PW SKILLS

Prerequisites :

1) Practical OOPs
↓
User defined data type + constructors private, static

Linked List

→ Border Line

Arrays

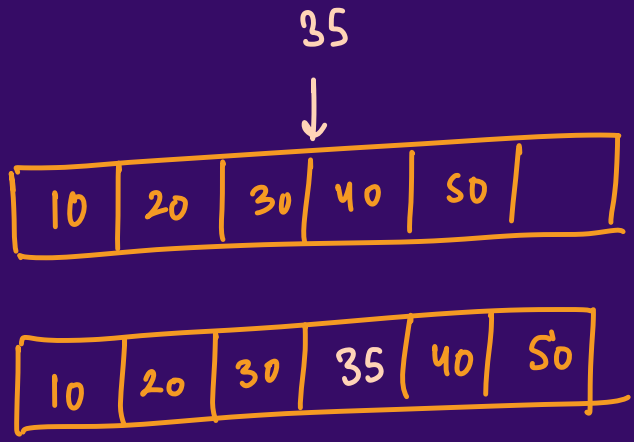
Limitations

1) Fixed Size

2) Insertion / Deletion

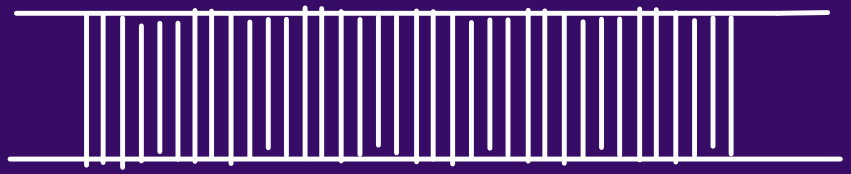
3) Continuous Memory Allocation

```
int[] arr = new int[10];
```




```
int x = 9;
```

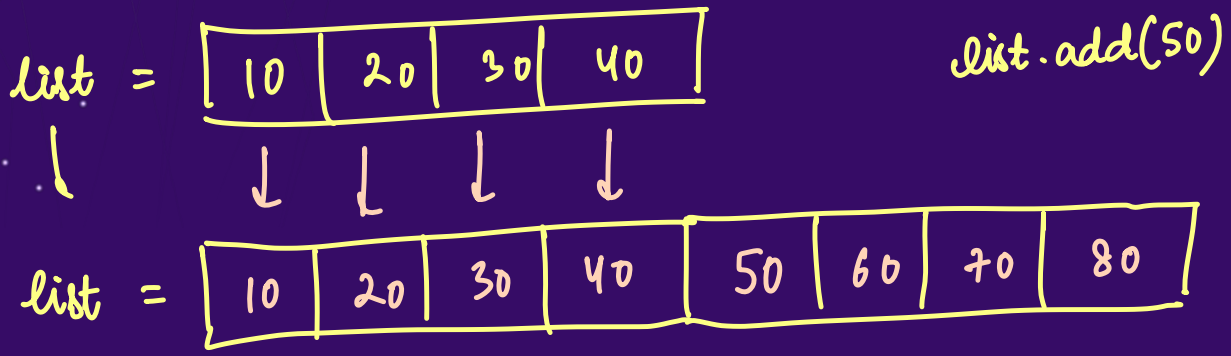
x will take 4 bytes in memory



Arrays

Limitations

How ArrayLists work ? Size & Capacity



list.add(60)
(70)
(80)
(90)

Arrays

Need for a new linear data structure → Truly Unlimited Size

↓ Continuous memory allocation ∝

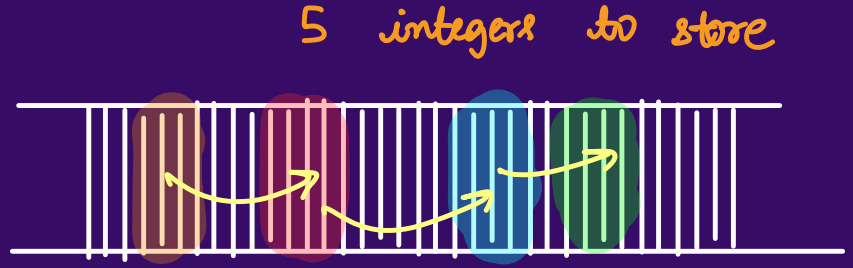
↘ Insertion, Deletion with O(1) space

Linked List

Introduction to Linked List

PW SKILLS

Idea of linking two non-contiguous memory locations (nodes)



Nodes

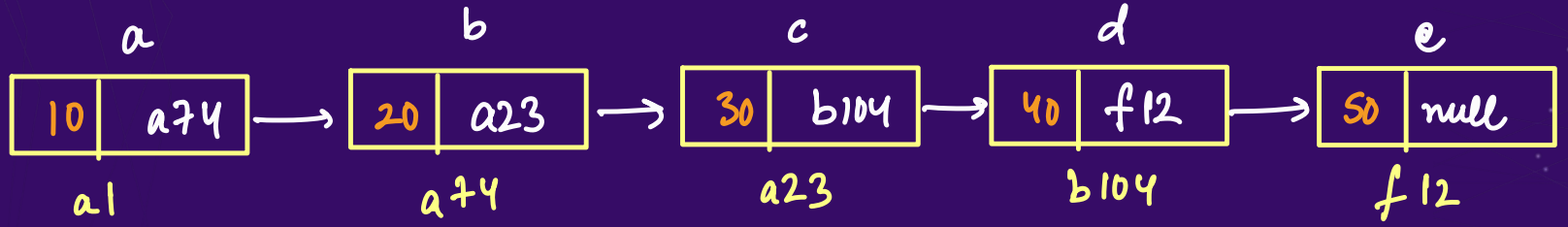
↓

value & address of next node

Introduction to Linked List

PW SKILLS

Idea of linking two non-contiguous memory locations (nodes)



these 5 objects are Nodes

One sided Love

Introduction to Linked List

PW SKILLS

Creation of a linked list

done, simple creation

Implementation

PW SKILLS

Node class

```
class Node{
    int val;
    Node next;
    Node(int val){
        this.val = val;
    }
}
```

With Parameterised constructor

Implementation

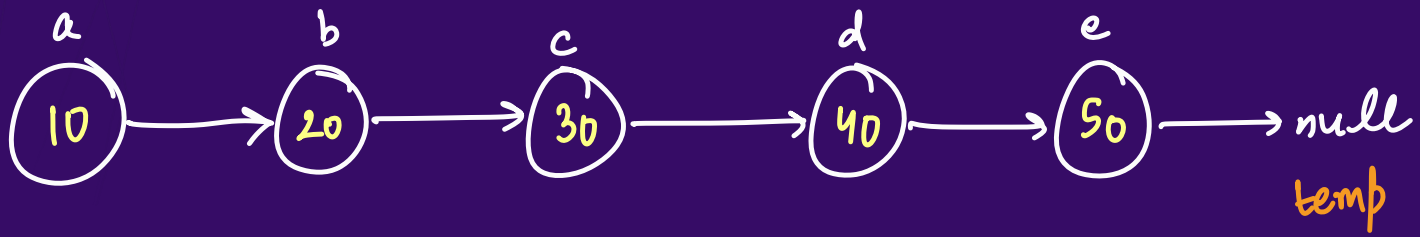
Linking nodes to form a linked list

```
Node a = new Node(10);
Node b = new Node(23);
Node c = new Node(37);
Node d = new Node(48);
Node e = new Node(53);
a.next = b; // 10->20
b.next = c; // 10->20->30
c.next = d; // 10->20->30->40
d.next = e; // 10->20->30->40->50
```

Displaying a Linked List

Shallow copy of node (temp)

Used to traverse the LL



temp = temp.next

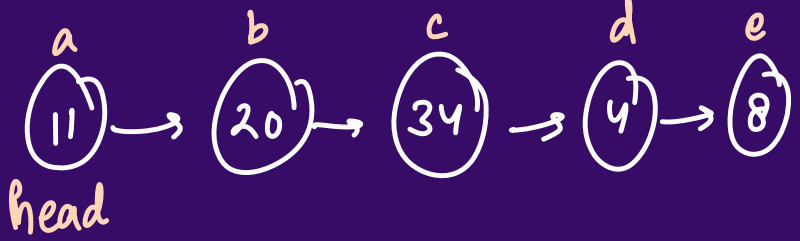
1) Node temp = a; ✓

2) Node temp = new Node(10); ✗

Displaying a Linked List

PW SKILLS

Can we do it recursively ?



What will this function do?

Do it yourself

PW SKILLS

```
void display(Node head) {
    if (head == null)
        return;
    display(head.next);
    System.out.print(head.val + " ");
}
```

A. **Print all the elements of the linked list.**

B. **Print all the elements except last one.**

C. **Print alternate nodes of linked list**

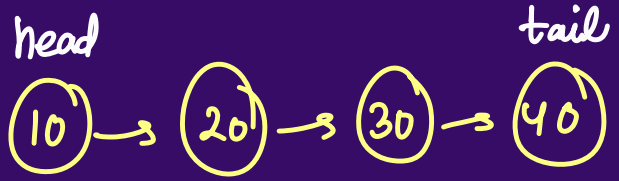
D. **Print all the nodes in reverse order**

Implementation

Linked List class

```
class LinkedList {
    ____
    ____
    ____
    ____

}
```



Display method

Implement display method to print all the elements

↓

T.C. = O(n)

S.C. = O(1)

Length method

Implement a method to find out the length of a Linked List (Iterative and Recursive)

↓

Implemented Linked List → T.C. = O(1)

If only head is provided → T.C. = O(n)

InsertAtEnd Function

PW SKILLS

Implement a method to insert a node at the end of a linked list.

Node temp = new Node(val)

1) if LL is empty

- head = tail = temp

T.C. = O(1)

S.C. = O(1)

when we have tail

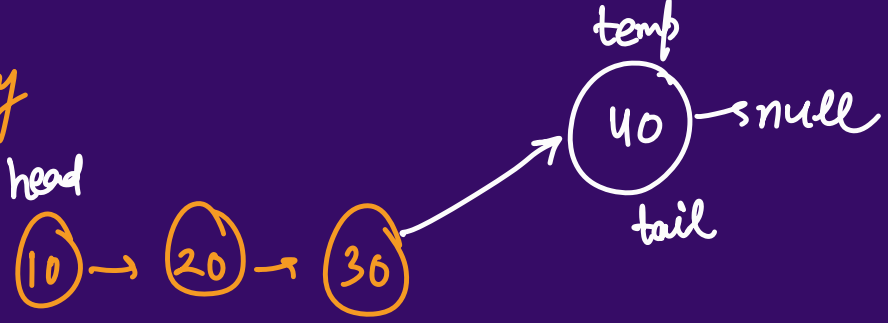
T.C. = O(n)

if tail is not provided

2) If LL is not empty

tail.next = temp

tail = temp



InsertAtBeginning Function

Head

PW SKILLS

Implement a method to insert a node at the start of a linked list.

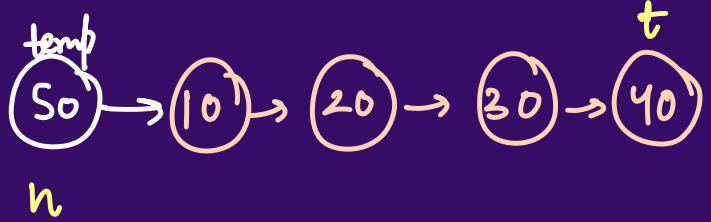
Node temp = new Node(val)

1) if LL is empty

- head = tail = temp

T.C. = O(1)
S.C. = O(1)

2) LL is not empty :



list.addHead(50);

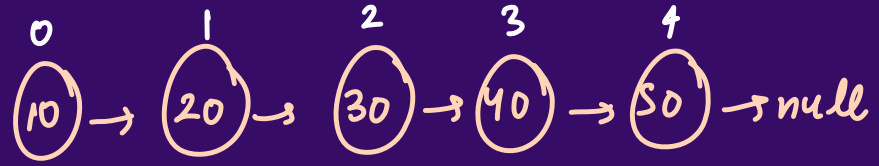
temp.next = head

h = temp

Insert method [0-based indexing]

PW SKILLS

Implement a method to insert a node at any given index.

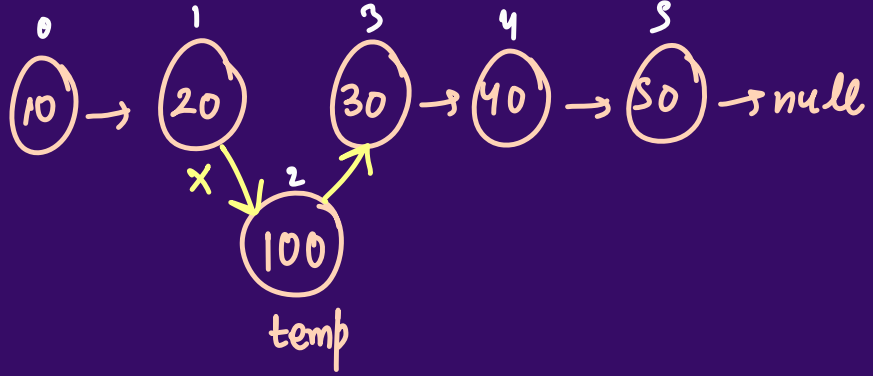


Create a temporary node 'x'.
↓
traverse x till 'idx-1' pos

list.insert(2, 100)

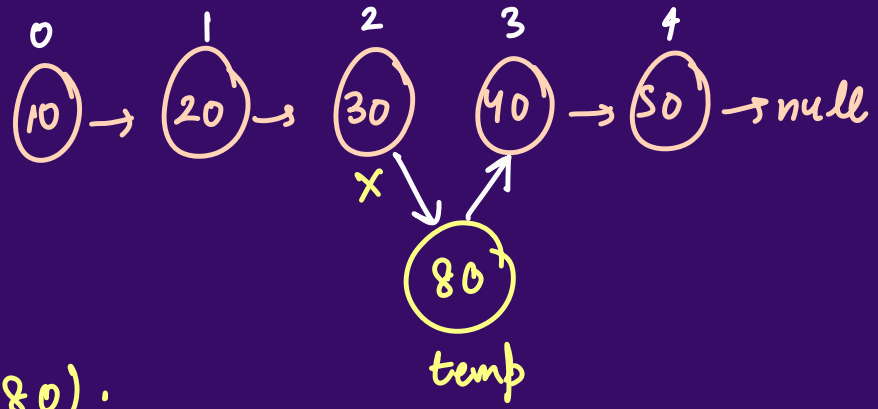
temp.next = x.next

x.next = temp



Insert method

Implement a method to insert a node at any given index.



list.insert (3, 80);
idx val

T.C. = O(n)
S.C. = O(1)

Insert method

PW SKILLS

Implement a method to insert a node at any given index.

Base Cases :

1) if (idx == 0) → insert at Head

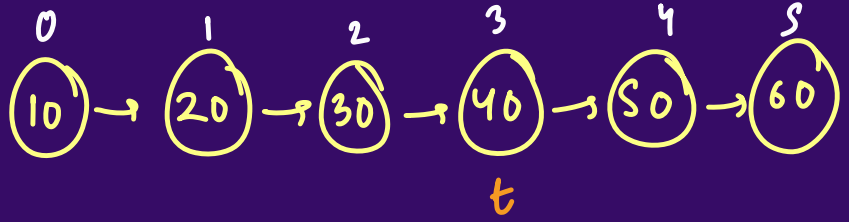
2) if (idx == size) → insert at tail

3) if (idx > size) → Error throw,
msg throw, ki invalid index

getElement method

PW SKILLS

Implement a method to return the element at any given index of the linked list.



idx

Sout (list . get (3)) ;

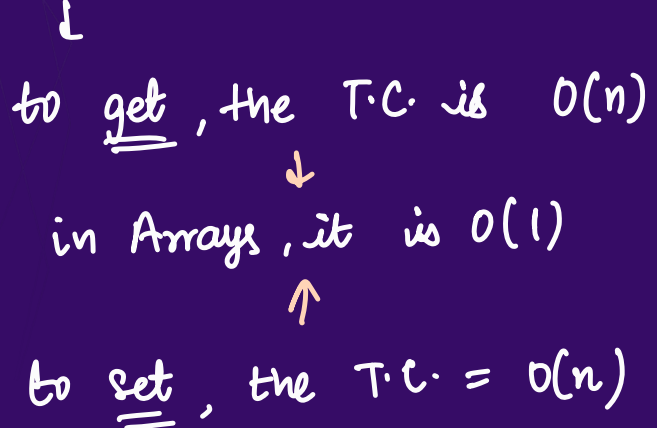
3 base cases → idx → 0, idx → size - 1, idx >= size || idx < 0

Extras

Error

T.C. = O(n)

An evident limitation of LinkedList



list.set(3, 200)

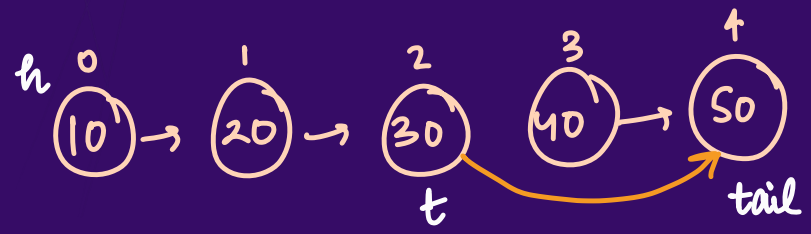
deleteAtIndex method

PW SKILLS

Implement a function to delete a node at a given index

deleteAtHead() → h = h.next

Size --



deleteAtIndex(3)

temp.next = temp.next.next

Size --

to delete a node at i^{th} index, we need the $(i-1)^{th}$ node

deleteAtIndex method

PW SKILLS

Implement a function to delete a node at a given index

Homeworks :

1) implement LL atleast 3-4 times yourself

2) Make a LL by just connecting & then write a code to find tail

```
class SLL{ // User defined data structure
    private Node head;
    Node tail;
    private int size;
    void insertAtTail(int val){...}
    void insertAtHead(int val){...}
    void insert(int idx, int val){...}
    int get(int idx) throws Error{...}
    void set(int idx, int val) throws Error{...}
    void deleteAtHead() throws Error{...}
    void delete(int idx) throws Error{...}
    void display(){...}
    void size() { System.out.println("Size : " + size); }
}
```

```java
void insertAtTail(int val){
    Node temp = new Node(val);
    if(head==null) head = tail = temp;
    else{
        tail.next = temp;
        tail = temp;
    }
    size++;
}
```

```java
void insertAtHead(int val){
    Node temp = new Node(val);
    if(head==null) head = tail = temp;
    else{
        temp.next = head;
        head = temp;
    }
    size++;
}
```

```java
void insert(int idx, int val){
    if(idx==0){
        insertAtHead(val);
        return;
    }
    if(idx==size){
        insertAtTail(val);
        return;
    }
    if(idx>size || idx<0){
        System.out.println("Invalid Index!!");
        return;
    }
    Node temp = new Node(val);
    Node x = head;
    for(int i=1;i<=idx-1;i++){...}
    temp.next = x.next;
    x.next = temp;
    size++;
}
```

```
int get(int idx) throws Error{
    if(idx==size-1) return tail.val;
    if(idx>=size || idx<0){
        throw new Error("Bhai ERROR");
    }
    Node temp = head;
    for(int i=1;i<=idx;i++){
        temp = temp.next;
    }
    return temp.val;
}
```

```
void set(int idx, int val) throws Error{
    if(idx==size-1){
        tail.val = val;
    }
    if(idx>=size || idx<0){
        throw new Error("Bhai ERROR");
    }
    Node temp = head;
    for(int i=1;i<=idx;i++){
        temp = temp.next;
    }
    temp.val = val;
}
```

```
void deleteAtHead() throws Error{
    if(head==null) throw new Error("List is Empty");
    head = head.next;
    size--;
}
```

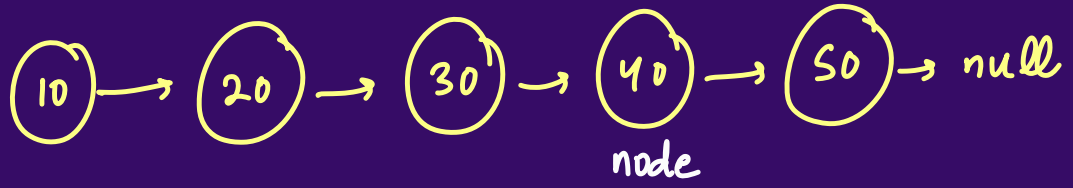
```
void delete(int idx) throws Error{
    if(idx==0){
        deleteAtHead();
        return;
    }
    if(idx<0 || idx>=size)
        throw new Error("Invalid Index");
    Node temp = head;
    for(int i=1;i<=idx-1;i++){
        temp = temp.next;
    }
    if(temp.next==tail) tail = temp;
    temp.next = temp.next.next;
    size--;
}
```

```
void display(){
    Node temp = head;
    while(temp!=null){
        System.out.print(temp.val+" ");
        temp = temp.next;
    }
    System.out.println();
}
void size(){
    System.out.println("Size : " + size);
}
```

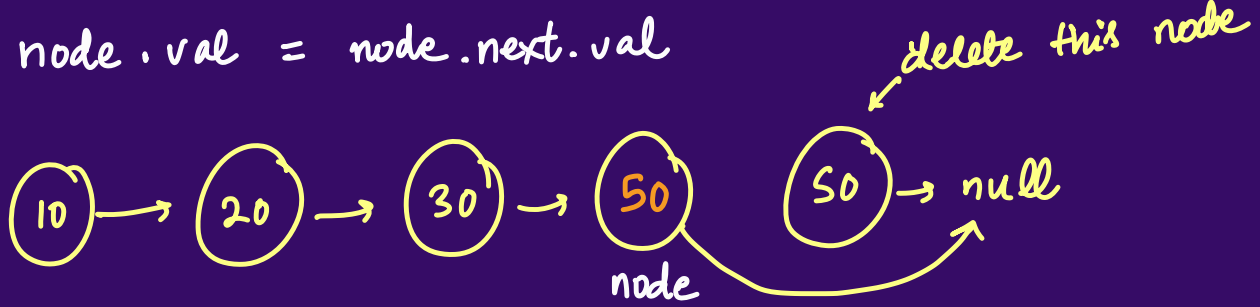
Ques:

PW SKILLS

Q1 : Delete Node in a Linked List



node . val = node . next . val



node . next = node . next . next

[Leetcode 237]

Ques:

PW SKILLS

Q1 : Delete Node in a Linked List

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

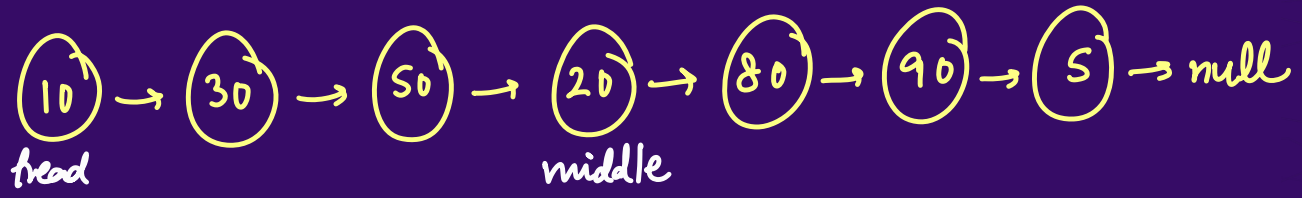
[Leetcode 237]

Ques: Very Famous

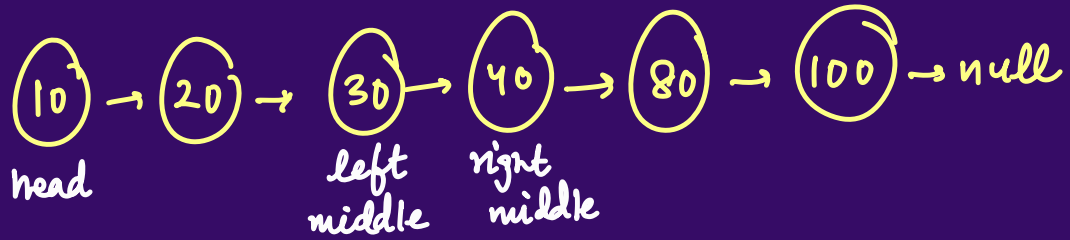
PW SKILLS

Q2 : Middle of the Linked List

n=7



n=6



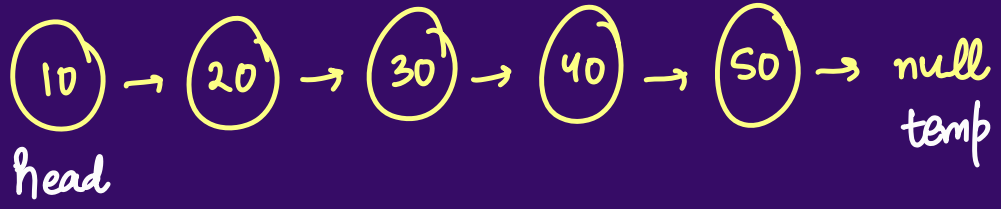
[Leetcode 876]

Ques:

PW SKILLS

Q2 : Middle of the Linked List

Length of a LL



len
↓
0
1
2
3
4
5

odd & even
↓
middle is $\left(\frac{n}{2}+1\right)^{th}$ node

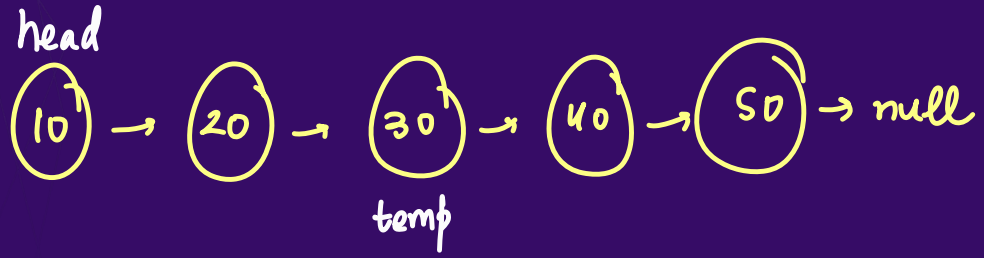
```
int len = 0;
while (temp != null){
    temp = temp.next;
    len ++;
}
```

[Leetcode 876]

Ques: Two pass solution

PW SKILLS

Q2 : Middle of the Linked List



len = 5

mid = $\frac{len}{2} + 1$;

```
for(int i=1; i<= mid-1 ; i++){
|     temp = temp.next;
3
```

T.C. = O(n) where 'n' is length of LL

Extra Space Used = O(1)

[Leetcode 876]

Ques:

Q2 : Middle of the Linked List

```java
public ListNode middleNode(ListNode head) {
    ListNode temp = head; // nothing is created
    int len = 0;
    while(temp!=null){
        temp = temp.next;
        len++;
    }
    int mid = len/2 + 1;
    temp = head;
    for(int i=1;i<=mid-1;i++){
        temp = temp.next;
    }
    return temp;
}
```

[Leetcode 876]

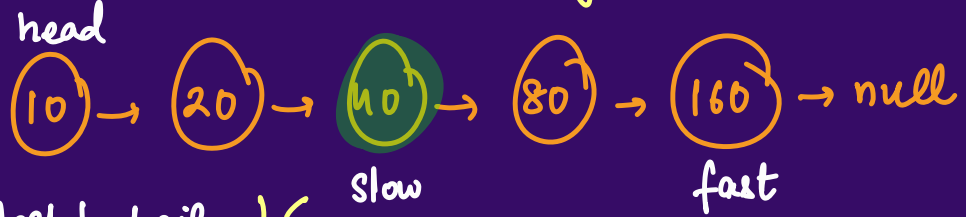
Ques:

PW SKILLS

Q2 : Middle of the Linked List

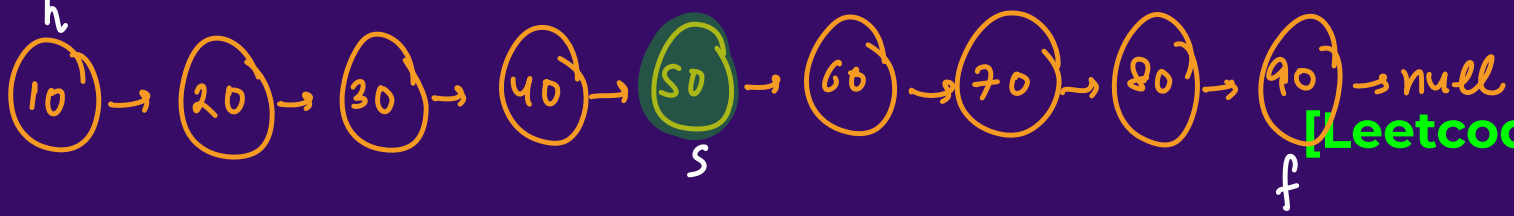
M-2 One pass solution / without finding the length / slow-fast approach

Odd Length :




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

fast.next != null



[Leetcode 876]

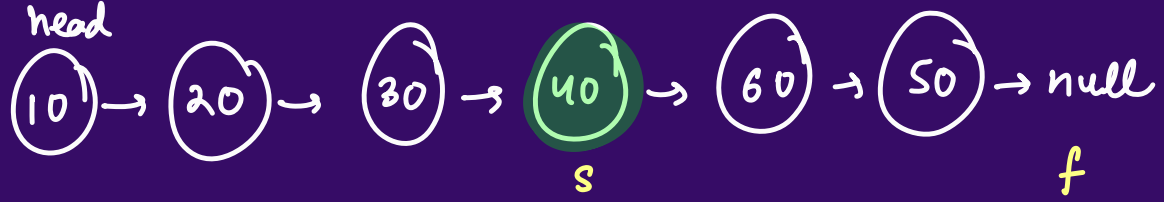
Ques: → Even → while (fast != null)

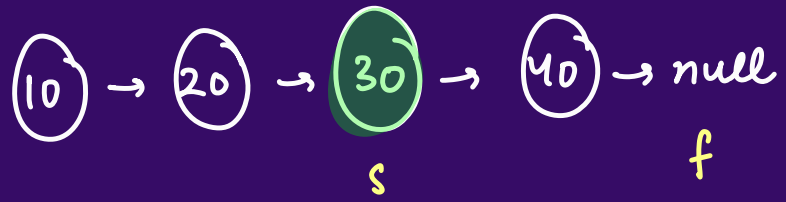
PW SKILLS

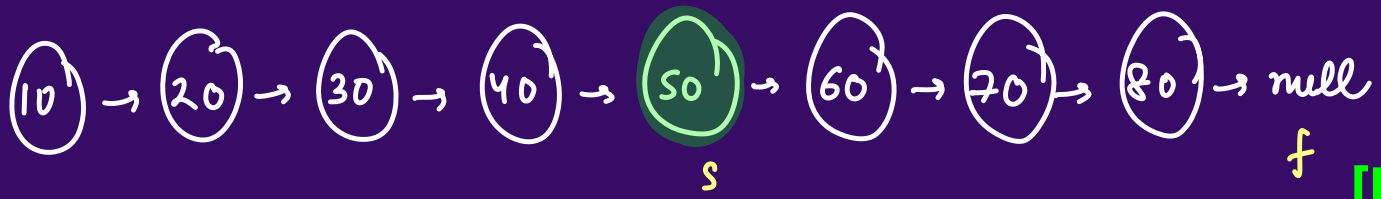
Q2 : Middle of the Linked List

Right Middle

Even length List :







[Leetcode 876]

Ques:

PW SKILLS

Q2 : Middle of the Linked List

```
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;
}
```

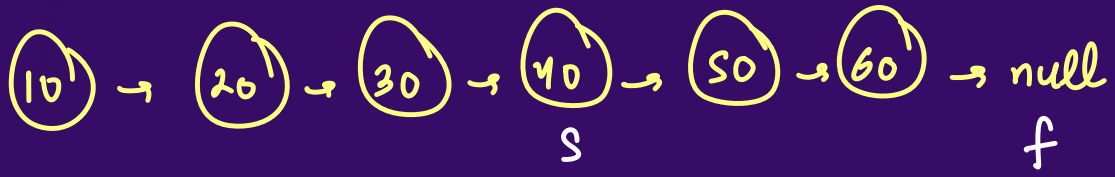
[Leetcode 876]

Ques:

PW SKILLS

Q2 : Middle of the Linked List

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



Error → null.next != null

[Leetcode 876]

Homework

Q1 : Find the left middle (or middle) of a linked list

PW SKILLS

Q2: Delete the Middle node of a Linked List

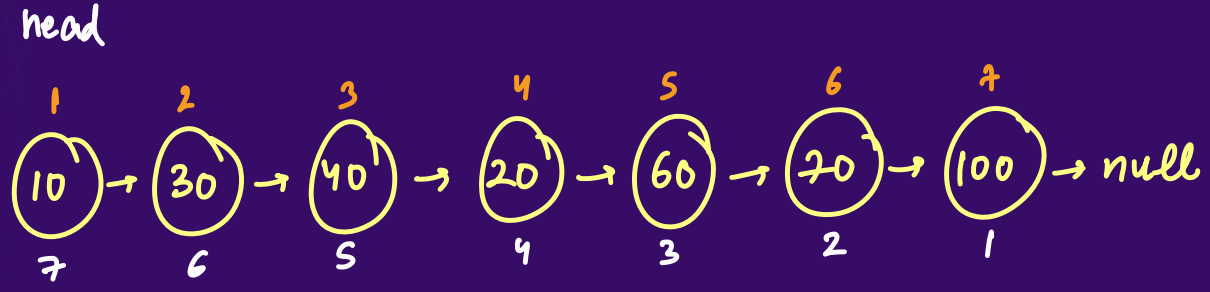
you have to delete the 'node' & the not just the 'val'.

[Leetcode 2095]

Ques:

PW SKILLS

Q3 : Remove Nth Node from End of List



n = 3

len = 7

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

to delete $(len - n + 1)^{th}$ node, we need $\underbrace{(len - n)^{th}}_{temp}$ node

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

[Leetcode 19]

Ques:

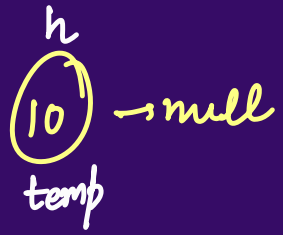
PW SKILLS

Q3 : Remove Nth Node from End of List

Edge Cases :

n = 1

len = 1



1st from end = (len - 1 + 1) from start

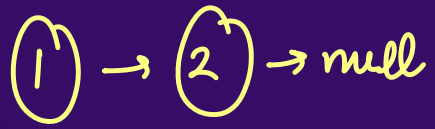
= 1st from start

[Leetcode 19]

Ques:

if (n == len) then it means we have to delete the head

Q3 : Remove Nth Node from End of List



return head.next

n = 2

len = 2

[Leetcode 19]

Ques:

PW SKILLS

Q3 : Remove Nth Node from End of List

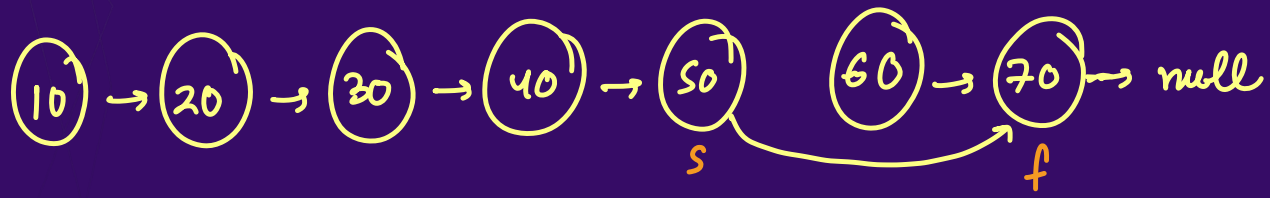
```java
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode temp = head;
    int len = 0;
    while(temp!=null){
        temp = temp.next;
        len++;
    }
    // edge cases
    if(len==n) return head.next;
    // n from end = (len-n+1) from start
    // we need a temp = len-n
    temp = head;
    for(int i=1;i<=len-n-1;i++){
        temp = temp.next;
    }
    // deletion
    temp.next = temp.next.next;
    return head;
}
```

[Leetcode 19]

Ques:

PW SKILLS

Q3 : Remove Nth Node from End of List (Slow - fast Approach)



n = 2

1) Move 'fast' n steps ahead
2) Move 'slow' & 'fast' together till fast.next!=null
3) Slow.next = slow.next.next

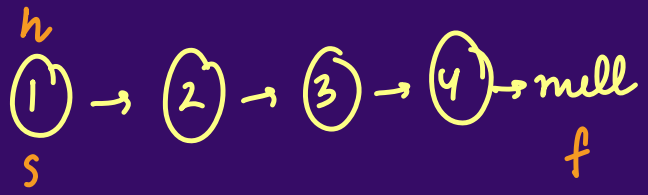
[Leetcode 19]

Ques:

PW SKILLS

Q3 : Remove Nth Node from End of List

Edge Cases : when (n == len)



n=4

[Leetcode 19]

Ques:

PW SKILLS

Q3 : Remove Nth Node from End of List

```java
public ListNode removeNthFromEnd(ListNode head, int n) {
    ListNode slow = head;
    ListNode fast = head;
    // move fast n steps ahead
    for(int i=1;i<=n;i++){
        fast = fast.next;
    }
    if(fast==null){ // n == len
        // I have to delete the head
        return head.next;
    }
    // move slow and fast together until fast reaches tail
    while(fast.next!=null){
        slow = slow.next;
        fast = fast.next;
    }
    slow.next = slow.next.next;
    return head;
}
```

[Leetcode 19]

Homework

PW SKILLS

Q : Swapping Nodes in a Linked List

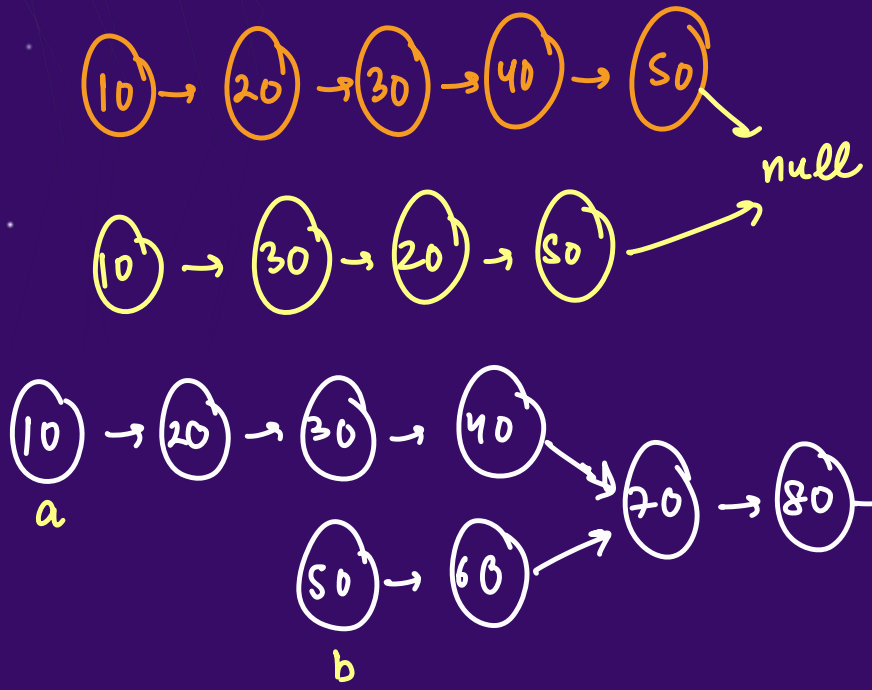
Q/ : Find the n^{th} node from end of a LL

[Leetcode 1721]

Ques:

PW SKILLS

Q4 : Intersection of two Linked Lists (Very Good)



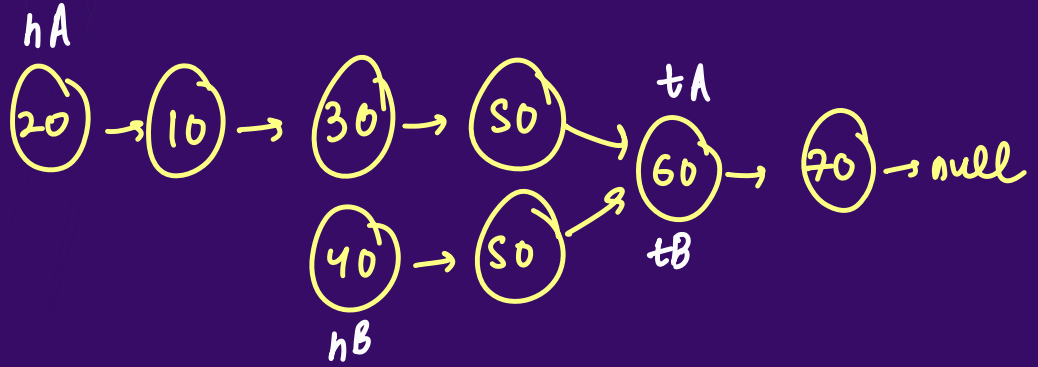
Hint : 1) Calculate lengths of A, B

2) Diff. of lengths

[Leetcode 160]

Ques:

Q4 : Intersection of two Linked Lists




```
while (tA != tB) {
    tA = tA.next
    tB = tB.next
}
return tA;
   |
   v
   or
   tB
```

lenA = 6
lenB = 4

diff → 2

[Leetcode 160]

Ques:

PW SKILLS

Q4 : Intersection of two Linked Lists

```
int lenA = 0;
ListNode tempA = headA;
while(tempA!=null){
    tempA = tempA.next;
    lenA++;
}
int lenB = 0;
ListNode tempB = headB;
while(tempB!=null){
    tempB = tempB.next;
    lenB++;
}
```

1

```
tempA = headA;
tempB = headB;
if(lenA > lenB){
    for(int i=1;i<=lenA-lenB;i++){
        tempA = tempA.next;
    }
}
else{ // lenB >= lenA
    for(int i=1;i<=lenB-lenA;i++){
        tempB = tempB.next;
    }
}
```

2

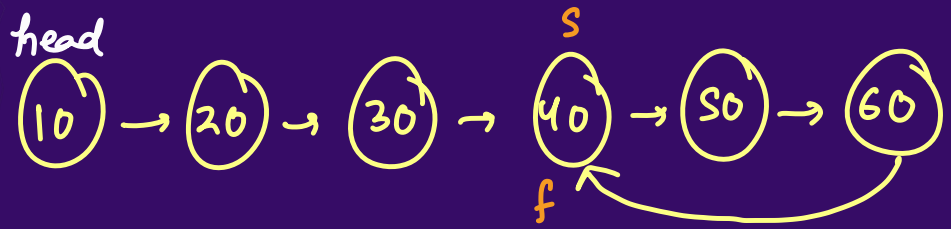
```
while(tempA!=tempB){
    tempA = tempA.next;
    tempB = tempB.next;
}
return tempA;
```

3

[Leetcode 160]

Ques:

Q5 : Linked List Cycle (slow & fast approach)




```
while ( fast != null && fast.next != null ){
    s = s.next;
    f = f.next.next;
    if(s == f) return true;
}
return false;
```

[Leetcode 141]

Ques:

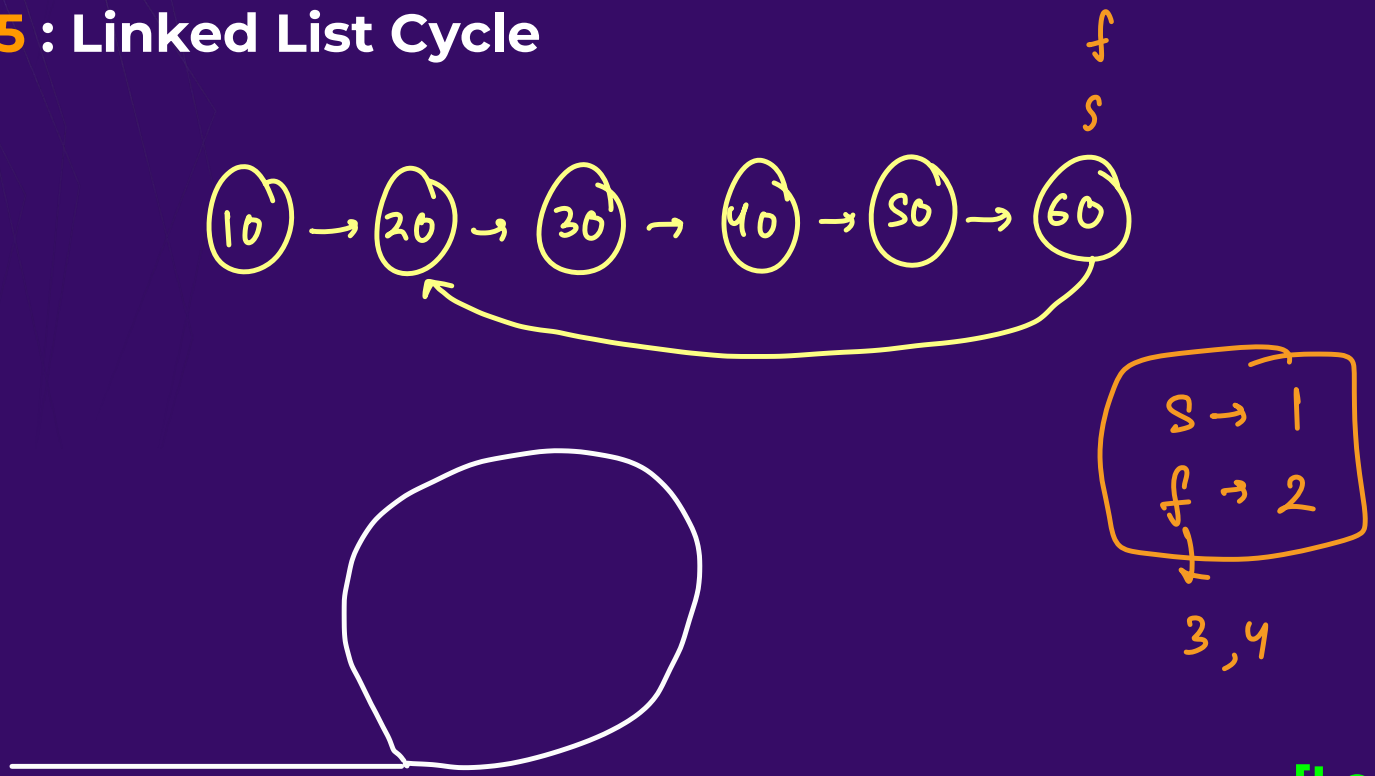
Q5 : Linked List Cycle

```java
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;
}
```

[Leetcode 141]

Ques:

Q5 : Linked List Cycle



[Leetcode 141]

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

These codes also work

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

Ques:

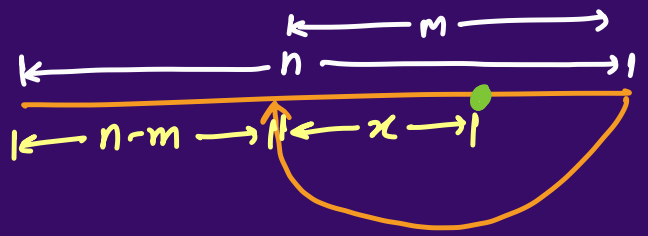
PW SKILLS

Q5 : Linked List Cycle (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

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

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

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

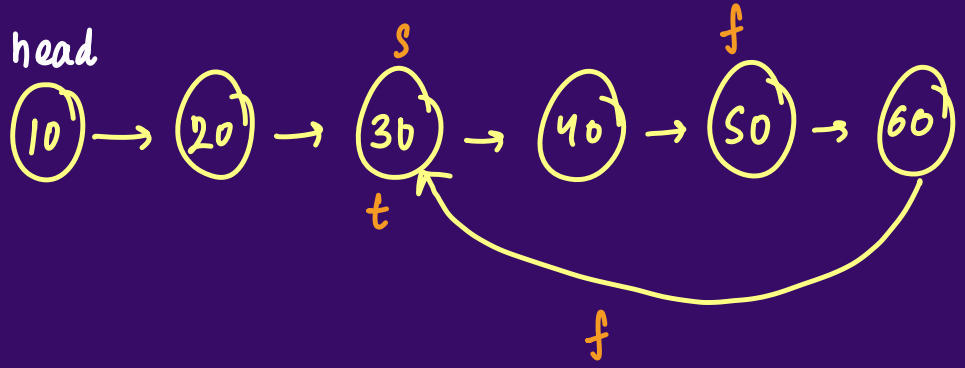
$$\Rightarrow \boxed{x = 2m - n}$$

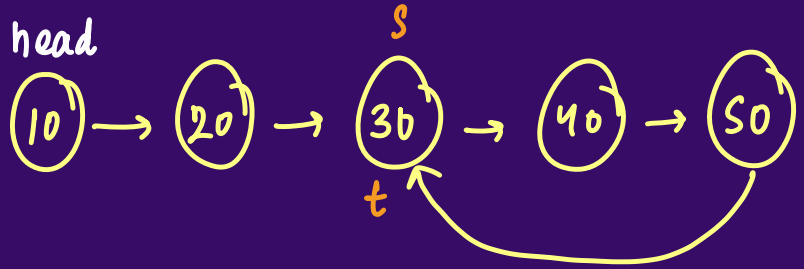
[Leetcode 141]

Ques:

PW SKILLS

Q6 : Linked List Cycle II

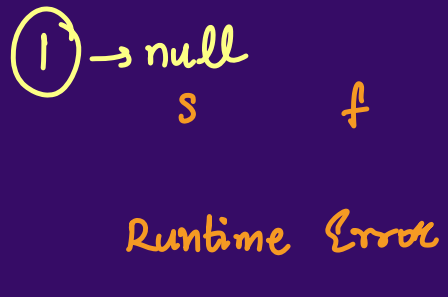




[Leetcode 142]

Ques:

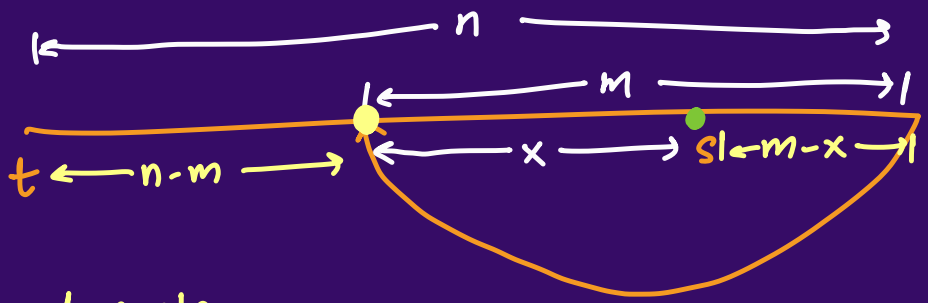
Q6 : Linked List Cycle II



[Leetcode 142]

Ques:

Q6 : Linked List Cycle II



to reach start of cycle

Slow → m - x = m - (2m - n) = m - 2m + n = n - m

temp → n - m

[Leetcode 142]

Ques:

Q6 : Linked List Cycle II

```java
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;
}
```

[Leetcode 142]

Homework

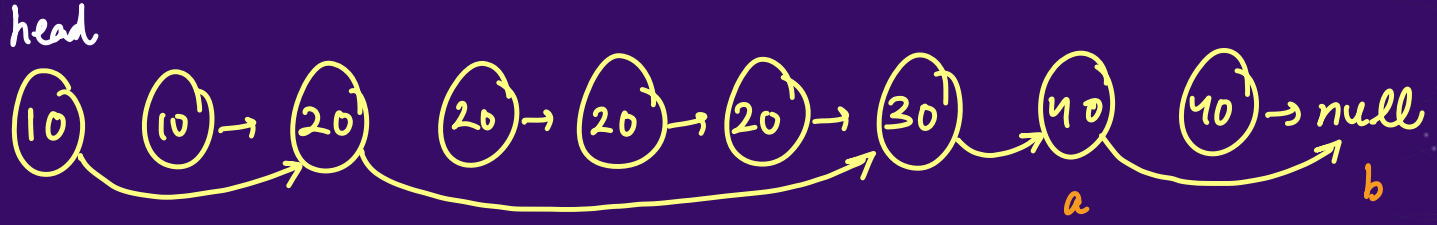
PW SKILLS

Q : Happy Number

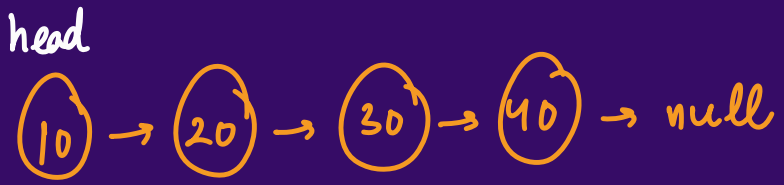
[Leetcode 202]

Ques: hint : 2 pointer approach

Q7 : Remove Duplicates from Sorted List



ans ⇒



[Leetcode 83]

Ques:

Q7 : Remove Duplicates from Sorted List

```java
public ListNode deleteDuplicates(ListNode head) {
    if(head==null) return head;
    ListNode a = head;
    ListNode b = head;
    while(b!=null){
        if(b.val==a.val) b = b.next;
        else{
            a.next = b;
            a = b;
        }
    }
    a.next = null;
    return head;
}
```

[Leetcode 83]

Homework:

PW SKILLS

Q : Remove Duplicates from Sorted List II

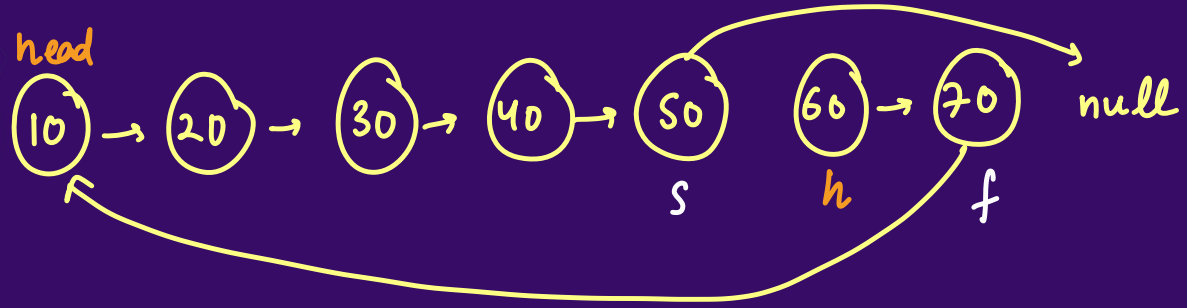
↓

Dummy Node

[Leetcode 82]

Ques:

Q8 : Rotate List



K = 2

if (K >= n) K %= n

Hint

nth node from the end delete.

[Leetcode 61]

Ques:

PW SKILLS

Q8 : Rotate List

```
if(head==null || head.next==null) return head;
ListNode temp = head;
int n = 0;
while(temp!=null){
    temp = temp.next;
    n++;
}
k %= n;
if(k==0) return head;
```

```
ListNode slow = head;
ListNode fast = head;
for(int i=1;i<=k;i++){
    fast = fast.next;
}
while(fast.next!=null){ // fast will be at tail
    slow = slow.next;
    fast = fast.next;
}
ListNode newHead = slow.next;
slow.next = null; // slow is now the new tail
fast.next = head;
return newHead;
```

[Leetcode 61]

Homework:

Q : Spiral Matrix IV

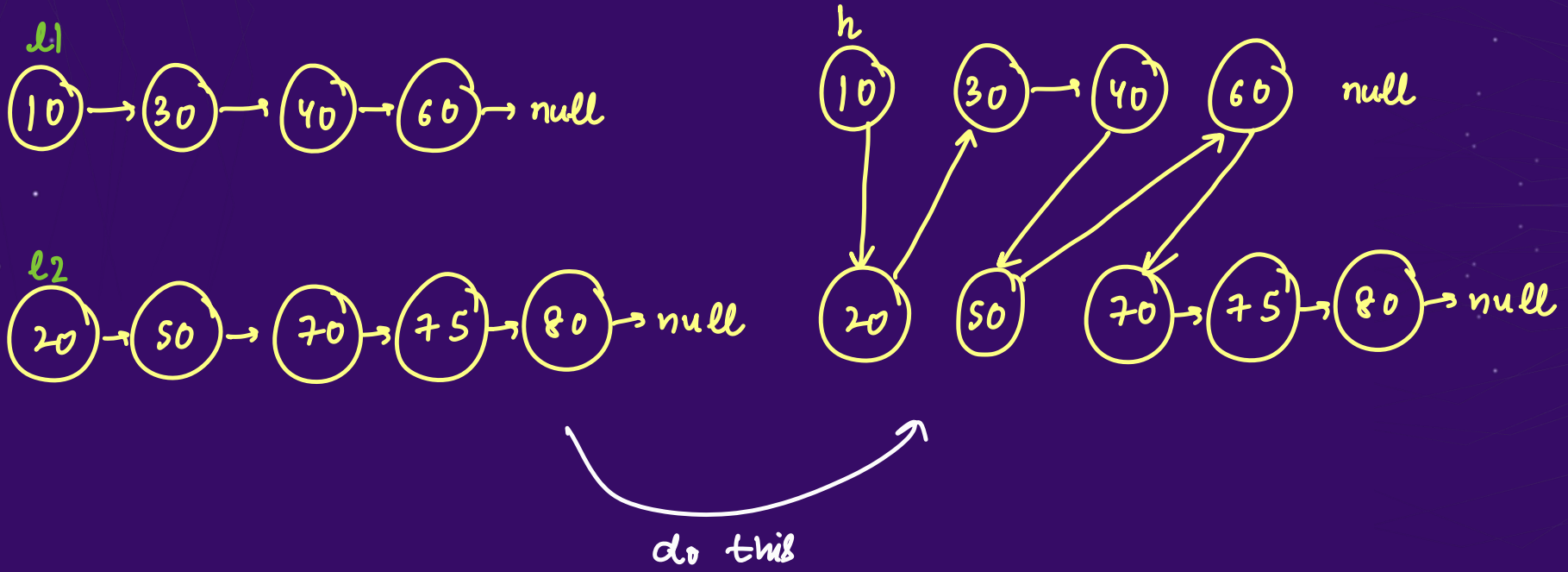
[Leetcode 2326]

Ques:

Prerequisites : Merge 2 sorted arrays
Merge Sort

SKILLS

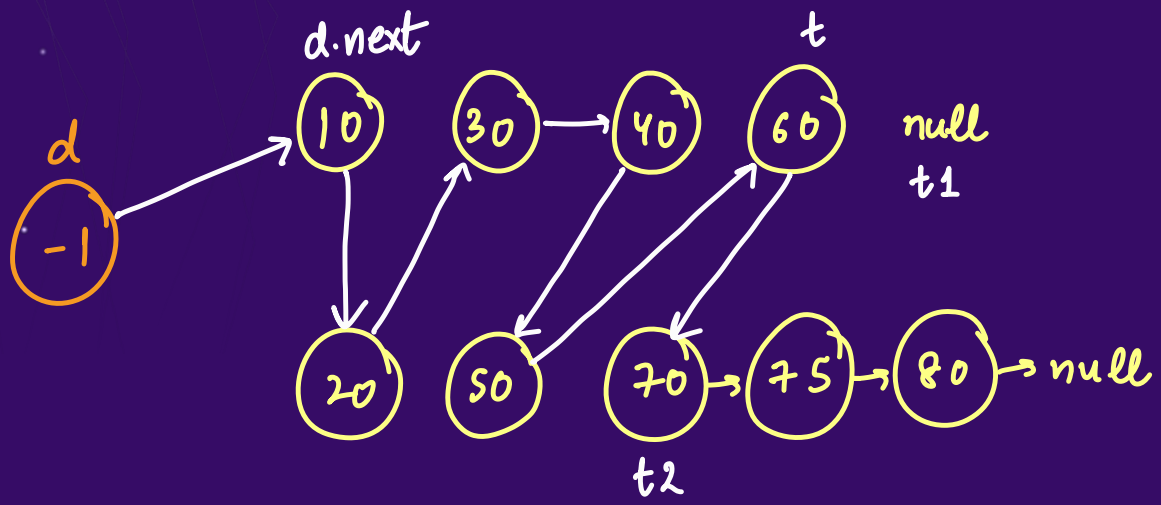
Q10 : Merge 2 sorted lists



[Leetcode 21]

Ques:

Q10 : Merge 2 sorted lists




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

[Leetcode 21]

Ques:

PW SKILLS

Q10 : Merge 2 sorted lists

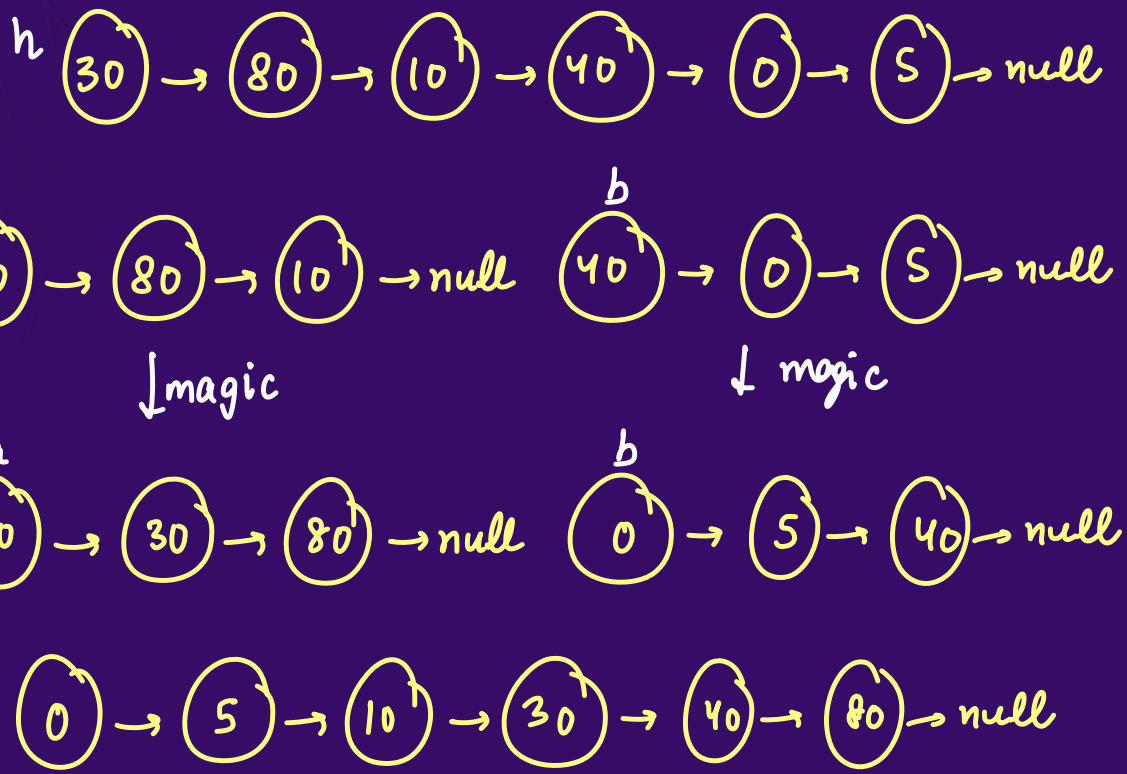
```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    ListNode dummy = new ListNode(100);
    ListNode temp = dummy;
    ListNode temp1 = list1;
    ListNode temp2 = list2;
    while(temp1!=null && temp2!=null){
        if(temp1.val<=temp2.val){
            temp.next = temp1;
            temp1 = temp1.next;
        }
        else{
            temp.next = temp2;
            temp2 = temp2.next;
        }
        temp = temp.next;
    }
    if(temp1==null) temp.next = temp2;
    else temp.next = temp1;
    return dummy.next;
}
```

[Leetcode 21]

Ques: #Hint : Merge Sort Algo & Merge 2 Sorted array

PW SKILLS

Q11 : Sort List



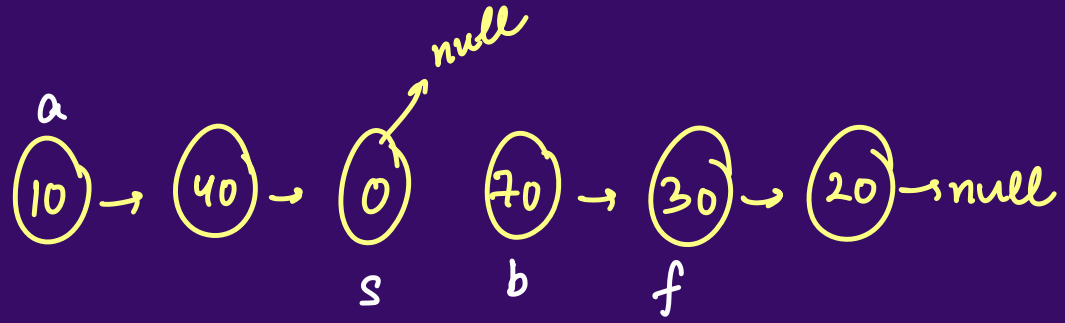
[Leetcode 148]

Ques:

PW SKILLS

Q11 : Sort List

Even length :




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

b = slow.next
slow.next = null
```

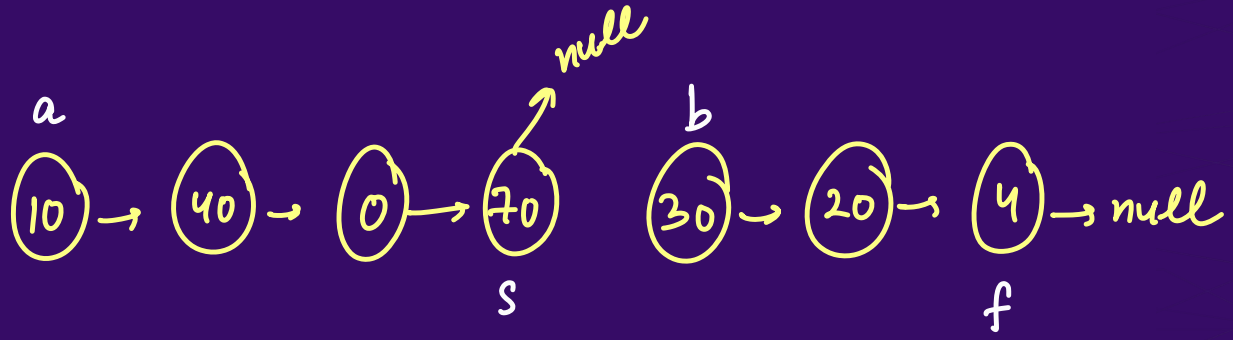
[Leetcode 148]

Ques:

PW SKILLS

Q11 : Sort List

Odd length :




```
while (fast.next != null) {
```

3

b = s.next

s.next = null

[Leetcode 148]

Ques:

Q11 : Sort 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 = mergeTwoLists(firstHalf,secondHalf);
    return ans;
}
```

[Leetcode 148]

Homework:

PW SKILLS

Q : Merge k sorted lists

arr = { l1, l2, l3, l4 }

arrlist = { l1, l2, l3, l4 }

⇒ { l1, l2, l34 }

⇒ { l1, l234 }

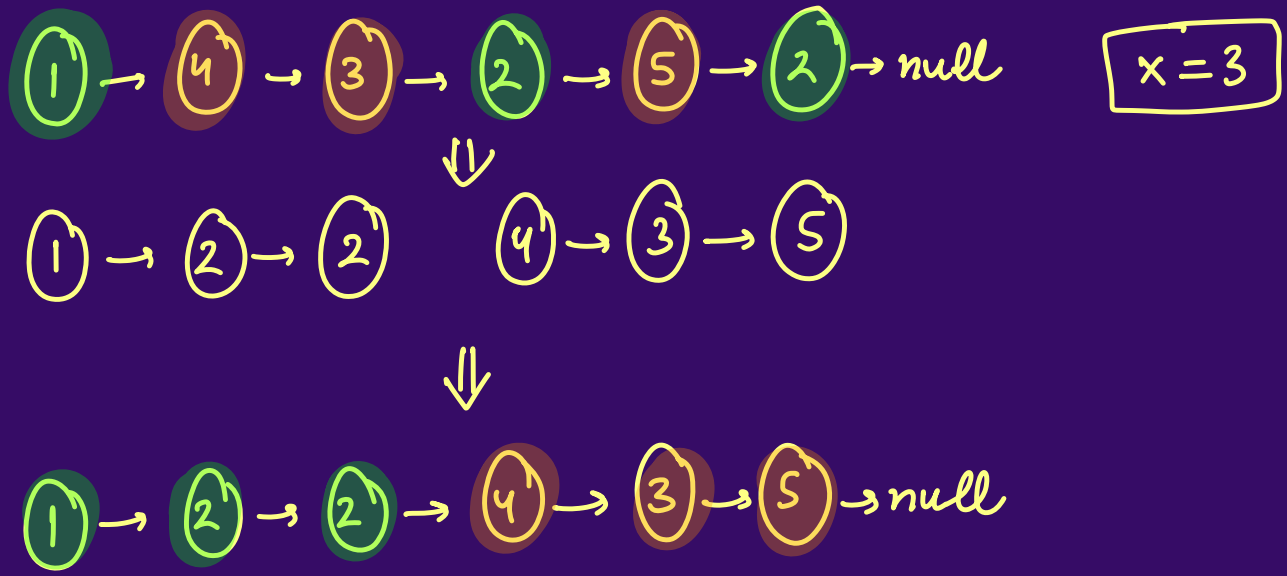
⇒ { l1234 }

[Leetcode 23]

Ques:

PW SKILLS

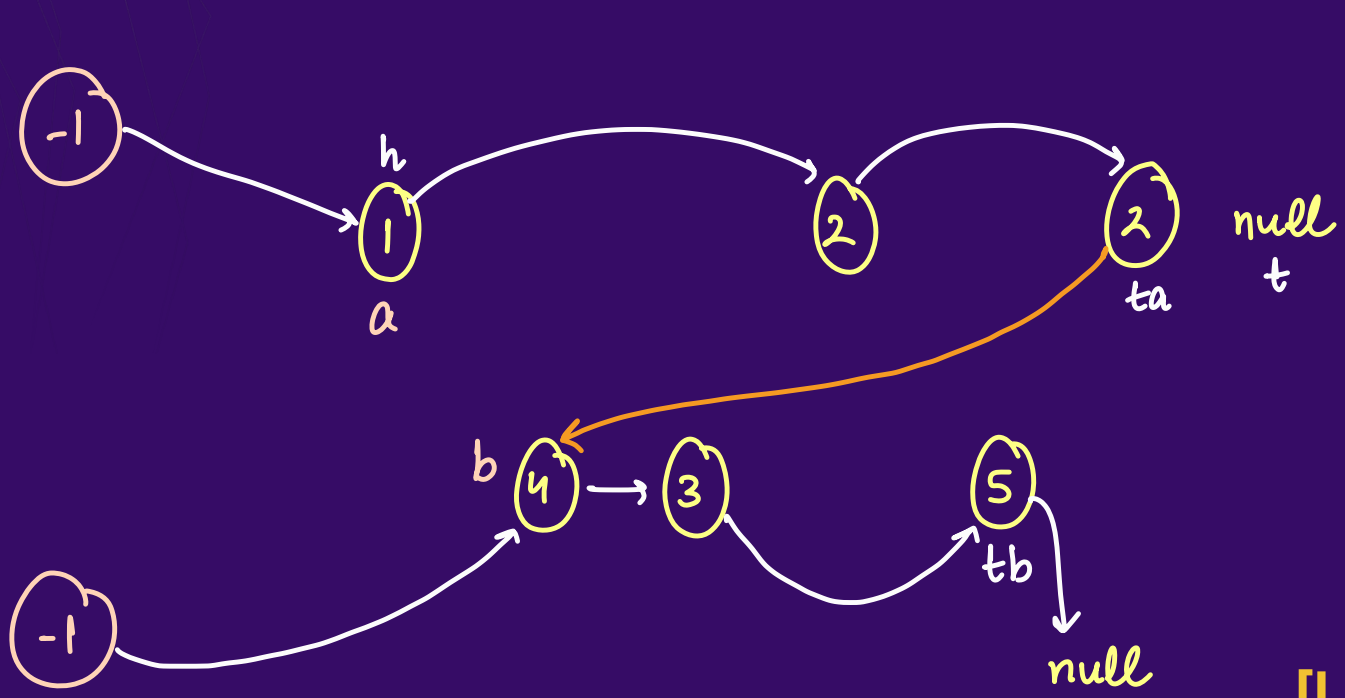
Q12 : Partition List



[Leetcode 86]

Ques: a vali LL me x se smaller elements honge
b vali LL me >=x elements

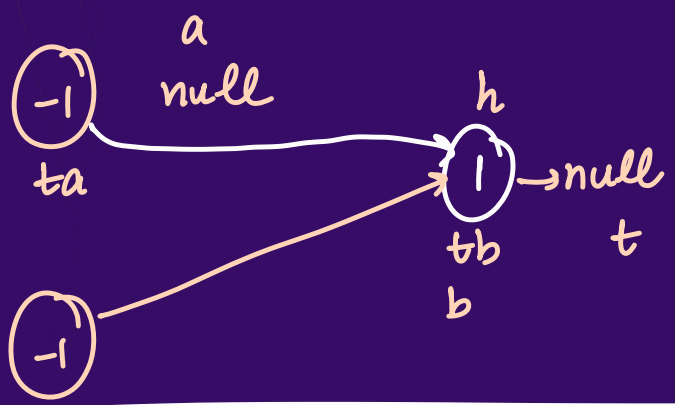
Q12 : Partition List



[Leetcode 86]

Ques:

Q12 : Partition List



x=0

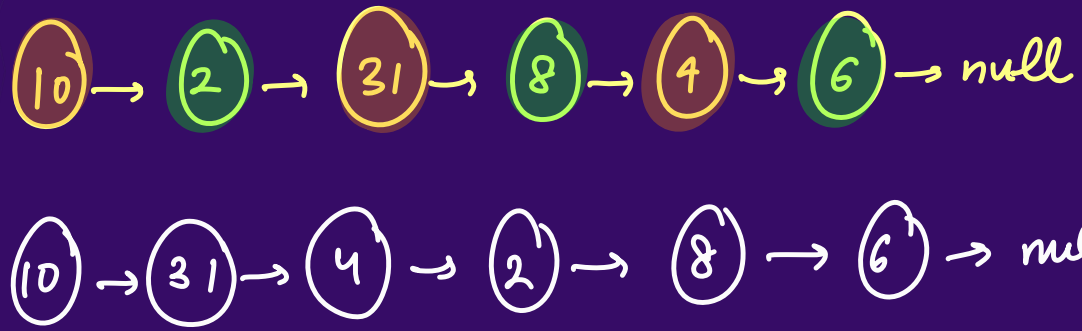
x=1

[Leetcode 86]

Homework:

PW SKILLS

Q : Odd Even Linked List

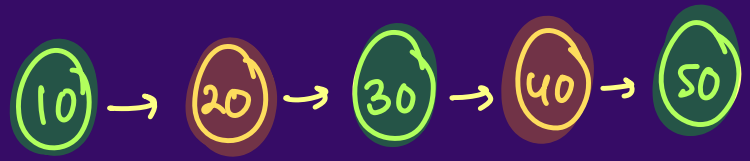


[Leetcode 328]

Homework:

PW SKILLS

Q : Swap Nodes in Pairs



[Leetcode 24]

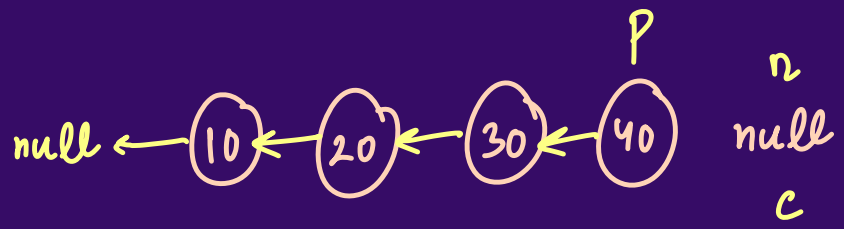
Ques:

PW SKILLS

Q13 : Reverse Linked List

M-I :

Iterative




```
while (c != null)
    n = c.next
    c.next = p
    p = c
    c = n
return p;
```

[Leetcode 206]

Ques:

PW SKILLS

Q13 : Reverse Linked List (Iterative Method)

```java
public ListNode reverseList(ListNode head) {
    ListNode curr = head;
    ListNode prev = null;
    ListNode Next = null; // head/null
    while(curr!=null){
        Next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = Next;
    }
    return prev;
}
```

T.C. = O(n)

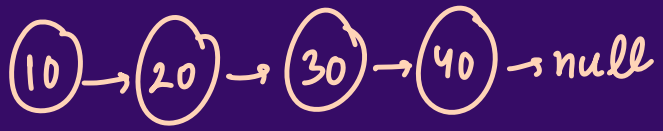
Extra Space Used = O(1)

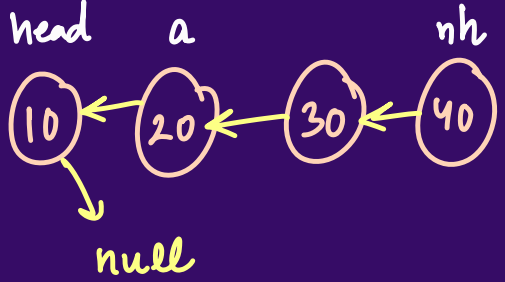
[Leetcode 206]

Ques:

Q13 : Reverse Linked List

believe in magic
↑
(Recursive Solution)






```
if(head==null || head.next==null)
    return head
a = head.next
ListNode nh = reverse(a)
a.next = head
head.next = null
```

[Leetcode 206]

Ques:

PW SKILLS

Q13 : Reverse Linked List

```java
public 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)

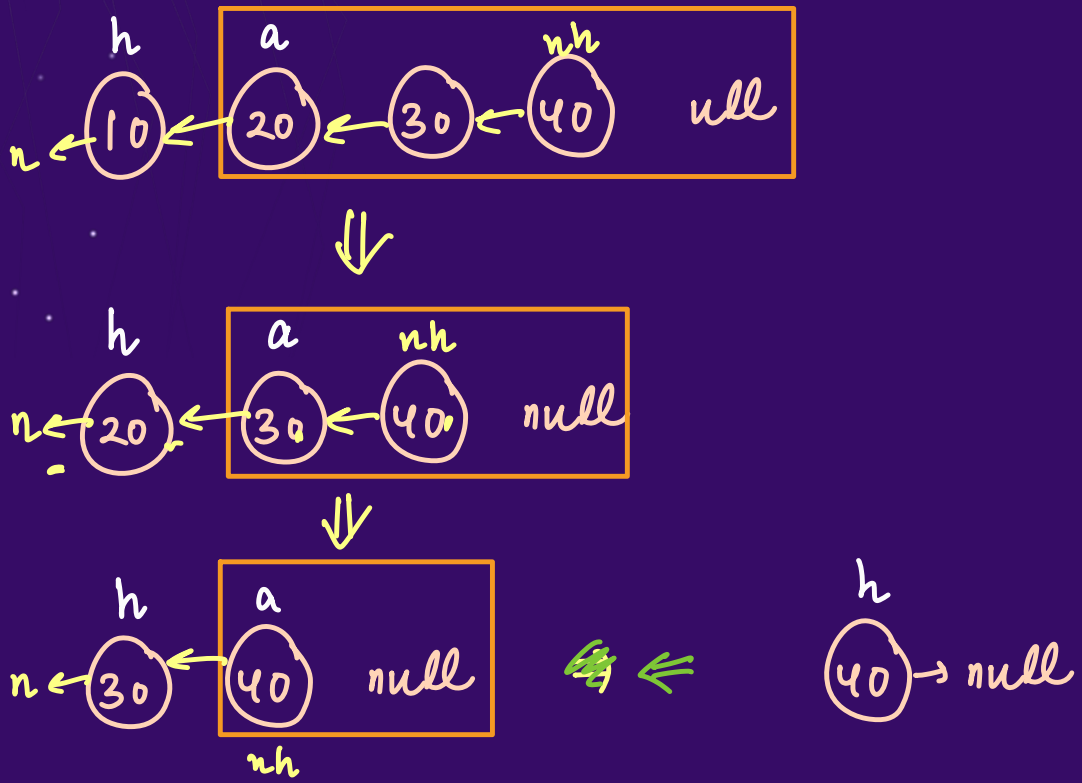
↓

call stack space

[Leetcode 206]

Ques:

Q13 : Reverse Linked List




```java
public 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;
}
```

[Leetcode 206]

Ques: M-0 : getNodeAt(idx) → T.C. = $O(n^2)$, S.C. = $O(1)$

PW SKILLS

Q14 : Palindrome Linked List

M-1 :

Creating an arrayList

temp

al = {10, 20, 30, 20, 10}

T.C. = $O(n)$

S.C. = $O(n)$

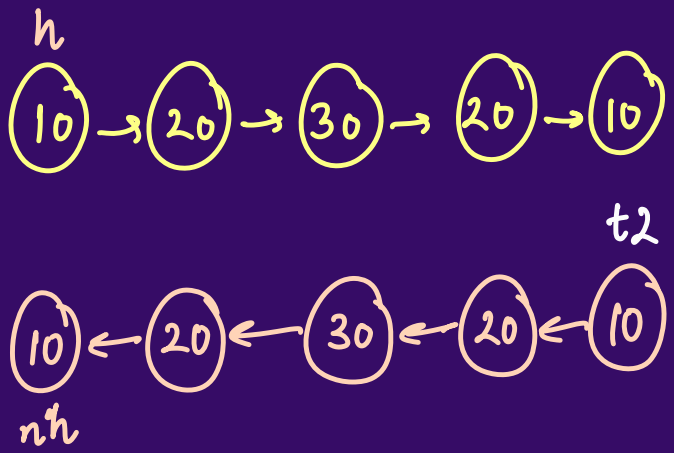
[Leetcode 234]

Ques:

Q14 : Palindrome Linked List

M-II

Create Deep Copy of given Linked List



T.C. = O(n)

S.C. = O(n)

[Leetcode 234]

Ques:

PW SKILLS

Q14 : Palindrome Linked List

M-III

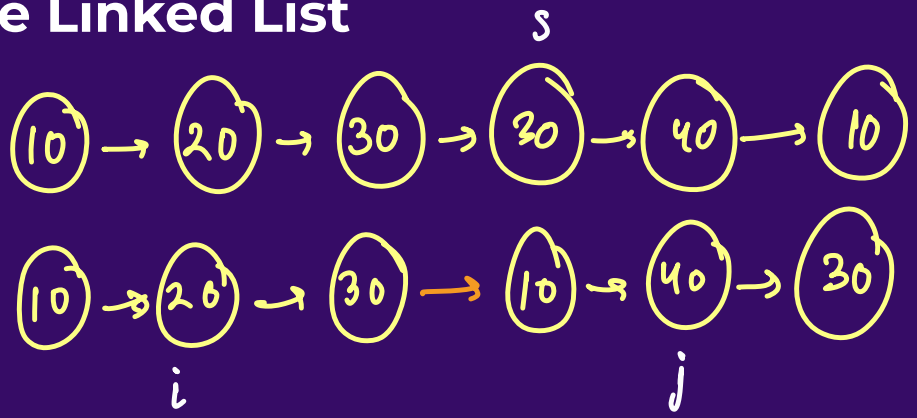
Reverse the second half of List

↓

Middle of LL

Reverse LL

2 pointer



[Leetcode 234]

Homework:

PW SKILLS

Q : Maximum Twin Sum of a Linked List

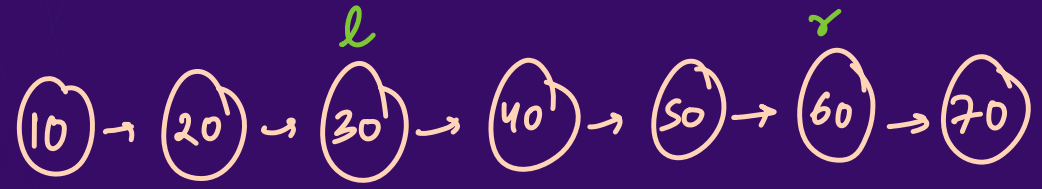
[Leetcode 2130]

Ques:

head, left, right

Q15 : Reverse Linked List II

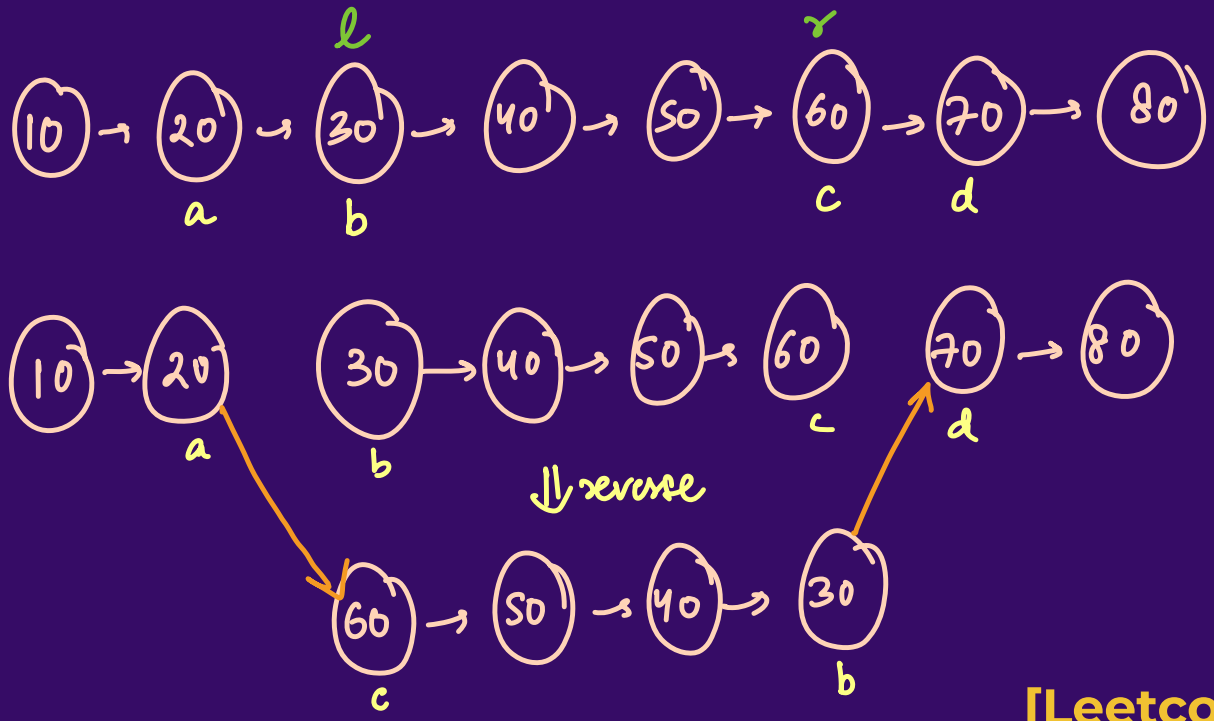
l = 3
r = 6



[Leetcode 92]

Ques:

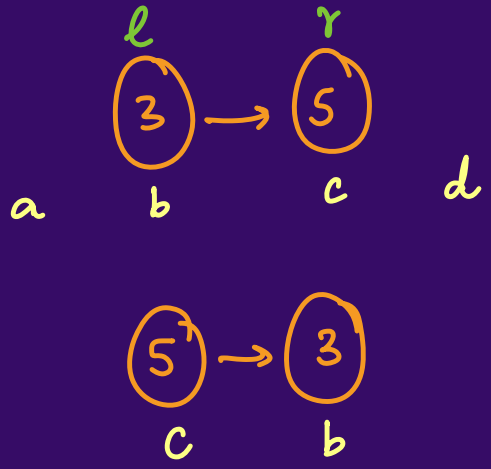
Q15 : Reverse Linked List II



[Leetcode 92]

Ques:

Q15 : Reverse Linked List II



return head → α

[Leetcode 92]

Ques:

Q15 : Reverse Linked List II

```
ListNode a = null, b = null, c = null, d = null;
int pos = 1;
ListNode temp = head;
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;
if(c!=null) c.next = null;
reverseList(b);
if(a!=null) a.next = c;
b.next = d;
if(a==null) return c;
return head;
```

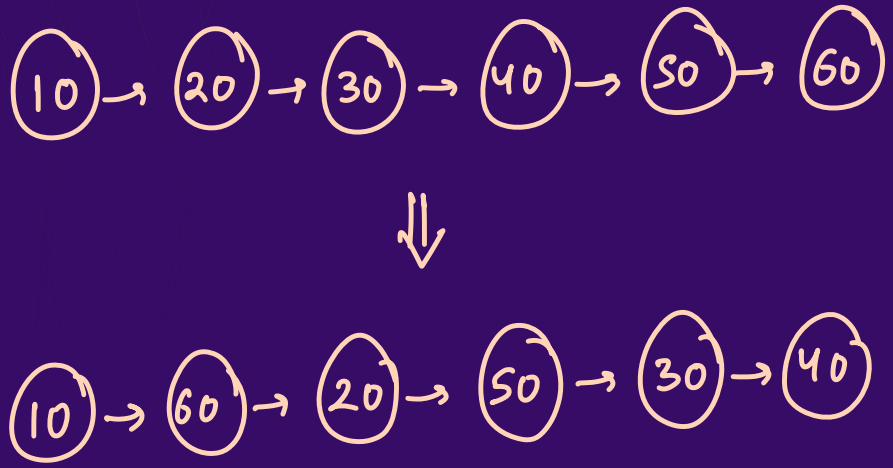
T.C. = O(n)

H.W. Do it in one pass

[Leetcode 92]

Ques:

Q16 : Reorder List



Hint

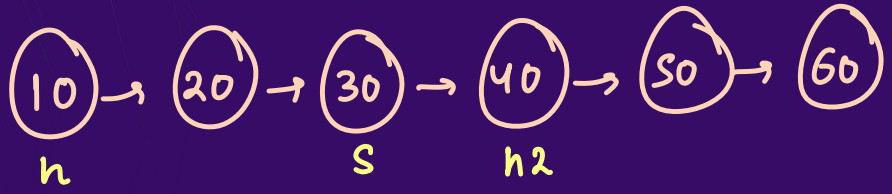
- Middle of LL
- Reverse LL
- Joining 2 LL

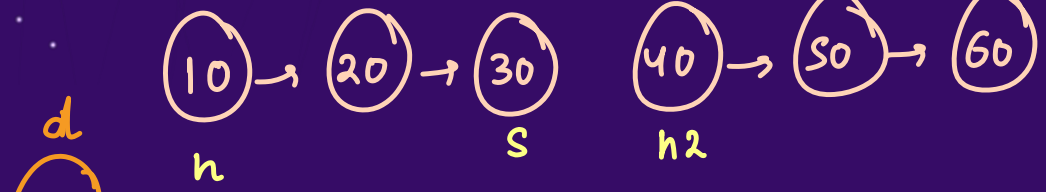
PW SKILLS

[Leetcode 143]

Ques:

Q16 : Reorder List

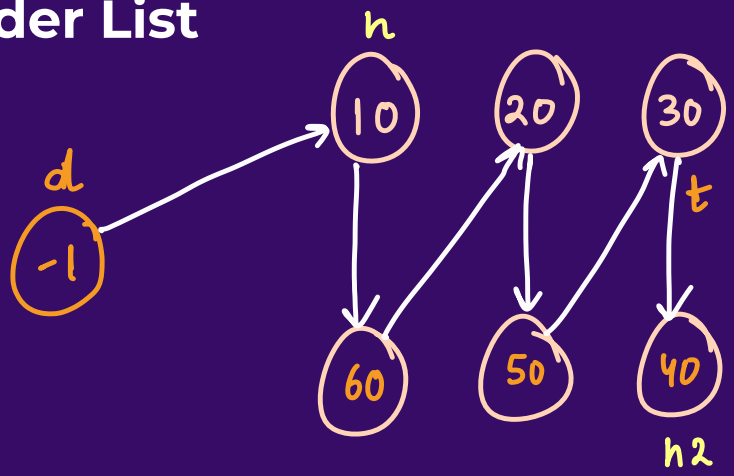




[Leetcode 143]

Ques:

Q16 : Reorder List



[Leetcode 143]

Ques:

Q17 : Add Two Numbers

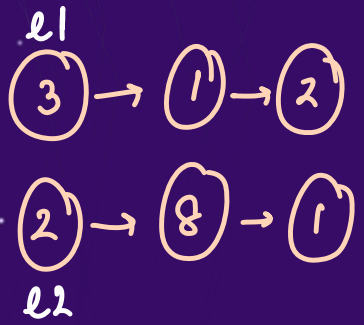
213 + 182 = 395

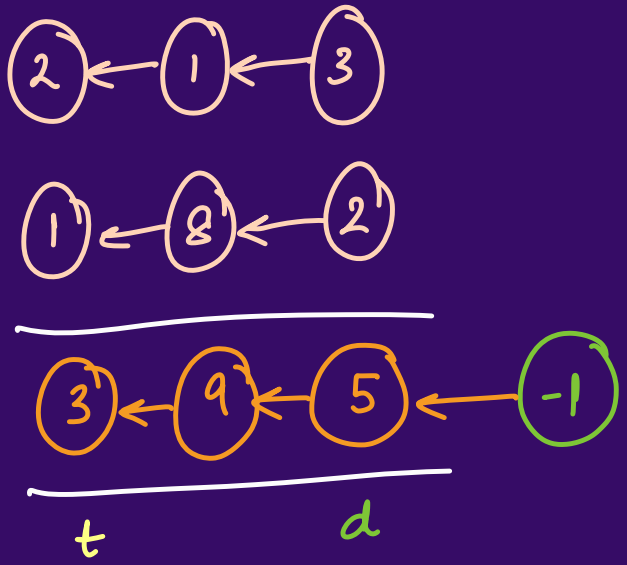

```
  2 1 3
  1 8 2
-------
  3 9 5
```

[Leetcode 2]

Ques:

Q17 : Add Two Numbers



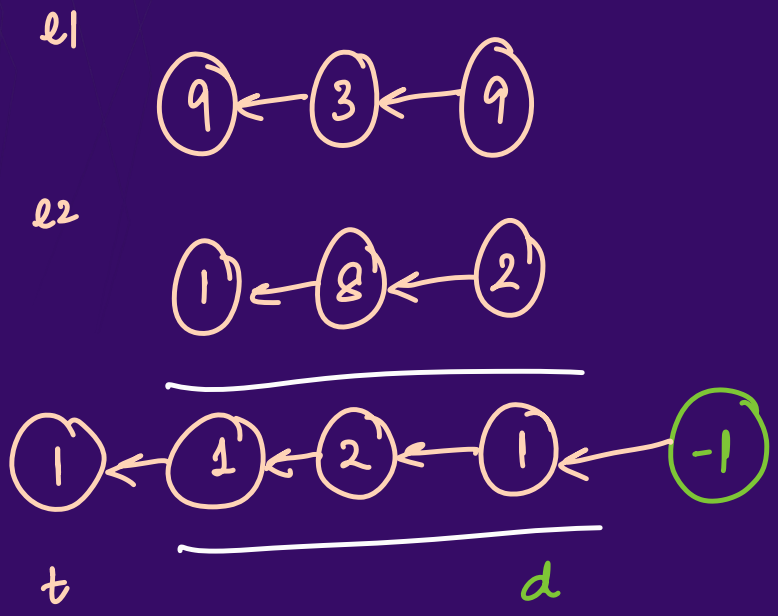


[Leetcode 2]

Ques:

Q17 : Add Two Numbers

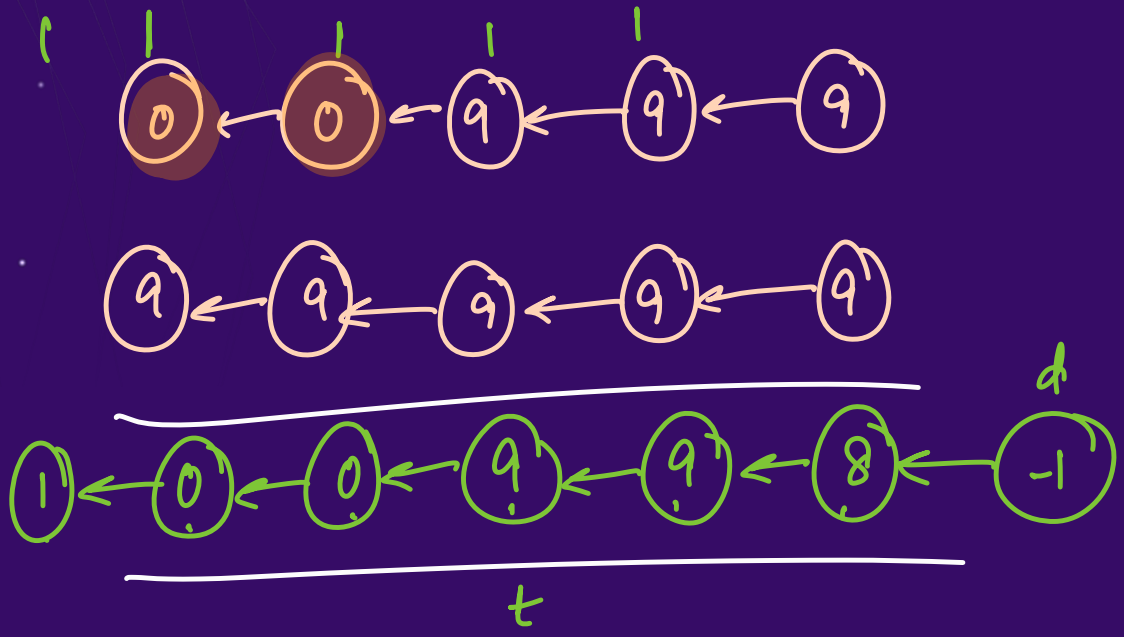
carry = 0 & & 1



[Leetcode 2]

Ques:

Q17 : Add Two Numbers



Carry = 0 1 1

00121
+ 43146

[Leetcode 2]

Ques:

Q17 : Add Two Numbers

```java
public 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==1){
        ListNode node = new ListNode(1);
        temp.next = node;
        temp = temp.next;
    }
    return dummy.next;
}
```

Homework:

PW SKILLS

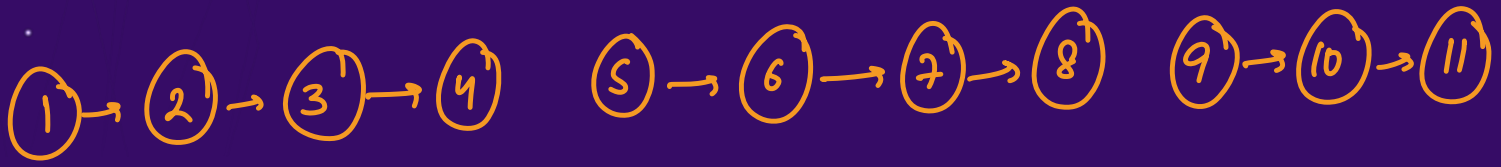
Q : Add Two Numbers II

[Leetcode 445]

Ques:

PW SKILLS

Q18 : Split Linked List in Parts



larger = $\frac{n}{k} + 1$

smaller = $\frac{n}{k}$

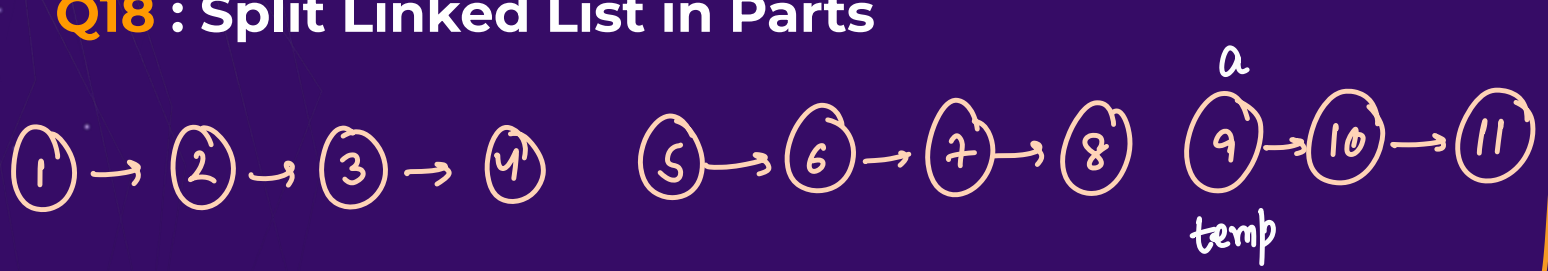
n = 11

no. of large parts = n%k = 2

[Leetcode 725]

Ques:

Q18 : Split Linked List in Parts



k = 3

size = 11/3 = 3

extra = 11 % 3

= 2 ≠ 0

len = 1 ✗

3 ✓

1 ✗ 1

```
s = size;
if(extra > 0) s++;
if(len == s) { work 3
else len++ ;
Node a = t.next
temp.next = null
temp = a
```

AL = { 1 , 5 , 9 }

[Leetcode 725]

```java
public ListNode[] splitListToParts(ListNode head, int k) {
    int n = lengthOfList(head);
    int size = n/k; // n/k + 1
    int extra = n%k;
    ListNode[] arr = new ListNode[k];
    int idx = 0;
    ListNode temp = head;
    int len = 1;
    while(temp!=null){
        int s = size;
        if(extra>0) s++;
        if(len==1) arr[idx++] = temp;
        if(len==s){
            ListNode a = temp.next;
            temp.next = null;
            temp = a;
            len = 1;
            extra--;
        }
        else{
            len++;
            temp = temp.next;
        }
    }
    return arr;
}
```

PW SKILLS

[Leetcode 725]

Homework:

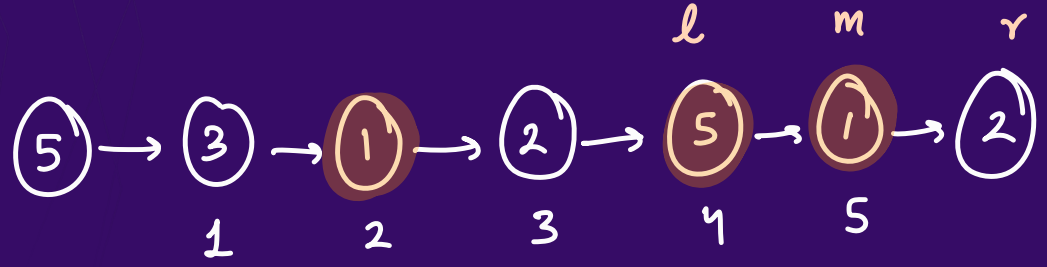
Q : Reverse Nodes in K-Groups

[Leetcode 25]

Ques: Critical point → local maxima
local minima

PW SKILLS

Q19 : Find the Minimum and Maximum Number of Nodes between Critical Points



int first = -1 2
int last = -1 2 4 5

dist = 5 - 4 = 1

ans = {-1, -1}

minD = 2 1

Consider only maxdistance

Now, min^m distance always comes b/w two consecutive critical nodes

[Leetcode 2058]

```java
public int[] nodesBetweenCriticalPoints(ListNode head) {
    ListNode left = head;
    ListNode mid = head.next;
    ListNode right = head.next.next;
    int first = -1, last = -1;
    int idx = 1;
    int[] arr = {-1,-1};
    int minDistance = Integer.MAX_VALUE;
    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 arr; // 0 or 1 critical point
    int maxDistance = last - first;
    arr[0] = minDistance;
    arr[1] = maxDistance;
    return arr;
}
```

SKILLS

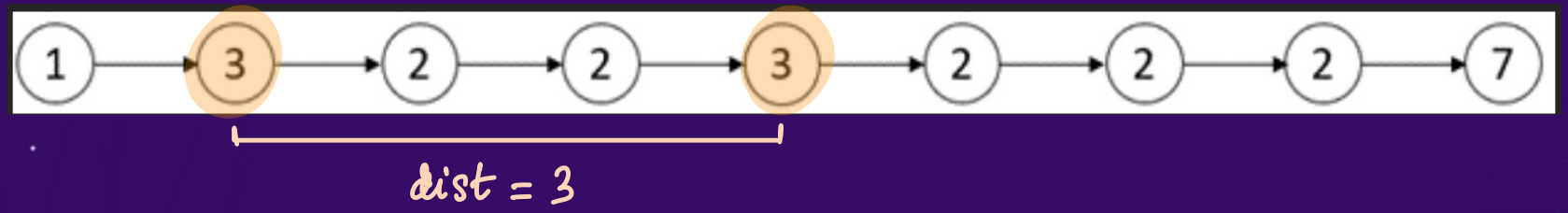
er of Nodes

[Leetcode 2058]

Ques:

PW SKILLS

Q19 : Find the Minimum and Maximum Number of Nodes between Critical Points



arr = [3, 3]

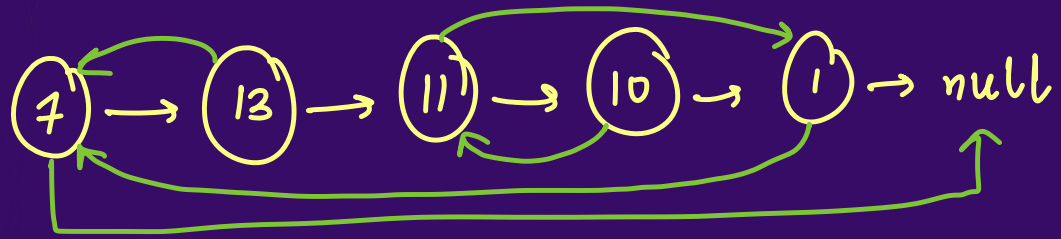
[Leetcode 2058]

Ques:

Q20 : Copy List with Random Pointer

Step - 1

Create deep Copy



[Leetcode 138]

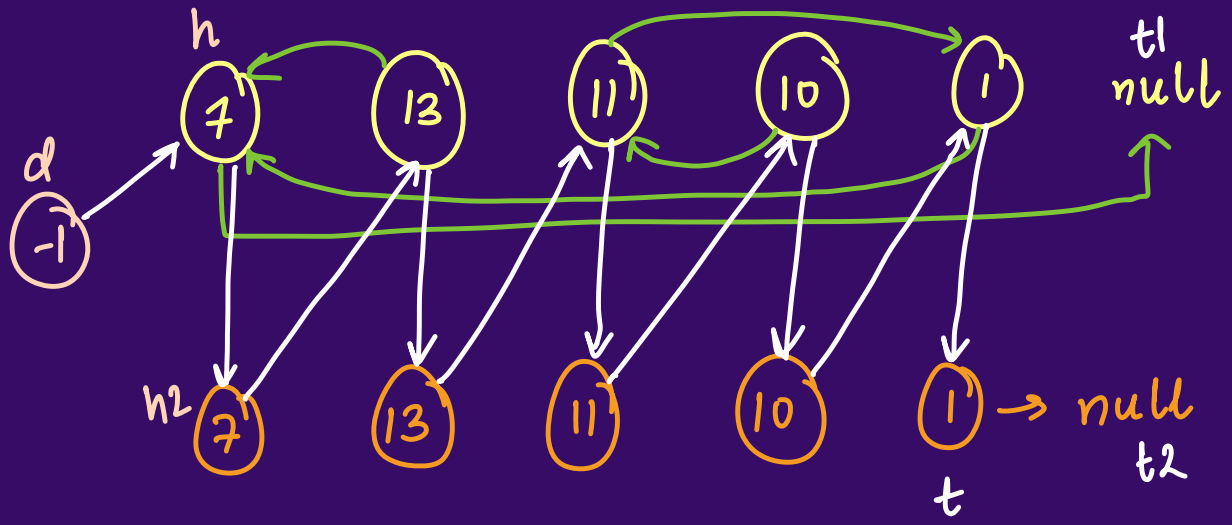
Ques:

Q20 : Copy List with Random Pointer

Step-2 :

↓

Join these linked list alternatively



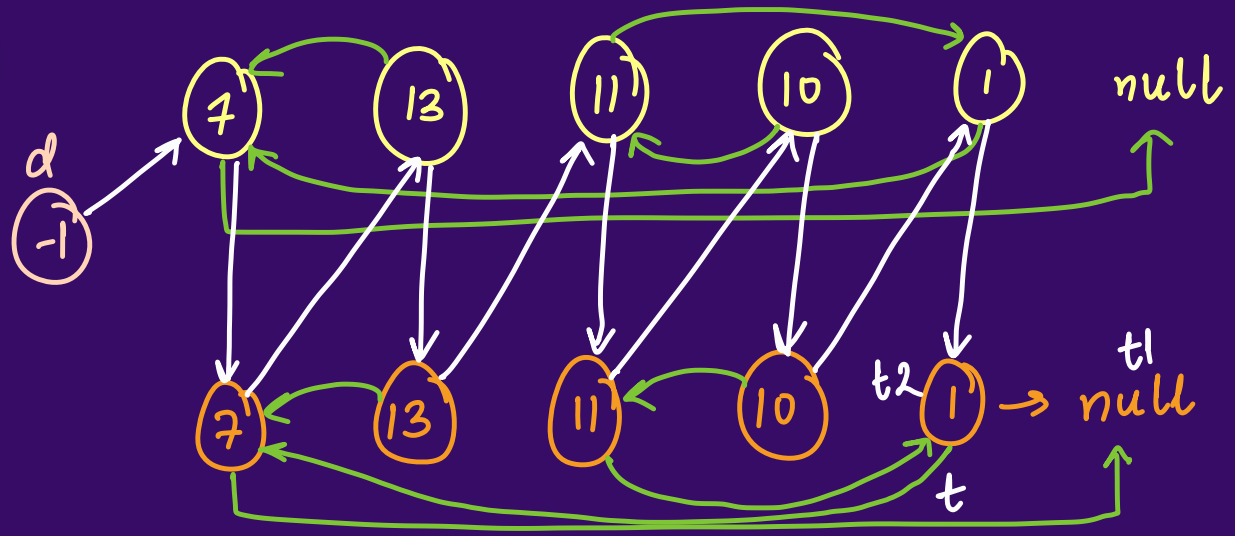
[Leetcode 138]

Ques:

Q20 : Copy List with Random Pointer

Step-3

Assign random pointers



t2.random = t1.random.next

[Leetcode 138]

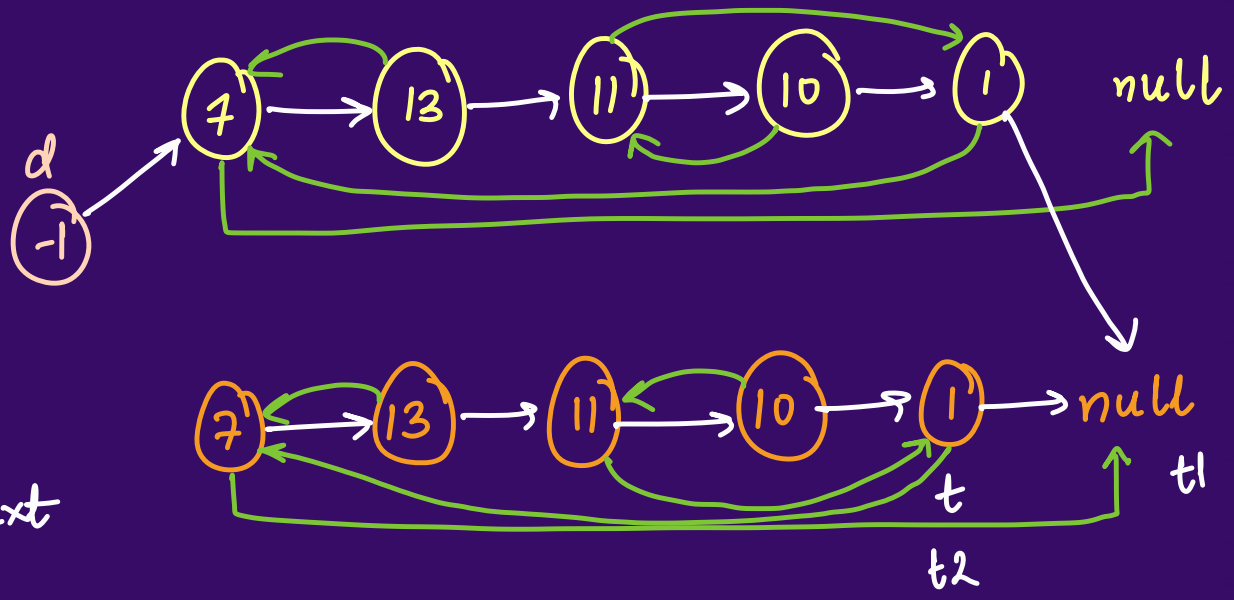
Ques:

PW SKILLS

Q20 : Copy List with Random Pointer

Step-4

Split the LL again



t1.next = t2.next

t1 = t1.next

t2.next = t1.next

t2 = t2.next

[Leetcode 138]

Ques:

Q20 : Copy List with Random Pointer

```java
public Node copyRandomList(Node head) {
    if(head==null) return null;
    // Step 1 : Create deep copy w/o random connections
    Node head2 = deepCopy(head);
    // Step 2 : Join head and head2 alternatively
    connectAlternatively(head,head2);
    // Step 3 : Assign random pointers
    assignRandom(head,head2);
    // Step 4 : Split the Linked List
    split(head,head2);
    return head2;
}
```

T.C. = O(n)

S.C. = O(n)

We are not using any extra space

[Leetcode 138]

```java
public Node deepCopy(Node head){
    Node head2 = new Node(head.val);
    Node t1 = head.next;
    Node t2 = head2;
    while(t1!=null){
        Node temp = new Node(t1.val);
        t2.next = temp;
        t2 = t2.next;
        t1 = t1.next;
    }
    return head2;
}
```

```java
public void connectAlternatively(Node head, Node head2){
    Node t1 = head;
    Node t2 = head2;
    Node dummy = new Node(-1);
    Node t = dummy;
    while(t1!=null && t2!=null){
        t.next = t1;
        t1 = t1.next;
        t = t.next;
        t.next = t2;
        t2 = t2.next;
        t = t.next;
    }
}
```

```java
public void assignRandom(Node head, Node head2){
    Node t1 = head;
    Node t2 = head2;
    while(t1!=null){
        t2 = t1.next;
        if(t1.random!=null) t2.random = t1.random.next;
        t1 = t1.next.next;
    }
}
```

```java
public void split(Node head, Node head2){
    Node t1 = head;
    Node t2 = head2;
    while(t1!=null){
        t1.next = t2.next;
        t1 = t1.next;
        if(t1==null) break;
        t2.next = t1.next;
        t2= t2.next;
    }
}
```

SKILLS

38]

Revisiting the limitations of "Singly Linked List"

PW SKILLS

get → O(n)

set → O(n)

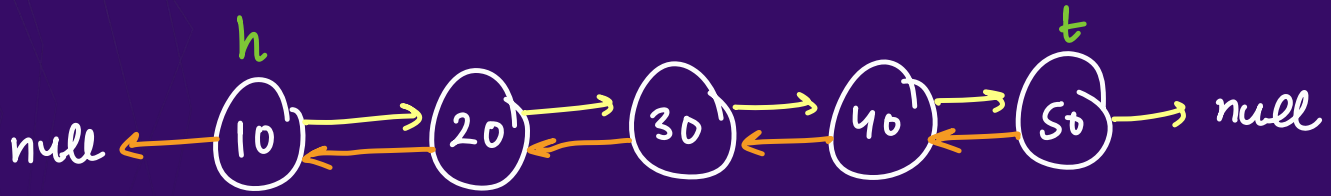
we cannot go back & we always need head to traverse entire list

delete tail → O(n)

Introducing Doubly Linked List

PW SKILLS

Can we make our insertion and deletion more efficient?

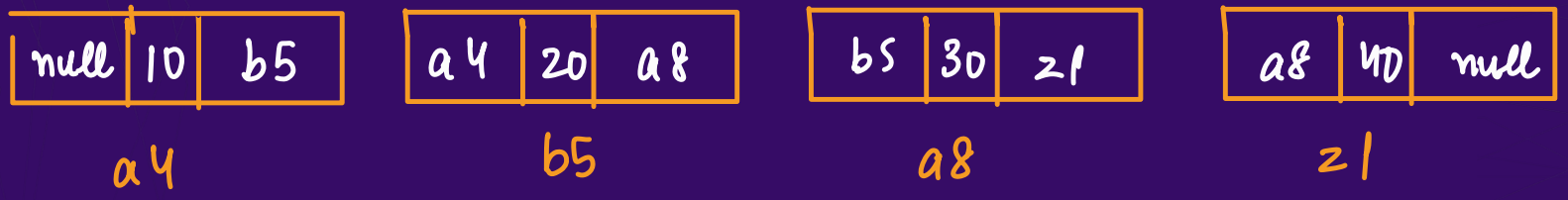



```
class Node{
    int val;
    Node next;
    Node prev;
}
```

Introducing Doubly Linked List

PW SKILLS

Node Class



Homework :

- Check if DLL is palindrome or not

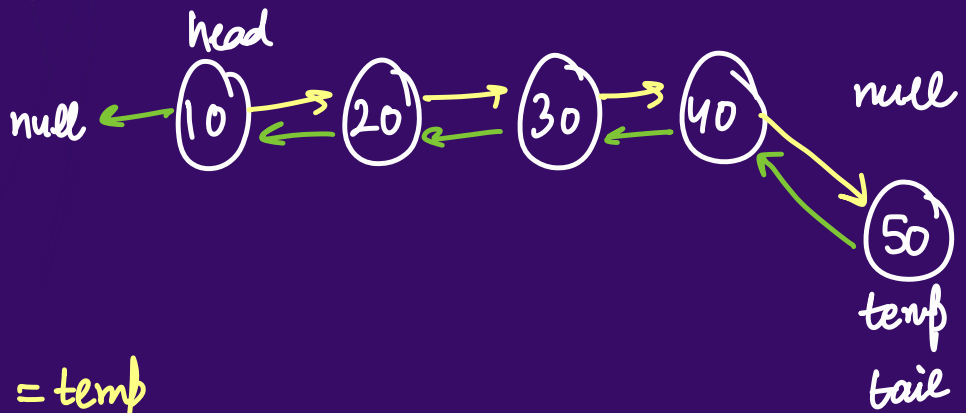
Note that we take up more memory

Implementation

PW SKILLS

Insertion and Deletion

insert At Tail



tail.next = temp
temp.prev = tail
tail = temp

Implementation

PW SKILLS

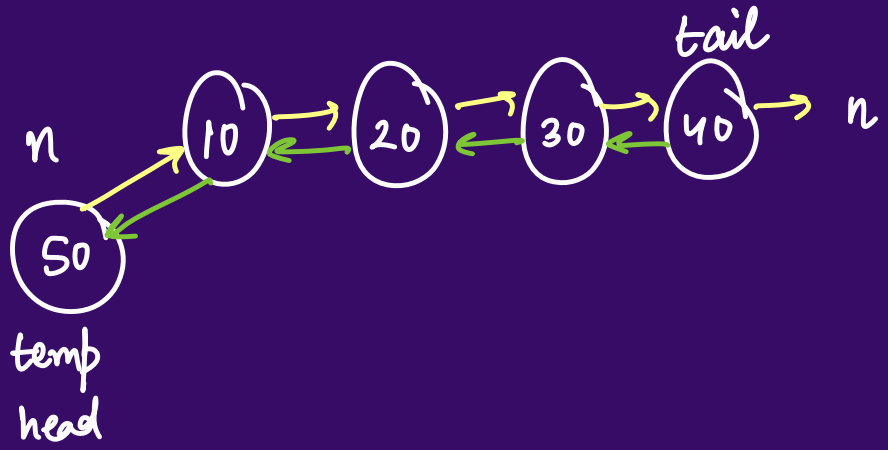
Insertion and Deletion

Insertion at Head

temp.next = head

head.prev = temp

head = temp



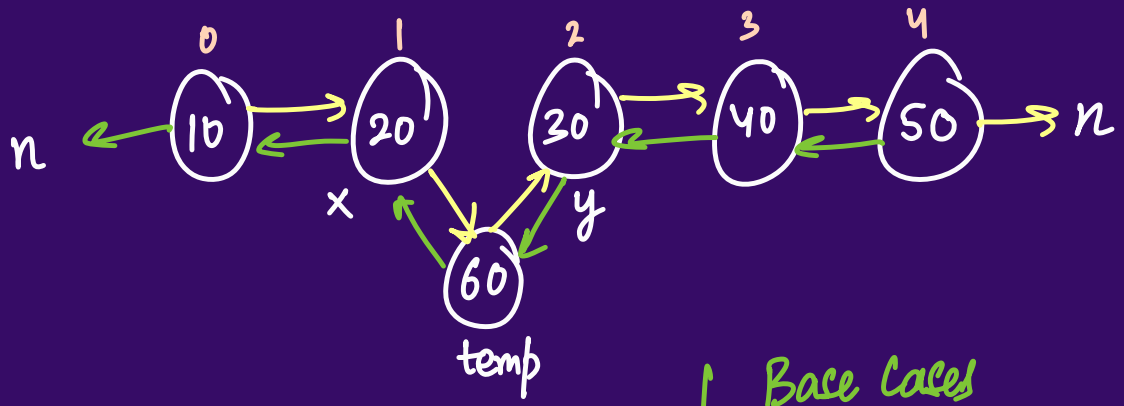
Implementation

PW SKILLS

Insertion and Deletion

1

At any index



insert(2, 60)

1) Place x & y at idx-1 & idx respectively

2) x.next = temp , temp.prev = x

3) y.prev = temp , temp.next = y

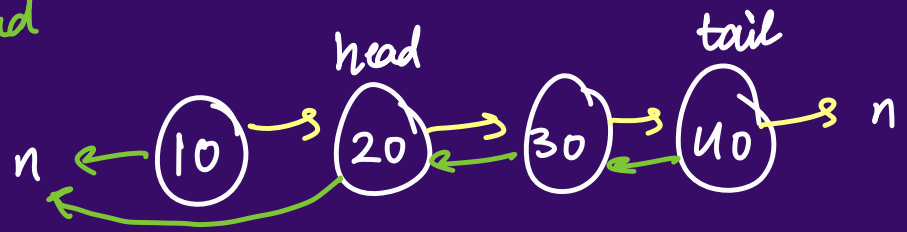
Base Cases

Implementation

PW SKILLS

Insertion and Deletion

1) Delete at Head



head = head.next

head.prev = null

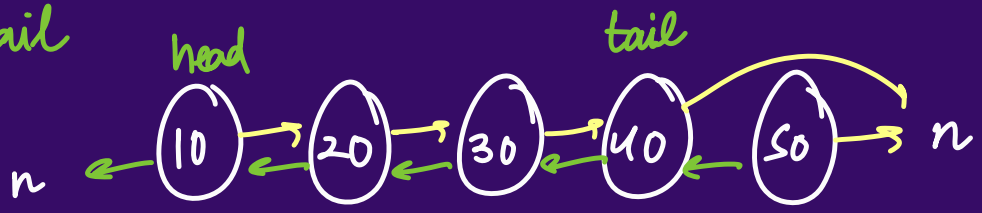
Implementation

PW SKILLS

Insertion and Deletion

x

2) Delete at Tail




```
tail = tail.prev
tail.next = null
```

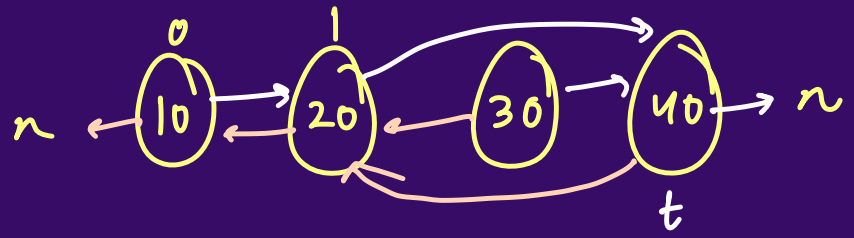
H.W. get → head, tail, idx

set → head, tail, idx

Implementation

PW SKILLS

Insertion and Deletion



t.next = t.next.next

t = t.next

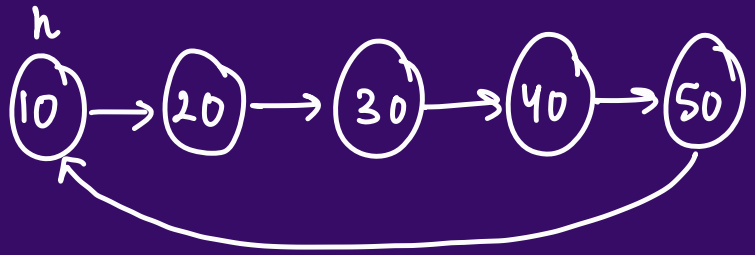
t.prev = t.prev.prev

Other types of Linked List

PW SKILLS

Circular Linked List :

1) Singly CLL :

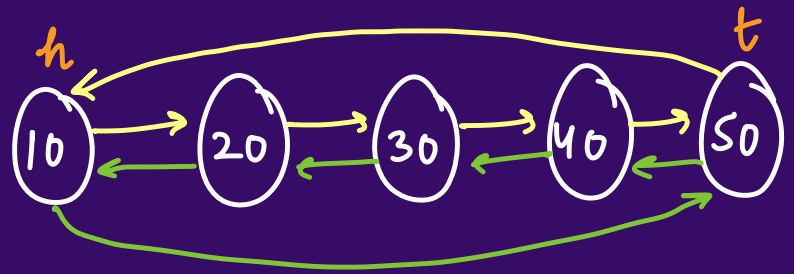


Ques : Given head of Singly Circular LL, print the list

Ques : Given head of SCLL, insert at Head, tail, idx
delete at Head, tail, idx

Other types of Linked List

2) Doubly CLL :

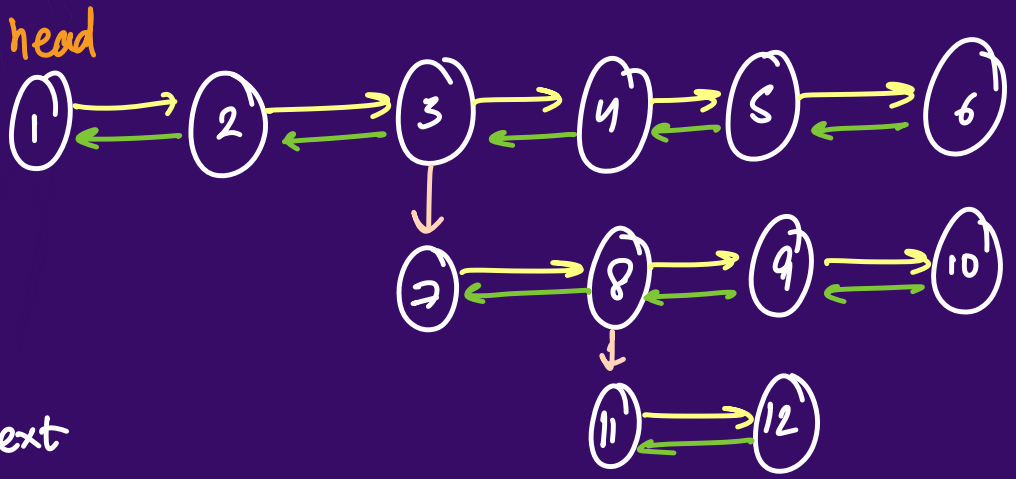


H.W: make DCLL class with following methods

1) display
2) insert → idx, head, tail
3) delete → idx, head, tail
4) get & set → idx

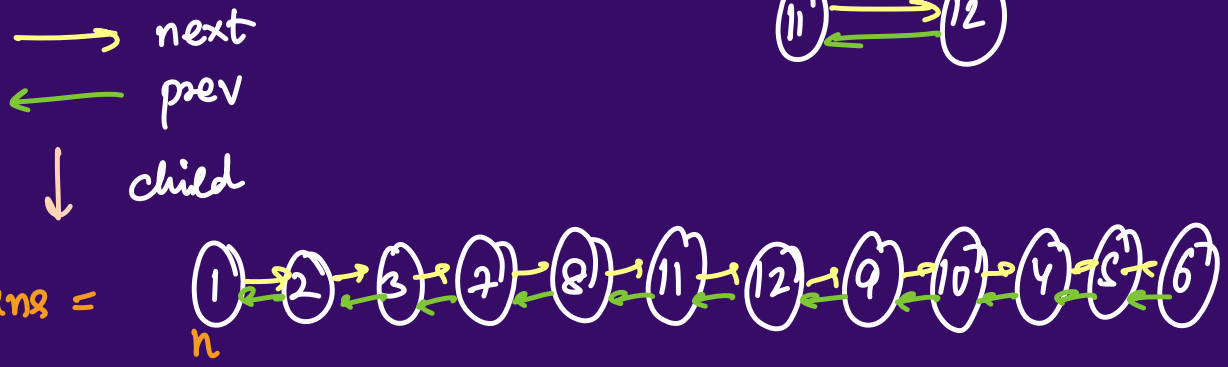
Ques:

Q21 : Flatten a Multilevel Doubly Linked List




```
class Node{
    int val;
    Node next;
    Node prev;
    Node child;
}
```

3

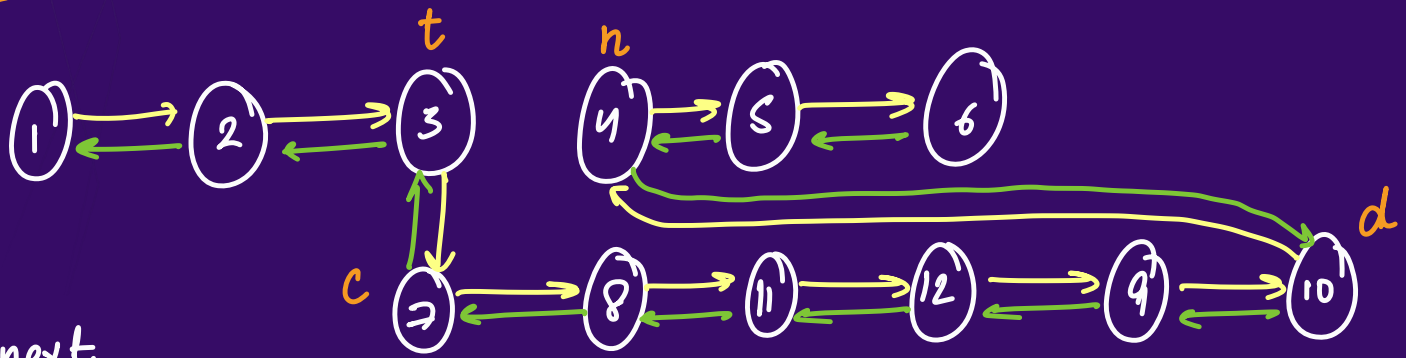


[Leetcode - 430]

Ques:

Q21 : Flatten a Multilevel Doubly Linked List

Intuition → Recursion



n = t.next

c = t.child

c = flatten(c)

Now place 'd' at flattened 'c''s tail

t & c d & n

[Leetcode - 430]

THANK YOU