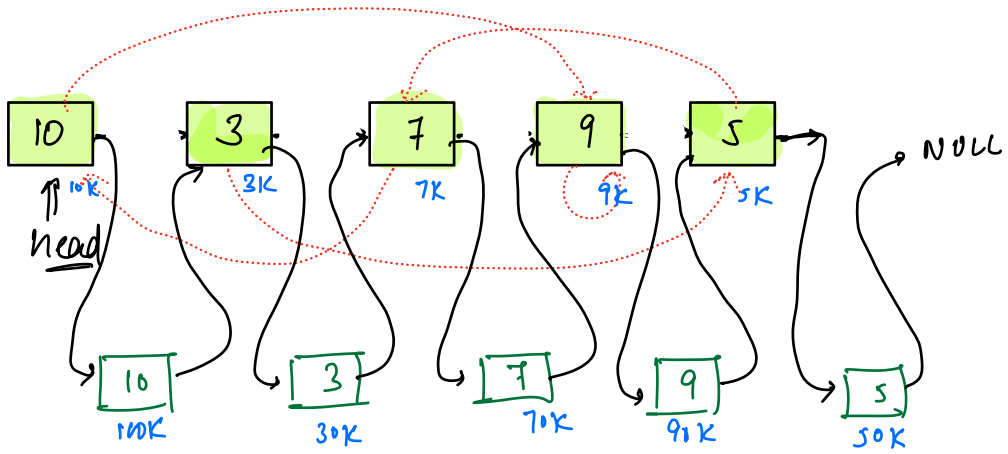
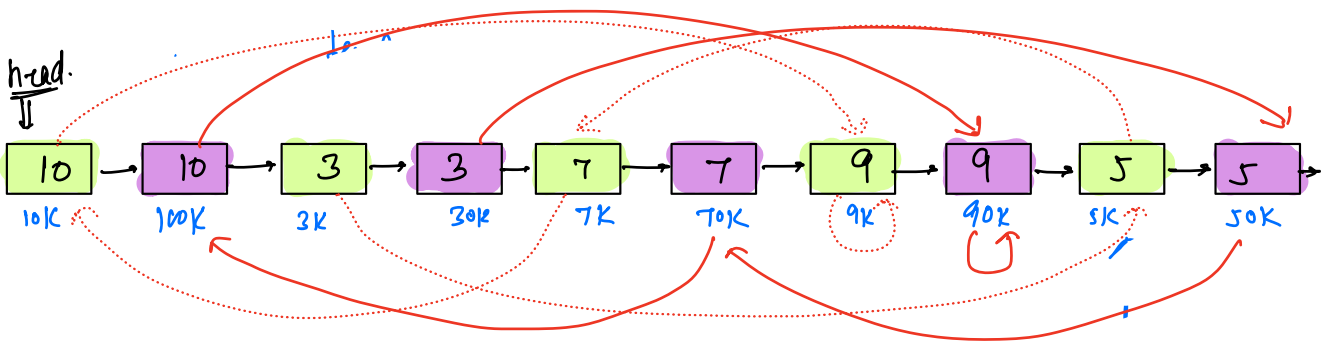
Clone Linked List with Random Pointer





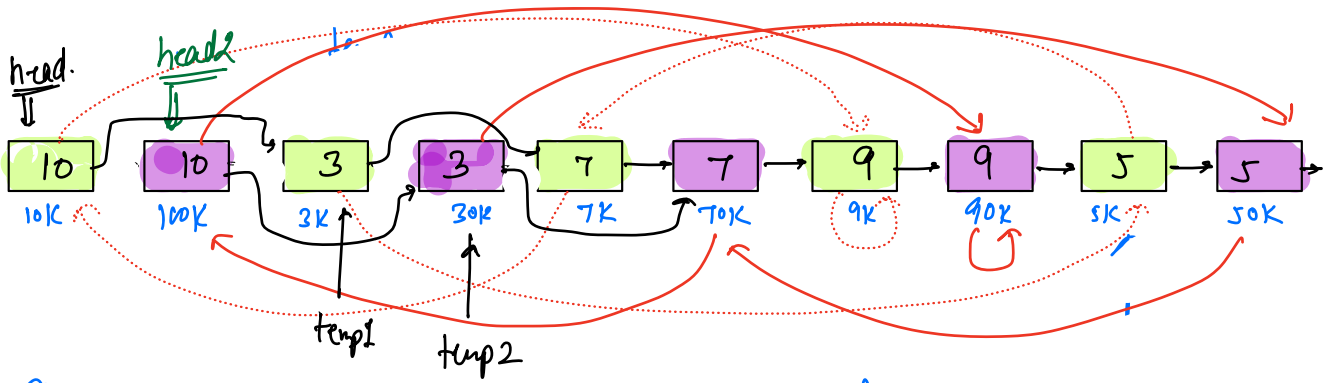
① → Insert cloned nodes in the original linked list.] N.

② Set random of cloned nodes using random of original nodes.

≠ Every cloned node is lying next to the original (old) node

```
while (temp != null){
    temp.next.random = temp.random.next
    temp = temp.next.next
}
```

] N



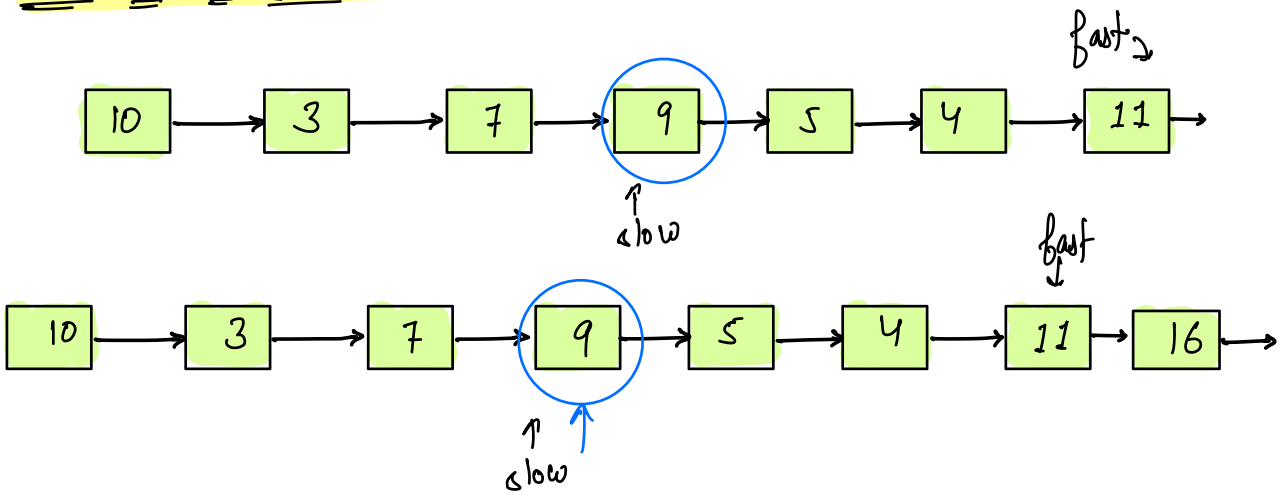
(5) Separate old & new linked list.

```
head2 = head.next ;
temp2 = head2 ;
temp1 = head ;

while (temp1 != NULL && temp2 != NULL) {

    temp1.next = temp1.next.next
    temp2.next = temp2.next.next

    //update temp1 & temp2
    temp1 = temp1.next
    temp2 = temp2.next

}

return head2 ;
```

N.

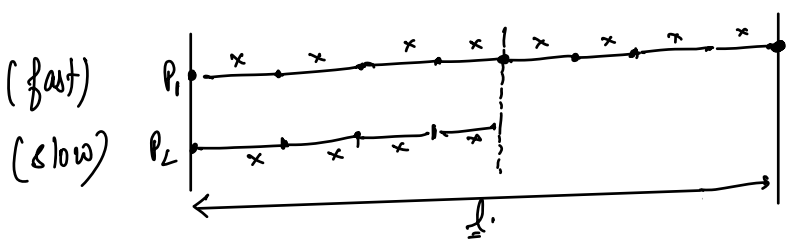
T.C → O(N)
S.C → O(1)

Middle of A Linked List.



idea-1. Find length of LL → l.
ans - element present at $\left(\frac{l-1}{2}\right)^{th}$ idx.

[T.C → O(N)
S.C → O(1)]



speed of P_1 = 2 · speed of P_2

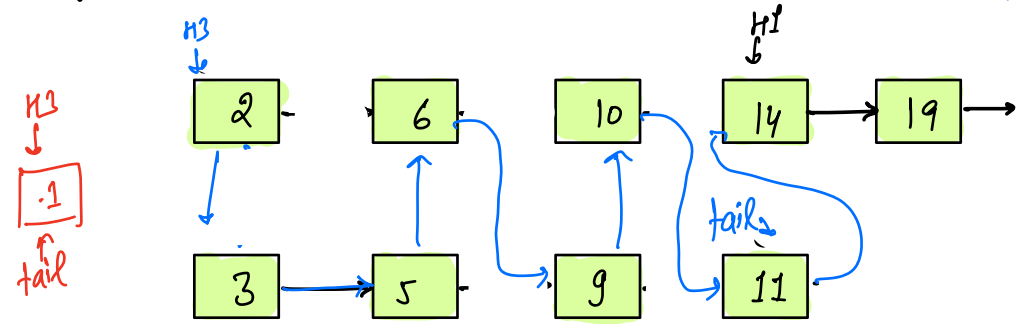
```
slow = fast = head
while (fast.next != NULL && fast.next.next != NULL) {
        slow = slow.next
        fast = fast.next.next
}
return slow;
```

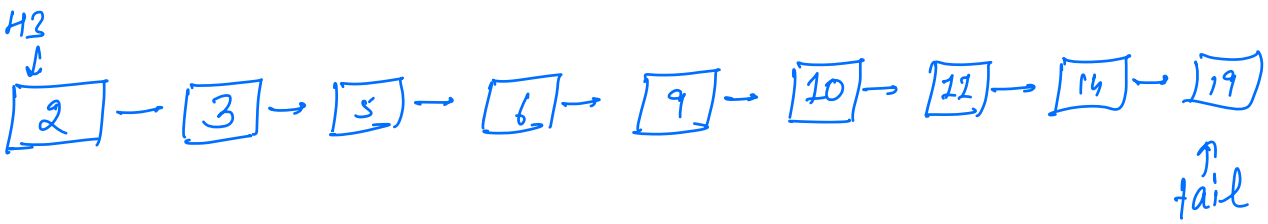
odd

even

[T.C → O(N)
S.C → O(1)]

Merge Two Sorted Linked-lists





pseudo-code.

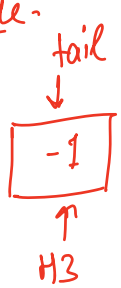
```
Node H3 = null
Node tail = null.

// Node dummy = new Node(-1)
// H3 = tail = dummy;

if (H1 == null) { return H2 }
if (H2 == null) { return H1 }

// if (H1.val ≤ H2.val) { H3 = H1, tail = H1, H1 = H1.next }
// else { H3 = H2, tail = H2, H2 = H2.next } // x

while (H1 != NULL && H2 != NULL) {
    if (H1.val ≤ H2.val) {
        tail.next = H1
        H1 = H1.next
        tail = tail.next
    } else {
        tail.next = H2
        H2 = H2.next
        tail = tail.next
    }
}
```

```
if ( H2 == NULL){
{
        tail.next = H1
}
if ( H1 == NULL){
{
        tail.next = H2
}
return H3;  // return H3.next
```

total no. of nodes.

T.C → O(N)
S.C → O(1)

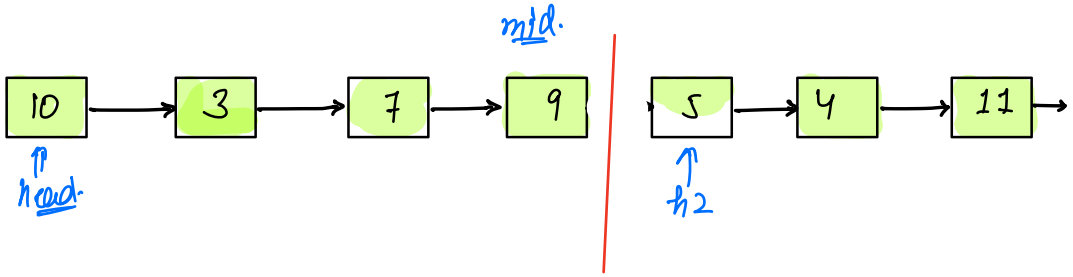
dummy node.



[Break. → 10:40 → 10:47]

Merge Sort a linked list →

//Merge Sort.




```
Node mergeSortLL(Node head){
    if(head == null || head.next == null){return head;}
    Node mid = getMiddle(head);
    Node h2 = mid.next;
    mid.next = null
    head = mergeSortLL(head)
    h2 = mergeSortLL(h2)
    return mergeSortedLL(head, h2)
}
```

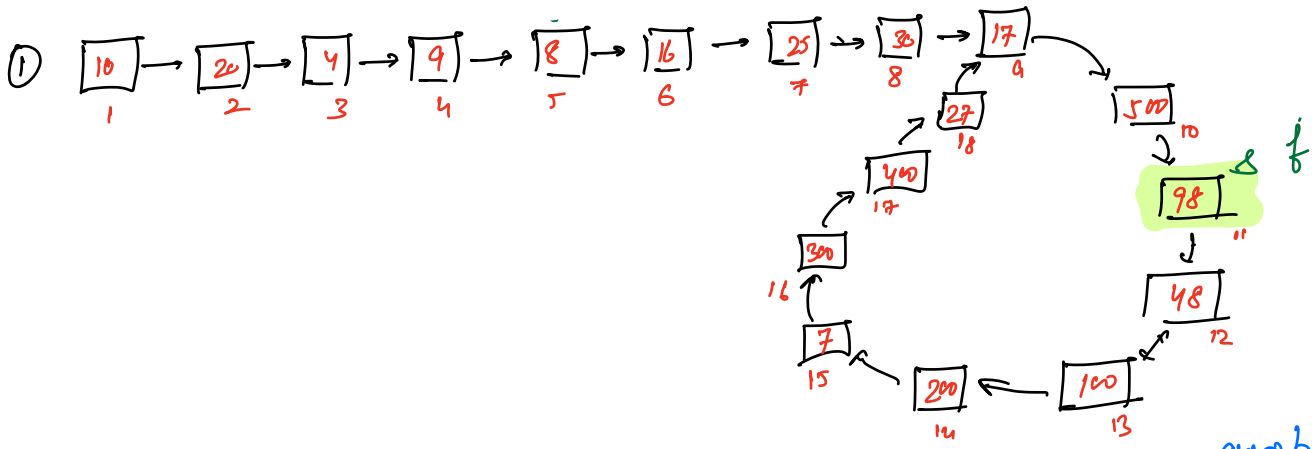
[# todo → dry run this code]

[T.C → O(N log N)
  S.C → O(log N)]

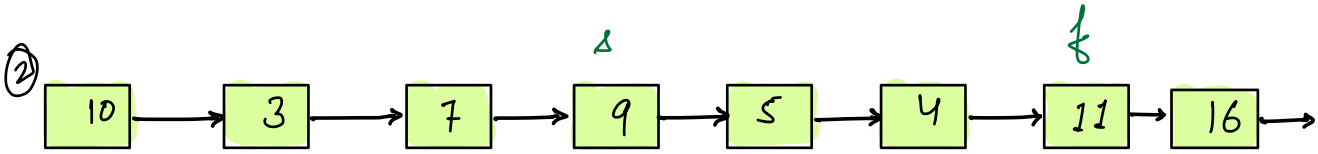
↓

Recursive stack

Check if there is a loop

① 

ans → true.

② 

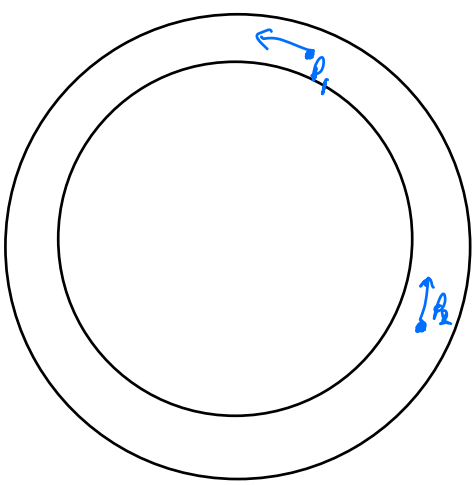
ans → false.

idea-1. → Use HashSet / HashMap to store the reference of all nodes.

If reference of any node is already stored, return true;

otherwise. temp will become NULL eventually.

[T.C → O(N), S.C → O(N)]



[speed of P_1 ≠ speed of P_2]

⇓

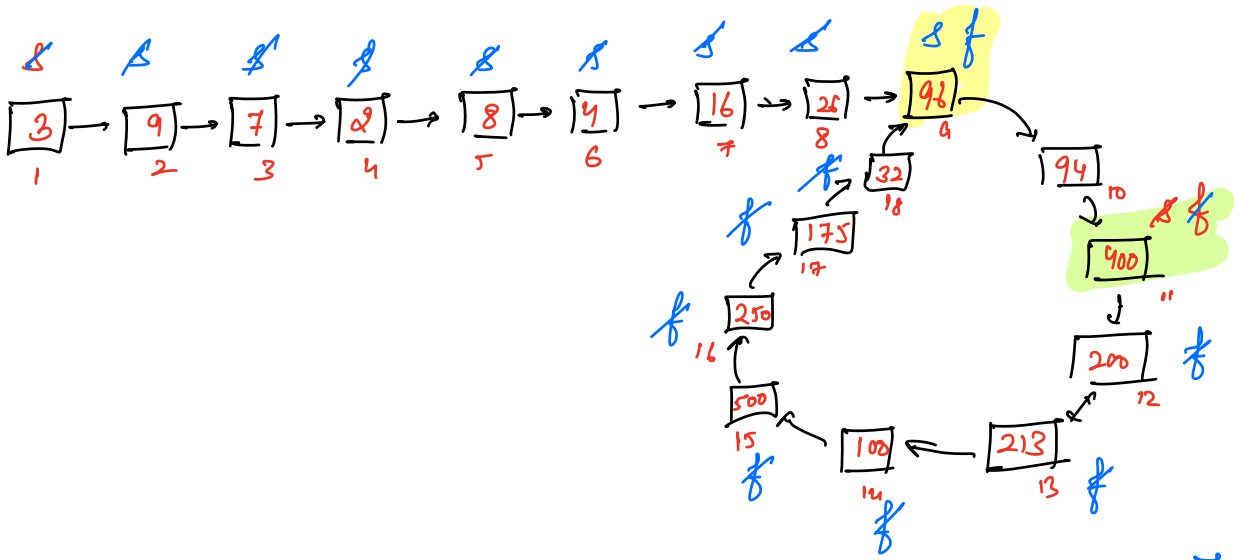
they will meet at one point definetly.

pseudo-code

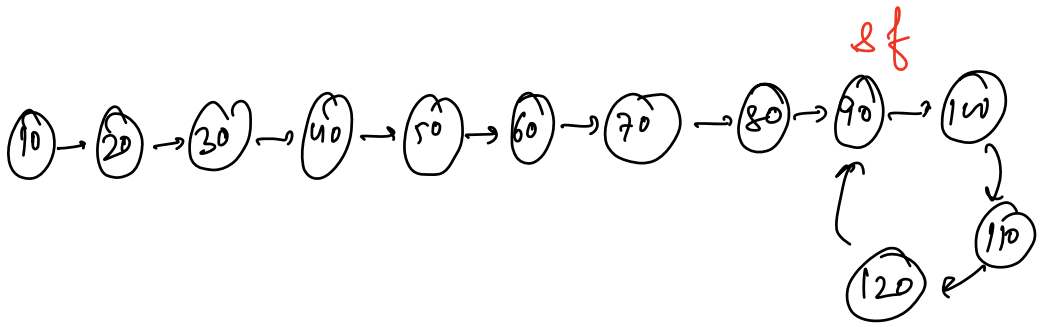
```
slow = fast = head
while ( fast.next != NULL && fast.next.next != NULL){
        slow = slow.next
        fast = fast.next.next
        if (slow == fast){
                return true
        }
}
return false;
```

T.C → O(N)
S.C → O(1)

Find the start point of the loop

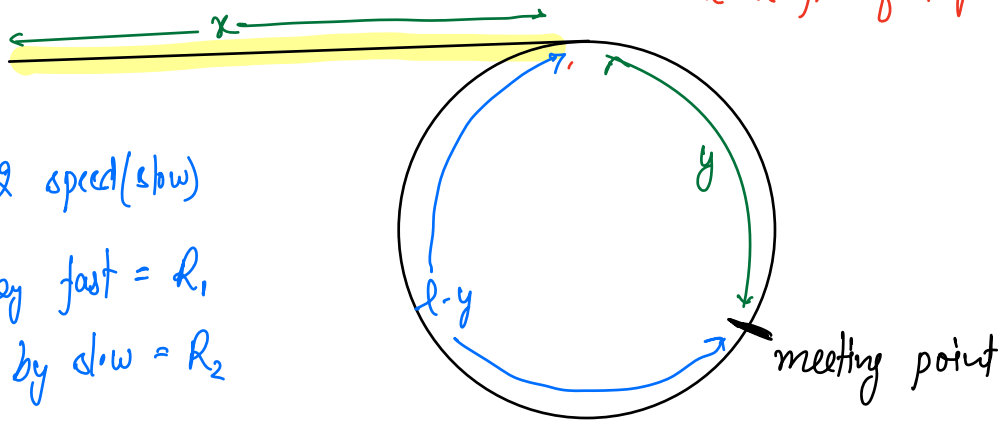


- Reset slow pointer to head.
- Move slow & fast pointer simultaneously by 1 step.
- Their meeting point ⇒ start point of the loop.



proof →

l = length of loop



speed(fast) = 2 speed(slow)

no of rotations by fast = R_1

no of rotations by slow = R_2

distance covered by fast = $x + (R_1 \cdot l) + y$.

distance covered by slow = $x + (R_2 \cdot l) + y$.

distance covered by fast = 2 * distance covered by slow

$$x + (R_1 \cdot l) + y = 2 * (x + R_2 \cdot l + y)$$

$$x + y + R_1 \cdot l = 2x + 2y + 2R_2 \cdot l$$

$$R_1 l - 2R_2 l = x + y$$

$$\underbrace{(R_1 - 2R_2)}_{R_3} \cdot l = x + y$$

$$\boxed{R_3 \cdot l - y = x}$$

$$\Rightarrow \boxed{x = R_3 l - y}$$

(R_3: constant.)

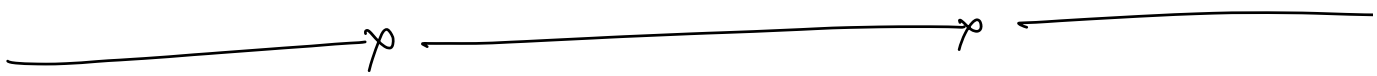
$R_3 = 1$. $x = l - y$.

$R_3 = 2$. $x = 2l - y$

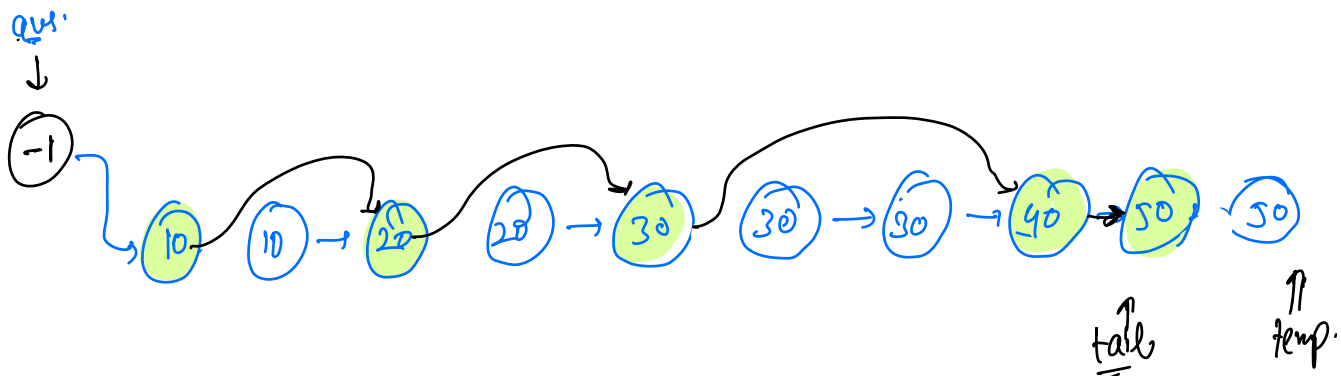
$x = l + (l - y)$

$R_3 = 3$. $x = 3l - y$

$x = 2t + (1-y)$



- check if a loop is there.
- find s.p of loop
  - proof.




```
while (temp != null) {
    if (tail.val != temp.val) {
        tail.next = temp., tail = tail.next
    }
    temp = temp.next;
}
tail.next = null
return ans.next;
```