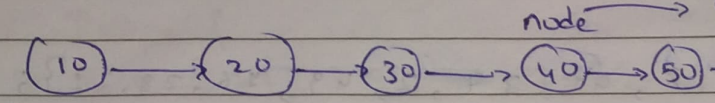
#Linked - List. LeetCode Question

M T W T F S S | Page No.: | Date: | YOUVA

(1) LeetCode Q No. : 237. {Delete Node in a Linked List}

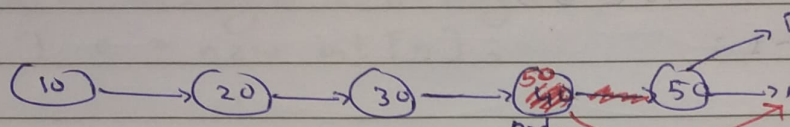
Ex ① :-



Logic :-

node.val = node.next.val.

step ① → Change Node.val to Node.next.val.



step ② → Then delete the Node.next. To do so, Connect ~~Node.next~~ Node.next to node.next.next.

node.next = node.next.next.

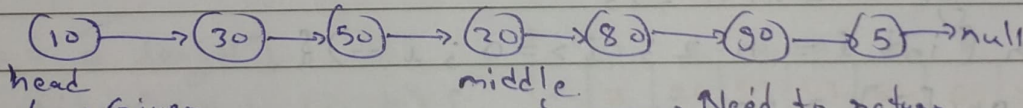
⇒ Code :- Class Solution {

```
    public void deleteNode(ListNode node){
        node.val = node.next.val;
        node.next = node.next.next;
    }
}
```

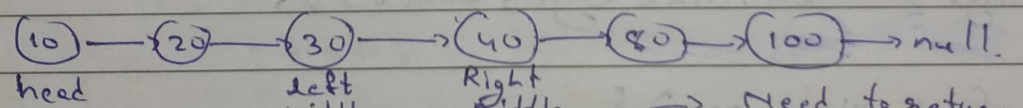
---

(2) LeetCode Q No. :- 876. Very Famous. {Middle of the Linked List}.

n = ~~7~~ 7



n = 6



1st → Calculate length.
then mid ⇒ len/2 + 1.
2nd → Run loop from 1 to mid.
return temp.

T.C = O(n) → len of L.L.
Extra space used } O(1).

Two Pass Solution

M-1

⇒ Code :- Public ListNode middleNode(ListNode head){

```
    ListNode temp = head;
    int len = 0;
    while(temp != null){
        temp = temp.next;
        len++;
    }                                   } 1st
    int mid = len/2 + 1;
    temp = head;
    for(int i = 1; i < mid; i++){       } 2nd
        temp = temp.next;
    }
    return temp;
}
```

M-2

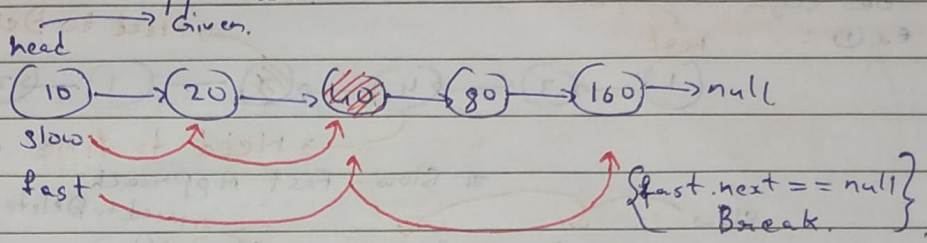
M T W T F S S | Page No.: | Date: | YOUVA

One pass ~~Sotln~~ Solution / without finding length

Slow - fast Approach

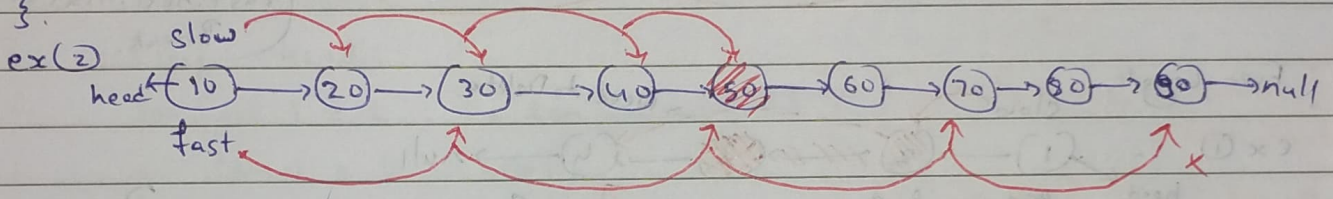
ex ① :-

Odd Length :-



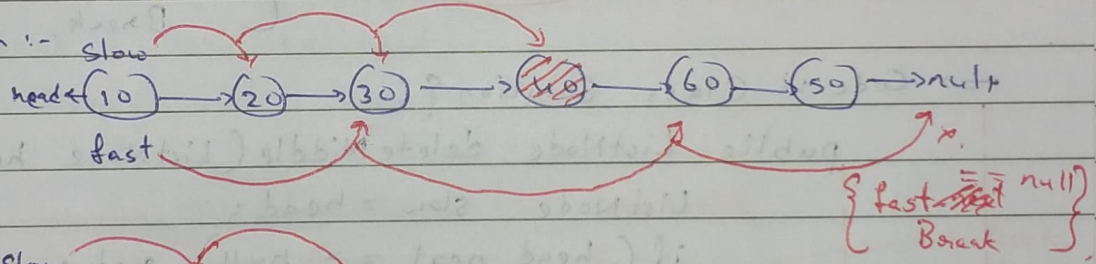

```
while(fast != null){
    slow = slow.next;
    fast = fast.next.next;
}
```

ex (2)

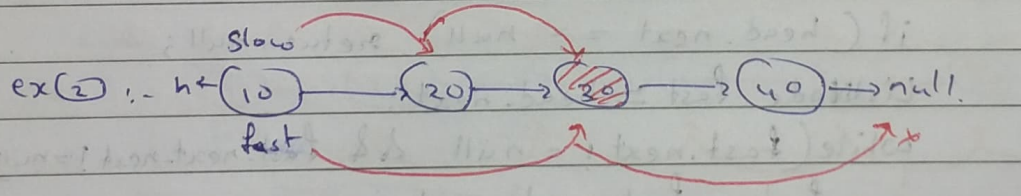


Even Length :-

ex (1) :-



ex (2) :-



=> Code :-

```
Class Solution{
    public ListNode middleNode(ListNode head){
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null & fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

If you write this like

```
while(fast.next != null && fast != null){
    slow = slow.next;
    fast = fast.next.next;
}
```

It will show Error

=> Error Because null.next != null

Nothing.

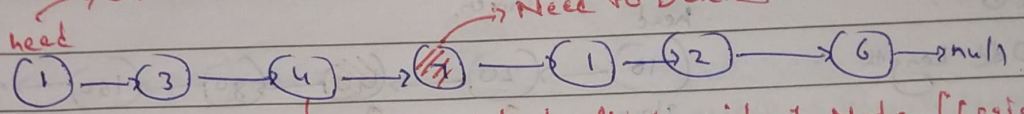
Home Work

M T W T F S S | Page No.: | Date: | YOUVA

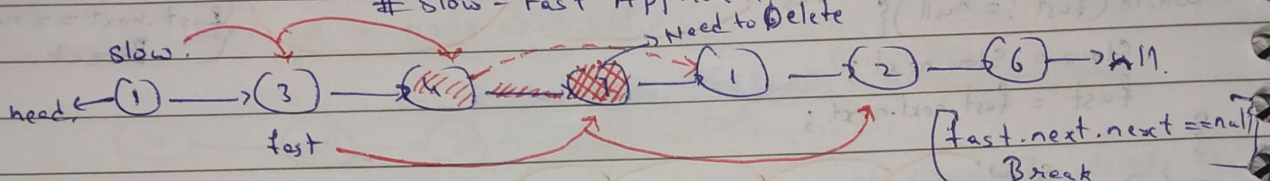
(3) LeetCode Q No :- 2095 { Delete the Middle Node }

odd Length

ex (1) :-

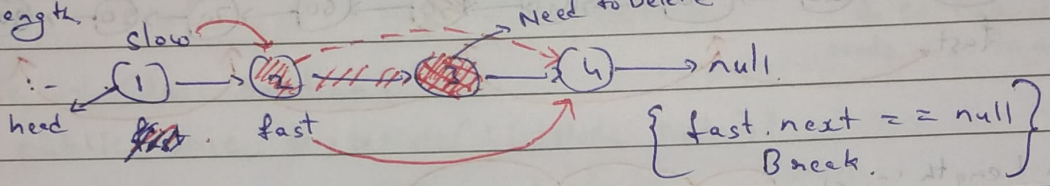


Slow - Fast Approach



Even Length

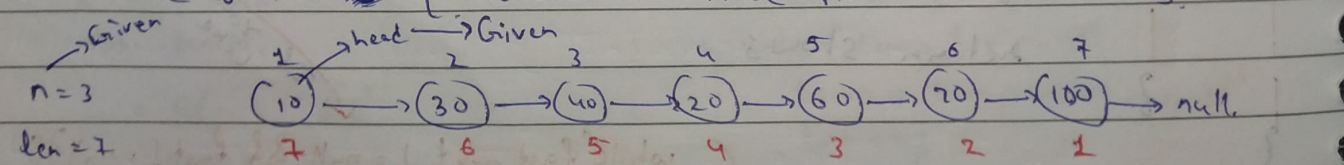
ex (1) :-



⇒ Code :- Class Solution {

```
public ListNode deleteMiddle(ListNode head){
    ListNode slow = head;
    if(head.next == null) return null;
    ListNode fast = head.next;
    while(fast.next != null && fast.next.next != null){
        fast = fast.next.next;
        slow = slow.next;
    }
    slow.next = slow.next.next;
    return head;
}
}
```

(4) LeetCode Q No. :- 19 { Remove n^{th} Node from End of List }



⇒ n^{th} Node from end = $(len - n + 1)^{th}$ Node from start.

⇒ To Delete $(len - n + 1)^{th}$ Node, we need $(len - n)^{th}$ Node (temp)

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

M - 1

=> Code :- Public ListNode removeNthFromEnd(ListNode head, int n){

```
ListNode temp = head;
int len = 0;
while (temp != null){
    temp = temp.next;
    len++;
}
```

→ [1st Calculate length]

Return head.next

if(len == n) it means we need to delete head.

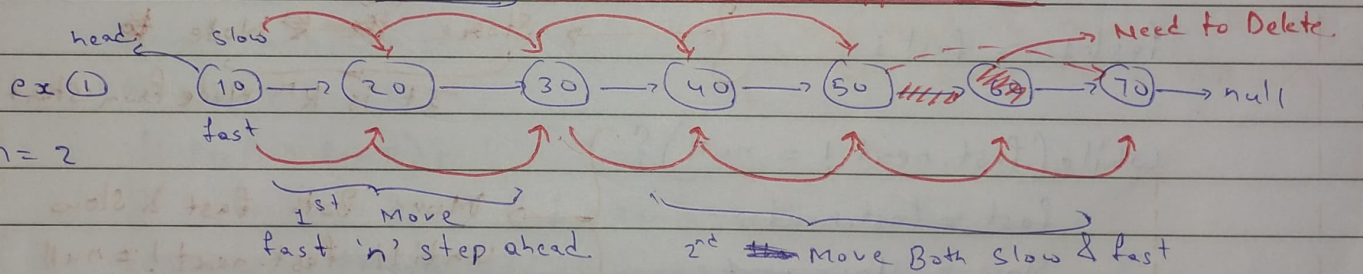
```
if(len == n) return head.next; // Edge Case
temp = head;
for(int i = 1; i < len - n; i++){
    temp = temp.next;
}
```

→ n from end = (len - n + 1) from start

→ We need temp = len - n

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

M - 2 Slow - fast Approach.



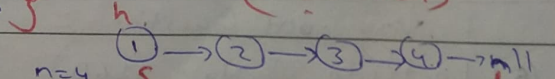
=> Code :- public ..... (ListNode head, int n){

```
ListNode slow = head;
ListNode fast = head;
for(int i = 0; i < n; i++){
    fast = fast.next;
}
```

Move fast 'n' step ahead.

```
if(fast == null) return head.next;
```

→ Edge Case when (n == len)



⇒ Need to Delete head.

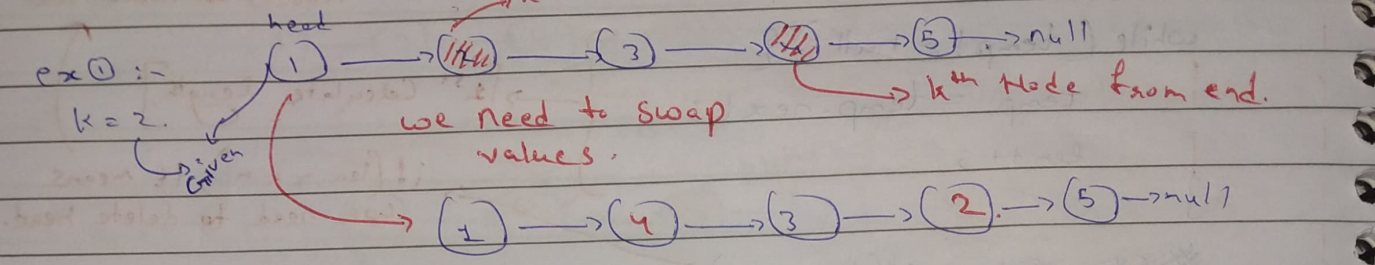
```
while(fast.next != null){
    slow = slow.next;
    fast = fast.next;
}
```

→ Move slow & fast together until fast reaches tail.

```
slow.next = slow.next.next;
return head;
}
```

(5) LeetCode Q. No.:- 1721 { Swapping Nodes in a Linked List }:

we need to Swap value of k^{th} Node from Begining & k^{th} Node from end.



⇒ Code :-

```
public ListNode swapNodes(ListNode head, int k){
    ListNode fast = head;
    ListNode slow = head;
    ListNode fast1;

    for(int i = 1; i < k; i++){
        fast = fast.next;
    }
    fast1 = fast;

    while(fast.next != null){
        fast = fast.next;
        slow = slow.next;
    }
    int temp = fast1.val;
    fast1.val = slow.val;
    slow.val = temp;
    return head;
}
```

1st move fast to 'k-1' step ahead.

Store k^{th} Node

It is k^{th} Node from start.

Move Both fast & slow until fast.next != null

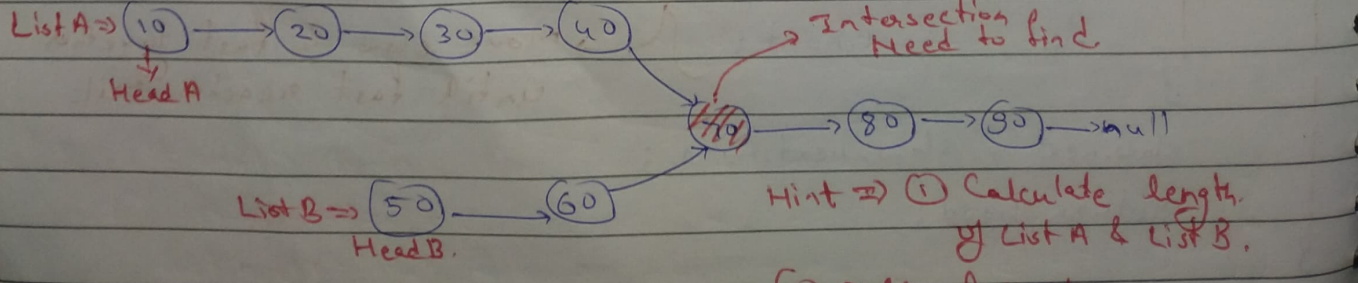
After this slow will be at k^{th} Node from end.

Swap fast1 & slow

Very Good Question.

(6) LeetCode Q. No.:- (160.) { Intersection of two Linked List }

ex①



Hint ⇒ ① Calculate length of List A & List B.

② Diff. of lengths.

ex(2) ⇒ (10) → (20) → (30) → (40) → (50) → null

(10) → (30) → (20) → (50) →

M T W T F S S	
Page No.:	YOUVA
Date:	

⇒ Code:- public ListNode getIntersectionNode(ListNode headA, ListNode headB){

```
int lenA = 0;
ListNode tempA = headA;
while(tempA != null){
    tempA = tempA.next;
    lenA++;
}
```
} Calculate Length of List A.

```
int lenB = 0;
ListNode tempB = headB;
while(tempB != null){
    tempB = tempB.next;
    lenB++;
}
```
} Calculate Length of List B.

⇒ Compare both the length.
if(lenA > lenB)
we need to move tempA ahead till i = 0 to i < lenA - lenB.

```
tempA = headA;
tempB = headB;
if(lenA > lenB){
    for(int i = 0; i < lenA - lenB; i++){
        tempA = tempA.next;
    }
}
else{ // lenB >= lenA
    for(int i = 0; i < lenB - lenA; i++){
        tempB = tempB.next;
    }
}
```
we need to move tempB ahead till i < lenB - lenA

7 > 5
lenA > lenB
7 - 5 => 2

ex(1) ⇒

(10) → (20) → (30) → (40) → (50) → (60)
tempA tempB

we need to move tempA two times.

tempA = tempB

```
while(tempA != tempB){
    tempA = tempA.next;
    tempB = tempB.next;
}
```
} Move Both tempA & tempB together until tempA == tepB

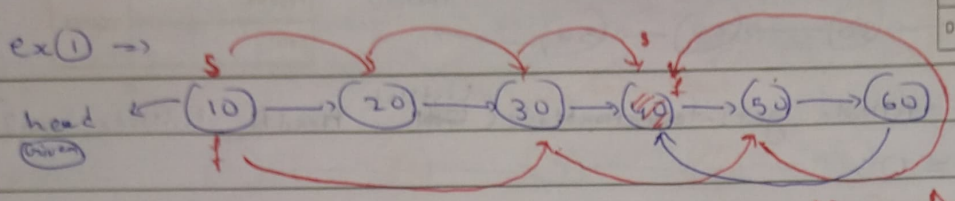
After this move Both tempA & B Together until tempA == tempB.

```
return tempA;
}
```

Slow fast approach.

(7) LeetCode Q.No.:- 141 { Linked List Cycle }.

ex(1) →



if(s == f) it is Cycle list. return true.

=> Code:-

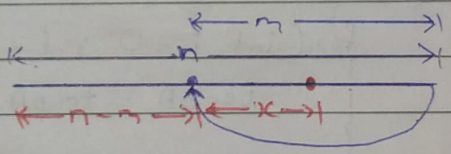
```
public boolean hasCycle(ListNode head){
    ListNode slow = head;
    ListNode fast = head;
    while(fast!=null && fast.next!=null){
        slow = slow.next;
        fast = fast.next.next;
        if(fast == slow) return true;
    }
    return false;
}
```

fast is moving 2x faster than slow & if there is Cycle in List then fast will definately catch the slow at some point.

Mathematical Proof.

slow = 1

fast = 2 / 3 / 4



distance travelled by fast in same time 't' will be double

Slow distance = n - m + x

fast distance = n + x

$n + x = 2(n - m + x)$

$\Rightarrow n + x = 2n - 2m + 2x$

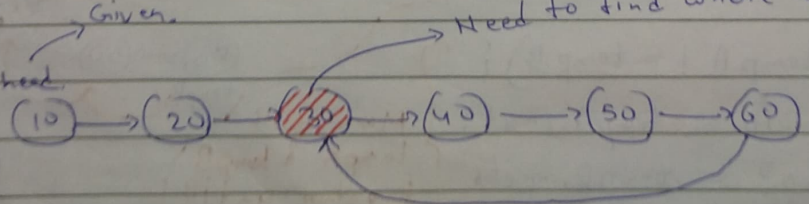
$\Rightarrow 2m = n + x$

$\Rightarrow x = 2m - n$

---

(8) LeetCode Q.No :- 142. { Linked List Cycle II }

ex(1) :-



** 1st put temp, slow, fast at head

2nd move slow = slow.next & fast = fast.next.next until fast.next != null && or fast != null & if (slow == fast) Break;

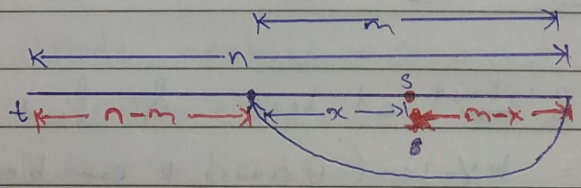
By Doing you will Check is their any cycle or not.

3rd move slow.next & temp.next until temp != slow.

=> Code :-

```
public class Solution{
    public ListNode detectCycle(ListNode head){
        if(head == null || head.next == null) return null;
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
            if(fast == slow) break;
        }
        if(fast != slow) return null;
        ListNode temp = head;
        while(temp != slow){
            slow = slow.next;
            temp = temp.next;
        }
        return slow;
    }
}
```

Proof.



To Reach start of cycle

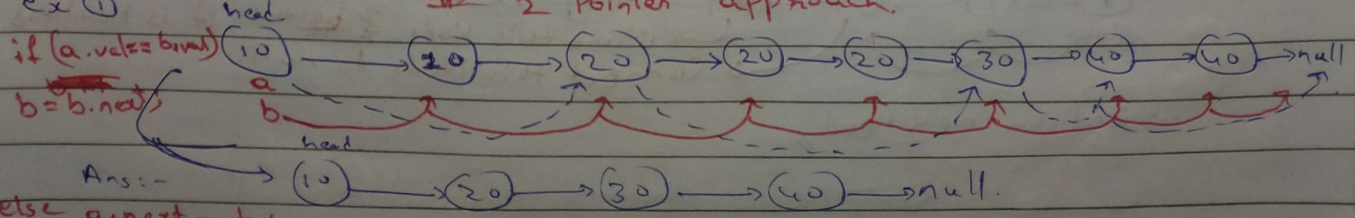
Slow ⟶ $m - x = m - (2m - n) = m - 2m + n =$ **n - m**

temp ⟶ **n - m**.

---

(9) LeetCode Q.No :- 83 {Remove Duplicates from Sorted List}

2 Pointer approach.



=> Code :-

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode i = head;
        ListNode j = head;
        if(head == null) return null;
        while(j != null){
            if(i.val == j.val){
                j = j.next;
            }
            else{
                i.next = j;
                i = j;
            }
        }
        i.next = j;
        return head;
    }
}
```

---

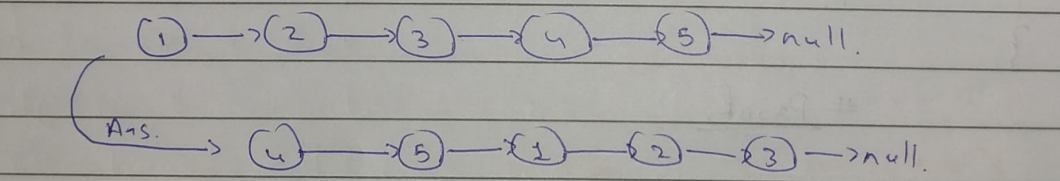
(10) LeetCode Q.No. 61. { Rotate List } [Similar to Rotate Array.]

Need to Rotate k times.

ex(1).

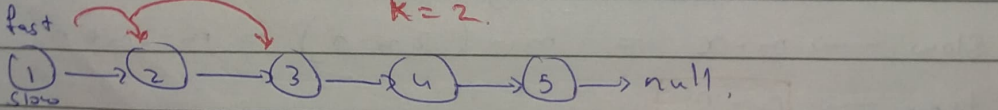
k = 2 (Given)

head → Given

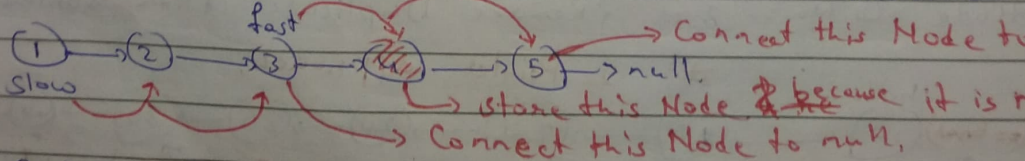


Step(1). put slow & ~~fast~~ at head & find length of LinkedList & k = k%len (Because k can be more than length)

Step(2). move fast k times



Step(3) move both slow & fast until. fast.next != null



=> Return newHead //.

=> Code :- Class Solution {

```
    public int lenOfList(ListNode head){
        int len = 0;
        ListNode temp = head;
        while(temp != null){
            len++;
            temp = temp.next;
        }
        return len;
    }

    public ListNode rotateRight(ListNode head, int k){
        if(head == null) return null;
        ListNode slow = head;
        ListNode fast = head;
        int len = lenOfList(head);
        k = k % len;
        if(k == 0) return head;
        for(int i = 0; i < k; i++){
            fast = fast.next;
        }
        while(fast.next != null){
            slow = slow.next;
            fast = fast.next;
        }
        ListNode ansHead = slow.next;
        slow.next = null;
        fast.next = head;
        return ansHead;
    }
}
```

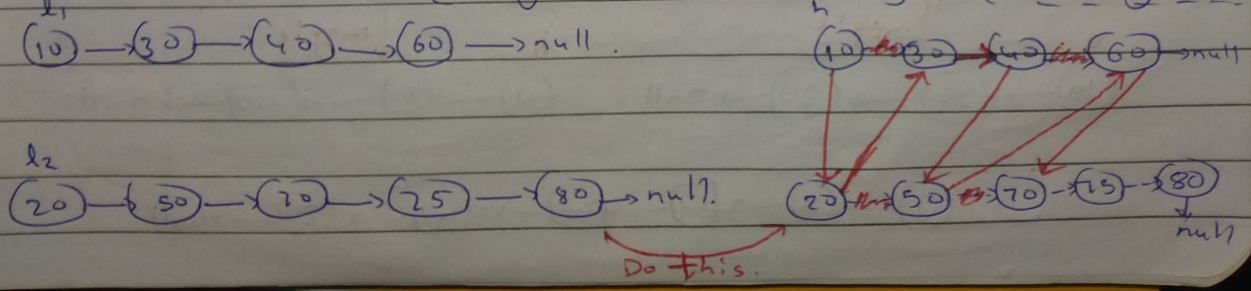
step (1).

step (2).

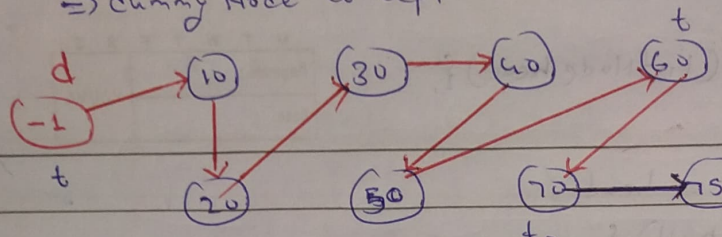
step (3).

(11) LeetCode Q.No. 21 {Merge 2 sorted List}

Similar to merge 2 sorted Arrays.



=> dummy Node Concept.




```
if(t1.val <= t2.val){
    t.next = t1
    t1 = t1.next
}
else {
    t.next = t2
    t2 = t2.next.
}
t = t.next
```

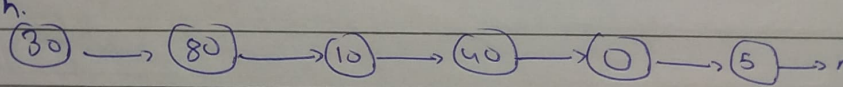
=> Code :- Class Solution {

```
public ListNode merge( ListNode list1, ListNode list2){
    ListNode dummy = new ListNode();
    ListNode t1 = list1;
    ListNode t2 = list2;
    ListNode t = dummy;
    while( t1 != null && t2 != null){
        if( t1.val <= t2.val){
            t.next = t1;
            t1 = t1.next;
        }
        else {
            t.next = t2;
            t2 = t2.next;
        }
        t = t.next;
    }
    if( t1 == null) t.next = t2;
    else t.next = t1;
    return dummy.next;
}
}
```

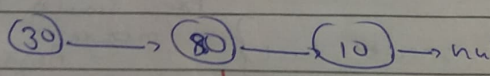
(12) LeetCode Q. No 148 {Sort List}

Hint:- merge Sort Algo & merge 2 sorted array.

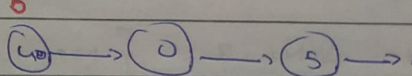
h.



a

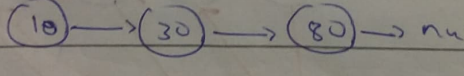


b



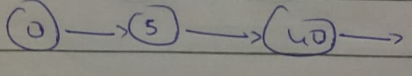
magic

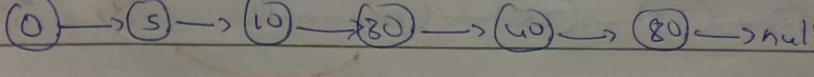
a



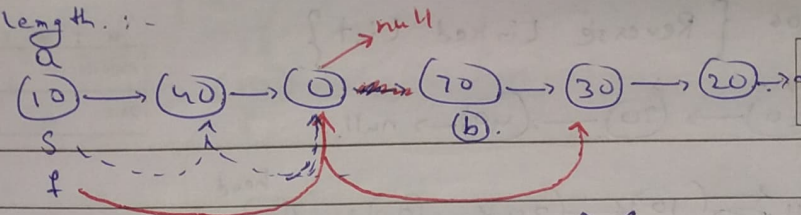
b

magic



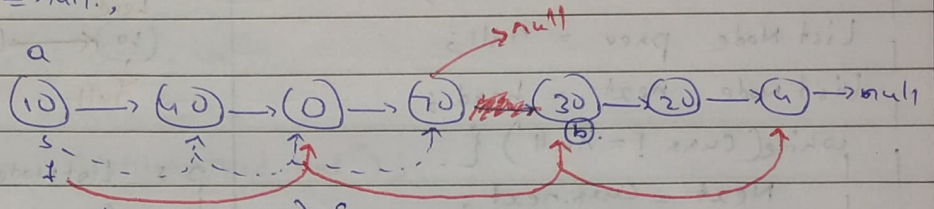


=> Even Length :-




```
while( fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
}
b = slow.next;
slow.next = null;
```

=> Odd Length :-




```
while( fast.next != null) {
        .
        .
}
b = slow.next;
slow.next = null;
```

H.W.

Leet Code 9 No.

→ 328
→ 24
→ 86

Easy Peasy.

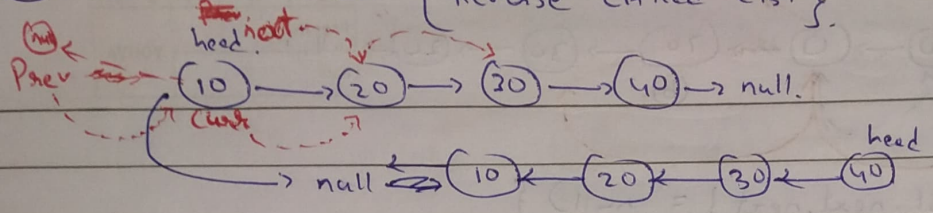
=> Code :-

~~public ListNode mergeList(ListNode list1, list~~

```
public ListNode sortList( ListNode head) {
    if( head == null || head.next == null) return head;
    ListNode firstHalf = head;
    ListNode slow = head;
    ListNode fast = head;
    while( fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
    }
    ListNode secondHalf = slow.next;
    slow.next = null;
    firstHalf = sortList( firstHalf);
    secondHalf = sortList( secondHalf);

    ListNode ans = merge( firstHalf, secondHalf);
    return ans;
}
```

(14) Leetcode Q.No. 206 {Reverse Linked List}.



M-1

Iterative.

```
ps ListNode reverseList(ListNode head){
    ListNode curr = head;
    ListNode prev = null;
    ListNode next = head;
    while(curr != null){
        Next = Curr.next;
        Curr.next = Prev;
        Prev = Curr;
        Curr = Next;
    }
    return prev;        // New Head
}
```

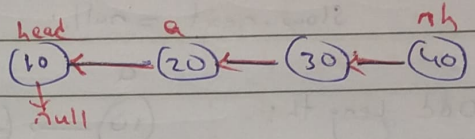
T.C = O(n)

& S.C = O(1).

M-2

Recursive Solution.

↳ Beleive in magic




```
ps ListNode reverseList(ListNode head){
    if(head == null || head.next == null)
        return head;
    ListNode a = head.next;
    ListNode newHead = reverseList(a);
    a.next = head;
    head.next = null;
    return newHead;
}
```

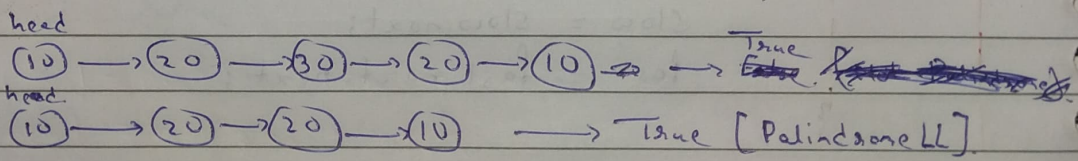
T.C = O(n).

Extra Space = O(n)

↓

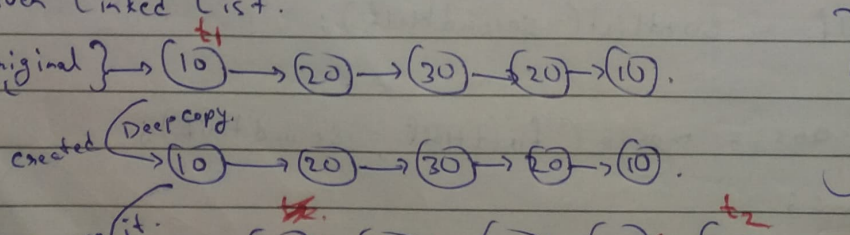
Call stack space

(15) Leetcode Q.No. ~~234~~. {Palindrome Linked List}



M-I

Create Deep Copy of Given Linked List.



T.C => O(n)

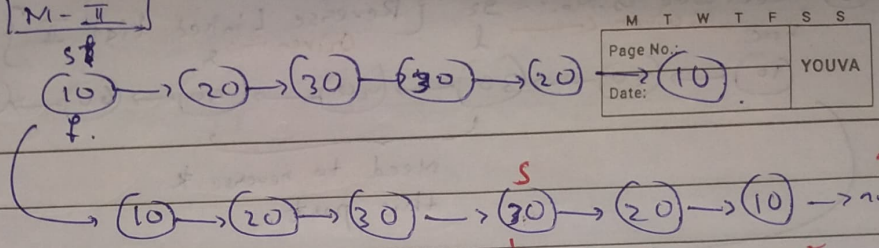
S.C => O(n).

M T W T F S S
Page No.:
Date:
YOUVA

M - II

Reverse the 2nd half of the List

Logics
→ Middle of LL
→ Reverse of LL
→ 2 pointer.



Then use Two pointer Technique

→ Codes:-

M - I

```
p s boolean isPalidrome(ListNode head){
    if(head.next == null) return true;
    ListNode newHead = new ListNode(Head.val);
    ListNode t1 = head.next;
    ListNode t2 = newHead;
    while(t1 != null){
        ListNode temp = new ListNode(t1.val);
        t2.next = temp;
        t2 = t2.next;
        t1 = t1.next;
    }
    newHead = reverseList(newHead);
    t1 = head;
    t2 = newHead;
    while(t1 != null){
        if(t1.val != t2.val) return false;
        else{
            t2 = t2.next;
            t1 = t1.next;
        }
    }
    return true;
}
```

Creating Deep Copy.

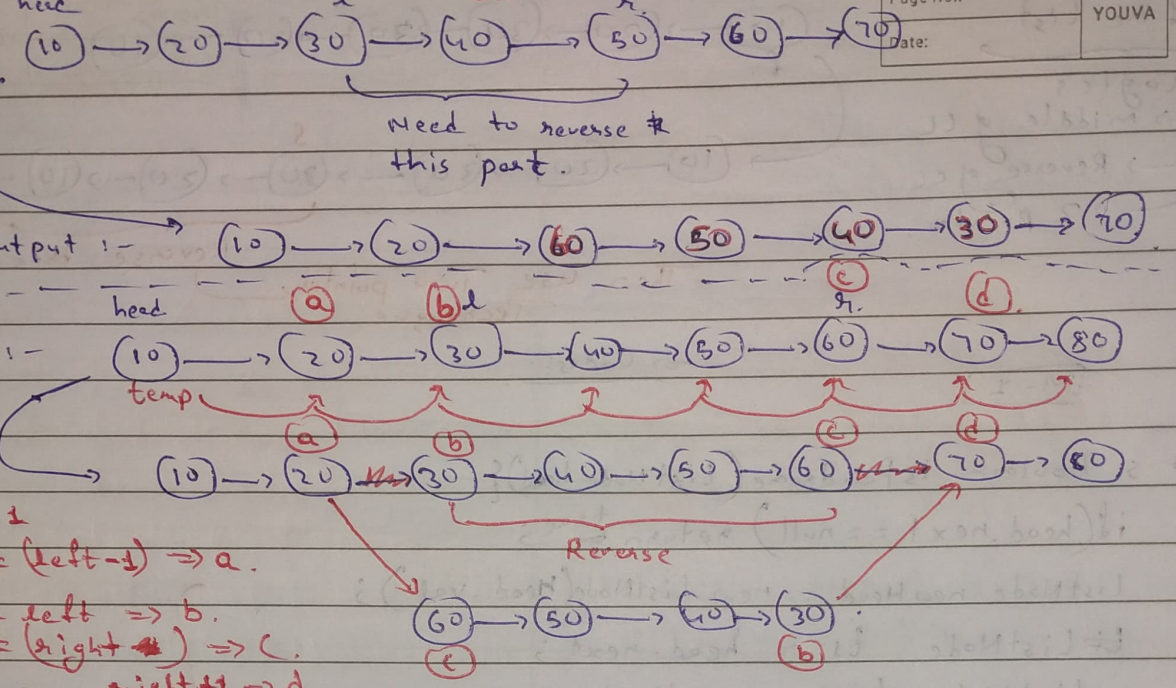
Reversing Deep Copied LL.

M - II

```
p s b isPalindrome(ListNode head){
    ListNode slow = head;
    ListNode fast = head;
    while(fast != null && fast.next != null){
        slow = slow.next;
        fast = fast.next.next;
    }
    ListNode j = reverseList(slow);
    ListNode i = head;
    while(j != null){
        if(i.val != j.val) return false;
        i = i.next;
        j = j.next;
    }
    return true;
```

H.W
Leet Code Q. No.:- 2130
Similar Logic //.

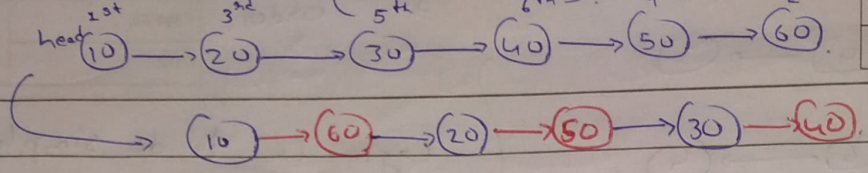
~~15~~ (16) LeetCode Q.No.:- 92. {Reverse Linked list II}.



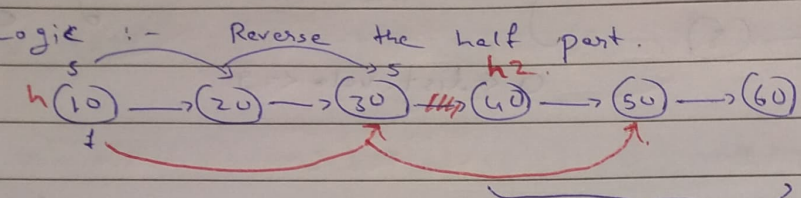

```
p s LN reverseBetween(ListNode head, int left, int Right){
    ListNode a = null, b = null, c = null, d = null;
    ListNode temp = head;
    int pos = 1;
    while(temp != null){
        if(pos == left-1) a = temp;
        if(pos == left) b = temp;
        if(pos == right) c = temp;
        if(pos == right+1) d = temp;
        temp = temp.next;
        pos++;
    }
    if(a != null) a.next = null;
    c.next = null;
    reverseList(b);
    b.next = d;
    if(a == null) return c;
    a.next = c;
    return head;
}
```

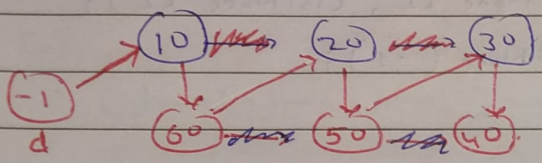
T.C ⇒ O(n) //

(17) Leetcode Q.No. :- 143 {Reorder List}.



Logic :- Reverse the half part.





⇒ Code :-

```
public void reorderList(ListNode head){
    ListNode slow = head;
    ListNode fast = head;
    while(fast.next != null && fast.next.next != null){
        slow = slow.next;
        fast = fast.next.next;
    }
    ListNode head2 = slow.next;
    slow.next = null;
    head2 = reverseList(head2);
    ListNode d1 = new ListNode();
    ListNode t = d1;
    ListNode t1 = head;
    ListNode t2 = head2;
    while(t2 != null){
        t.next = t1;
        t1 = t1.next;
        t = t.next;
        t.next = t2;
        t2 = t2.next;
        t = t.next;
    }
    t.next = t1;
}
```

(18) LeetCode Q.No.:- 2 { Add Two Numbers } → Lists are already given in reverse Order.

M T W T F S S

Page No.: YOUVA

Date:

l1: (2) → (4) → (3)

\+ l2: (5) → (6) → (4)

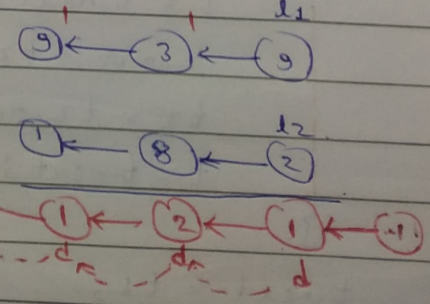
(7) → (0) → (8)

Logic is to do simple normal addition using carry

0 <= list.val <= 9.

=> Code :-

```
p s  ListNode addTwoNumbers ( ListNode l1 , ListNode l2 ){
        ListNode dummy = new ListNode (-1);
        ListNode temp = dummy;
        int carry = 0;
        while( l1 != null || l2 != null ){
            int val1 = 0 , val2 = 0;
            if( l1 != null) val1 = l1.val;
            if( l2 != null) val2 = l2.val;
            int num = val1 + val2 + carry;
            ListNode node = new ListNode (num % 10);
            temp.next = node;
            temp = temp.next;
            if( num > 9) carry = 1;
            else carry = 0;
            if( l1 != null) l1 = l1.next;
            if( l2 != null) l2 = l2.next;
        }
        if( carry > 0 ){
            ListNode node = new ListNode (carry);
            temp.next = node;
            temp = temp.next;
        }
        return dummy.next;
}
```



H.W

Leet Code Q. No. :- 445

(19). LeetCode Q. No. :- 725. {Split Linked List In Parts}

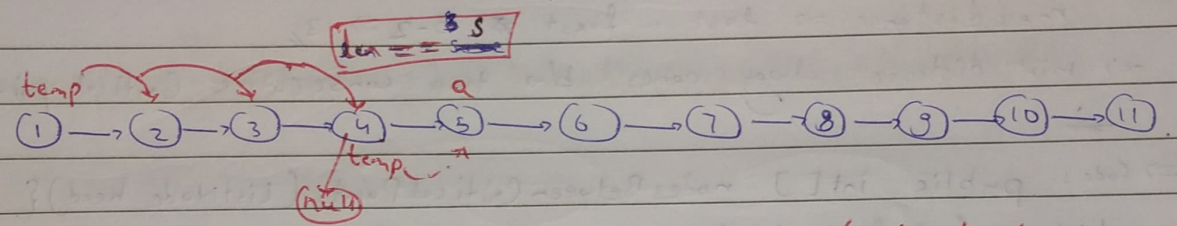
=> Split List into k-part. Size of parts should be equal as possible or should have size differing by ~~Not. One~~ No more than One

=> Ex :-

k = 3 (1)→(2)→(3)→(4)→(5)→(6)→(7)→(8)→(9)→(10).

Output. in [(1)→(2)→(3)→(4), (5)→(6)→(7), (8)→(9)→(10)]

Ex :- temp



k = 3.

s = size = $\frac{n}{k}$ (n → actual length of List) = $\frac{11}{3}$ => 3.

len = 1 1 1 1 1 1 .....

if(extra > 0) s++;

extra = n % k => 11 % 3 = 2 1 0.

=> Code :-

```
public static ListNode[] SplitListToParts(ListNode head, int k){
    int n = length of List(head);
    int size = n/k;    int extra = n%k;
    ListNode[] ans = new ListNode[k];
    ListNode temp = head;    int len = 1;    int idx = 0;
    while(temp != null){
        int s = size;
        if(extra > 0) s++;
        if(len == 1){
            ans[idx] = temp;
            idx++;
        }
        if(len == s){
            ListNode a = temp.next;
            temp.next = null;
            temp = a;
            len = 1;
            extra--;
        }
        else{
            len++;
            temp = temp.next;
        }
    }
    return ans;
}
```

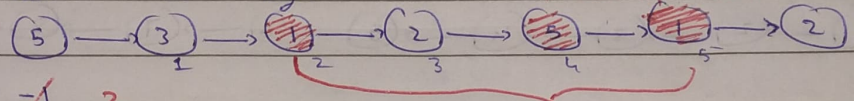
(20) LeetCode Q. No:- 2058. { Find the minimum & maximum Number of Nodes b/w Critical Points }

Page No.: Date: YOUVA

Critical Points :- (3) > (4) < (2). (2) < (5) > (1)

(4) Critical Point. (5) Critical Point

Ex :- left mid right.



Min^m distance b/w two C.P.

Max^m distance b/w two Critical Points

int first => ~~-1~~ 2

int last => ~~-1~~ ~~2~~ ~~4~~ 5

maxdistance => last - first => 5 - 2 => 3 //

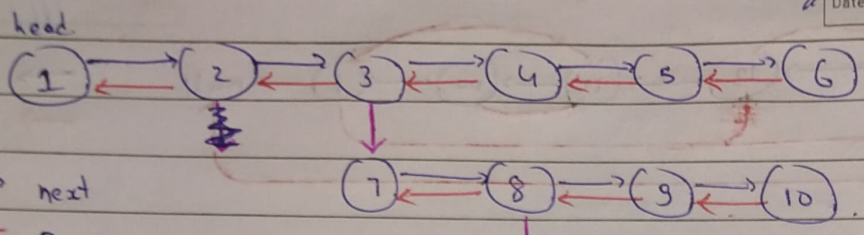
=> Min^m distance always comes b/w two consecutive Critical point.

=>

=> Code :-

```
public int[] nodesBetweenCriticalPoints(ListNode head){
    ListNode left = head;
    ListNode mid = head.next;
    ListNode right = head.next.next;
    int first = -1, last = -1; int minDistance = Integer.MAX_VALUE;
    int idx = 1; int[] ans = {-1,-1};
    while(right != null)
    {
        if(mid.val < left.val && mid.val < right.val || mid.val > left.val &&
                                                        mid.val > right.val){
            if(first == -1) first = idx;
            if(last != -1){
                int dist = idx - last;
                minDistance = Math.min(minDistance, dist);
            }
            last = idx;
        }
        idx ++
        left = left.next;
        mid = mid.next;
        right = right.next;
    }
    if(first == last) return ans;
    int maxDistance = last - first;
    ans[0] = minDistance;
    ans[1] = maxDistance;
}   return ans;
```

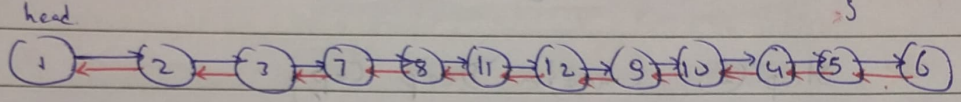
(21) Leet Code Q.No. 430 {Flatten a Multilevel Doubly Linked List}



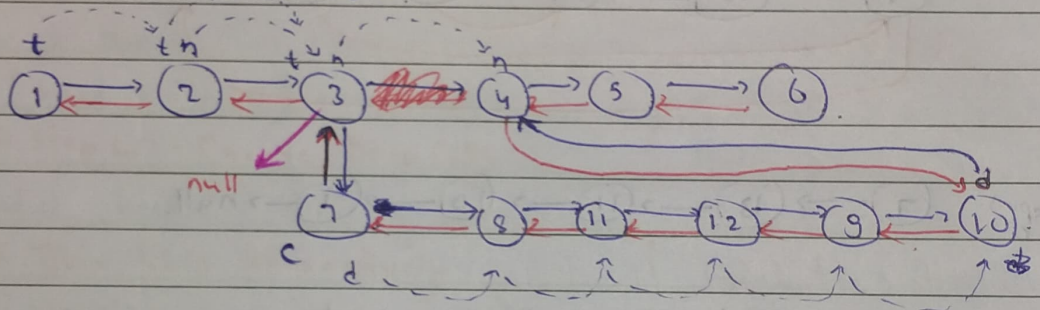
→ next
← Pev
↓ Child

```
Class Node {
    int val;
    Node next;
    Node prev;
    Node Child;
}
```

ans :-



→ Intuition → Recursion



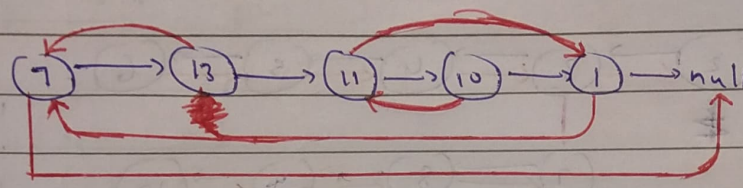
→ Code :-

```
public Static NodeList flatten(NodeList head){
    NodeList temp = head;
    while(temp != null){
        NodeList n = temp.next;
        if(temp.child != null){
            NodeList c = flatten(temp.child);
            NodeList d = c;
            while(d.next != null) d = d.next;
            temp.next = c ; c.prev = temp;
            d.next = n ;
            if(n != null) n.prev = d;
            temp.child = null;
        }
        temp = n;
    }
}
```

(22) Leet Code Q. No. 138. {Copy List with Random Pointer}

Page No.: YOUVA
Date:

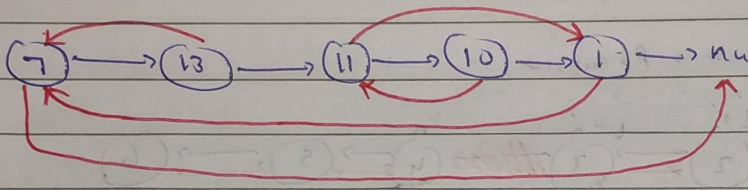
Need to Create a Deep Copy of this with Random



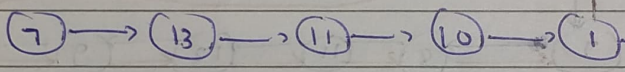
→ next
⇢ Random.

Logic

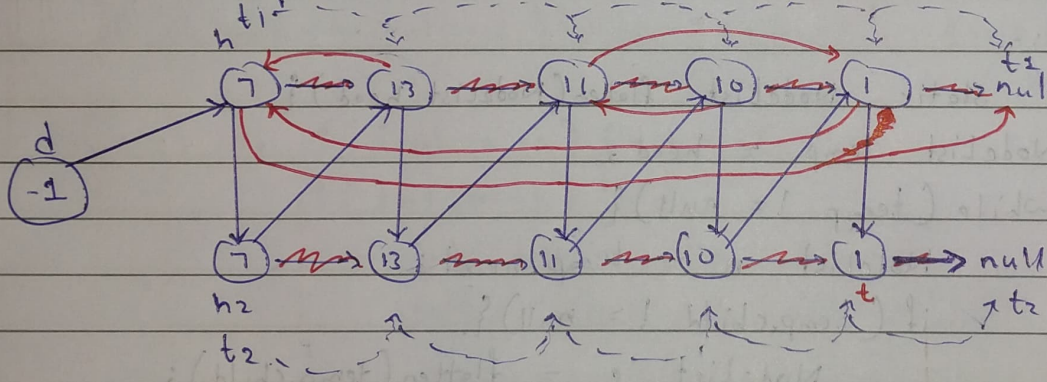
Step (1) Create Deep Copy.



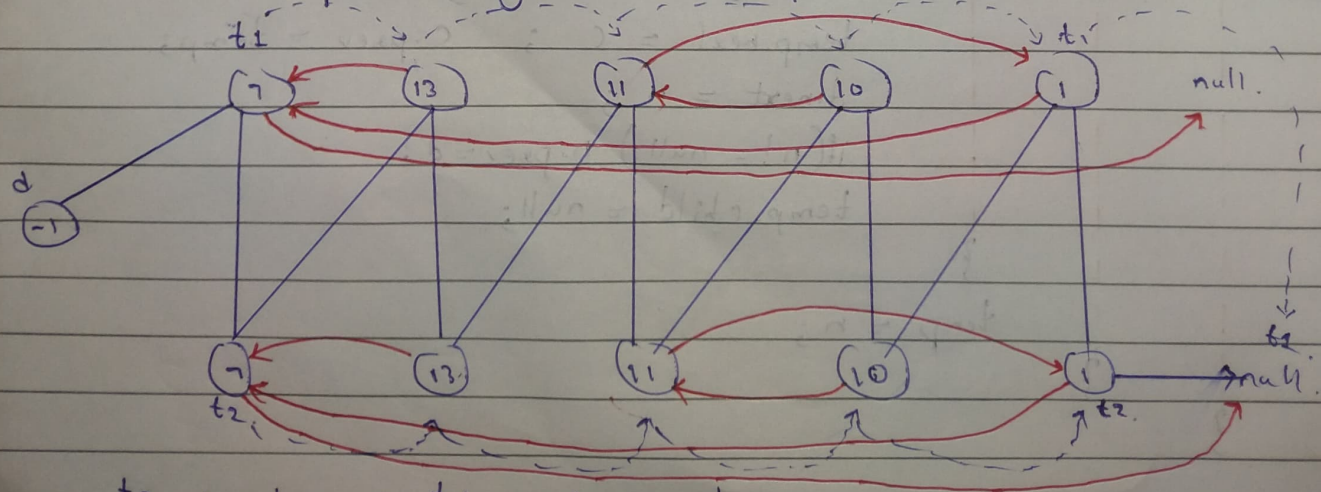
Deep Copy :-



Step (2) Join these Linked-List Alternatively.

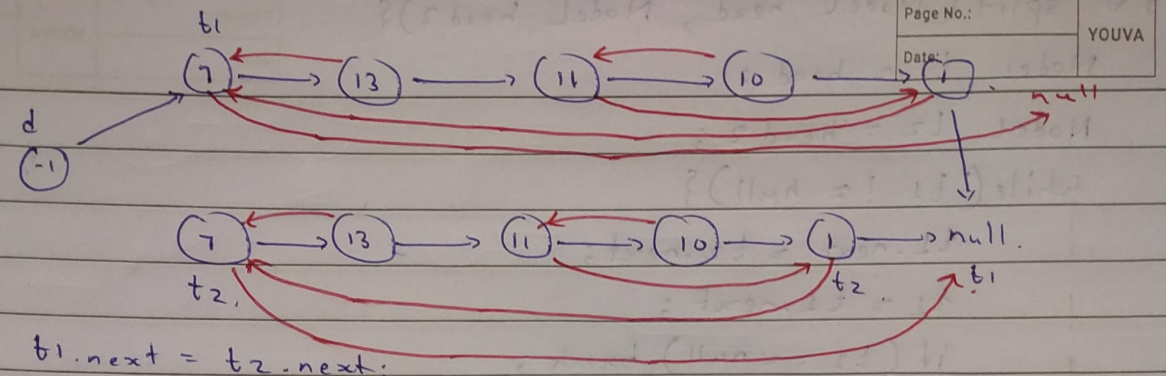


Step (3) Assign Random pointers.



t2.random = t1.random.next.

Step 4 Split the Linked-Lists Again.




```
t1.next = t2.next;
t1 = t1.next;
t2.next = t1.next;
t2 = t2.next;
```

T.C => O(n)

S.C => O(n).

we are not using any extra space.

=> Code:-

```
public S Node1 CopyRandomList(Node1 head){
    if (head == null) return null;
    // step 1 :- Create Deep Copy.
    Node1 head2 = deepCopy(head);
    // Join LL's Alternatively.
    ConnectAlternatively(head, head2);
    // Step 3 :- Assign Random Pointers.
    assignRandom(head, head2);
    // Step 4 :- Split the LL.
    Split(head, head2);
    return head2;
}
```

```
// Step (1).
p s Node1 deepCopy(Node1 head){
    Node1 head2 = new Node1(head.val);
    Node1 t1 = head.next;
    Node1 t2 = head2;
    while (t1 != null){
        Node1 temp = new Node1(temp.val);
        t2.next = temp;
        t2 = t2.next;
        t1 = t1.next;
    }
    return head2;
}
```

```
// Step (2).
p s v ConnectAlternatively(Node1 head, Node1 head2){
    Node1 dummy = new Node1(-1);
    Node1 t1 = head;
    Node1 t2 = head2;
    Node1 t = dummy;
    while (t1 != null && t2 != null){
        t.next = t1;
        t = t.next;
        t1 = t1.next;
        t.next = t2;
        t = t.next;
        t2 = t2.next;
    }
}
```

```
// Step (3).
p s v assignRandom(Node1 head, Node1 head2){
    Node1 t1 = head;
    Node1 t2 = head2;
    while(t1 != null){
        t2 = t1.next;
        if(t1.random != null)
            t2.random = t1.random.next;
        t1 = t1.next.next;
    }
}
```

// Step (4).

```
psv split(Node1 head, Node1 head2){
    Node1 t1 = head;
    Node1 t2 = head2;
    while(t1 != null){
        t1.next = t2.next;
        t1 = t1.next;
        if(t1 == null) break;
        t2.next = t1.next;
        t2 = t2.next;
    }
}
```