Linked List 3: Doubly Linked List

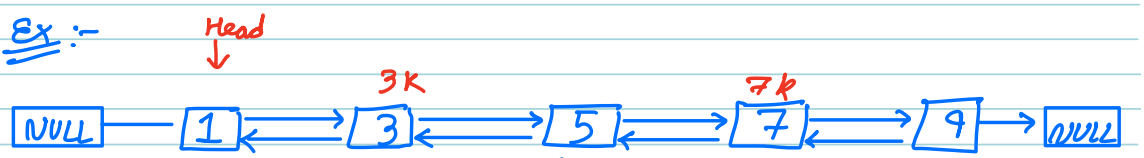
AGENDA

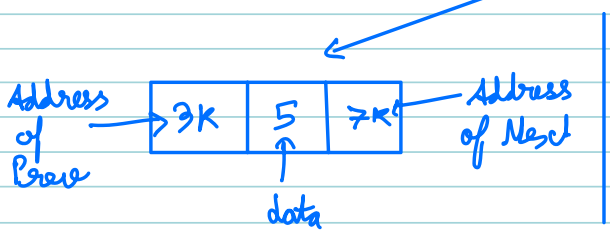
- what is Doubly Linked List ?
- How is Doubly Linked List different from singly Linked List ?
- 4 coding problems related to doubly Linked List !

DOUBLY LINKED LIST

↳ A doubly Linked List is a type of data structure used in computer science & programming to store & organize a collection of elements, such as nodes. It is similar to a Singly Linked List but with an additional feature: each node in a doubly Linked List contains pointer or references to both the next & the previous nodes in the list.

Ex :-



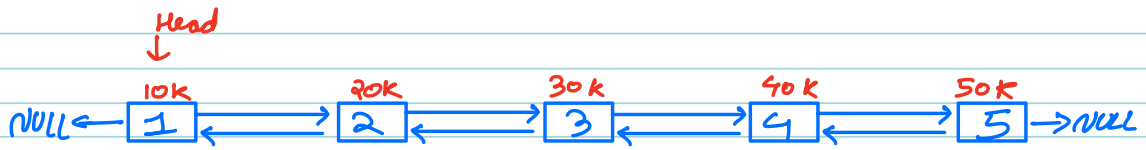



```
class Node {

    int data;
    Node Next;
    Node Prev;

}
```

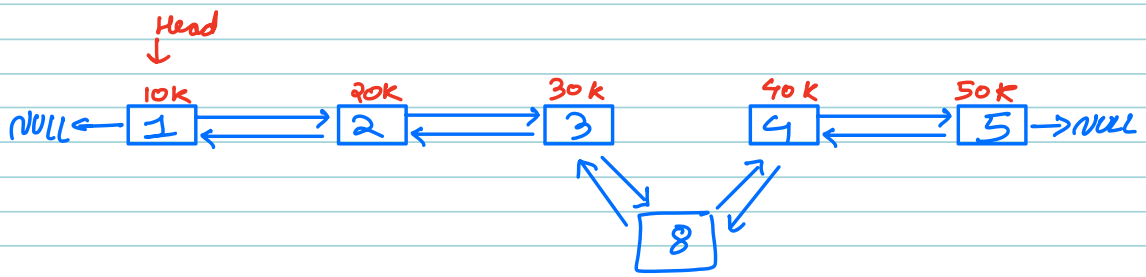
Quiz 1 :- Prev Pointer of Head of Doubly Linked List points to:

↳ NULL

Question 1 :- A doubly Linked List is given. A node is to be inserted with data x at position K. The range of K is between 0 & N where N is length of the doubly Linked List.



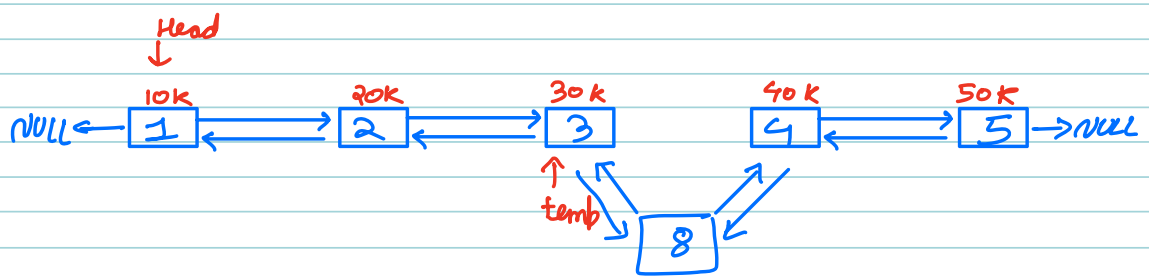
Ex :- x = 8 , K = 3



Quiz 2 :- In a doubly Linked List, the no. of pointers affected for an insertion operation b/w two nodes will be ?

↳ 4

IDEA :-



Steps :- ① move temp till prev of insertion
② Update Prev & Next of New Node.
③ Update Prev of Node ahead of New Node.
④ update Next of Prev of New Node.

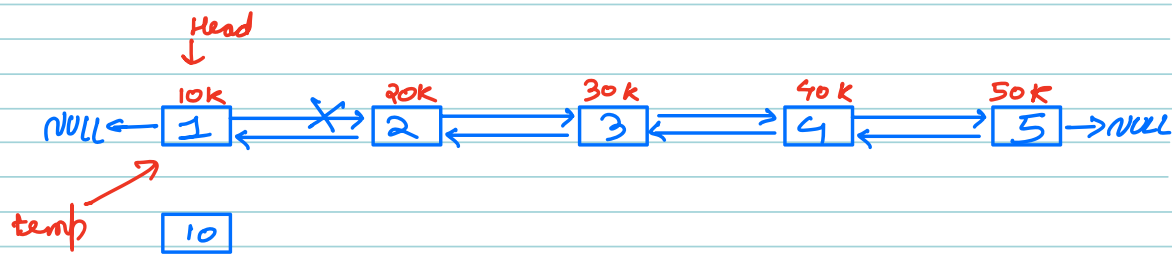
```
nn.next = temp.next
nn.prev = temp;
temp.next.prev = nn;
temp.next = nn;
```

PSEUDO CODE

```
nn = new Node(x)
if( head == null) { return nn; }
// Edge Case 1
temp = head;
for( i=1; i<k; i++){
        temp = temp.next;
}
nn.next = temp.next
nn.prev = temp;
temp.next.prev = nn;
temp.next = nn;
```

Edge Case

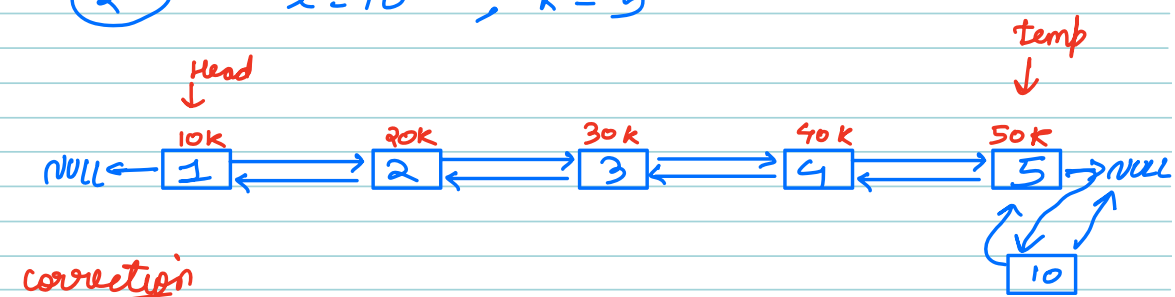
① $x = 10$, $k = 0$



Correction

```
if (k==0) {
    nn.next = head;
    head.prev = nn;
    head = nn;
    return head
}
```

② $x = 10$, $k = 5$



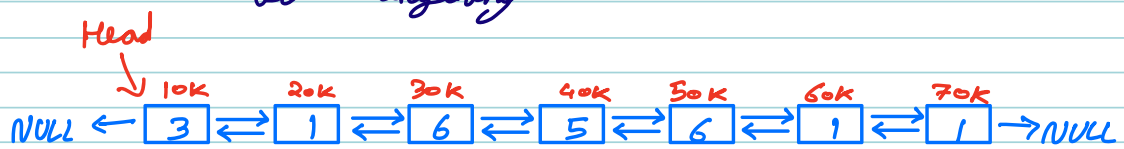
correction

```
nn.next = temp.next
nn.prev = temp;
if (temp.next != null){
    temp.next.prev = nn;
}
temp.next = nn;
```

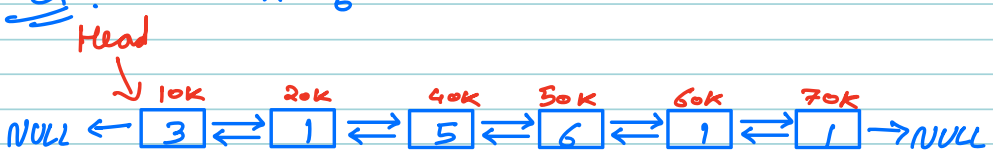
TC — O(N)

SC — O(1)

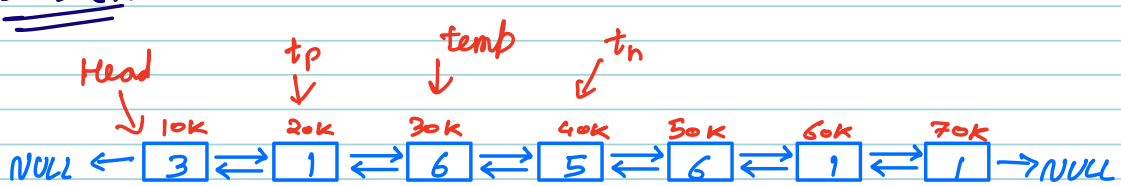
Question 2 :- We have been given a doubly Linked List of length N, we have to delete the first occurrence of data X from the given doubly Linked List. If element X is not present, don't do anything.



Ex :- X = 6



IDEA



Steps :- ① Goto Exist occurance.

② Initialize : temp.Prev & temp.next

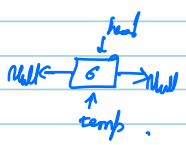
③ Delete

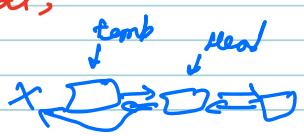
```
tp = temp.prev;
tn = temp.next;
tp.next = tn;
tn.prev = tp;
```

PSEUDO CODE

```
temp = head;
// Searching
while (temp != null){
    if (temp.data == x){
        break;
    }
    temp = temp.next;
}
if (temp == null) {
    return head;
} else if (temp.prev == null &&
           temp.next == NULL){
    head = null;
    return null;
} else if (temp.prev == null) {
    temp.next.prev = null
    head = temp.next;
    return head;
} else if (temp.next == null){
    temp.prev.next = null;
    return head;
}
```

Edge Cases





```
else {
    tp = temp.prev;
    tn = temp.next;

    tp.next = tn;
    tn.prev = tp;
}
```

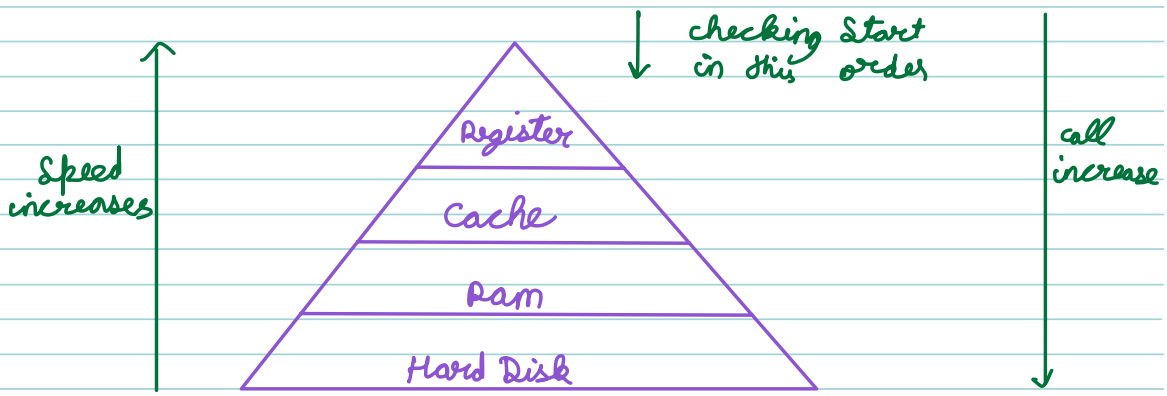
TC → O(N)

SC → O(1)

LRU CACHE

Question 3 :- We have been given running stream of integers & the fixed memory of size M, we have to maintain the most recent M elements. In Case the current memory is Full, we have to delete the last recent elements & insert the current data into the memory (as the most recent item.)

LRU CACHE [Least Recently Used Cache]



Data Stream :- 7 3 2 9 6 10 14 2 15 10 8 11 10 19 .....

Max Capacity = 5

Policy :- least Recently used

~~7~~ ~~3~~ ~~2~~ 9 6 10 14 15

CACHE

↳ Problem :- Need to do something for no. already existing.

Data Stream :- 7 3 2 9 6 10 14 2 15 10 8 11 10 19 .....

Correct Way

Max = 5 ele

~~7~~ ~~3~~ ~~2~~ ~~9~~ ~~6~~ ~~10~~ ~~14~~ ~~2~~ 15 ~~10~~ 8 11 10 19

CACHE

Steps :- ① Insert if capacity < 5

② Delete the element if you got it & existing previously (Basically re-insertion)

③ If got element which is new then Delete LRU & insert new

FLOW CHART

[X] → element to insert

Search (X)

x present

↳ Remove (x)

↳ Insert_last (X)

x absent

if (cache.size() == Limit) {

yes

↳ Delete_first()

↳ Insert_last (x)

No

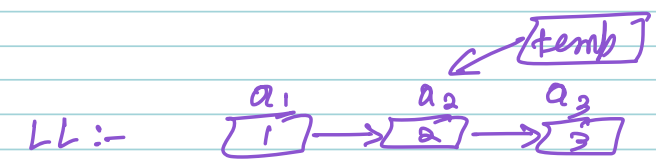
↳ Insert_last (X)

Quiz 3 :- what is the behaviour of an LRU cache memory when a new item is inserted & the cache is already full?

↳ New item added & Least Recently used will be removed.

⇒ Function which is Required for LRU implement-ation

operations	Arrays	Linked-List	Linked List – Hashmap
① Search(x)	O(N)	O(N)	O(1)
② Remove(x)	O(N)	O(N)	O(N)
③ Insert_Last(x)	O(1) (Arraylist)	O(1) (Tail is maintained)	O(1)
④ Delete_First()	O(N)	O(1)	O(1)



Hashmap :-

Key	Values
1	a_1
2	a_2
3	a_3

[Data] [Address]

★ The problem in LL + Hashmap Solution is Removal. This can be Solved using Doubly linked list + Hashmap

DRY RUN

Data :- 7 3 2 9 6 2 14 9 15 19 .....

Cap = 5

7

CACHE

Doubly Linked List	Hashmap.
	$7 \rightarrow a_1$
	$3 \rightarrow a_2$
	$2 \rightarrow a_3$
	$9 \rightarrow a_4$
	$6 \rightarrow a_5$

Steps :- ① By default maintain something like this :-

② Check if 7 present [This can be checked using map]. If not see Flow chart & perform operation

③ Run till capacity reaches. If Cap reaches delete from LL & map. check map for the Node to be deleted [you will get address.]

PSEUDO CODE

```
Node h = new Node(-1);
Node t = new Node(-1);

h.next = t;
t.prev = H;
```


```
Hashmap < Int, Node > hm;

Function LRU ( int x, int Limit) {

    if ( hm.contains (x)) {
        Node tp = hm[x];
        Delete Node (tp);
        Node nn = new Node(x);
        Insert_Last (nn, T);
        hm[x] = nn;
    } else {
        if ( hm.size() == Limit){
            Node t = h.next;
            hm.remove ( t.data);
            Delete Nod(t);
        }
        Node nn = new Node (x);
        Insert_Last ( nn, T);
        hm[x] = nn;
    }
```

```
}

void DeleteNode(Node temp){
    tp = temp.prev;
    tn = temp.next;
    tp.next = tn;
    tn.prev = tp;
}

void InsertLast(Node nn, Node tail){
    tp = tail.prev;
    nn.prev = tp;
    nn.next = tail;
    tp.next = nn;
    tail.prev = nn;
}
```

TC → Constant

Actual Copy

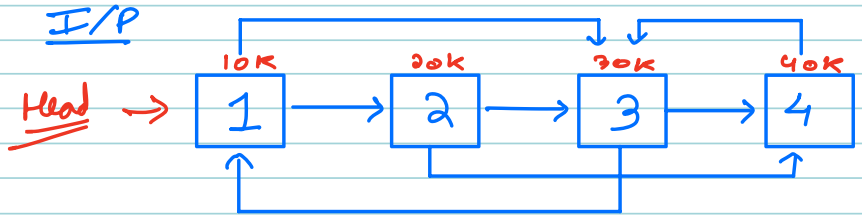
[Deep Copy of Linked List]

Question 4 :- We have to create a deep copy of the Doubly Linked List with random pointer. Here there is no certain next & prev pointer, a node can point to some other node.

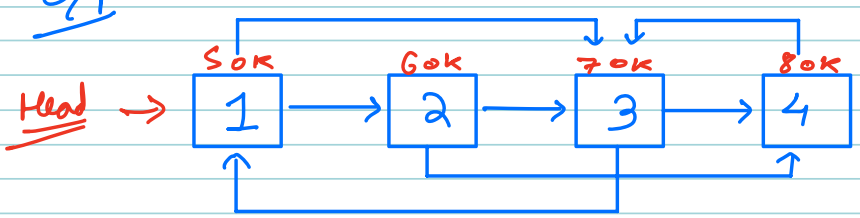
```
class Node {
    int data;
    Node next;
    Node rand;
}
```

Ex1

I/P

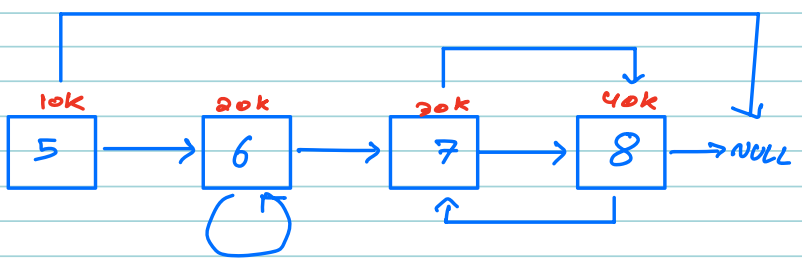


O/P

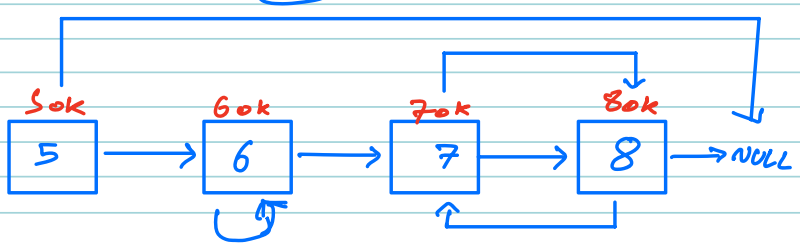


Ex2 :-

I/P



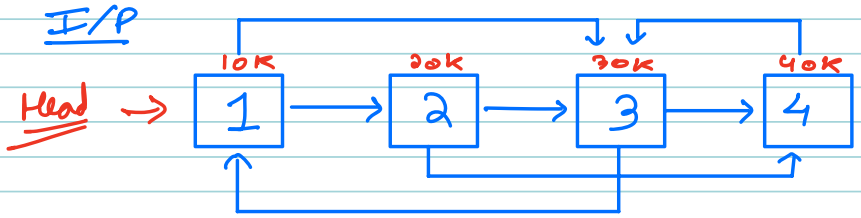
O/P



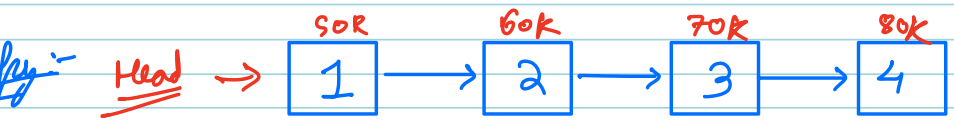
IDEA

Step 1 :- Create Copy of Doubly LL without random pointers.

I/P



Copy :-



Step 2 :- Update Random pointers

Approach 1 :- Use Hashmap

Hashmap < Node (old), Node (Copy) >

Hash Map

10K	50K
20K	60K
30K	70K
40K	80K

(L) Code for using hmap to update random

```
temp = old.rand;
New.random = hmap.get(temp);
```

PSEUDO CODE

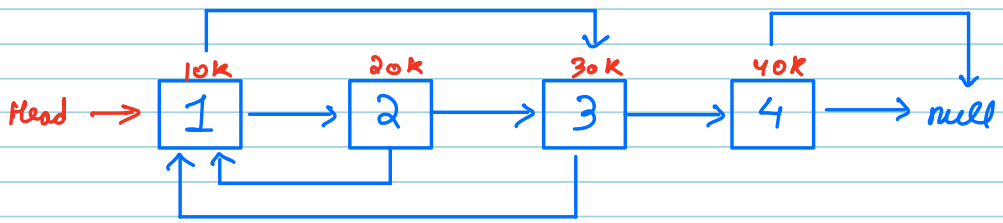
```
while( old != null){
    temp = old.random
    New.random = hmap.get(temp);
    old = old.next;
    new = new.next;
}
```

TC → O(N)

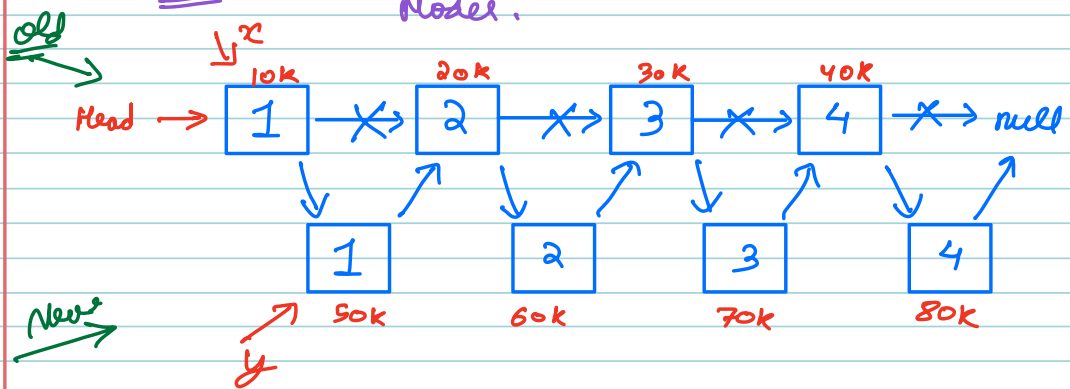
SC → O(N) → map

⇒ **Constraint** :- Do it in a constant space.

Approach 2



Step 1:- Insert new Nodes in b/w old Nodes.

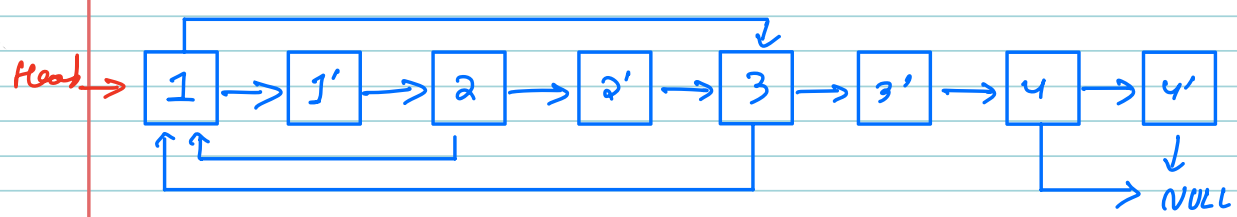


Code

```
while (x != null){
    y = new Node(x.data);
    y.next = x.next;
    x.next = y
    x = x.next.next;
}
```

* All old Node have random pointers intact. while New Nodes have not assigned random pointers.

Step 2 :- update Random pointer of newly created nodes



logical link

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

Pseudo Code

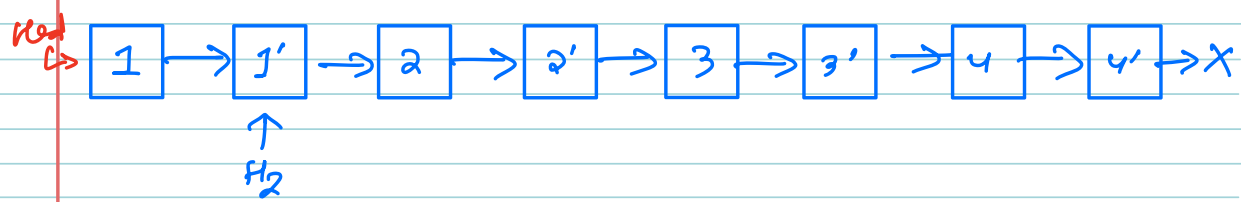
```
temp = head;
while (temp != null){
    else {
        temp.next.random =
            temp.random.next;
    }
    temp = temp.next.next;
}
```

Edge Case

correction :-

```
if (temp.random == null){
    temp.next.random
        = null;
}
```

Step 3:- Seperate old & new L.L.




```
H2 = head.next;
x = head;
y = H2;
while (x != null){
    x.next = x.next.next;
    if(y.next != null){
        y.next = y.next.next;
    }
    x = x.next;
    y = y.next;
}
return H2;
```

TC	→ O(N)
SC	→ O(1)

Ques 4:- what is the time complexity of creating a deep copy of a doubly linked list consists of N nodes with random pointers using extra space?

↳ O(N)