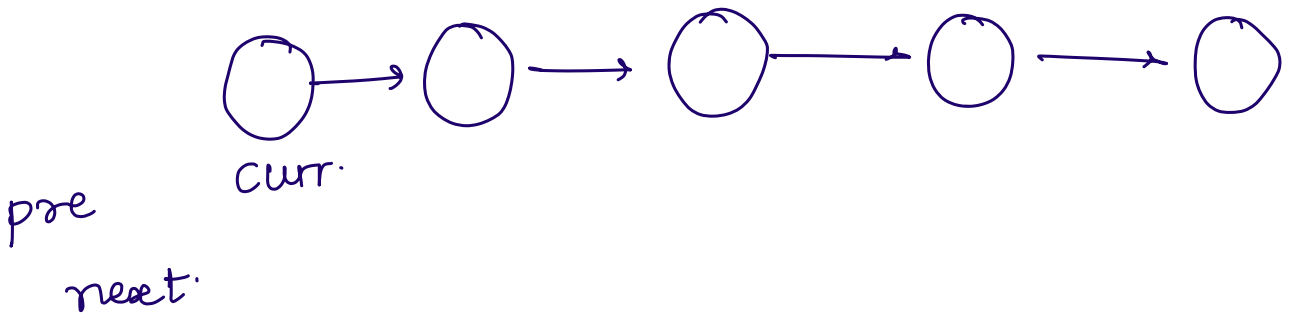
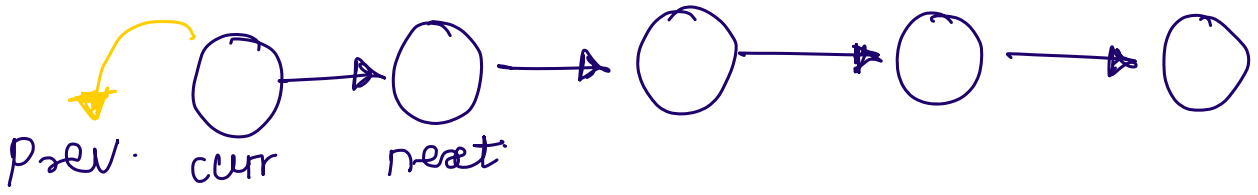
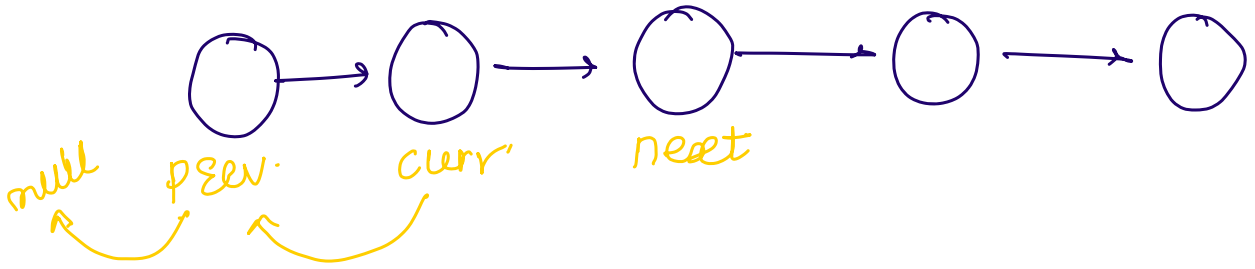
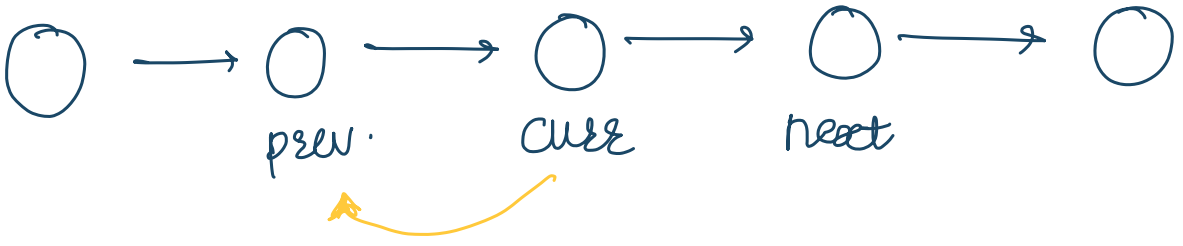
① Reverse linked list -





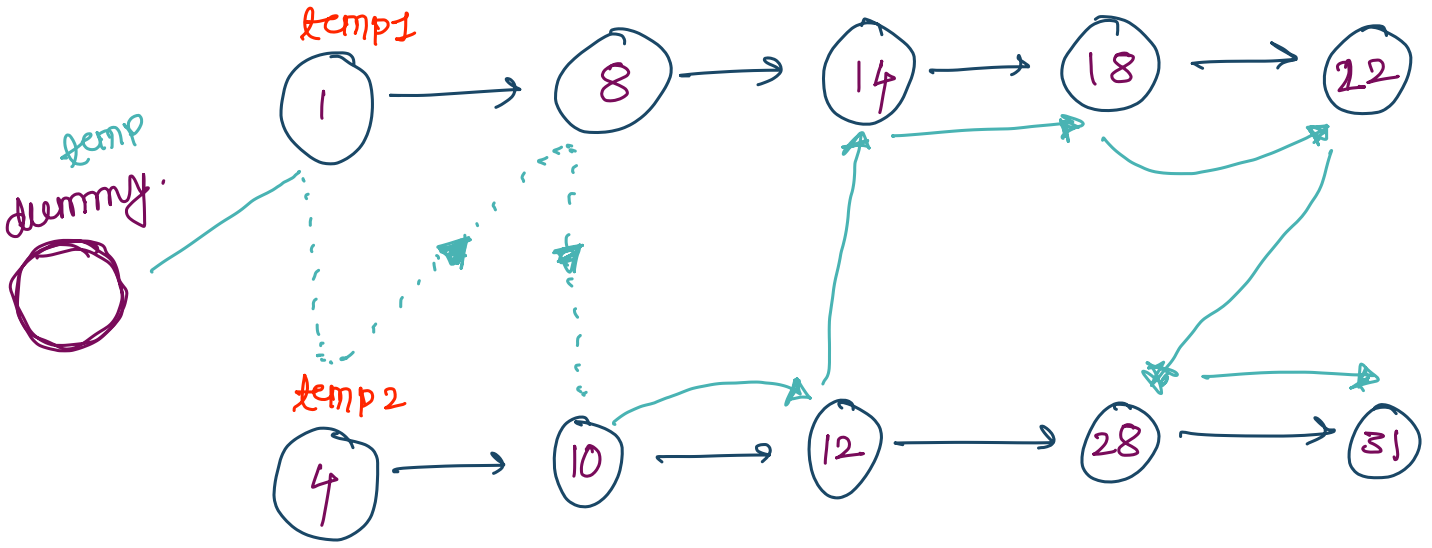




① just keep connecting current to prev

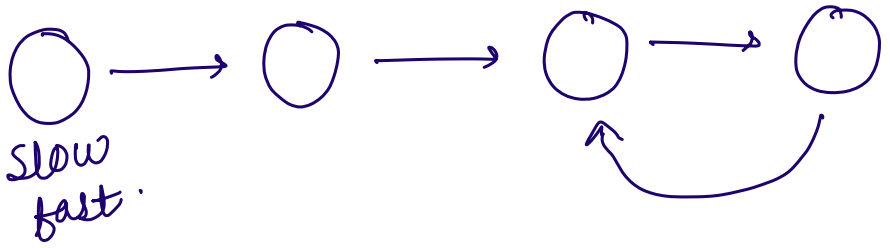
② & keep updating prev, curr & next.

② merge list



return dummy.next;

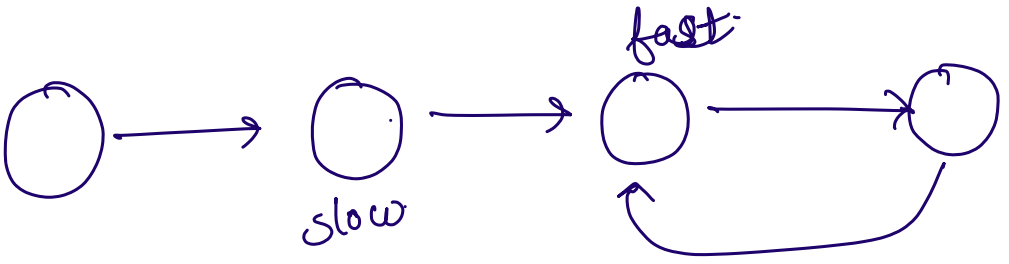
③ Cycle in linked list -

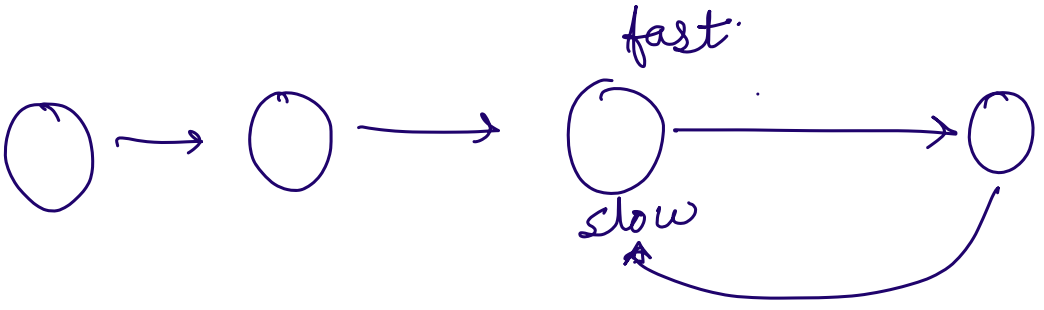


fast → 2x
slow → 1x.

while (fast != null)

if fast == slow return true.





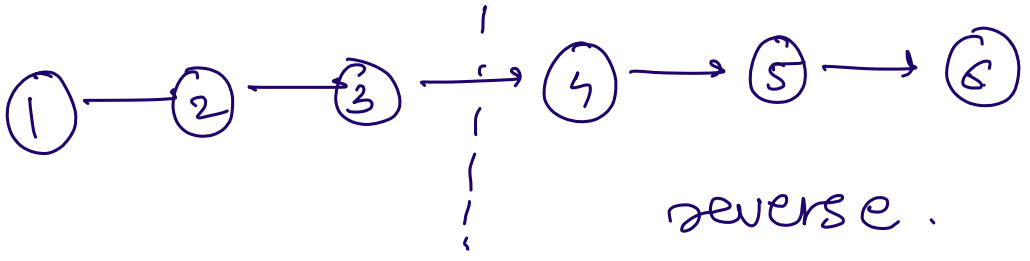
④ Reorder linked list.

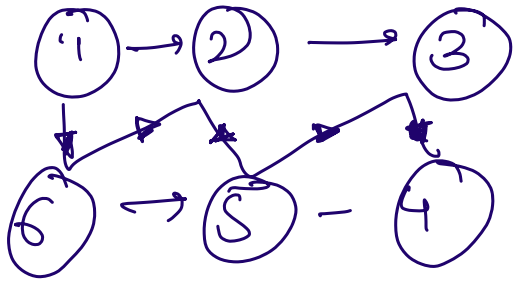
input ① → ② → ③ → ④ → ⑤

o/p ① → ⑤ → ② → ④ → ③

find halt → slow fast pointer

split the list





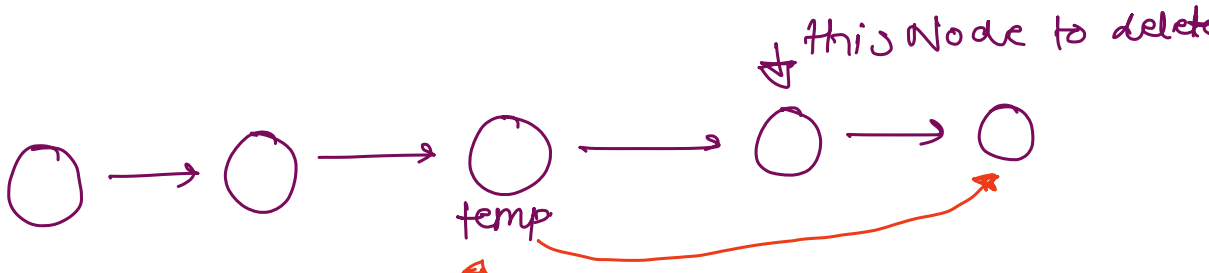
o/p. ① → ⑥ → ② → ⑤ → ③ → ④

① Cut list into halt with slow & fast pointers.

② reverse the second halt.

③ link the first & second halt alternatively.

⑤ Delete nth node from end of linked list.

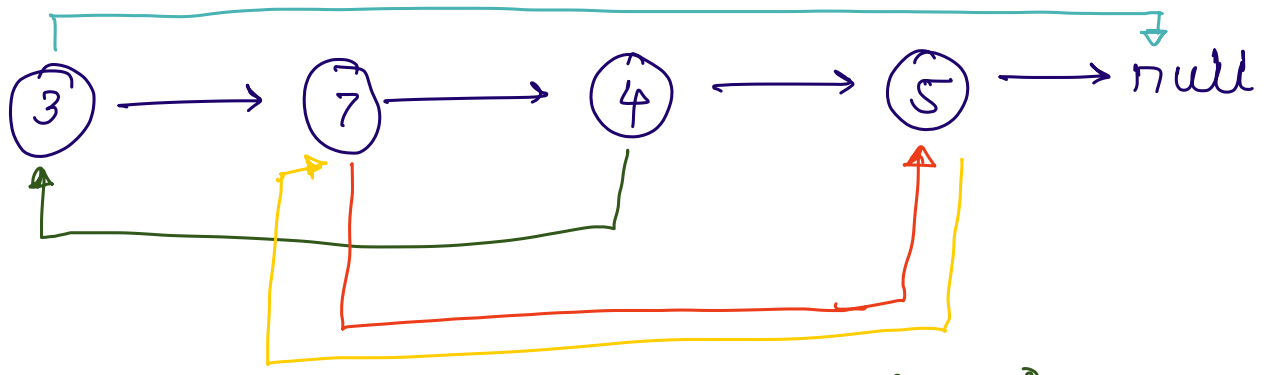


① get length of linked list by traversing

② now traverse towards temp.

& connect the node to next.next

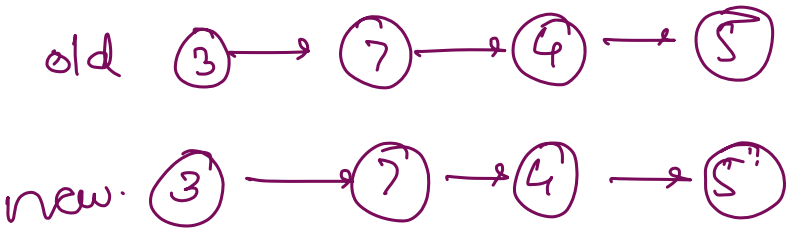
⑥ copy list with Random pointers:



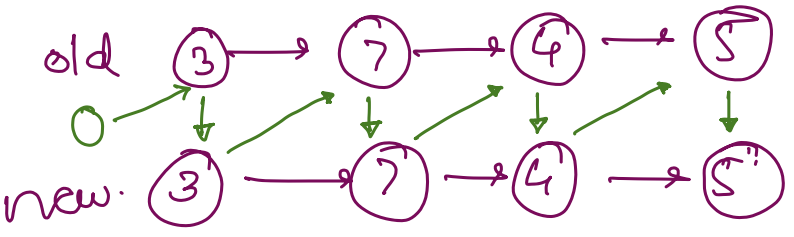
have to make deep copy of the list with random pointers

we can't directly copy random pointers

① make simple copy



② Now, attach the linked lists one-to-on.



so, for copying random pointers we use the relation.

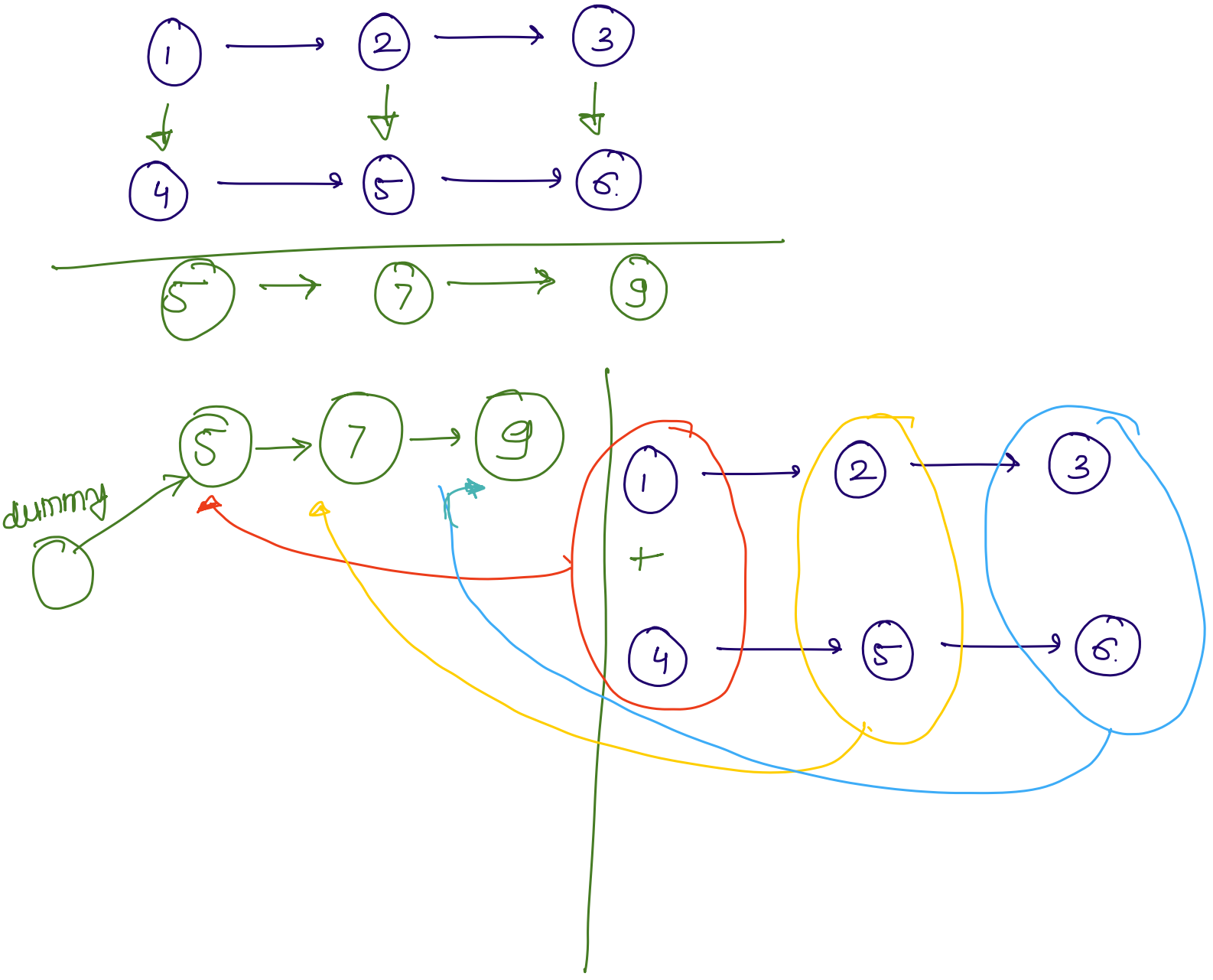
e.g new (4)'s random would be = new 4's → prev . random pointer . next
old '4' old 3. . new 3

this is how we assign random pointers to our new list

then we separate the list.

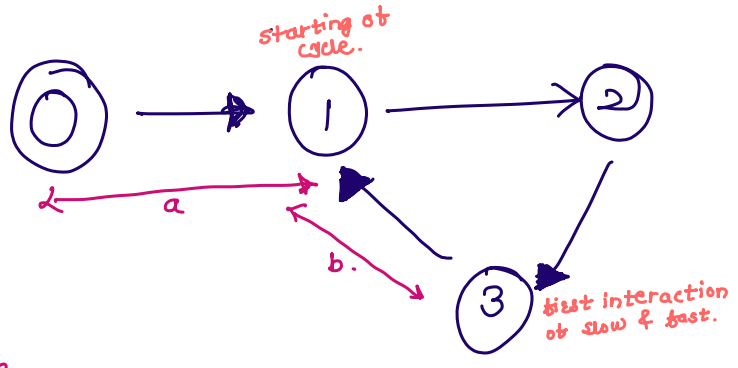
steps — ① make deep copy
② join original & deep copy
③ assign Random pointers of new list (By given logic)
④ separate two lists.

⑦ Add two numbers :



(8) Find the duplicate Numbers.

very non intutive problem.



a & b are same distances.

distance between the first interaction of slow & fast pointer &

distance between starting of list to the starting of the cycle

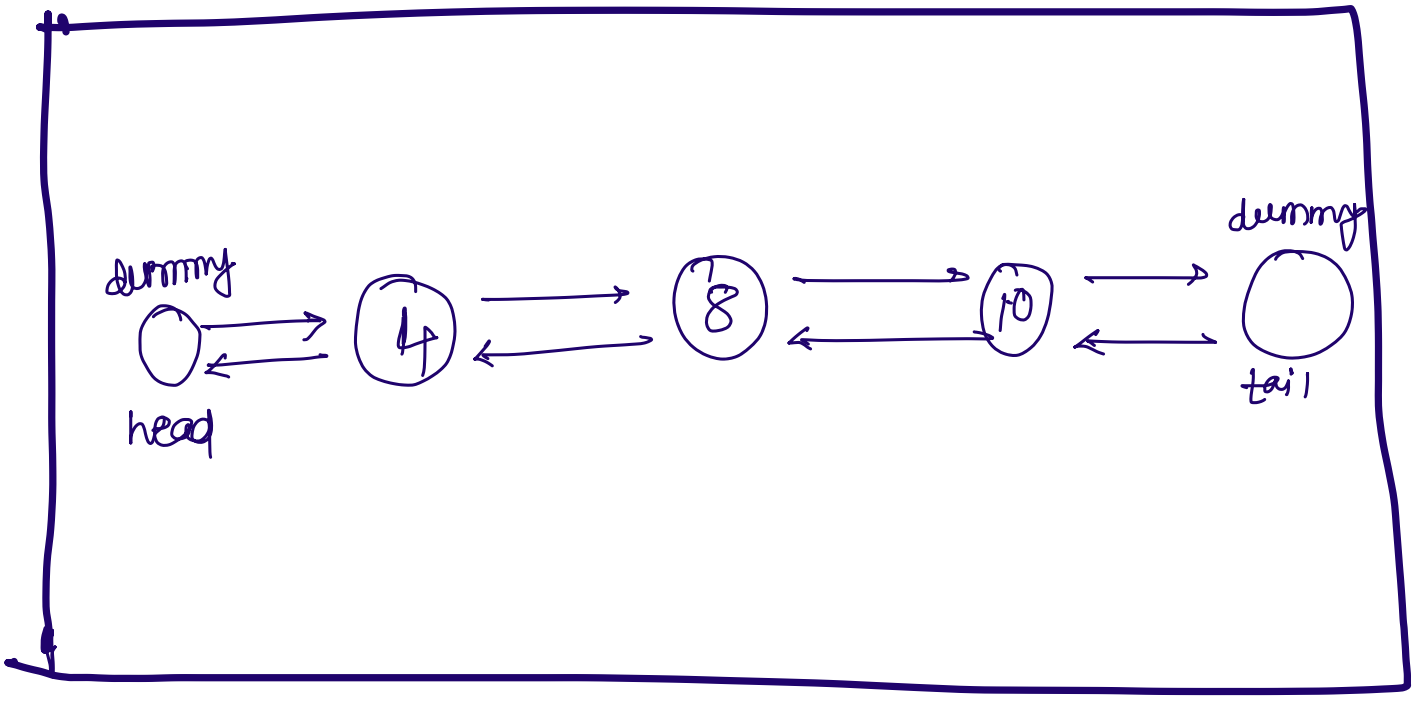
is same.

1. first run a while loop till both the pointers meet.
2. then start a pointer from start of LL & from the interaction point of slow & fast.
3. the point where they meet would be the start of cycle.

⑨ LRU Catch.

Least Recently used -

Storage Capacity (3)



Key	Node.
4	4's node.
8	8's node
10	10's node

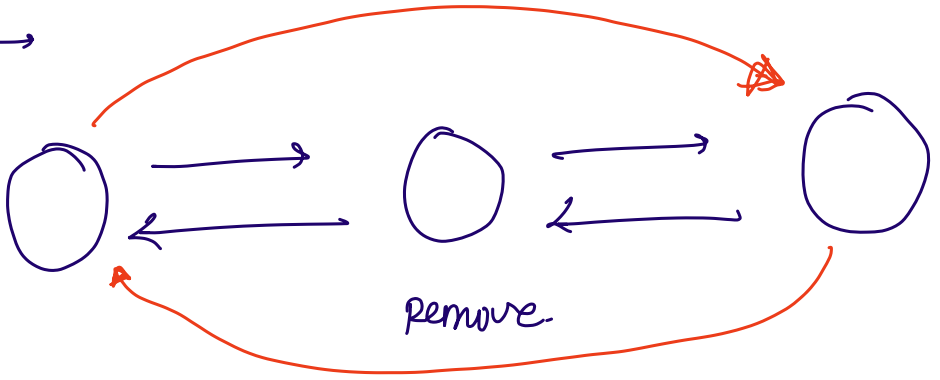
```
get (int val) {
    remove ( )
    insertInfront ( )
    return value
}.
```

```
put (int key, int value) {
```

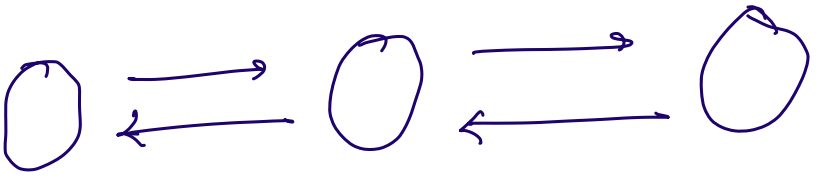
if present → (remove ( )
InsertInfront ( ))

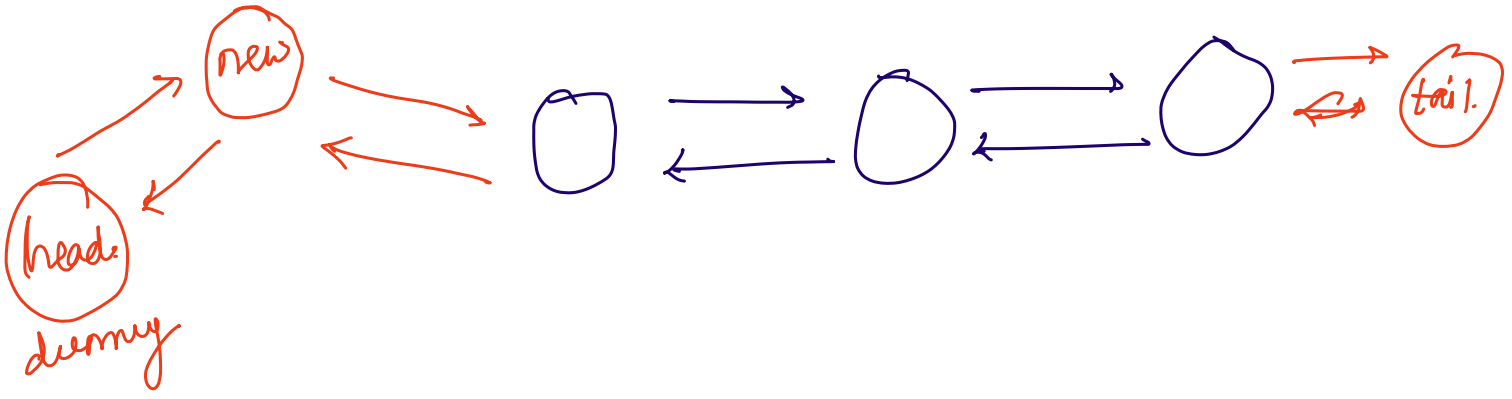
not present → (new node
capacity ++.
if size > capacity →
remove (tail)
insert new at front)

Remove →



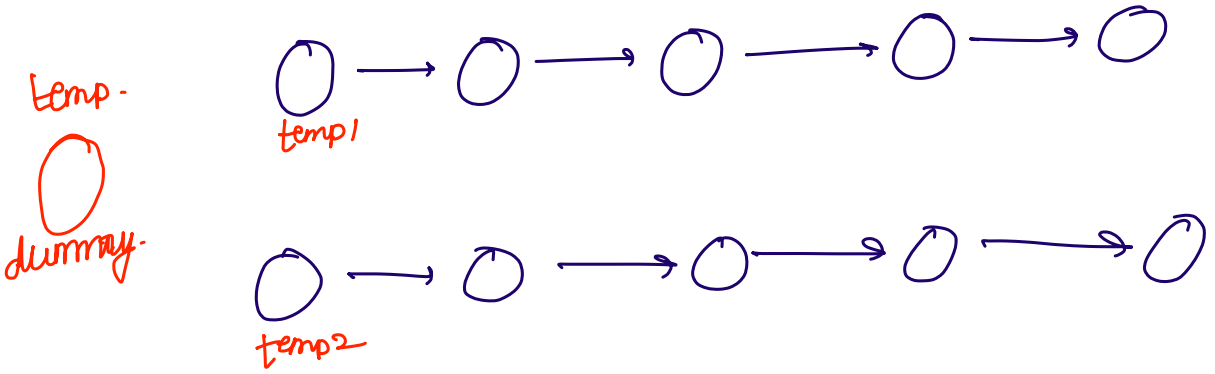
insert In front.





(10) merge K sorted list -

merge two Lists.



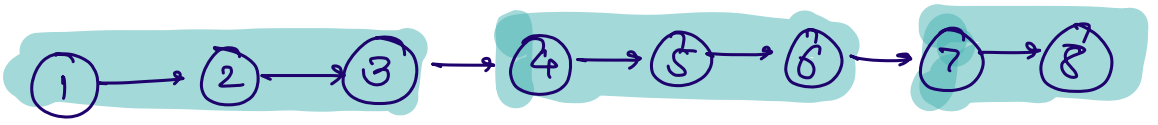
we traverse from -

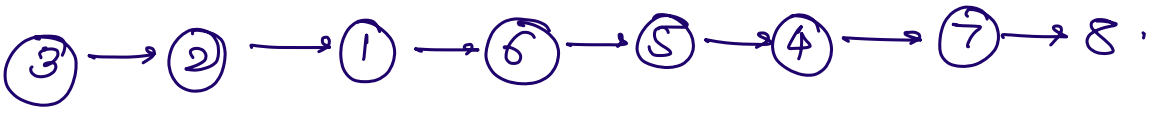
interval = 1

```
while (interval < K) {
    for (int i=0; i+interval < K; i+= interval *2)
        List[i] = mergeTwoList (list[i],
                                lists[i+interval])
    interval *= 2
}.
return List.length == 0 ? null : list[0],
```

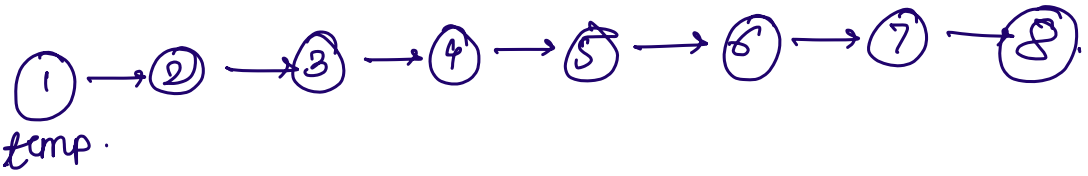
(11) Reverse nodes in 'K' groups

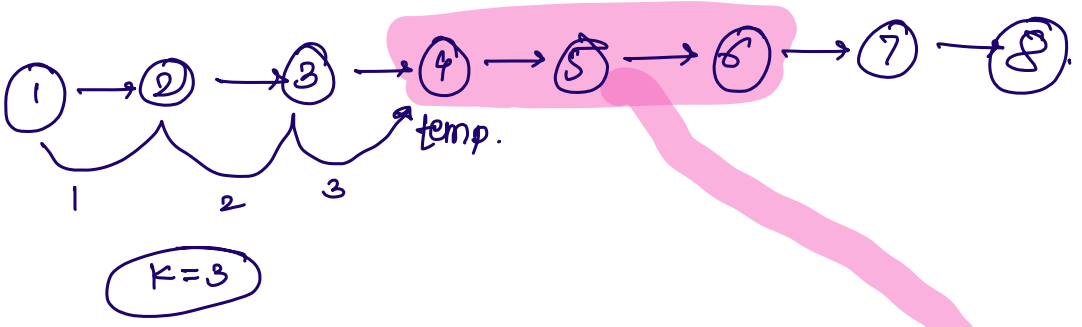
→ suppose K = 3

I/p 

O/p 

if we have three elements in linked list.





reverse the list which starts from temp. With recursion.