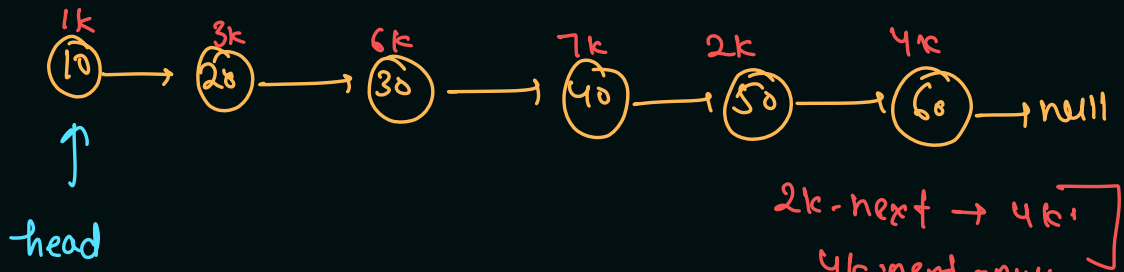
Reverse A Linkedlist

Middle Of A Linked List

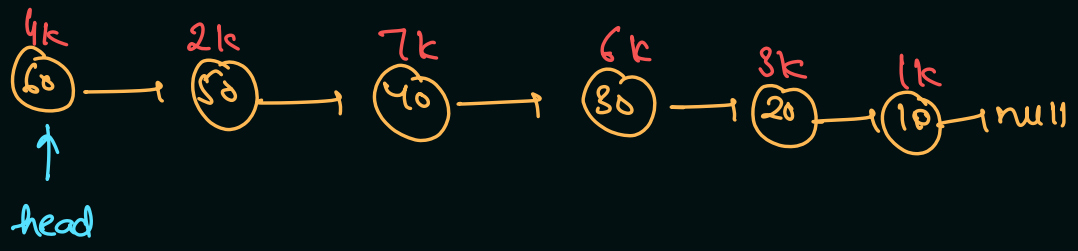
Palindrome Linkedlist



2k.next → 4k

4k.next = null

pointer Change.

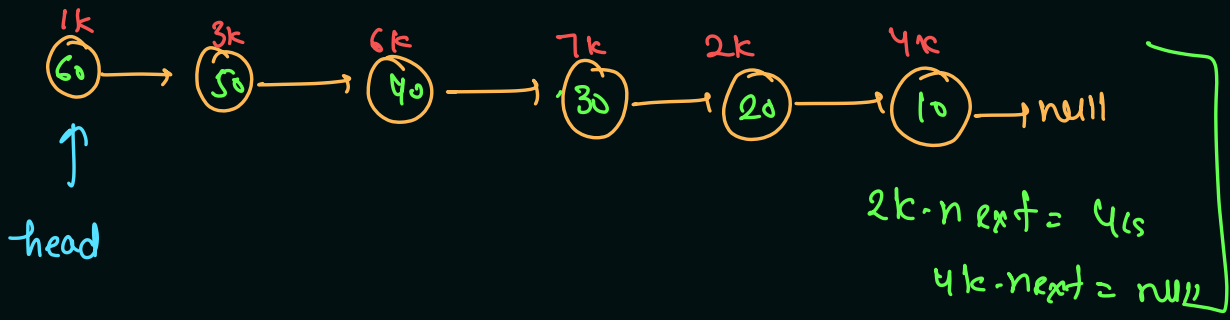


4k.next → 2k

2k.next = 7k

① Reverse using data Swapping.

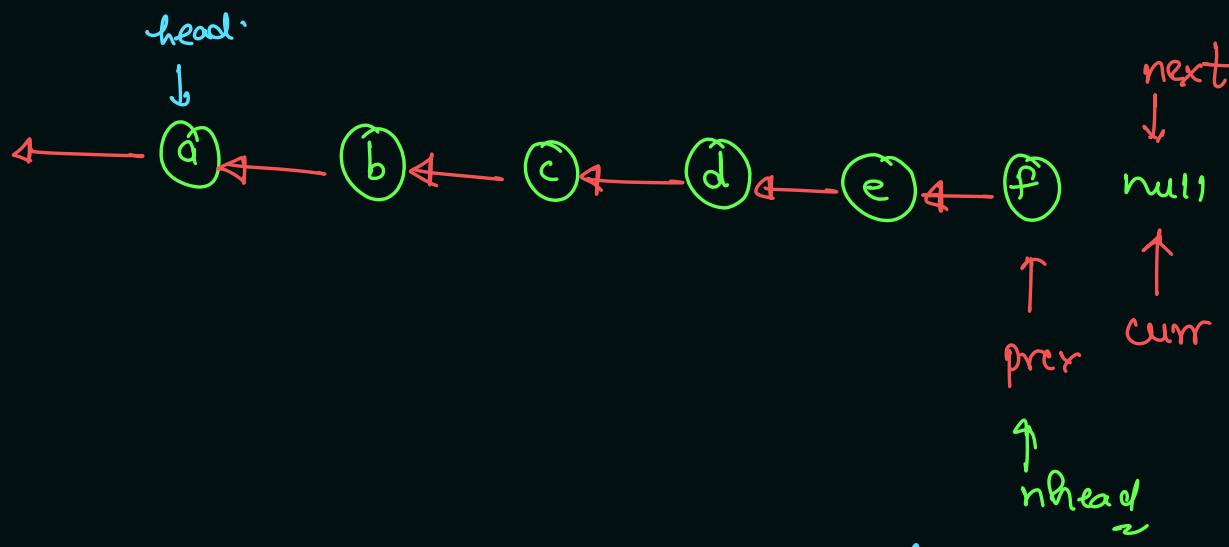
② Reverse using pointer



2k.next = 4k

4k.next = null

Pointer's are Same, but data reverse



① Make three pointer.

→ prev
→ curr
→ next] Running time.

loop, → Condition of Running loop → while (curr != null)

```
next = curr.next
curr.next = prev;
prev = curr;
curr = next
```

nhead = prev

order of Eq. is symmetric

Middle of a Linked List :

Object 1 → speed x

Object 2 → speed 2x

distance travelled by obj 1 in 't' time

dist. = speed x time

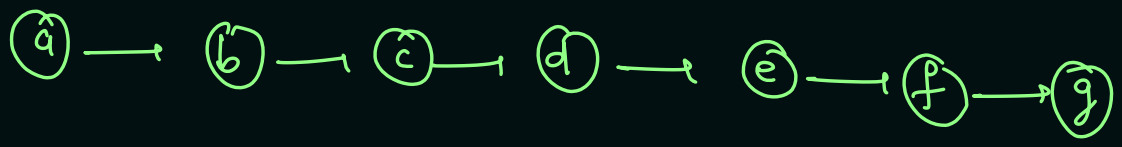
= x x t = xt

distance travelled by obj 2 in t time.

dist = speed x time

= 2x x t = 2xt

odd size



distance travel by obj 1 is half of distance travelled by obj 2.

If obj 2 travelled D dist. then obj 1 is present at 'Middle' of D distance.

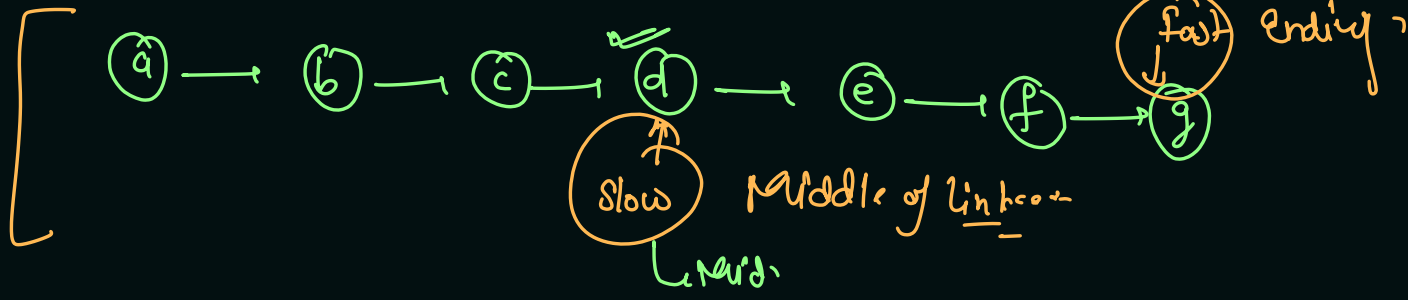
slow = x

fast = 2x

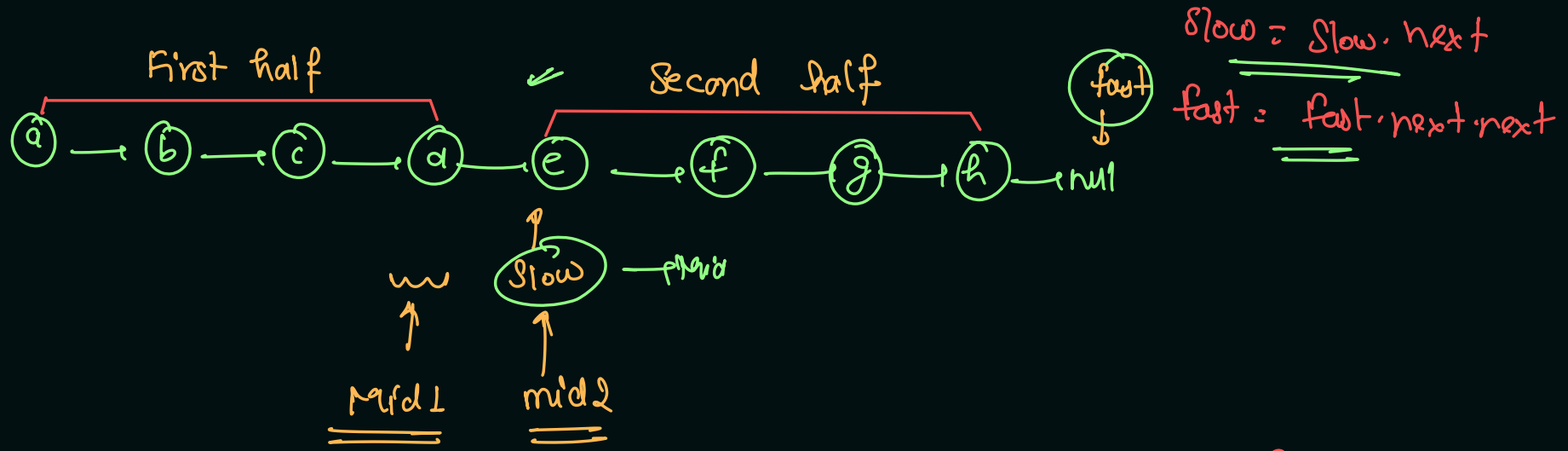
slow = slow.next

fast = fast.next.next

odd:



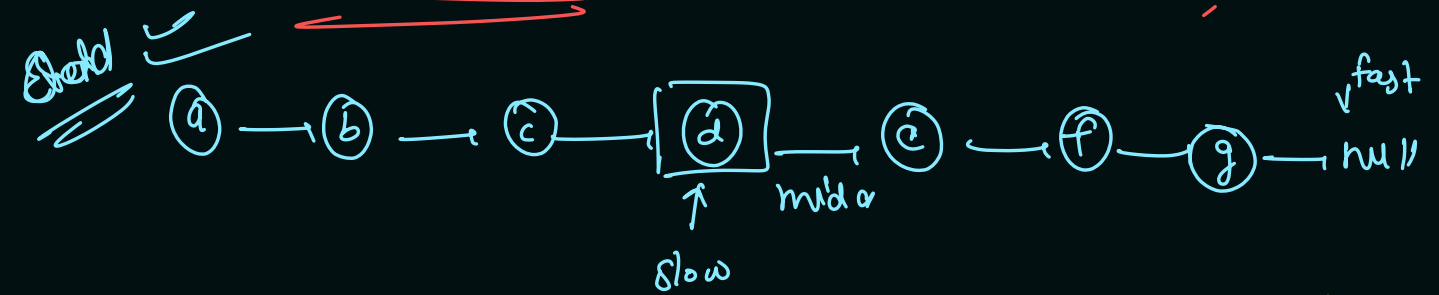
slow = head;
fast = head;



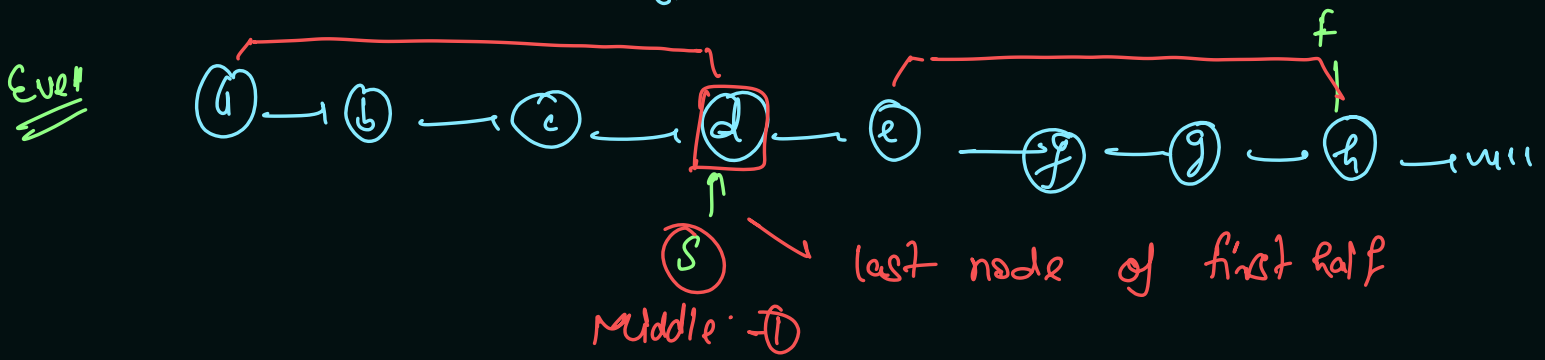
slow = slow.next
fast = fast.next.next

When fast is reached at end, that time slow is at first node of second half

odd → slow = head;
fast = head.next;



Even



Palindrome of a Linked list

Palindromic → 1 → 2 → 3 → 2 → 1 ✓

a → b → c → c → b → a ✓

Check
Palindromic array

left left

Right Right

left to Right → LL. in O(1)
Right to left → LL. in O(n)

left ++

→ If use of array is allowed, then it is solvable.

Case → Right to left

Reverse LinkedList and then solve the problem.

Concept → Mid & Midx

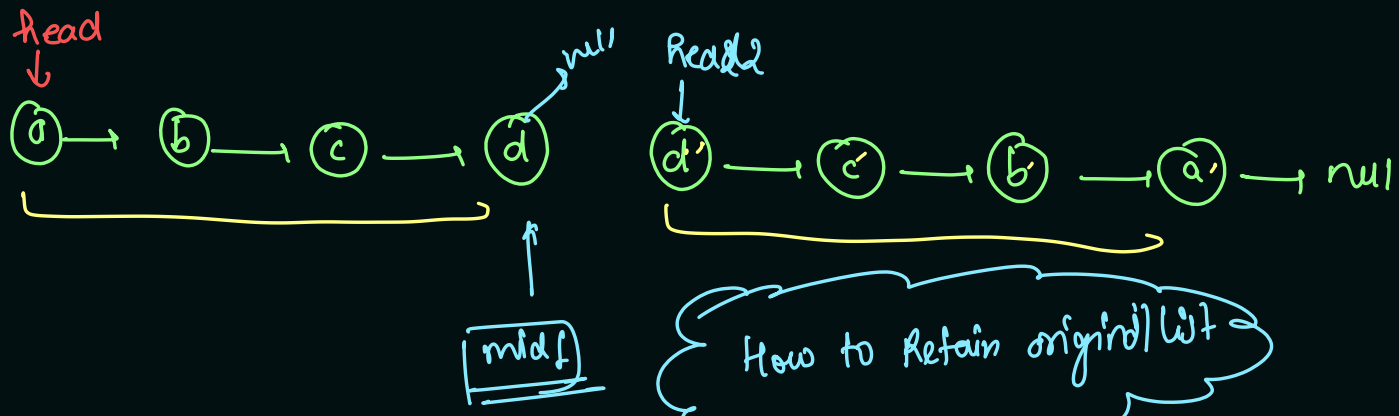
LinkedList

avoid [Array ✗ , Recursive ✗]

Check -

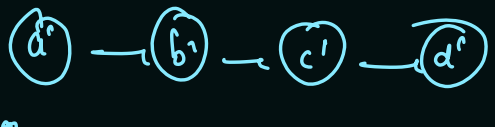
1. head = null
2. head.next == null
3. odd size.
4. Even size.
5. head.next.next = null

size=2 [5]

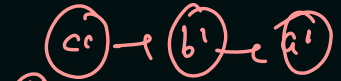


Steps

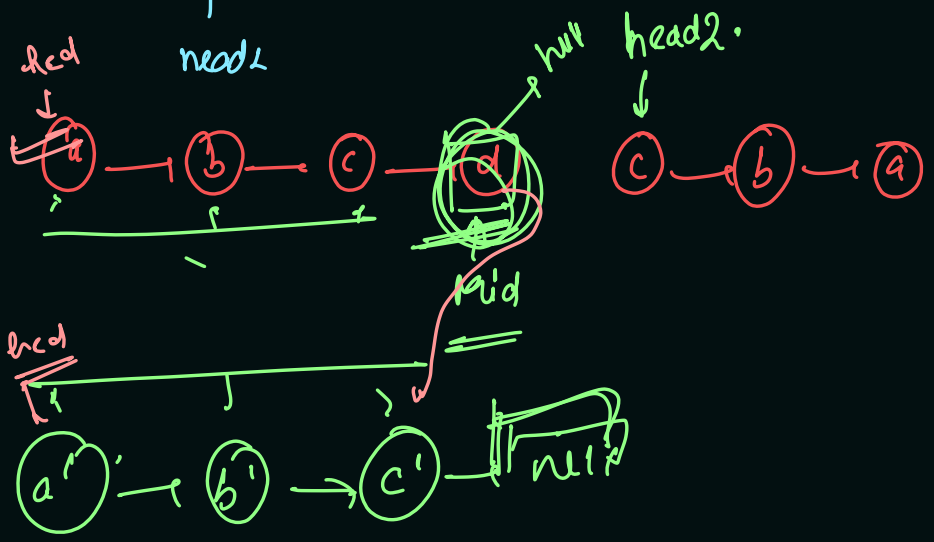
1. get mid
2. head2 = mid.next
3. mid.next = null
4. Reverse head(2)
5. Check pallindromic



1. head2 → reverse

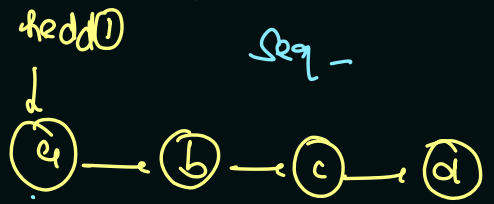


2. mid.next = head2

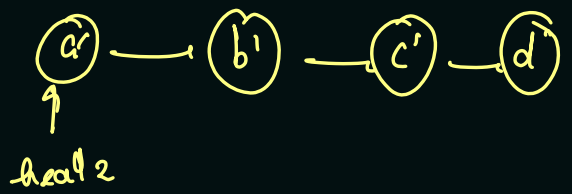


Seq -

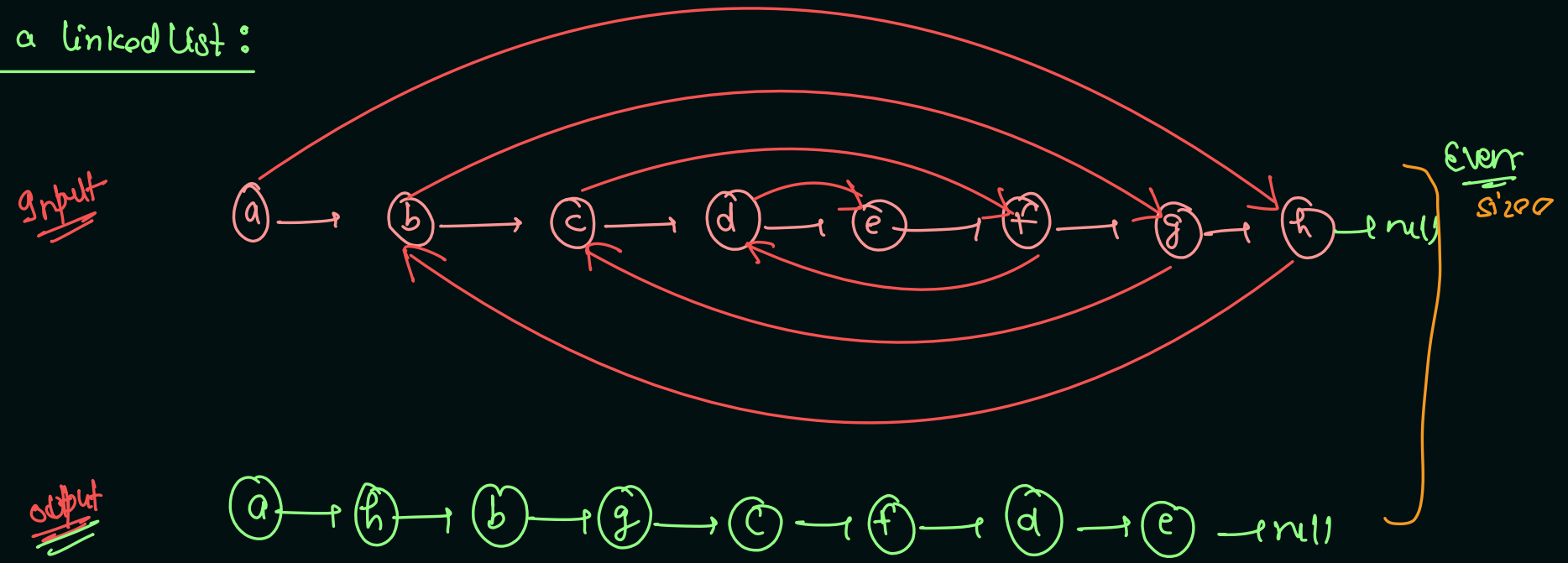
first list

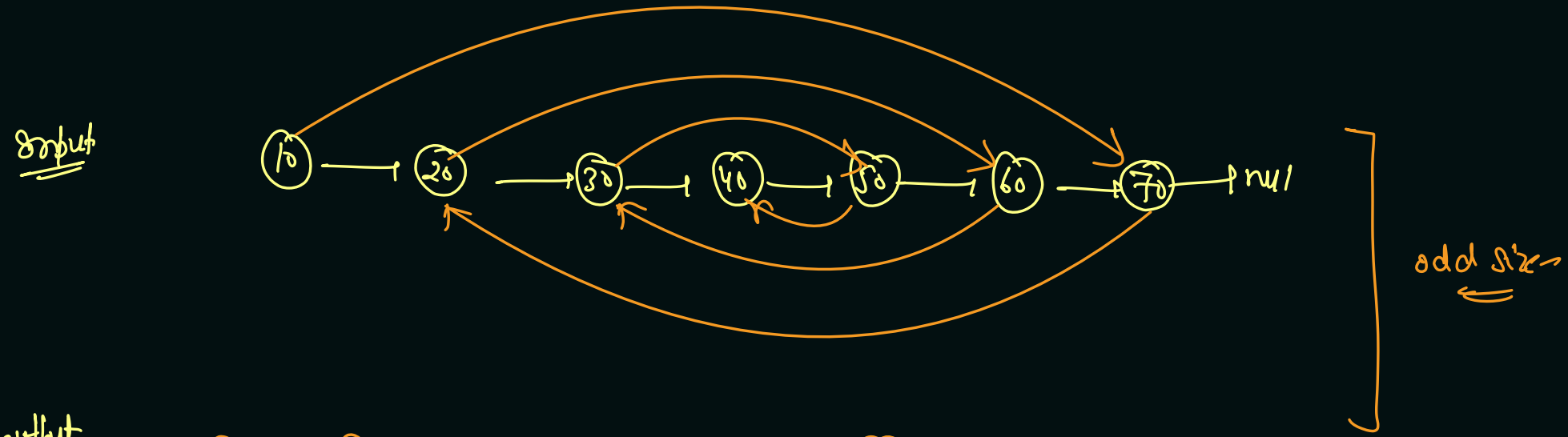


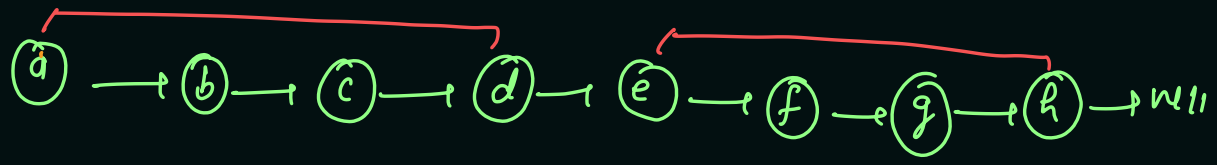
Second list

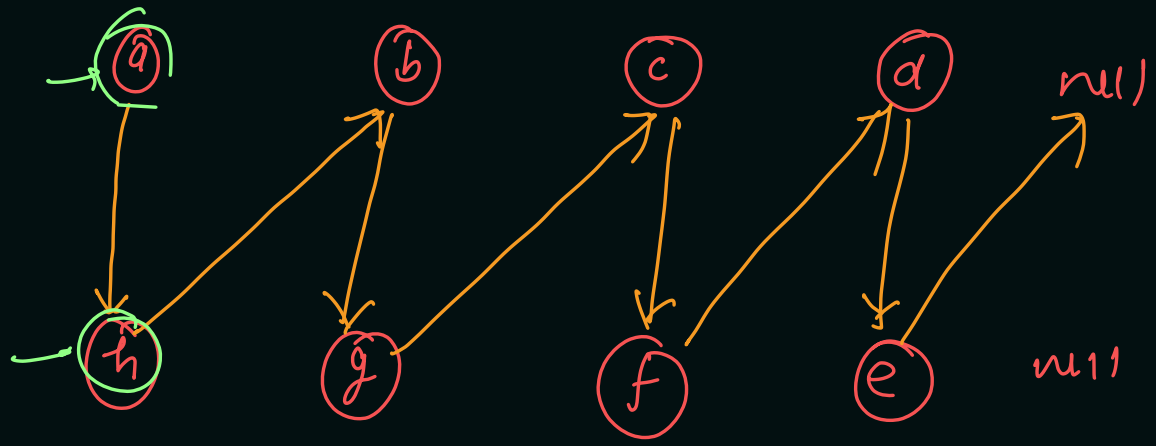


Fold of a linked list :



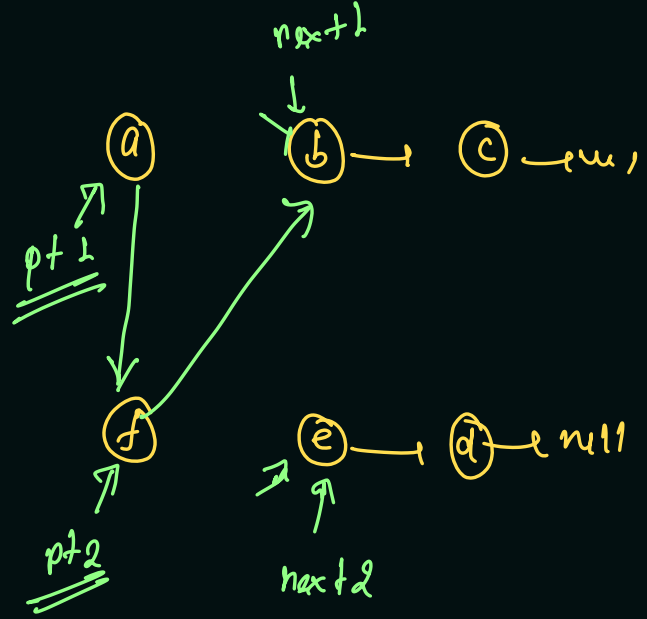




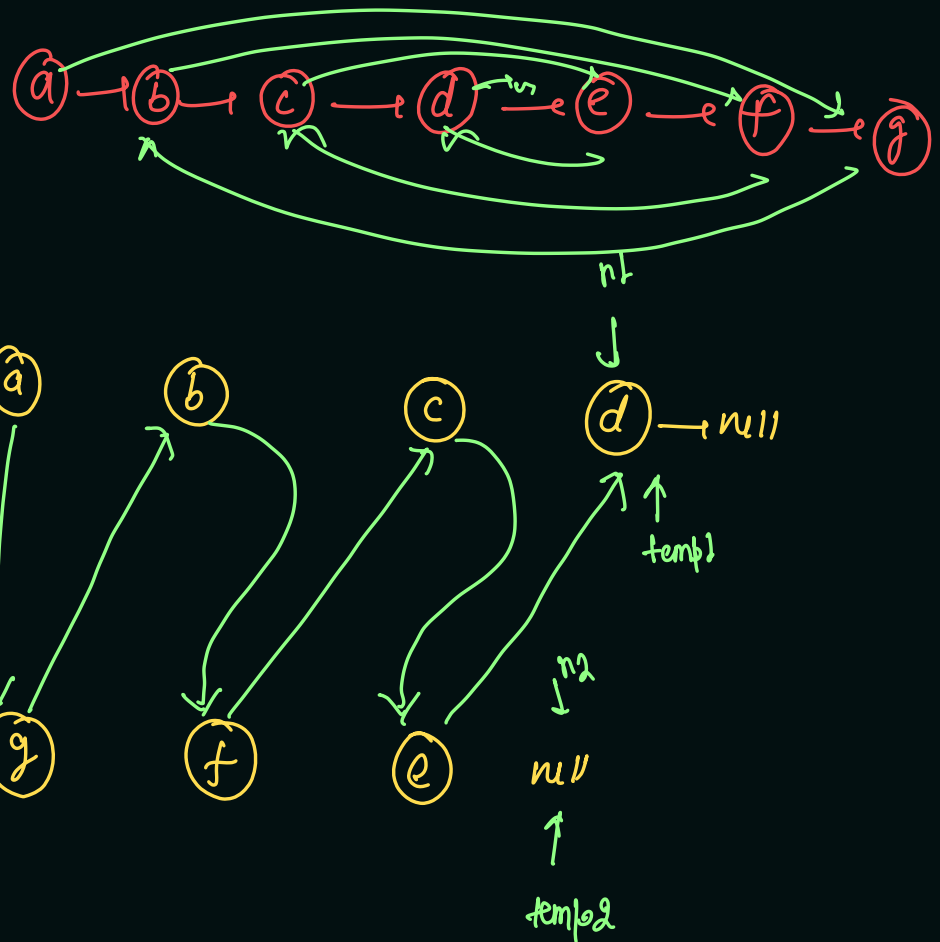


Steps:

1. Split from middle & reverse second half.
2. Merge them according to proper logic




```
pt1.next = pt2;
pt2.next = next1;
pt1 = next1;
pt2 = next2;
```



don't do anything

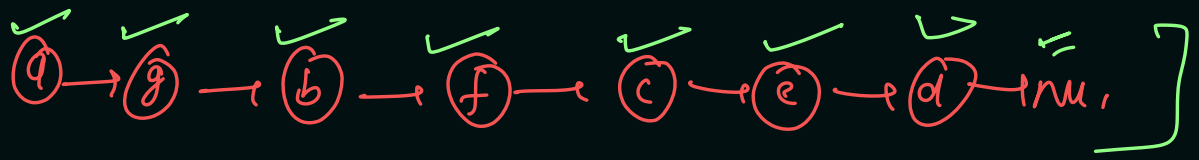
return head as it is

head == null ? → null

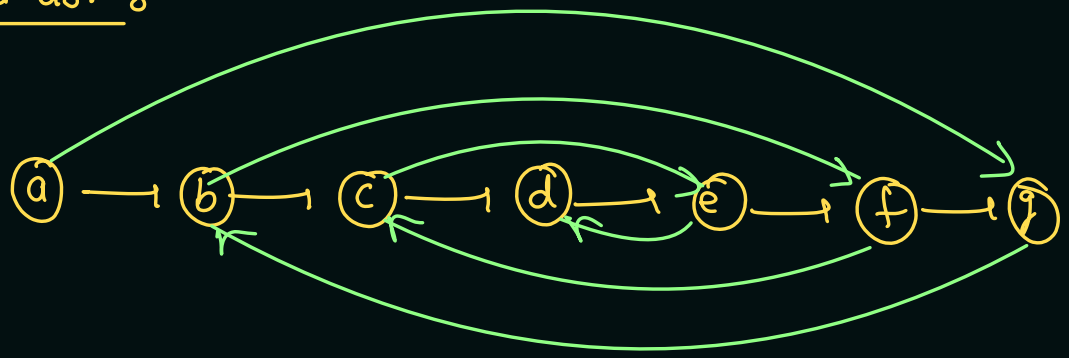
head.next == null → 10 → null

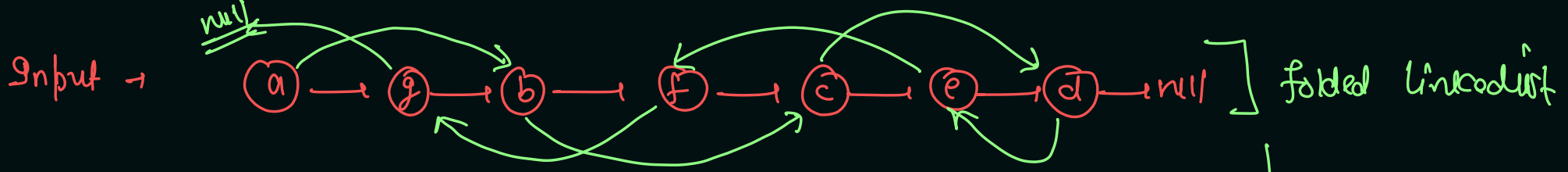
head.next.next == null → 10 → 20 → null

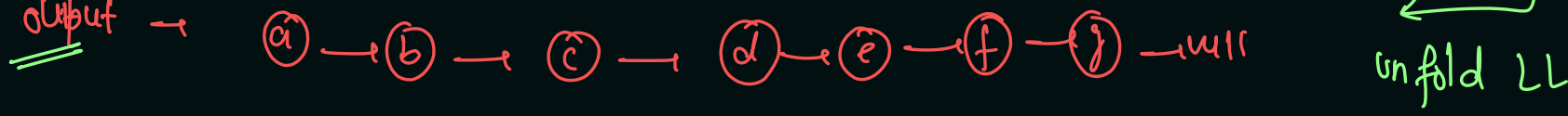
```
while(temp1 != null && temp2 != null) {
    ListNode next1 = temp1.next;
    ListNode next2 = temp2.next;

    temp1.next = temp2;
    temp2.next = next1;

    temp1 = next1;
    temp2 = next2;
}
```

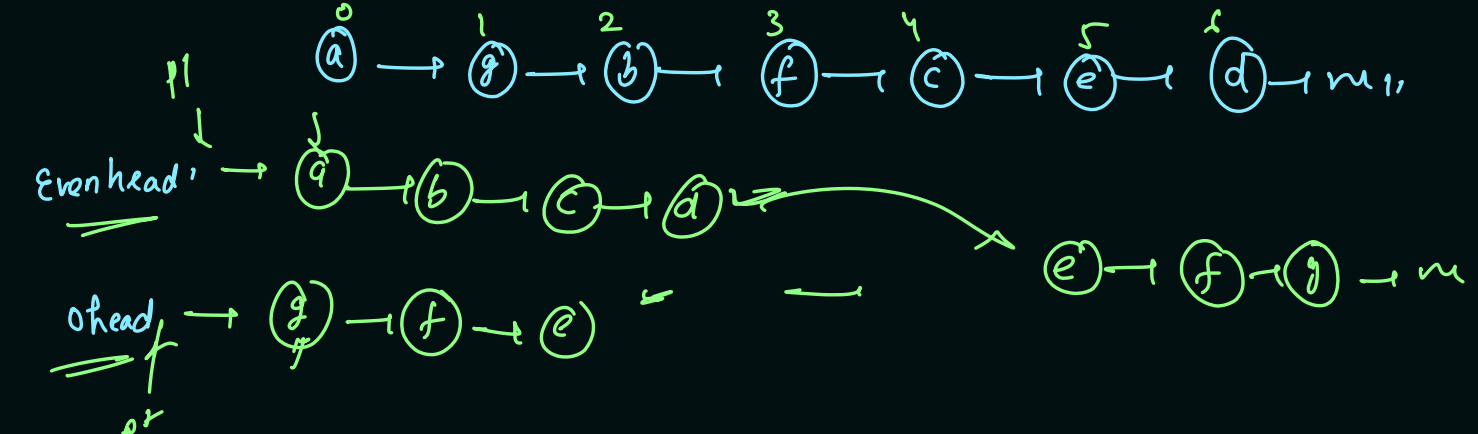


Unfold a linked list :

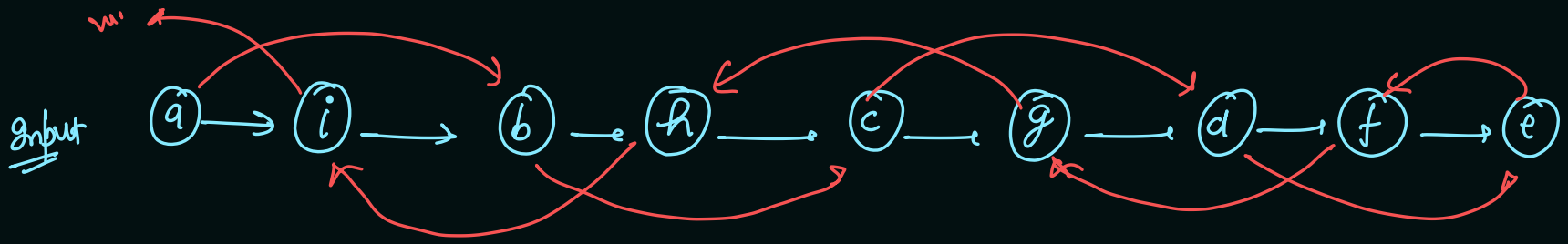




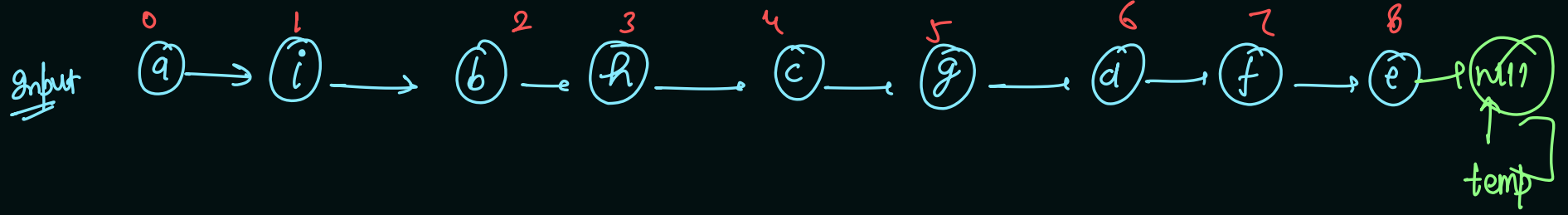




output → a → b → c → d → e → f → g → h → i → null

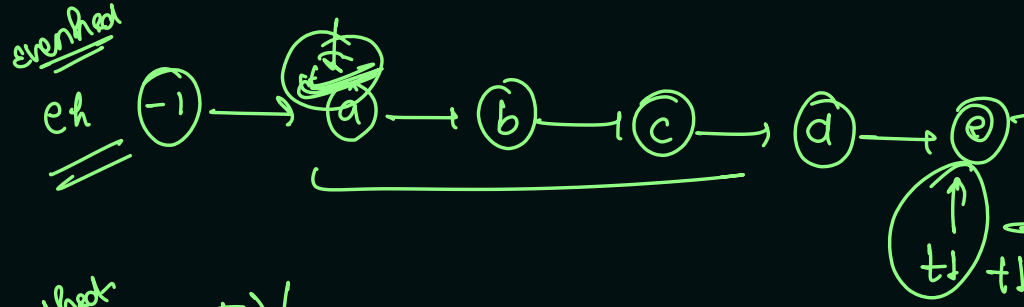


dummy Node



indx = 0 1 2 3 4 5 6 7 8

9



indx = even

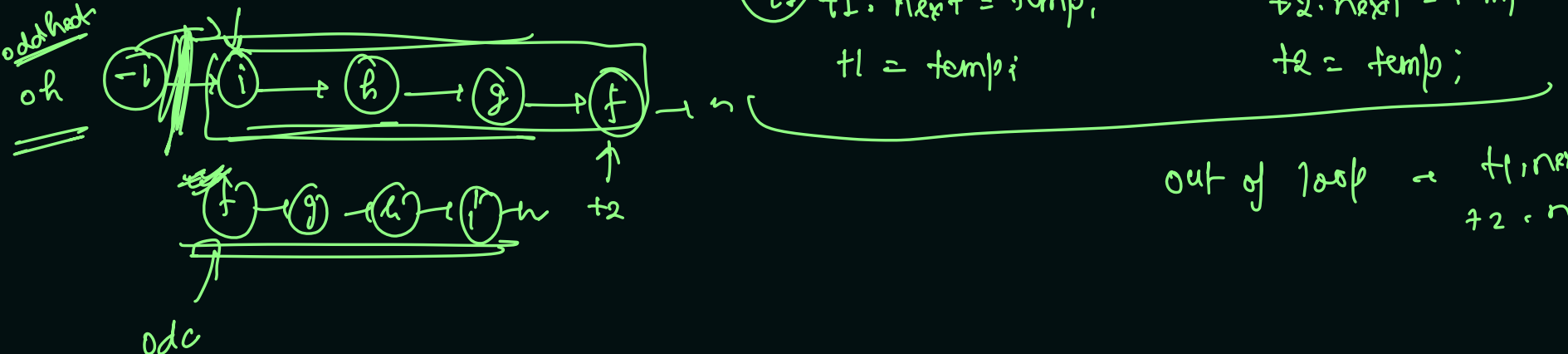
t1.next = temp;

t1 = temp;

indx = odd

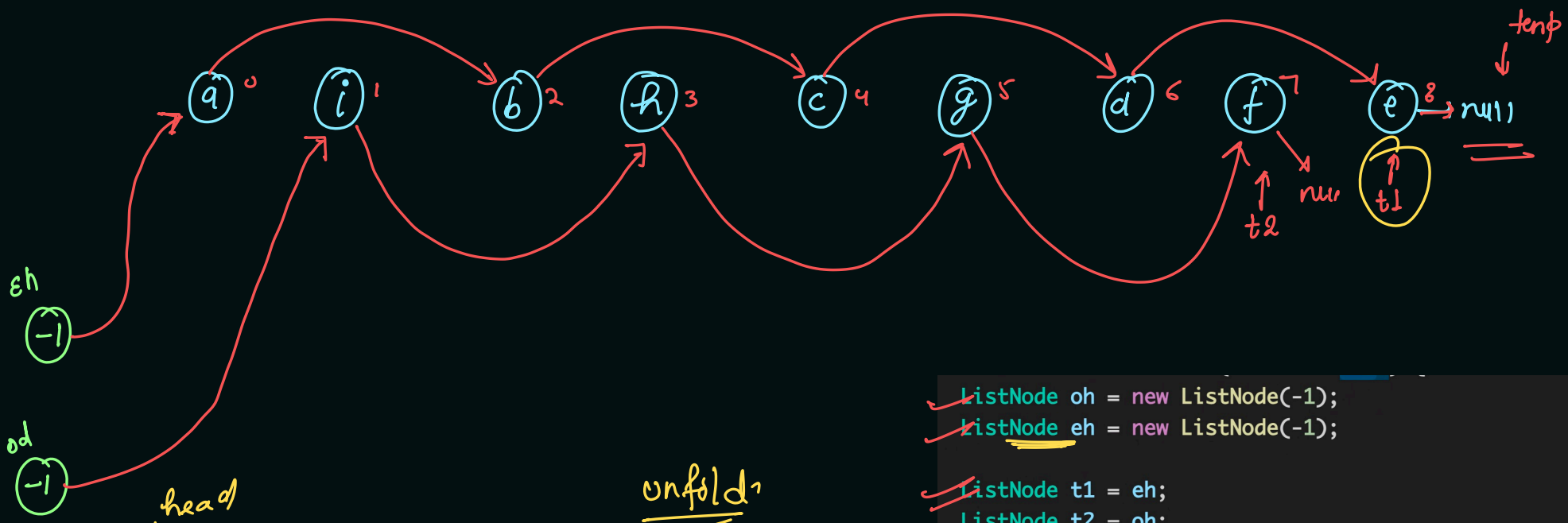
t2.next = temp.

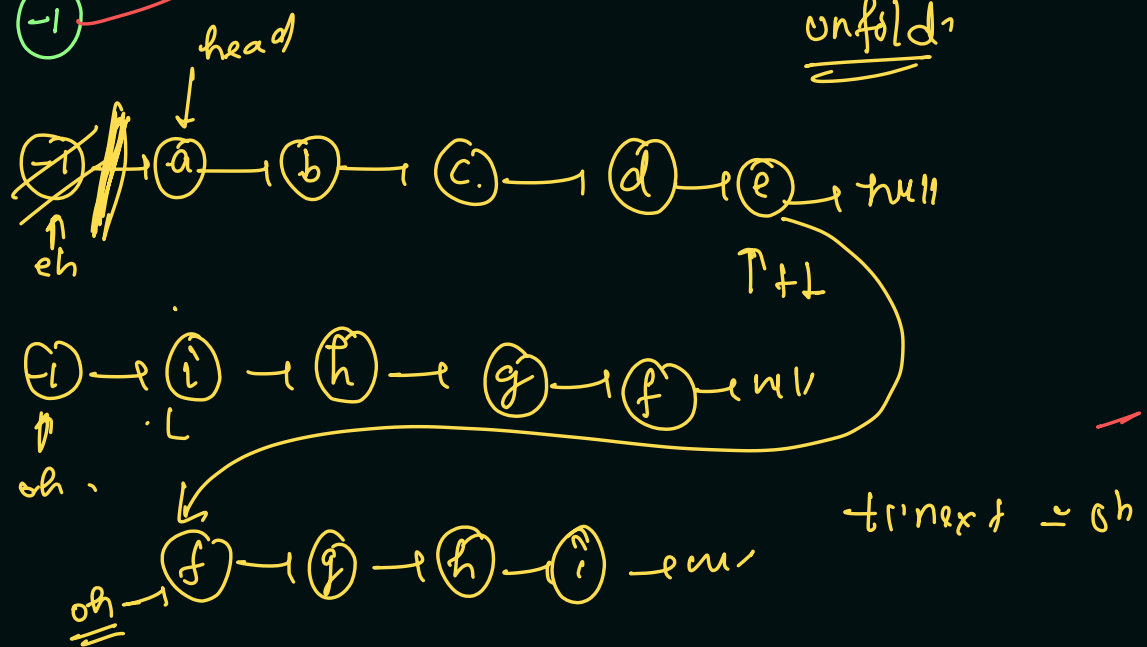
t2 = temp;



out of loop → t1.next = n

t2.next






```java
ListNode oh = new ListNode(-1);
ListNode eh = new ListNode(-1);

ListNode t1 = eh;
ListNode t2 = oh;
int indx = 0;
for(ListNode temp = head; temp != null; temp = temp.next, indx++) {
    if(indx % 2 == 0) {
        // even
        t1.next = temp;
        t1 = temp;
    } else {
        // odd
        t2.next = temp;
        t2 = temp;
    }
}
t1.next = null;
t2.next = null;
oh = reverseList(oh.next);
t1.next = oh;
```

Java Memory Managment

Mark & Swipe Algorithm

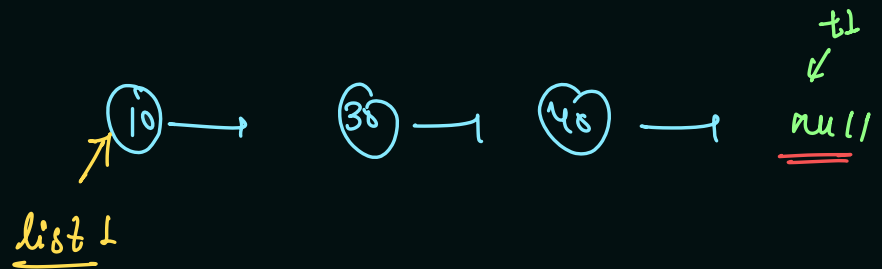
[head == null | head.next == null | head.next.next == null] Nothing to do

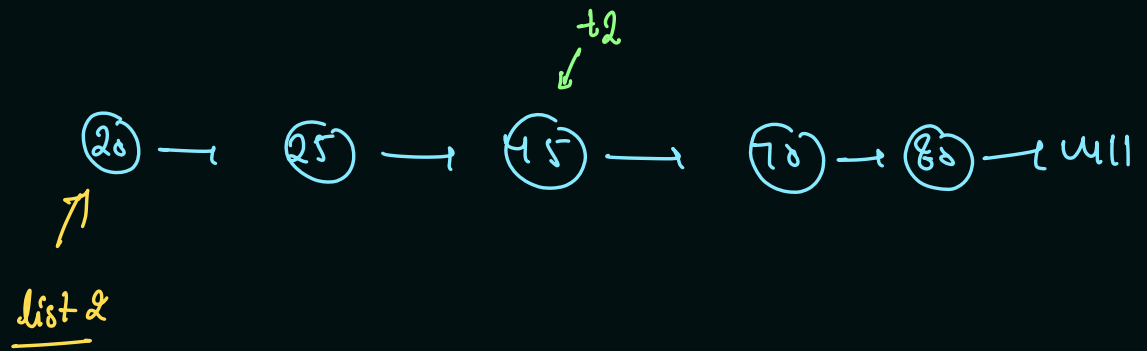
Merge two Sorted List:

Edge case :- (l1 == null || l2 null)

return l1 == null ? l2 ; l1

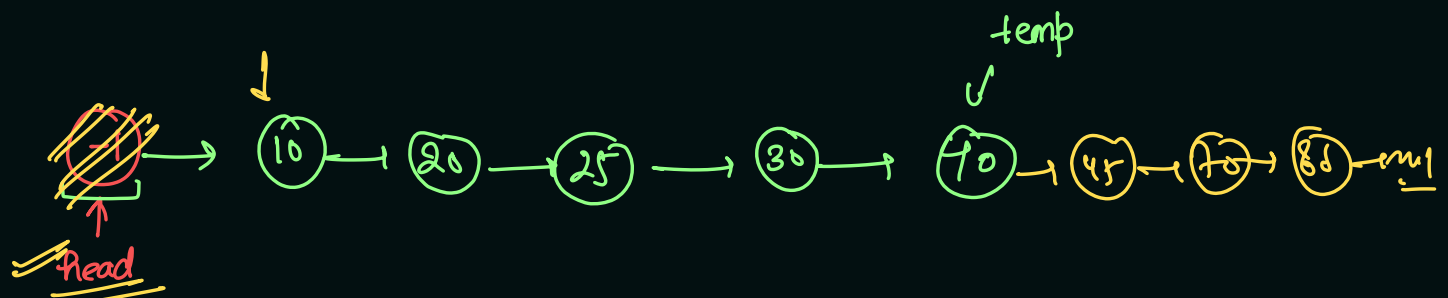
1. dummyNode
2. t1, t2 & temp-





loop:

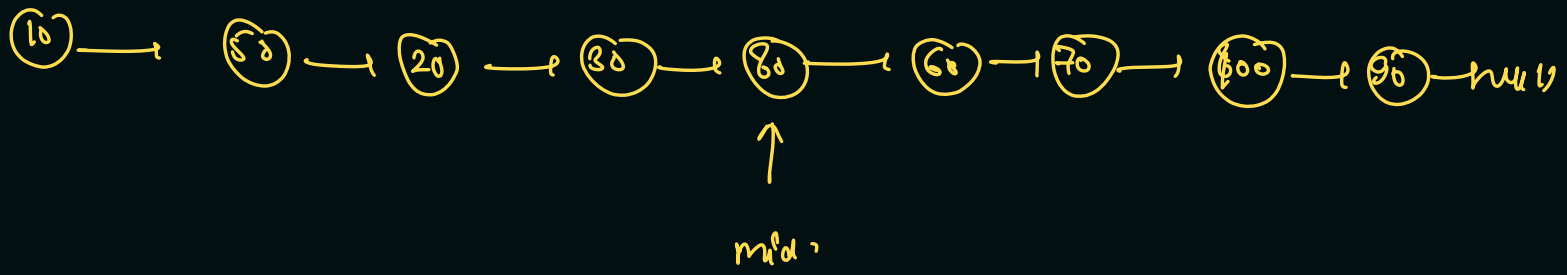
```
if(t1.val < t2.val){
  temp.next = t1;
  temp = t1;
  t1 = t1.next
} else {
  temp.next = t2;
  temp = t2;
  t2 = t2.next
}
```



temp.next = t1 == null ? t2 ; t1 ;

Result → head.next

Merge Sort



l1 10 → 50 → 20 → 30 ← 80 ← sort

l2 60 → 70 → 100 → 90 → sort

Merge

```java
public ListNode sortList(ListNode head) {
    if(head == null || head.next == null) return head;
    ListNode mid = midNode1(head);
    ListNode head2 = mid.next;
    mid.next = null;
    head = sortList(head);
    head2 = sortList(head2);
    ListNode res = mergeTwoLists(head, head2);
    return res;
}
```

n → 2n → n

T(n/2) , T(n/2)

$T(n) = n + T(n/2) + T(n/2) + n + k$

$T(n) = 2T(n/2) + 2n + k$

$2T(n/2) = 2^2 T(n/2^2) + 2n + 2k$

$2^2 T(n/2^2) = 2^3 T(n/2^3) + 2n + 2^2 k$

$2^3 T(n/2^3) = 2^4 T(n/2^4) + 2n + 2^3 k$

$2^{h-1} T(n/2^{h-1}) = 2^h T(n/2^h) + 2n + 2^{h-1} k$

h.

$T(n/2^h) = T(1)$

$\Rightarrow \frac{n}{2^h} = 1$, $T(1) = 1$

$\Rightarrow n = 2^h$

$\log_2 n = h \log 2_2$

$\Rightarrow \boxed{h = \log n}$

$a^{\log_a b} = b$

$T(n) = 2^h T(n/2^h) + 2nh + k(1 + 2 + 2^2 + 2^3 + \dots + 2^{h-1})$

G.P.

$\frac{k2^h}{2}$, $\frac{1(2)^{h-1}}{1}$, $\frac{ar^{n-1}}{(r-1)}$, $a=1$, $r=2$, $n=h$

$T(n) = 2^h T(1) + 2nh + \frac{k2^h}{2}$

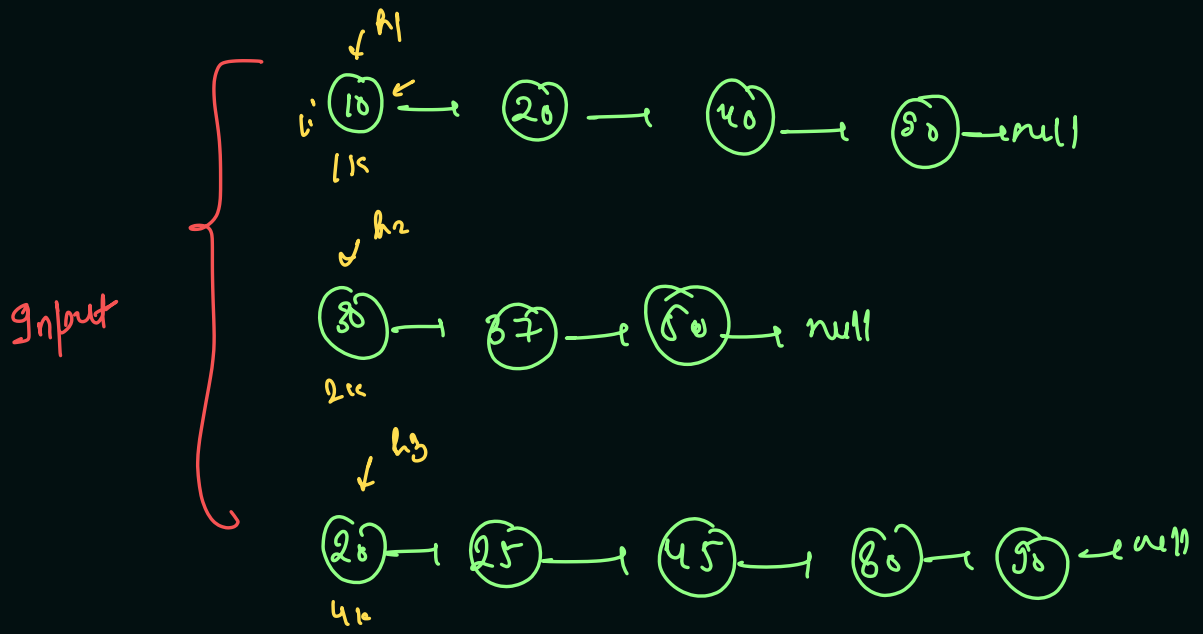
$= 2^h + 2nh + \frac{k2^h}{2}$

$= n + 2h\log h + \frac{kn}{2} = n + 2n\log n + kn/2 \Rightarrow O(n\log n)$

$[2^{\log n} = n]$

Merge k Sorted Linkedlist

k = 3



[1k, 2k, 4k]

[h1, h2, h3]

=

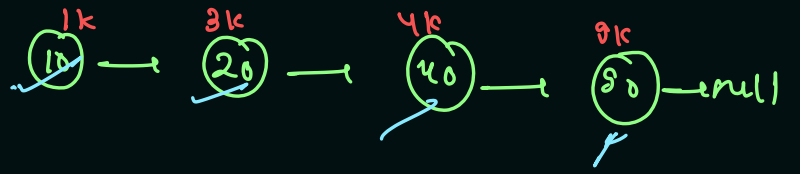
[h1, null, null, h2]

↑

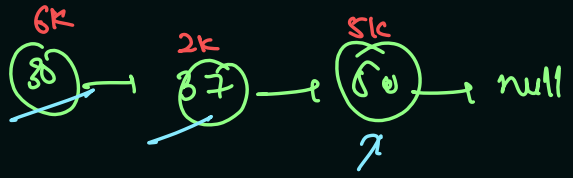
Reference

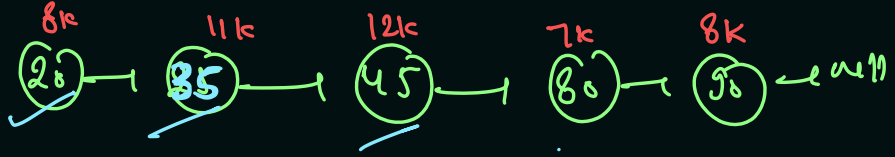
output { 10 → 20 → 25 → 30 → 37 → 40 → 45 → 50 → 80 → 90 → 90 → null

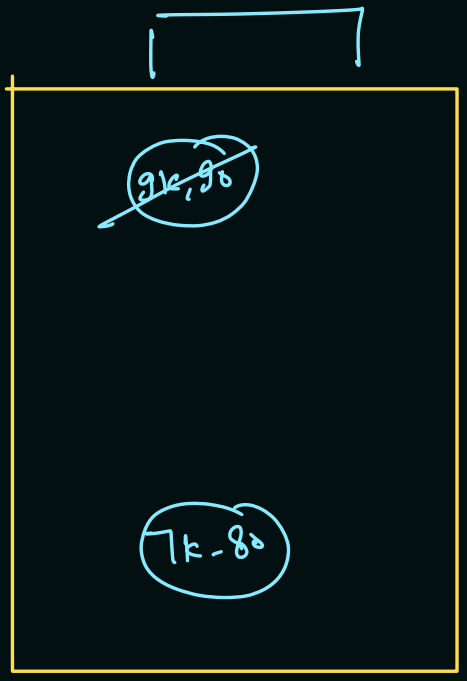
① Add heads of all k list in priority Queue.



②

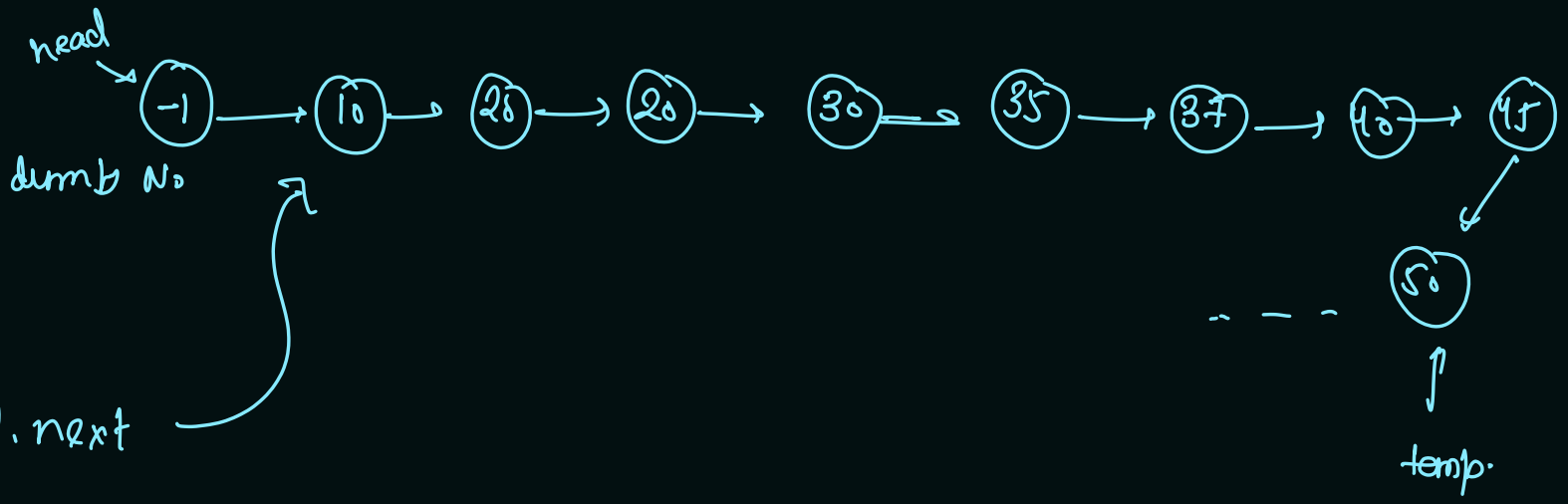






Min-Priority Queue on data.

Class → [Node] ←



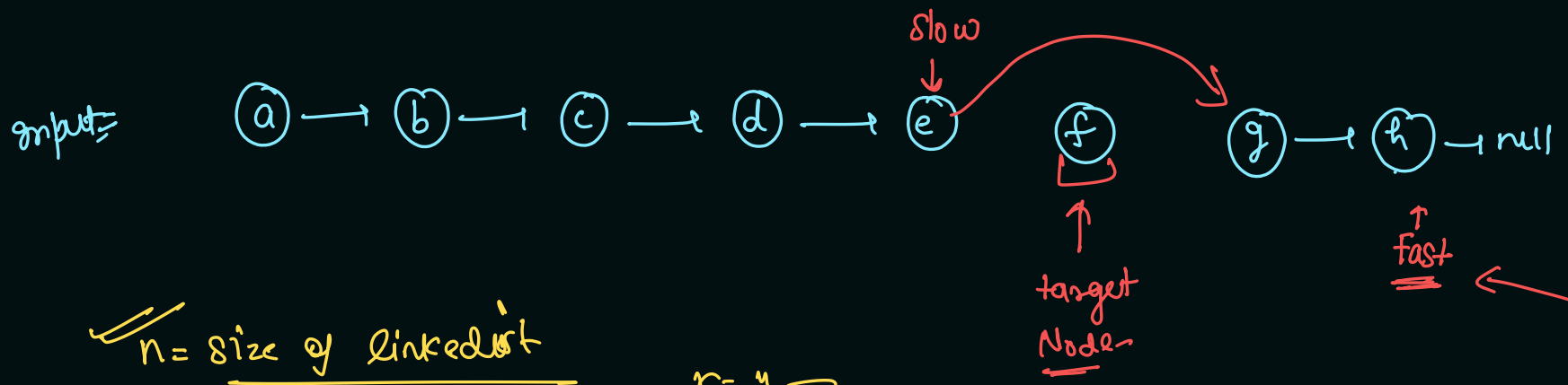
Result → head.next

Remove N^{th} node from end of linkedlist:

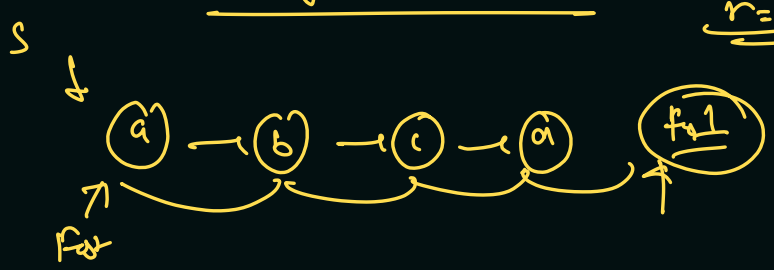
n=3

Iteration
↳ Single iteration.

↳ two iterations one not allowed.



n = size of linkedlist



Difference creation.

fast == null → n = size, Remove head

Return head next!

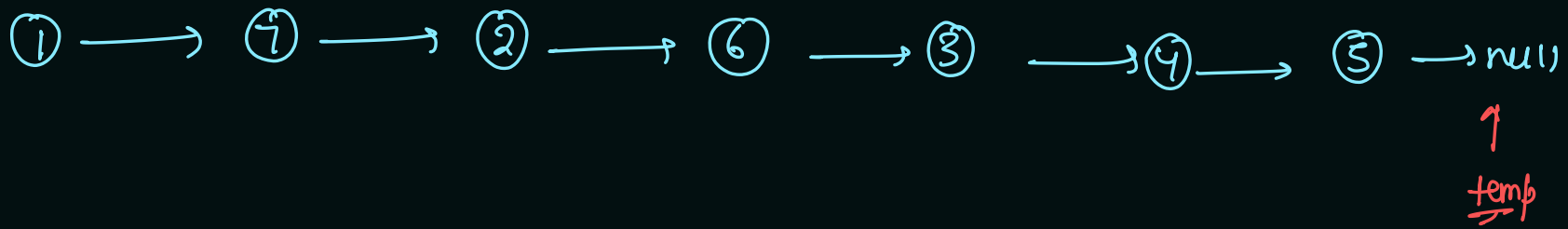
difference b/w fast & slow. = n

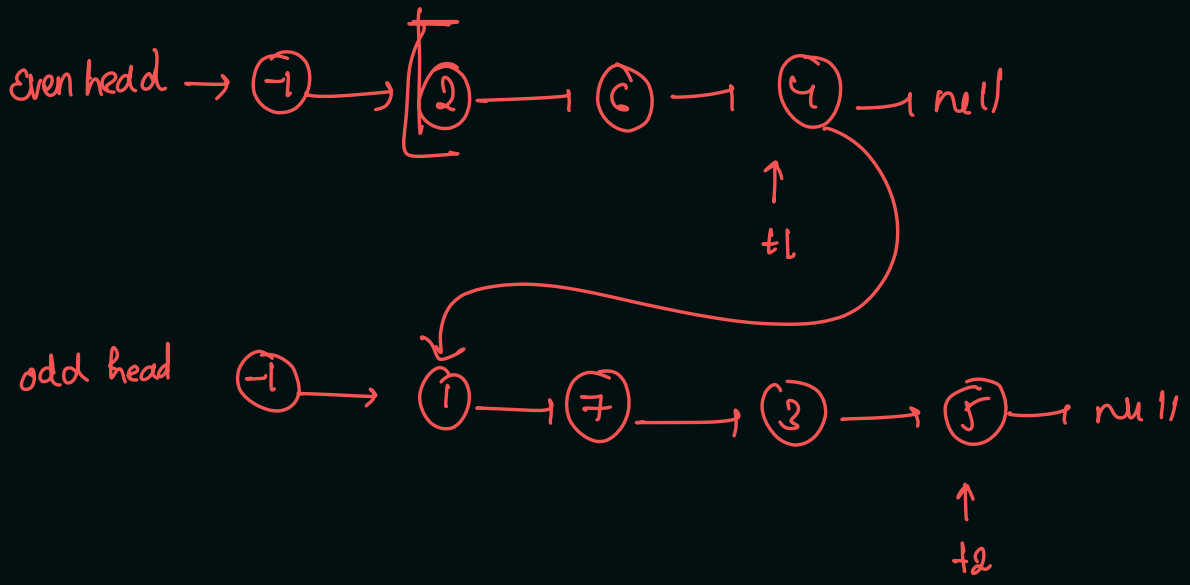
fast = end

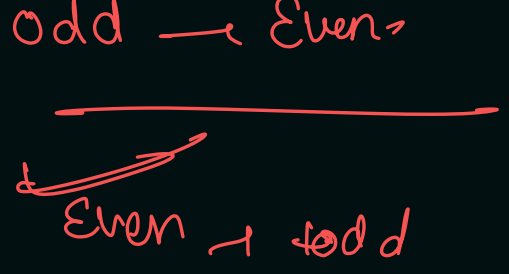
slow = end - n.

slow.next = slow.next.next

Segregate odd Even Nodes:



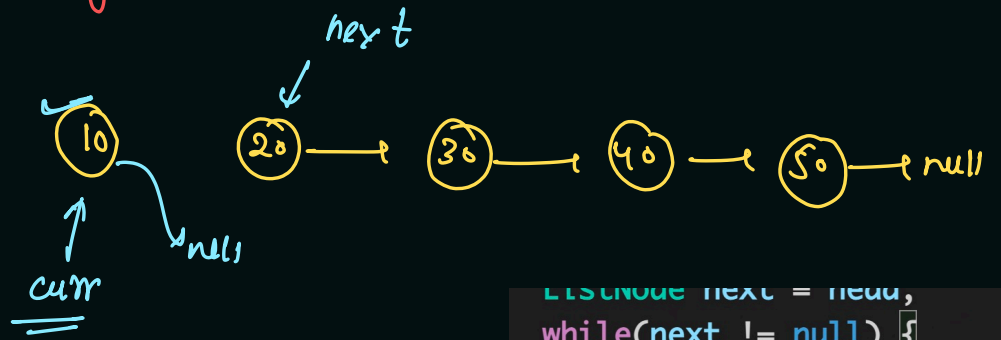




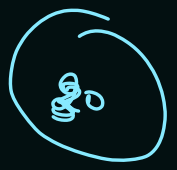
t1.next = ohead.next

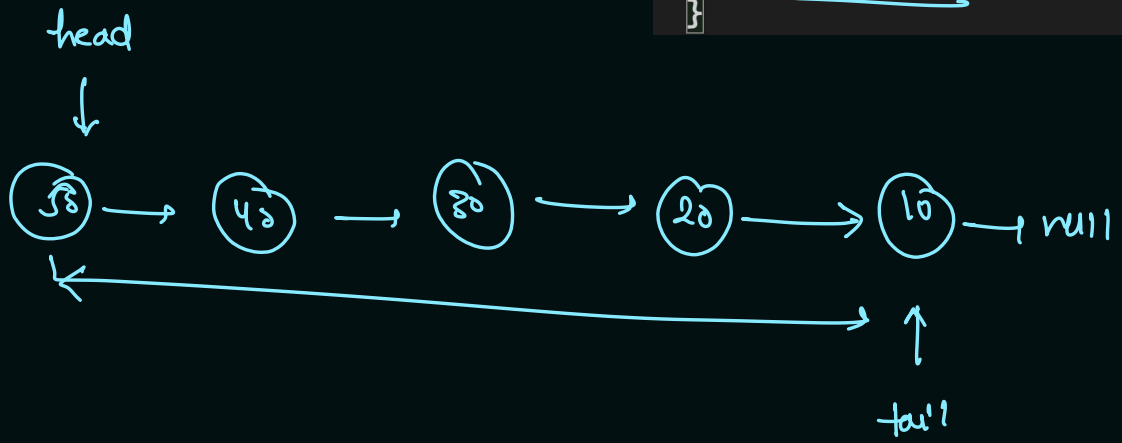
return ehead.next

Reverse using addFirst




```
ListNode next = head;
while(next != null) {
    ListNode curr = next;
    next = next.next;
    curr.next = null;
}
```





head
tail
=

addFirst

```
if (Head == null) {
    head = tail = node;
} else {
    node.next = head;
    head = node
}
```

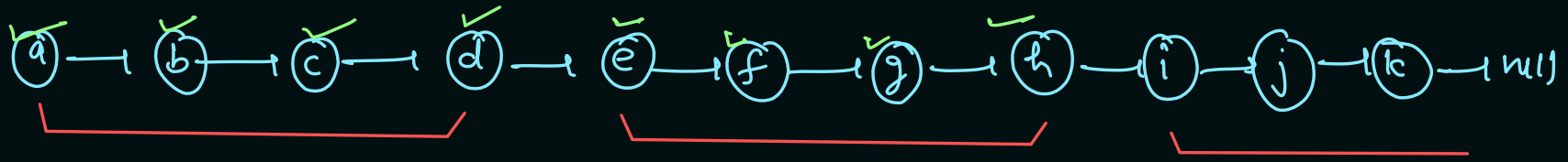
head & tail

temporary head

temporary tail

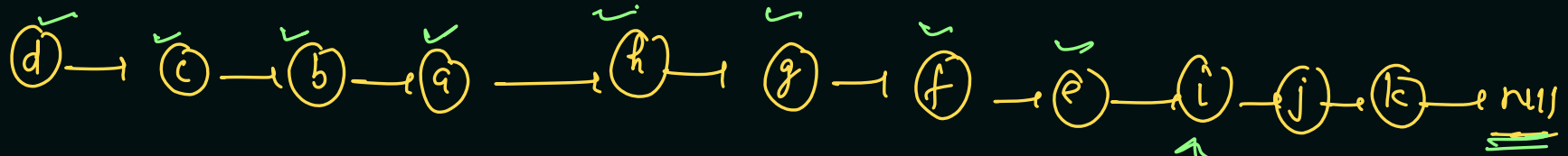
Reverse in k-Groups

k = 4.

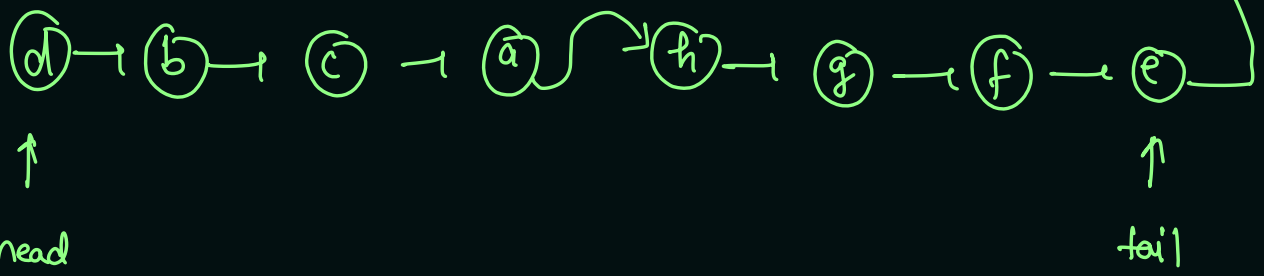


Complete → 4 element. | Complete 4-Ele. | Incomplete.

Reverse if Group is complete ↓ Reverse | ↓ Reverse | ↓ as it is.

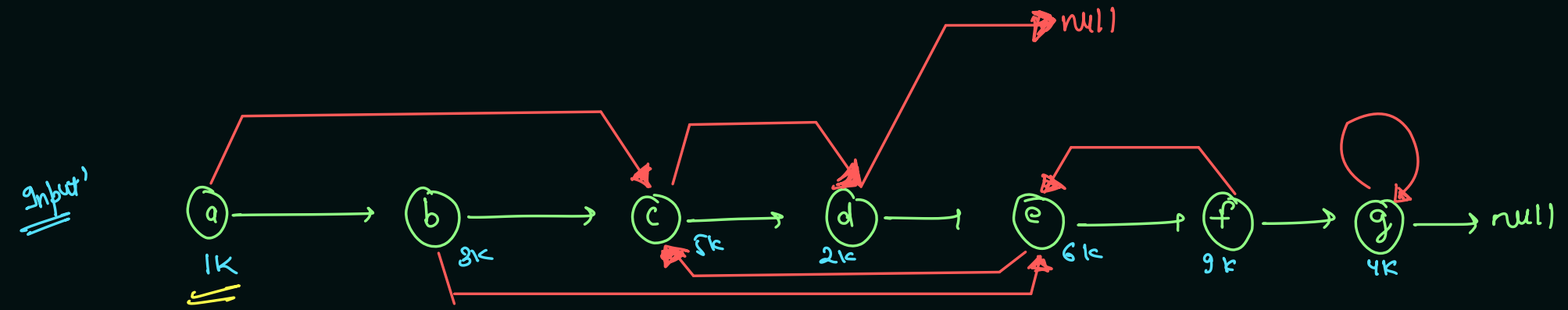


Size := 11

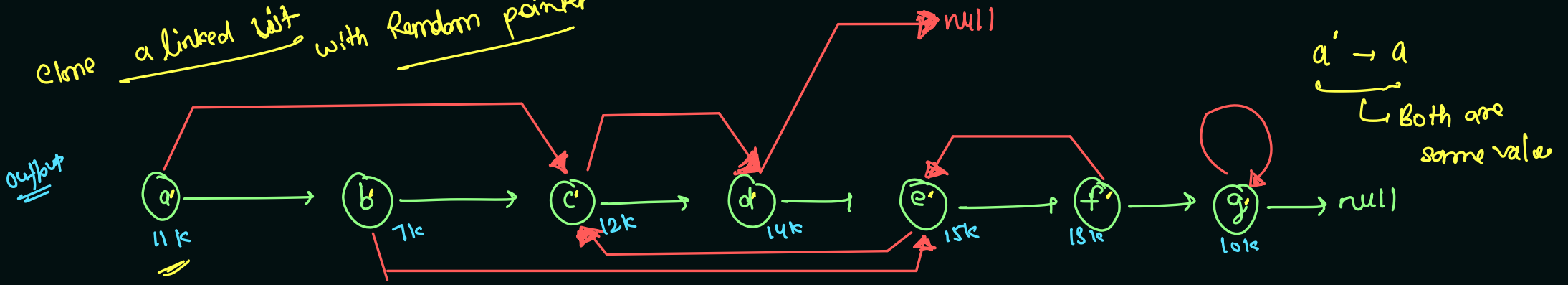


Reverse in Range] =

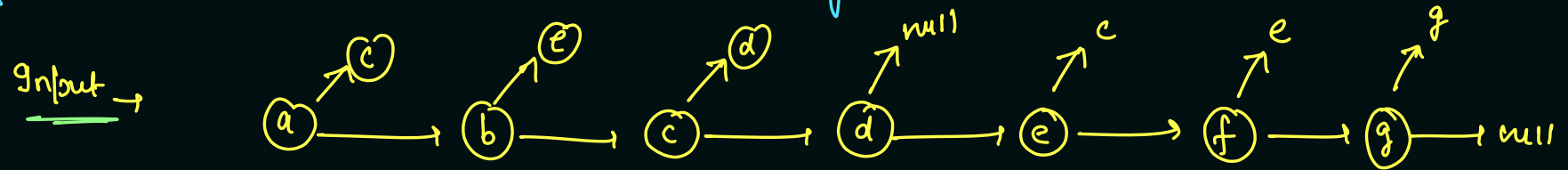
Copy linked list with Random pointer:

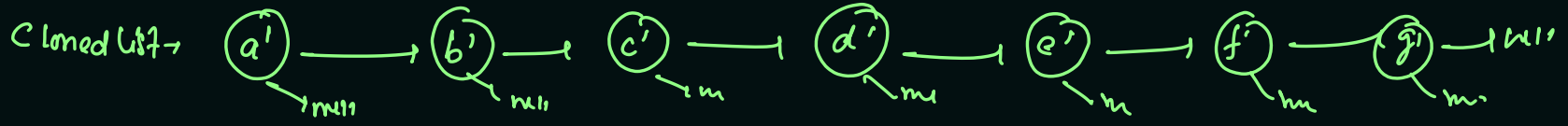


Clone a linked list with Random pointer

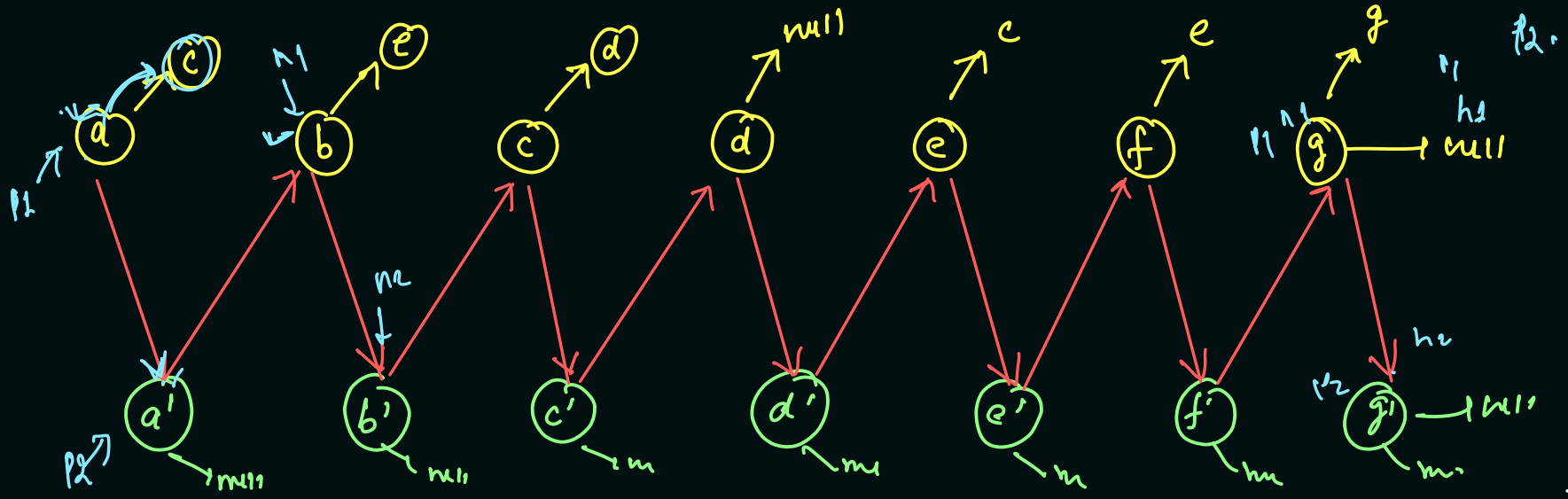


Step (i) → Clone LL with next only.

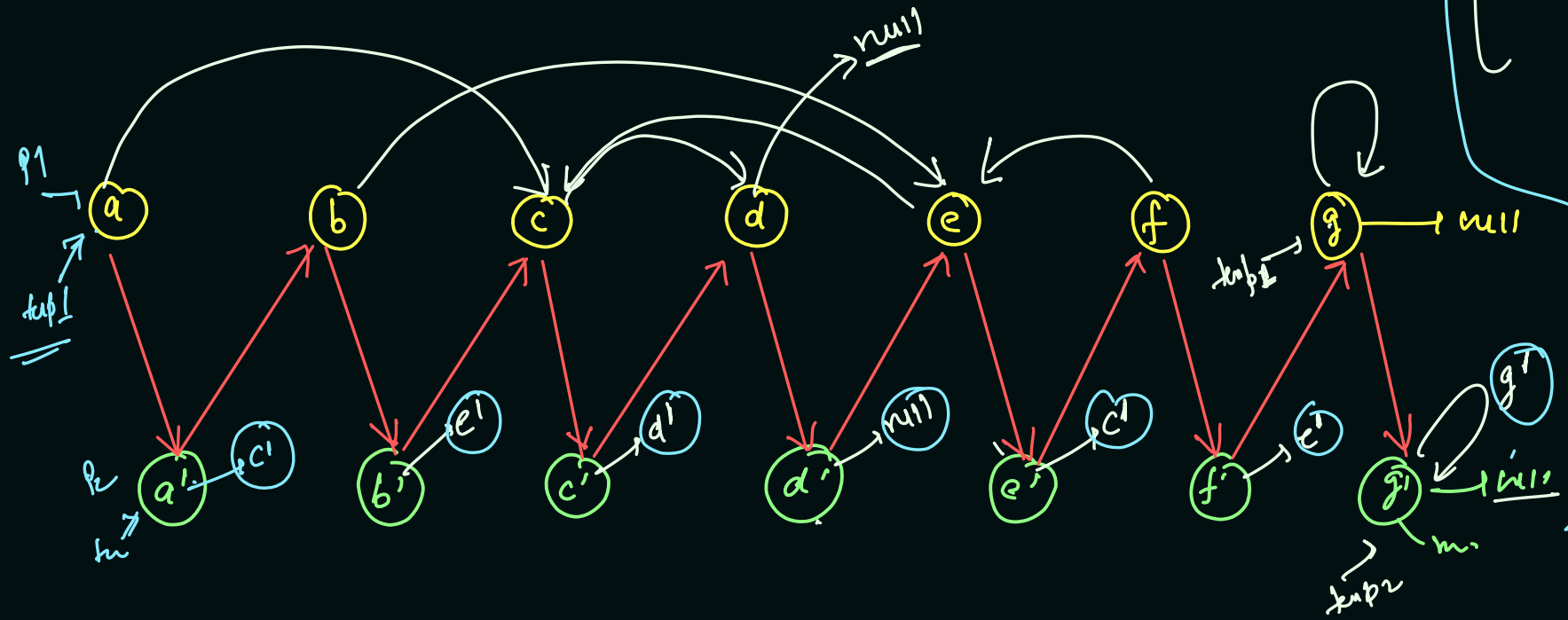




Step2 — connect nodes with zig-zag pattern



Step 3 - connect Random Pointers.



Check: temp1.random = null
temp2.random =

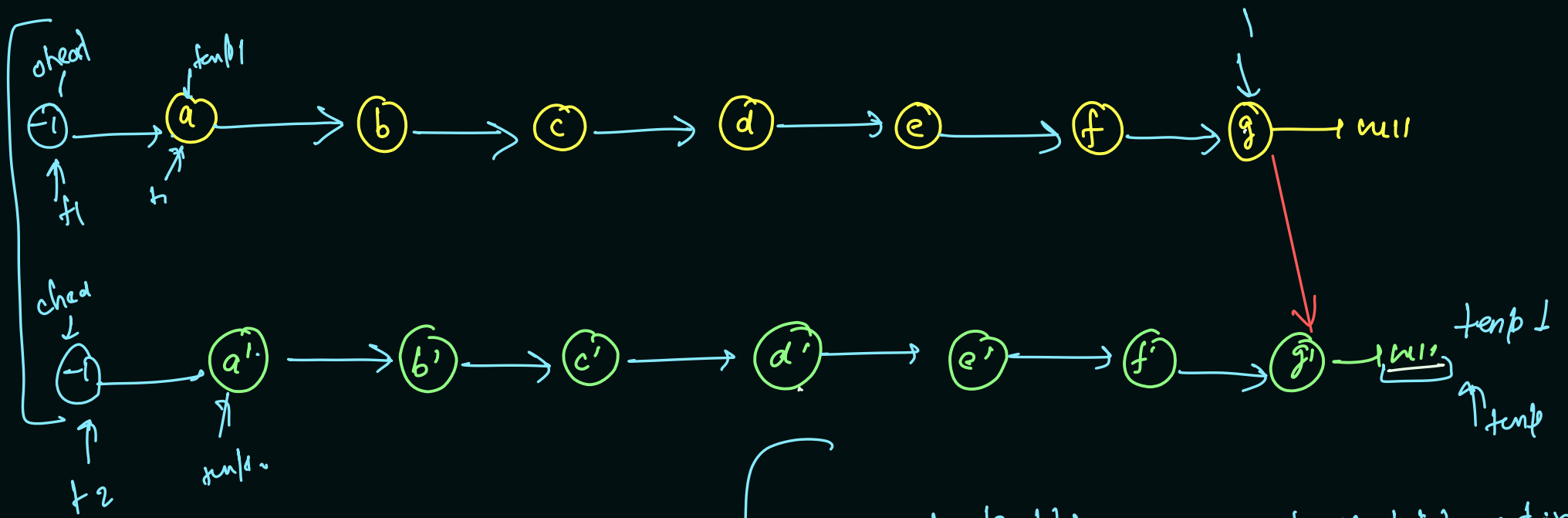
temp2.random = temp1.random.next

temp1 = temp1.next.next

temp2 = temp1.next.next

temp1.next != null &&
temp2.next != null

Step 4 Retain original list →




```
t1.next = temp1;
t1 = temp1;
t2.next = temp2;
t2 = temp2;
```

```
temp1 = temp1.next != null
```

```
temp2 = temp2.next.next
```

```
temp2 = temp2.next == null ?
        null : temp2.next.next
```

End

```
if .next = null
t2.next = null
```

Result = chead.next