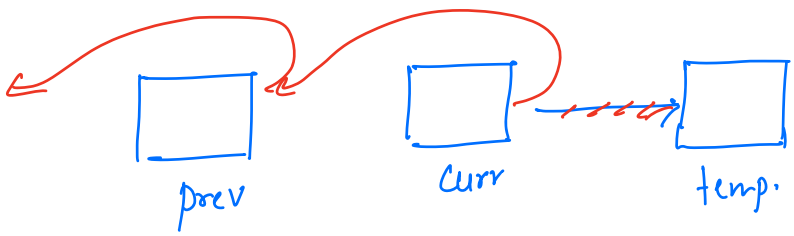


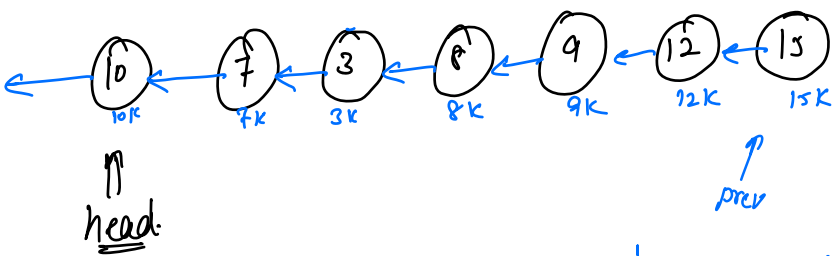
A1.:



traverse on all the nodes and do add first.

T.C → O(N), S.C → O(N).





temp = null

~~9K~~

prev = ~~12K~~ 15K

curr = ~~15K~~ null

~~12K~~

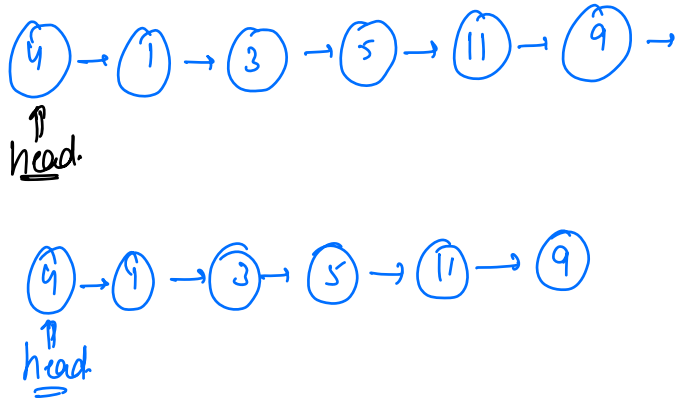
```
// Preserve curr.next (temp = curr.next)
    curr.next = prev

    [ prev = curr
      curr = temp ]

head = prev
```

pseudo-code.

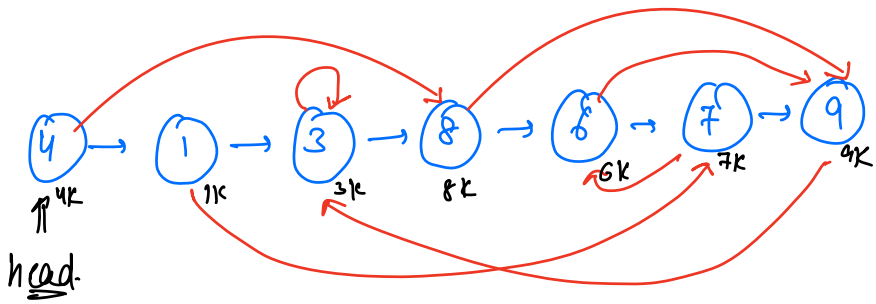
```
prev = null ,   curr = head ;

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



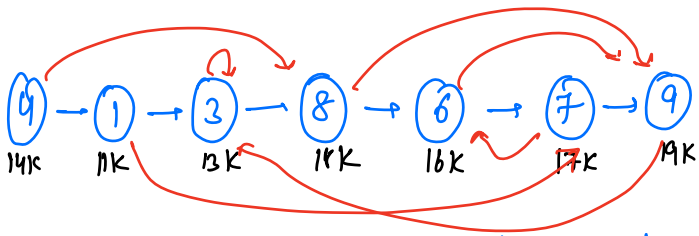
Clone A LinkedList with Random Pointer




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

Idea-1.

① Clone the l.l with next only.

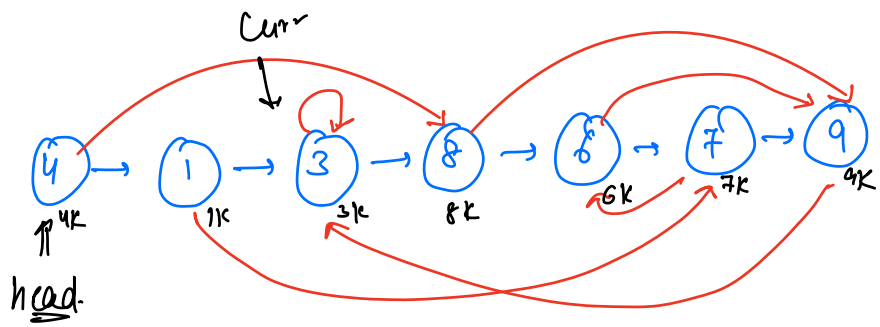


find the index of random node in the original l.l, then traverse upto that idx in c.l.l and set random pointer for every node in the cloned linked list.

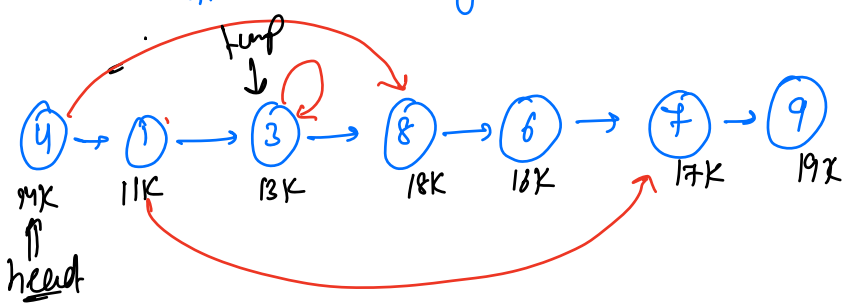
T.C → $O(N^2)$

for every old node, you know the reference of new Node

idea → Use Hashmap to store new Node against old Node



① Create a new linked only with next pointer & also create your Hashmap



old Node	new Node
4K →	14K
1K →	11K
3K →	13K
8K →	18K
6K →	16K
7K →	17K
9K →	19K

curr = head of o.l.l.

temp = head of c.l.l.

temp.random = hm[curr.random]

T.C → O(N)

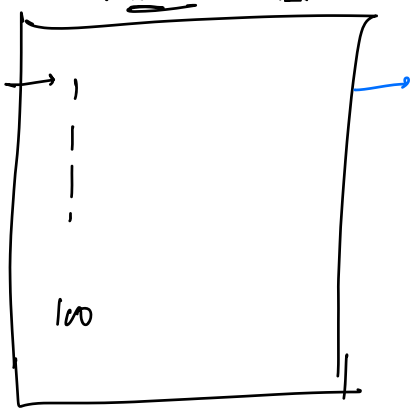
S.C → O(N)

O(1) S.C

L.L.

Trees

Revision Notes.



logic & pseudo-code in the notes.