



Linked List - Short Notes

only when require

✓ In Linked List memory is dynamically allocated so no extra space/wastage unlike in arrays/arraylist it is fixed & to add more elements in arraylist it doubles itself copy rest element then new added.

✓ In LL, they don't have any indices (they are randomly stored in Heap) & are pointed via reference variables.
L ↗ ✓ We can't get individual here as in arraylist (list.get(5));

* Single Linked List :-

- ✓ every single ^(node) item knows about next only.
- ✓ Head is a ref. variable pointing to first node.
- ✓ Tail " " " " " " " " last node & next to it is 'null'.

• Basic declaration :-

```
public class LL {
    private class Node {
        Node next;
        int value;
        // constructor
        Node both;
    }
    private Node head;
    Node tail;
    int size;
}

public LL() {
    this.size = 0;
}
```

✓ Insert First :-

- create Node with value
- determine "head" → $\text{node} \cdot \text{next} = \text{head}$ (or, determining node position + putting)
- assign head to node → ~~node = head~~ $\text{head} = \text{node}$
- null ptr. exception → if (~~tail~~ == null) {
 tail == null tail = head
 }
→ Don't forget to increase size → size++;

✓ Insert Last :- $(\text{tail} == \text{null})$

- "if list is empty (null ptr. exception) you can delegate this work to already made insertFirst();"
- else → create node of value
- $\text{tail} \cdot \text{next} = \text{node}$ → determine node position + put it
- $\checkmark \text{tail} = \text{node}$ → assign it as tail
- size++ → don't forget this.

✓ Insert at index :

- here we need node's value as well as index (const? of both used)
- if index == 0 → insert first
- if index == size → ~ last
- Create a temp node & move it till the index position to perform operation on that place ($O(n)$).
- ∴ Node temp = head; → create
- for (int i=1; i<index; i++) { → till index
 ↑ before index
 ↑ O is head only
 (here, temp at start)
 temp = temp.next; → moving
- Create node of value & next (temp.next) → place of index
 temp.next = node; → inserted
- size++; → dont forget

node->next = temp->next->next

head->next = 0

node->next = null

Date			
Page No.			

Ques
Ans
Date
Page No.

Delete first:

→ assign head->next to head (only step required)

→ size--;

write at start
cos head #
head pos will
change

int value = head->value (for returning purposes, no use)

head = head->next

if (head == null) { → null ptr exception (null → no ans.) }
tail = null;
size--;

return value;

kyo delete karoge agar
jaise :
assign it
to tail.

Delete last:

→ if size <= 1 → null items ∴ return;

→ else → get the second last node by iterating temp node from head
to particular position's node (i.e. for second last → size-2)

→ int value = tail->value (returning purposes)

→ assign secondlast to tail

→ & next to tail as null ∴ removed

→ size-- (Dont forget)

Better Delete at index:

→ if index at zero ∴ delete first

→ " " " size-1 (last) ∴ last

→ Make a prev. node at index-1 position
get

→ assign next to next node as next node (bich aala apne aap vdd gya)

i.e. prev->next = prev->next->next;
size--;

Display:

```
for (Node temp = head; temp != null; temp++) {
```

```
    cout (temp->value + " → ");
```

```
    temp = temp->next;
```

```
cout ("end");
```

* Doubly linked list

- same this just one more ref. variable added i.e. previous
- prev. of head \rightarrow null
- " " tail \rightarrow "
- Basic Declaration:- (no need to declare tail in DLL)

public class DLL {

private class Node {

Node next;

Node prev;

int value;

const of value

y " " all three

private Node head;

Insert First:-

- Create Node with value
- determining node position using $head \rightarrow node.next = head$
- assign head to node
- " " " " $\rightarrow node.prev = null$;
- if ($head != null$) { → * must for this
, ~~node~~ head.prev = node; } ←
- assign ~~head~~ to ~~node~~ at last \rightarrow ~~node = head~~
mode head head = node;

Insert last :-

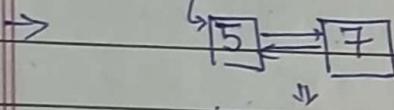
- Create node of value
- " last " from head $\rightarrow Node last = head$;
 $node.next = null$; \rightarrow " we're inserting this at last"
- null ptr exception \rightarrow if ($head == null$)
head = node;
y $node.prev = null$;

→ move 'last' till end → while ($\text{last} \cdot \text{next} \neq \text{null}$)
 }
 last = last · next;
 "updated finally"
 last value \leftarrow last · next \leftarrow node;
 assigned to last
 mode · prev = last;
 'last' aur bhi thodi jaoge
 walo bhi define karne using node.

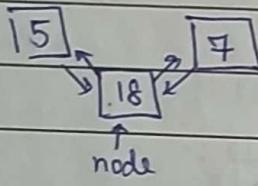
Insert at Index:

- method of after (int), value.
- Get node p after which we're to insert.
- If $p = \text{null}$ ∴ not exist ∴ return
- Create node of value

Note:- p is that node just before 'index')



① $18 \rightarrow 7 \therefore \text{node} \cdot \text{next} = p \cdot \text{next};$



② $5 \rightarrow 18 \therefore p \cdot \text{next} = \text{node};$

③ $18 \rightarrow 5 \therefore \text{node} \cdot \text{prev} = p;$

④ $7 \rightarrow 18 \therefore \text{node} \cdot \text{next} \cdot \text{prev} = \text{node}; \rightarrow \text{null ptr exception}$
 if ($\text{node} \cdot \text{next} \neq \text{null}$)

For Deletion in DLL (think on your own)

Display :-

- ~~create~~ declare ^{temp} node at head
- Declare last node as null
- while temp node not reaches null
 cout (~~temp~~ · mode · value + " → ");
 last = node; → ~~llj~~ last \neq last tak isi loop \rightarrow mardad krti pachha
 \downarrow
 node = node · next;
 cout ("end");
- cout ("In reverse")
 while (last != null) {
 cout (last · value + " ← ");
 if (last = last · prev)
 cout ("start");

* Circular linked list :-

- same as single LL + here no thing as "null".
- No big concept of index(specific) because we can choose head + tail.
- Basic declaration :-

```
public class CLL {
```

```
private class Node {
```

```
Node next;
```

```
int value;
```

```
} contr of value .
```

```
private Node head;
```

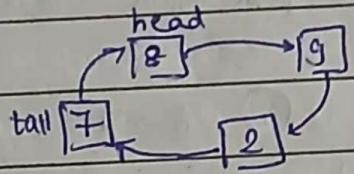
```
" " tail;
```

non-parameterized
contr.
of CLL

```
→ public CLL () {
```

```
this.head = head;
```

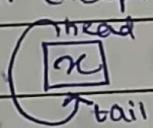
```
".tail = tail;
```



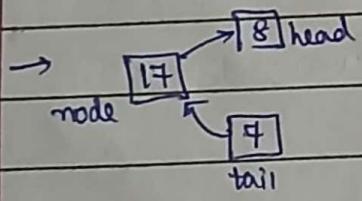
* Insert

→ Create node of value

→ null ptr exception → if (head == null) {



```
head = node;  
tail = node;
```



```
tail.next = node;
```

```
node.next = head;
```

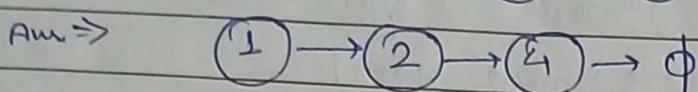
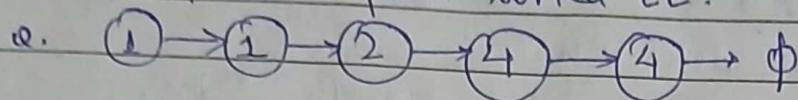
~~node = tail~~; tail = node;

~~tail = node~~
assign node as tail

- Delete, Display ⇒ concept of do while loop check video.

* Linked list imp. Questions → Imp. FAANG level Questions

1) Remove Duplicates from sorted LL.



Steps → Create a temp "node" to traverse along LL

Starting from head

→ till it reaches null, if duplicate, next^{value} will be node.next.next^{value}, else next continues.

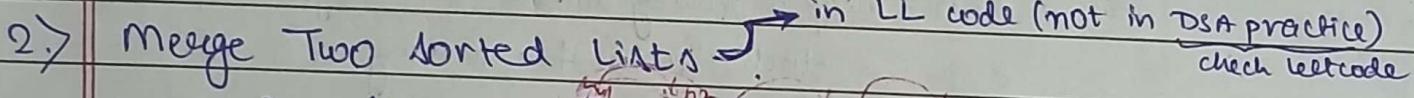
→ Don't forget to decrease size (at last only).

if tail { → assign that node as tail + complete by mentioning tail.next = \emptyset .

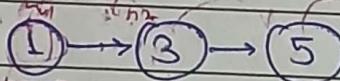
mentioned → otherwise return head only.

→ give null ptr exception at start.

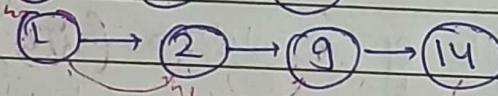
Note imp:- Check what you're traversing & what is given.

2) Merge Two Sorted Lists 

Q. List 1:

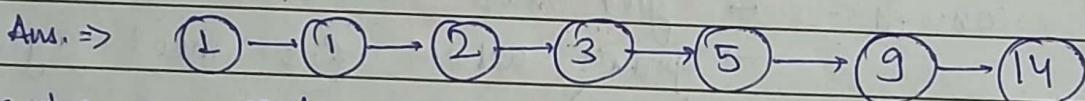


List 2:



in LL code (not in DSA practice)
check leetcode

{ already sorted



concept → → make a final ans list

→ Get head of both list, while both doesn't reach till null compare heads and "insert last" the smaller one & move forward in ans list

→ If any one reaches null^{list}, give while loop till other one reaches null & prints its remaining nodes

→ return ans.

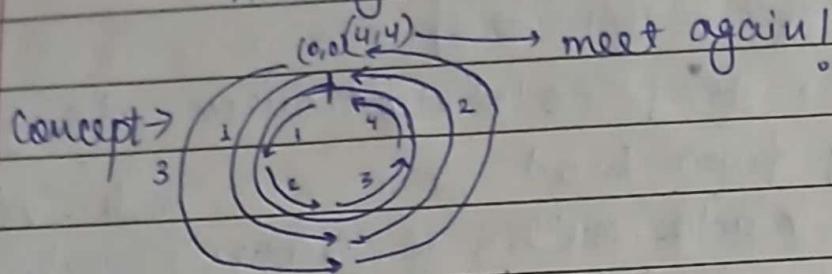
Linked

3) Linked list cycle

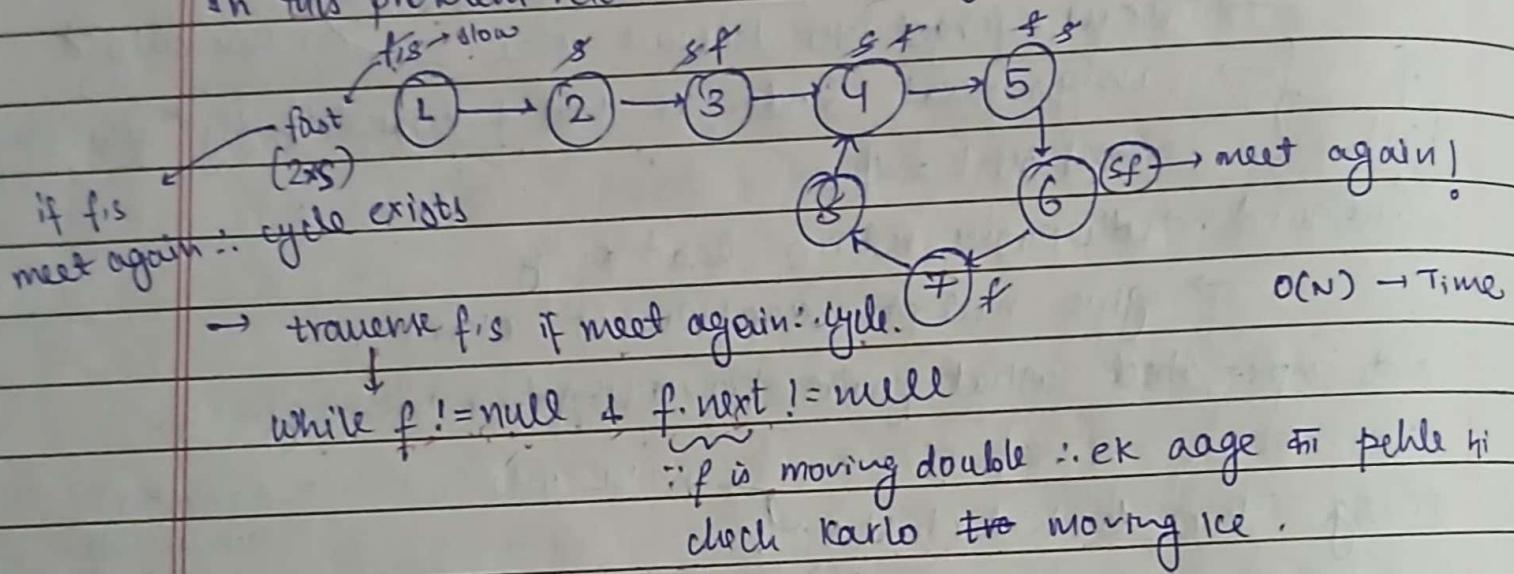
★ ★ Fast and slow pointer approach

→ Cycle detection

→ Finding a node in cycle, etc.



In this problem return true if it has cycle else false.



4). Find length of a loop (GFG)

concept:- → cycle must be there (write code for this)
 → point where f,s meet again, make (node)

a temp node as slow alongwith length (counter) as zero (at start)

→ traverse temp node with condition it does not again equals to slow using 'do while loop'

do {

 temp = temp.next

 length++;

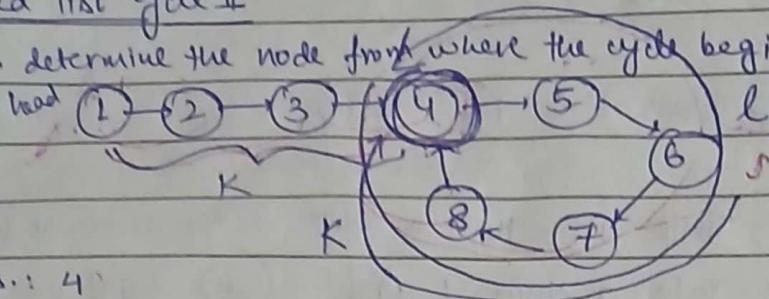
 while (temp != slow), Teacher's Signature.....

 return length

→ else return 0,

5) Linked list cycle II

a. determine the node from where the cycle begins.



Ans.: 4

while($\text{length} > 0$)
 $s = s.\text{next}$
 $\text{length} =$

Concepts :- \Rightarrow find the length of cycle = ?

$\star \rightarrow$ move s ahead by length of cycles times

\rightarrow Now keep ^{moving} f one by one from start & similarly
move s one by one from that particular position

- while $f \neq s$

\rightarrow return s

$\star \rightarrow$ check code

6) Happy Number Google 2017

Q. e.g.-1 $n=19$

$$1^2 + 9^2 = 1 + 81 = 82$$

$$8^2 + 2^2 = 64 + 4 = 68$$

$$6^2 + 8^2 = 36 + 64 = 100$$

$$1^2 + 0^2 + 0^2 = 1$$

Ans. true

$$\text{eg. } n=12 \rightarrow 1^2 + 2^2 = 1 + 4 = 5$$

$$5^2 = 25$$

$$2^2 + 5^2 = 4 + 25 = 29$$

$$2^2 + 9^2 = 4 + 81 = 85$$

$$8^2 + 5^2 = 64 + 25 = 89$$

$$5^2 + 8^2 = 25 + 64 = 89$$

$$8^2 + 9^2 = 64 + 81 = 145$$

$$1^2 + 4^2 + 5^2 = 1 + 16 + 25 = 42$$

$$4^2 + 2^2 = 16 + 4 = 20$$

$$2^2 + 0^2 = 4$$

$$4^2 = 16$$

$$1^2 + 6^2 = 37$$

$$3^2 + 7^2 = 58$$

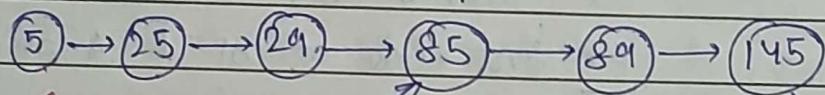
whary

repeated
 \therefore do terms

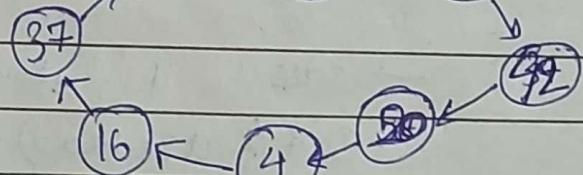
Ans. false

Concepts :-

(n=12)



v. whary



Use the concept of "hascycle"

that's it \rightarrow only diff. is that here LL is not mentioned

$\therefore \Rightarrow$ initially slow + fast = n [both]

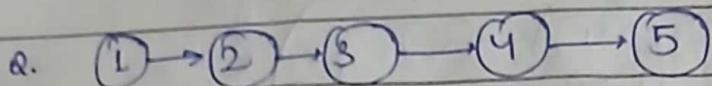
\rightarrow while slow != fast do slow \rightarrow find sq., fast \rightarrow find sq's sq.

\rightarrow if slow = 1 \rightarrow true, else false

\rightarrow For find sq. get individual value sq. it & return the stored value.

Teacher's Signature.....

7) Middle of the linked list



Ans. 3

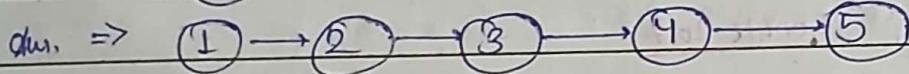
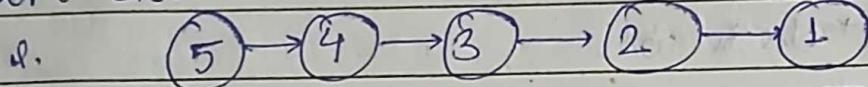
Note:- if there are two middle nodes \therefore return second middle

concept:- \rightarrow Take a slow & fast($2 \times \text{slow}$) pointer.

\rightarrow Traverse both from start till fast reaches null
or ($\text{fast} \cdot \text{next}$) \rightarrow to save time

\rightarrow The moment at which fast reaches null slow ptr
will be at $(\frac{1}{2}) \therefore$ return slow.

8) Sort List.



30

concept \Rightarrow \rightarrow use Bubble sort algo. approach.

\rightarrow Firstly make bubble sort method with no parameters &
one with two parameters row & col (obv. 0).

\rightarrow Par/initialize row as size-1 & col as 0 \rightarrow non-by
calling parameterized fn. from non-par. fn.

\rightarrow null ptr. exception ~~if size=1~~ i.e. if row == 0 \rightarrow return.

\rightarrow For condition col < row (till row reaches till end)
get first node at col & second at col+1.

\rightarrow case:1 if first value > second \therefore swap :-

① if (first == head) \rightarrow ① head = second, ~~second.next = first~~,

agar kisko pehle link dega
to first ka value garbage
main chala jayega

② first.next = second.next

③ second.next = first.

② else if (second == tail) \rightarrow ① make a prev node by getting at col-1

② prev.next = second

③ tail = first

④ first.next = null;

⑤ second.next = tail;

Teacher's Signature.....

Not first coz you're at the
end of the statement first tail been chulcha hai.

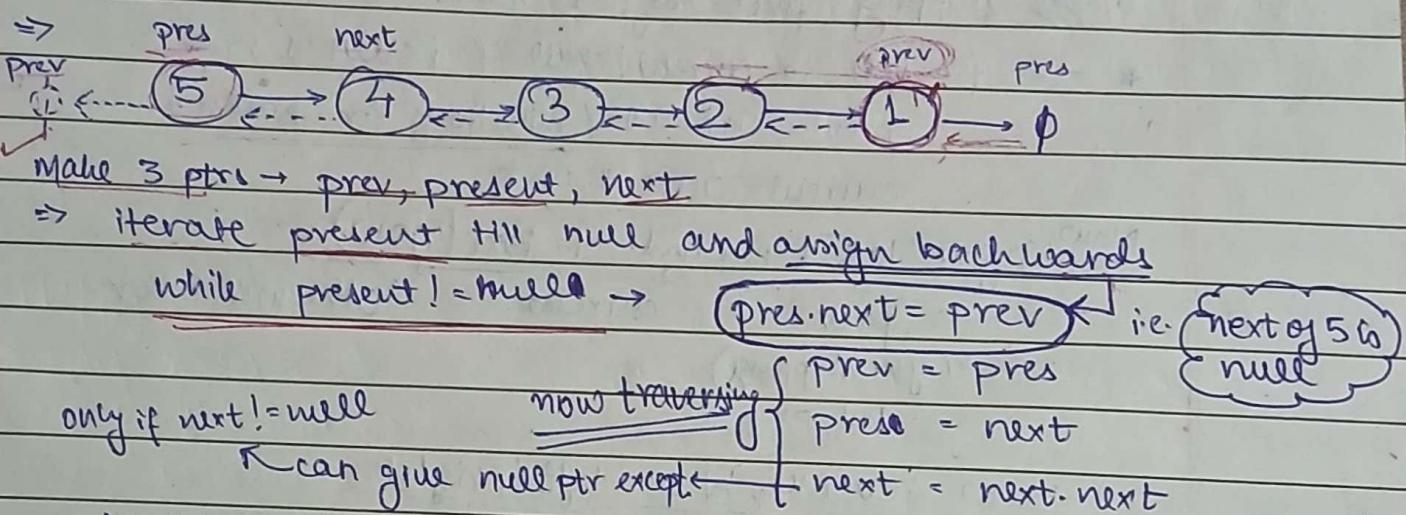
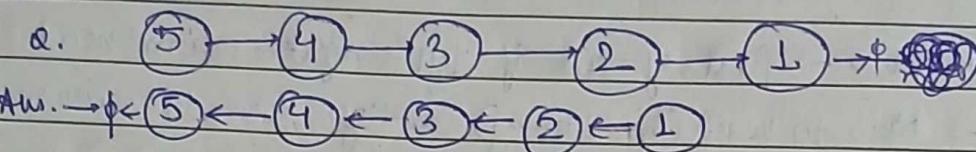
② elm → ①, ② or ② → elm if same

→ ③ first.next = second ④ second.next = first.

→ call bubblesort (row, col+1) → moving block to next (bubble sort concept)

→ else " " (row-1, col-1)

Reverse Linked List



in the end prev at 1 & pres at ϕ

$\therefore \text{head} = \text{prev}$

\Rightarrow base condition if size < 2 : return head

Recursive approach :

1. temp node pointing at head 2. if node is at tail $\therefore \text{head} = \text{tail}$, return

3. pass node.next to argument fn. (node is one step ahead now) \rightarrow till tail

4. tail.next = node \therefore while returning (returning values from stack) \therefore need to set tail to tail position

tail = node

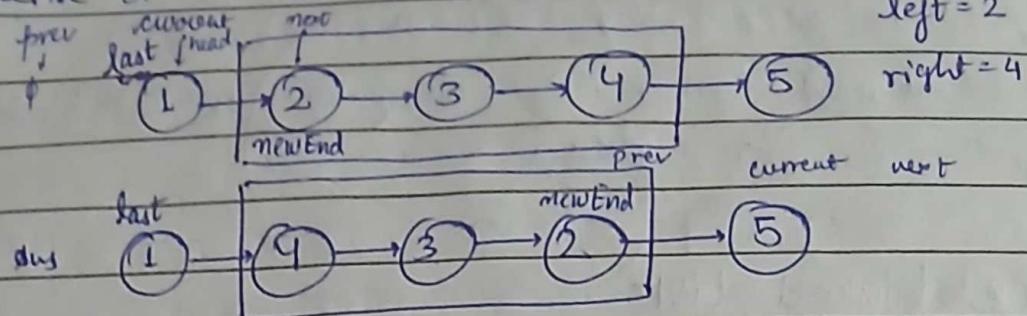
assign it to tail (stack)

tail.next = ϕ

this is happening when you're coming out of rec. call. \therefore reverse tail till end.

18. Reverse a Linked List II

last is
reverse home wale
jaga se just pehle
wala



concept \Rightarrow first of all if $left = right \rightarrow$ return head.

\rightarrow Now to reach (by skipping) at left then do reversing.

\therefore for skipping first $left - 1$ nodes

$\star \rightarrow$ for $i=0$ to $current != null$ if $i < left - 1$, i.e.

$\therefore prev = current$ (initialising for reverse)

$current = current.next$

Now, Define last + newEnd + next (now)

\downarrow \downarrow \downarrow
prev current next

(after skipping)

last = prev

Otherwise error

\rightarrow Now reverse b/w left + right

condition $\rightarrow current != null \& i < right - left + 1$

\rightarrow Note:- if last != null (must) | else, head = prev

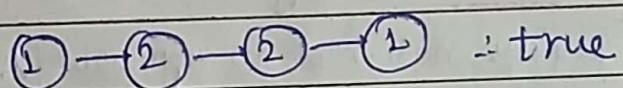
$\therefore last.next = prev$

$\rightarrow newEnd.next = current$

return head;

19.

Palindrome linked list.



concept \Rightarrow get middle node

\rightarrow reverse the list from mid + assign it as second head

\rightarrow while $head != \phi$ & $head.next != null$

if ($head.value == " "$)
 \quad break;

traverse head \rightarrow head.next
 \quad & head = head.next

\rightarrow now reverse that list from middle node

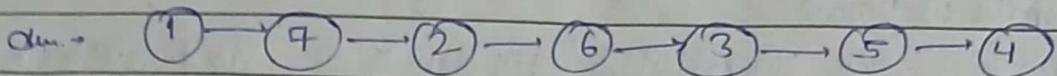
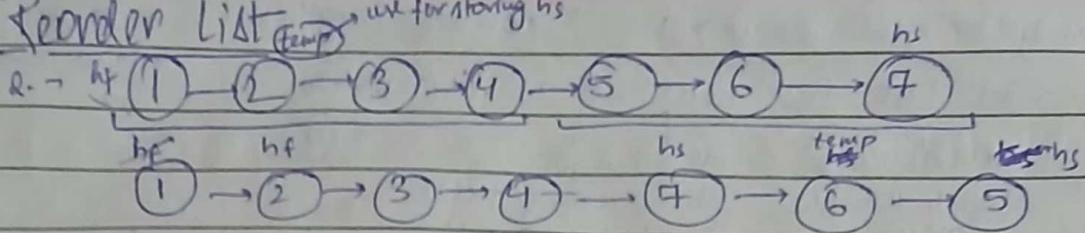
Hence, if both heads reached till null

either: true

else false.

Teacher's Signature.....

11.

Reorder List

concept: → if $head = \text{null}$ or $head.next = \text{null}$ (only 1 node) \therefore return

→ get mid node

→ hs (second head) \rightarrow reversed list from mid's head

→ hf (first head)

→ Now rearrange: till hf & hs != null

→ create a temp node next of hf then \leftarrow

→ put $hf.next = hs$ $\rightarrow hf = \text{temp}$ $\rightarrow \text{temp} = hs.next \rightarrow hs.next = hf$

rearranged 1

$\rightarrow hs = \text{temp}$

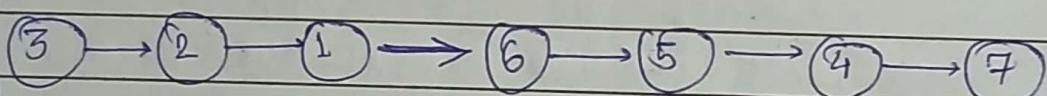
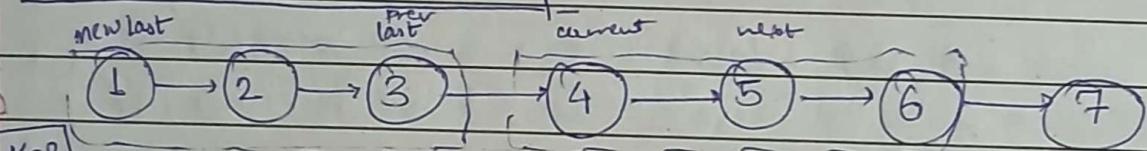
→ put tail at last as if ($hf \neq \text{null}$)

ie. hs reached
null first

rearranged 2

$hf.next = \text{null};$

13.

Reverse Nodes in K-Group

Concept: → first if all $K = 1$ or $head = \text{null}$ then return head

→ ~~last~~ get current + prev pointers at head + null \leftarrow

→ get length of list $\rightarrow count = \text{length}/K \rightarrow$ soch K must be $>$ length

→ till $count > 0$ [→ last = prev, newEnd = current, define 3rd pr next = current.next]

for loop till $i < K$ if $current \neq \text{null} \rightarrow$ assigning backwards $current.next = prev$

② Now continue traversing: $prev = current$, $current = next$, ($next = next.next$), nullptr exception

→ if $last \neq \text{null} \rightarrow last.next = prev$, else $head = prev$

$newEnd.next = current$, $prev = newEnd$

$count --$]

return head

Date			
Page No.			

Concept → till count > 0 $\rightarrow \text{count} = \text{length}/K \because K \text{ jo group } \& \text{ wo}$
 \rightarrow point last and new End → reverse (rev.LL2)
 atleast length ke jyada ho

reverse node

last jo

Q.10

block ke pehla
(prev)

obv. reverse se pehle

also make next node

first (current) pe hoga

→ Do reverse Linked List 2 till $i < K$ and decrease count

→ return head.

(14,15) → rev alt. K nodes & rotate list } → practice yourself