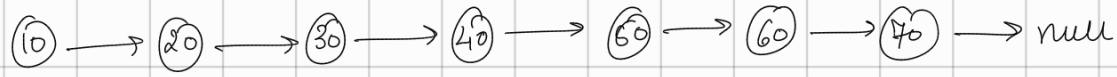
→ Floyd's cycle detection
- Cycle detection
- Starting pt. of cycle
- Mathematical Proof

→ Insertion Node

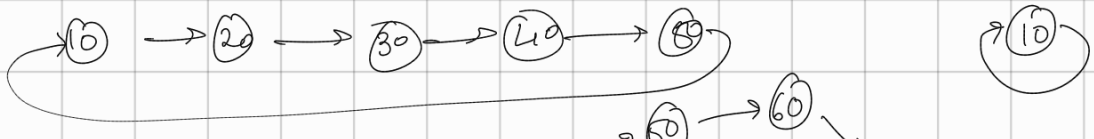
→ Clone LL
→ with extra space
→ w/o extra space

① FLOYD's CYCLE DETECTION

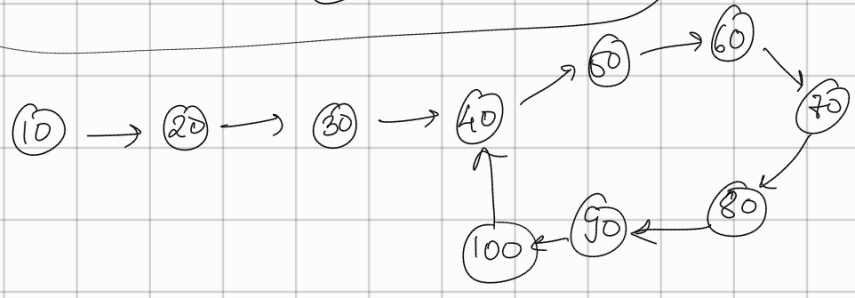
Case 1
SLL



Case 2
CSLL



Case 3
SLL + Tail
+
loop



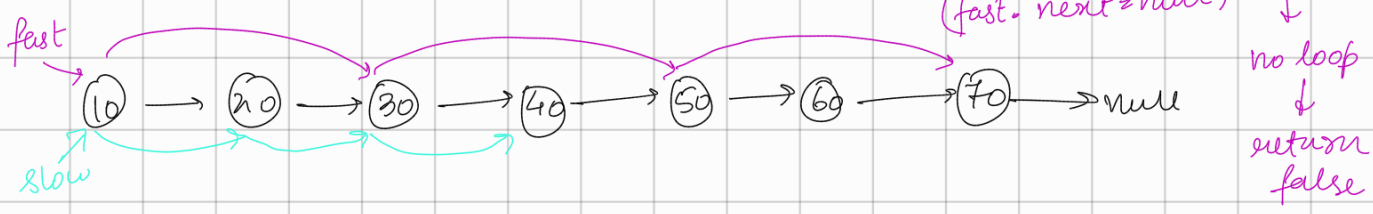
In a cycle, tail = prev node of head

→ Cannot use single pointer, because it will get stuck in an infinite loop.

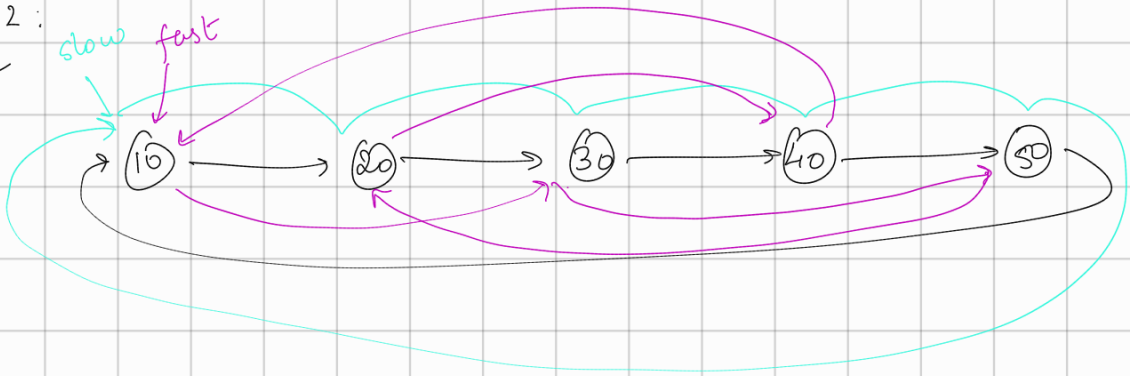
→ Can store all Node references in a map & on revisiting any we detect a valid cycle TC: O(N), SC: O(N)

Two Pointer Approach:

Case 1:
SLL

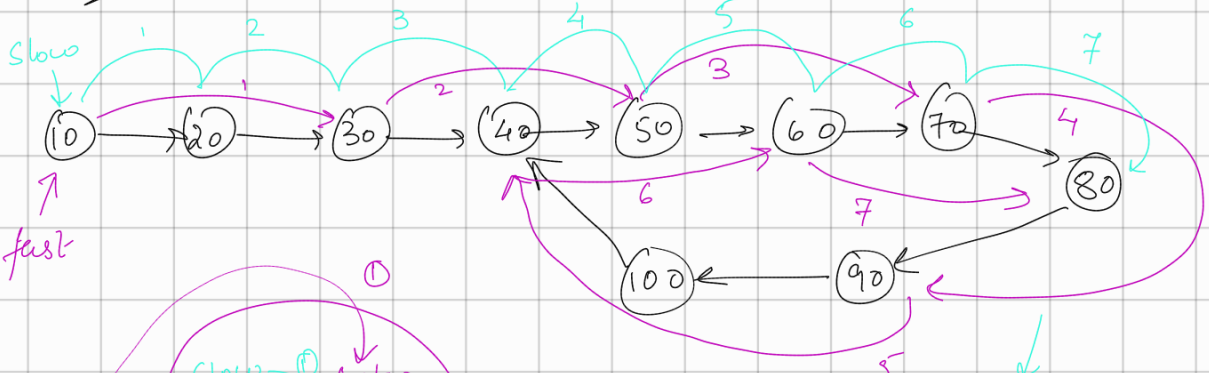


Case 2:
CSLL

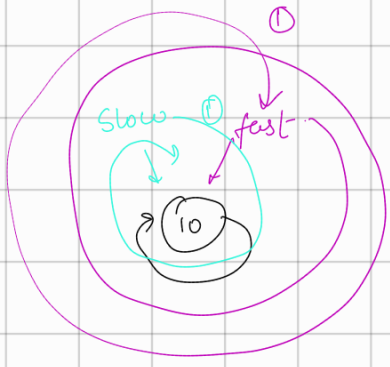


at 10 → slow == fast, return true

Case 3:
Tail
+Loop



Corner Case



→ slow == fast, return true

CODE:

```
if ( head == null || head.next == null)
    return false;
if ( head.next == head)   ← Self referential node
    return true;

ListNode slow = head, fast = head;
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    if ( fast == slow) return true;   ← Cycle detected
}
return false;   ← cycle not detected
```

Cannot use while (fast != slow) because fast = slow = head initially

TC: O(N) ← Not exactly, lower than that in case of cycle.
↳ not higher because fast moves at twice speed

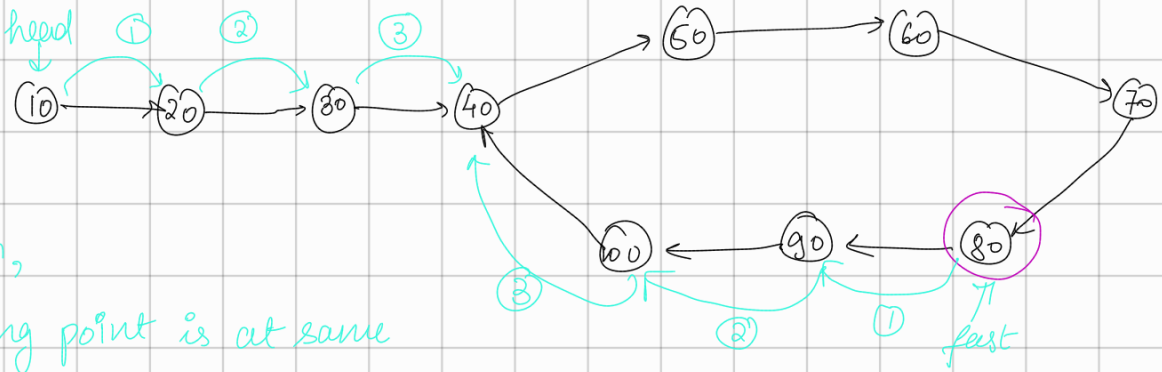
⭐ To check if Complete LL is circular, The same above code but also check if (fast == slow && slow == head)

② Linked list Cycle II — Leet Code Starting Node Of Loop

Case 1: return null

Case 2: return head

Case 3:



Observation,
loop starting point is at same distance from head as it is from meeting point of slow & fast in case of cycle detection.

Jumps from head to loop starting pt (3) = Jumps from fast to loop starting pt (3)

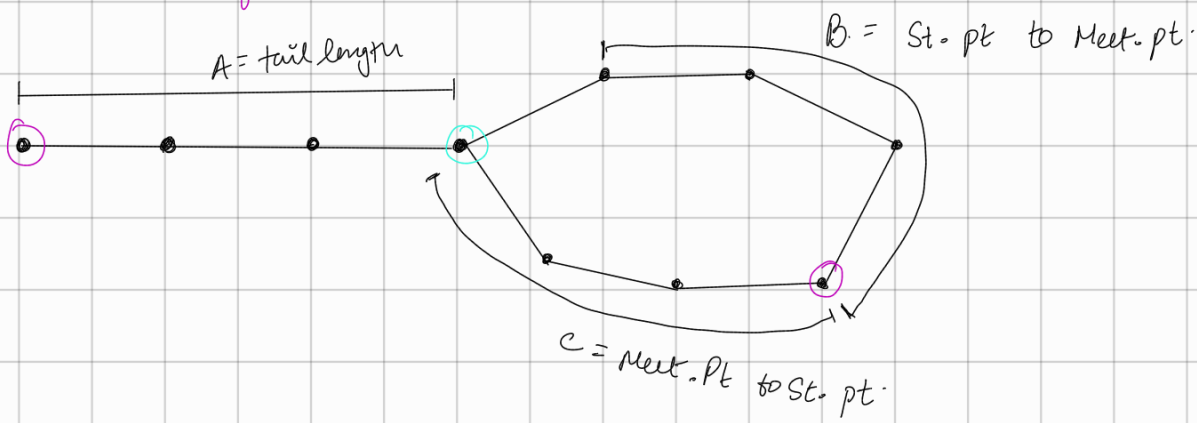
∴ Code:

```
List Node slow = head, fast = head;
while (fast != null && fast.next != null){
    slow = slow.next;
    fast = fast.next.next;
```

```
        if( slow == fast) break;          ← Cycle detected
    }
    if( fast == null || fast.next == null)   ← Cycle not detected
        return null;
    if (slow == head)                     ← if CompleteLL is a cycle
        return head;
    while ( head != slow ) {              ← Jumps simultaneously
        head = head.next;                   from head & slow
        slow = slow.next;
    }
    return slow;                          ← Found loop starting pt.
```

TC: $O(N)$, SC: $O(N)$

Mathematical Proof:



slow ⇒ 1 node/itr = S_s

fast ⇒ 2 node/itr = S_f

Time/itr taken by slow = Time/itr taken by fast ⇒ $\frac{d_{slow}}{speed_{slow}} = \frac{d_{fast}}{speed_{fast}}$

Total length of loop = B+C

Tail length = A

$d_{slow} = A + B$, $d_{fast} = A + B + k(B+C)$

i = no. of loops travelled by slow

j = no. of loops travelled by fast

Let $k = j - i$

k = no. of complete loop iterations travelled by fast − no. of complete loop iterations travelled by slow

$\therefore i = 0$, $k = j$

$= 0$ when $S_s = 1$ & $S_f = 2$

$$\frac{A + (B+C)i + B}{S_s} = \frac{A + (B+C)j + B}{S_f}$$

$$\frac{S_f}{S_s}\left[A + (B+C)i + B\right] = A + (B+C)j + B$$

$\frac{S_f}{S_s}$ → relative speed of fast wrt slow $= r$

$$\therefore rA + r(B+C)i + rB = A + (B+C)j + B$$

$$rA + rB - A - B = (B+C)(j - ri)$$

$$(r-1)A + (r-1)B = (B+C)(j - ri)$$

$$(r-1)(A+B) = (B+C)(j - ri)$$

$$\therefore (A+B) = \frac{(B+C)(j - ri)}{(r-1)} = \text{const}$$

$$\therefore A = \{(B+C)(\text{const})\} - B$$

$$\therefore A = (\text{Const} \times \text{Complete cycle distance}) - B$$

$\therefore$ Distance between Head & loop starting point = Distance between meeting point & loop starting point

or

If $d_{slow} = A + B$ for $S_s = 1$

& $d_{fast} = A + B + k(B+C)$ for $S_f = 2$

$$\Rightarrow \frac{A+B}{1} = \frac{A + B + k(B+C)}{2}$$

$$\Rightarrow 2A + 2B = A + B + k(B+C)$$

$$\Rightarrow A + B = k(B+C)$$

$$\therefore A = k(B+C) - B$$

k → const = 1 in case $S_s = 1$, $S_f = 2$

$(B+C)$ → Complete cycle Distance

for any K $\quad A = (B+C)k - B$ $\quad$ let $k = k'+1$

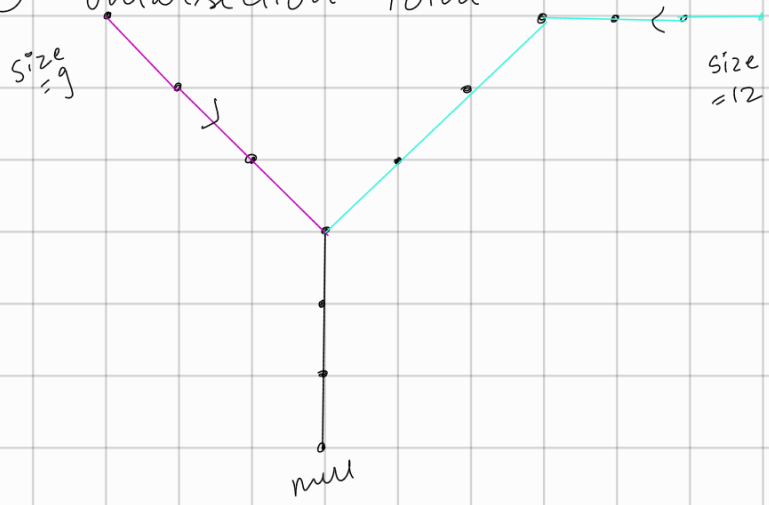
$$\therefore A = (B+C)k' + B + C - B$$

$$\therefore A = (B+C)k' + C$$ → Same conclusion

③ Find Length of Loop.

On detecting loop, just traverse loop completely & when reaching on the same point again return count of jumps +1

④ Intersection Point.



Corner case 1
No intersection

Corner case 2
Same head.

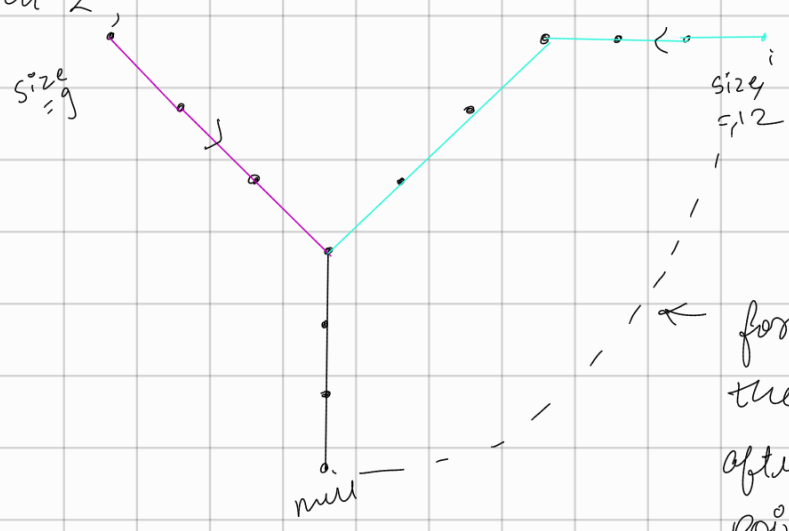
Approach 1
Traverse longer list upto difference between size1 & size2 the increment both list ptrs, when ptr1 == ptr2 intersection found.

3 Traversals <— size1, size2 → diff shift & common point

CODE 1:

```
int sizeA = getSize(headA), sizeB = getSize(headB);
if (sizeA > sizeB)
    for(int i=0; i< sizeA - sizeB; i++)
        headA = headA.next;
else
    for (int i=0; i< sizeB - sizeA; i++)
        headB = headB.next;
while (headA != headB){
    headA = headA.next;
    headB = headB.next;
}
return headA;
```

Approach 2



form a cycle & the use cycle detection after that use starting point logic.