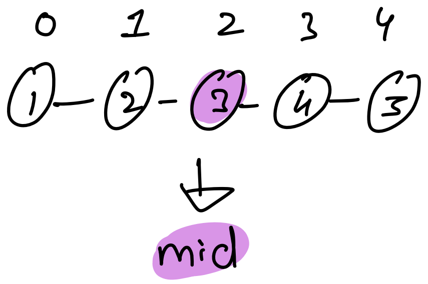
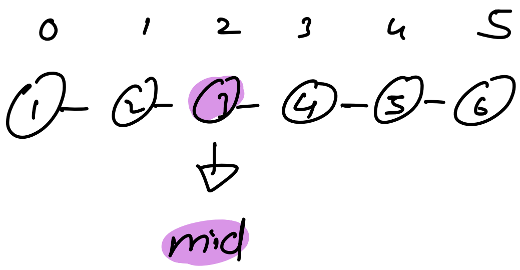
Q1 Given a linked list, find the middle element of linked list.

Ex1



Ex2



Approach 1 :

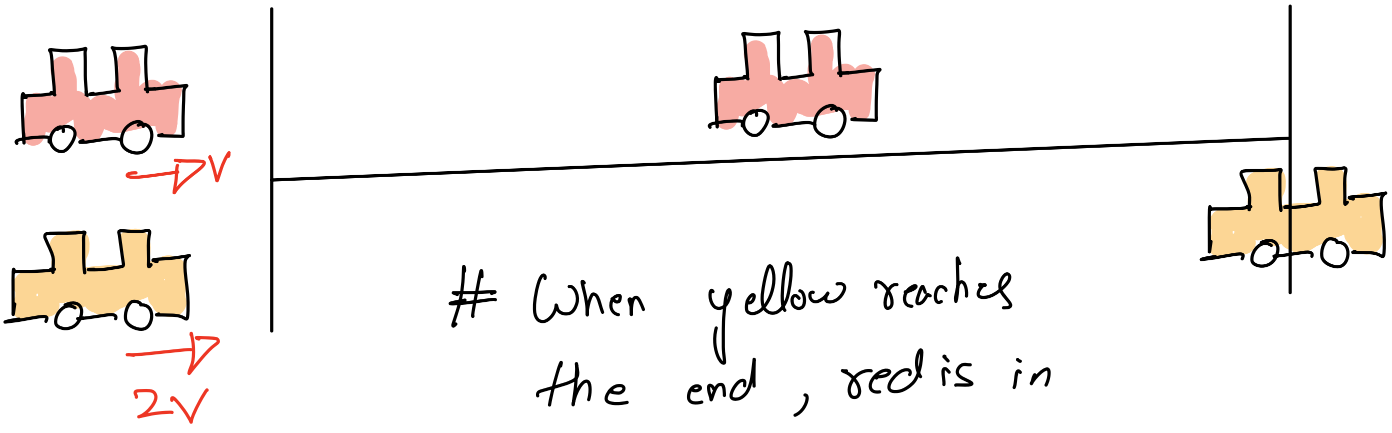
1) Find length after a traversal

2) Traverse again to find middle.

$$TC : O(n + n/2) = O(n)$$

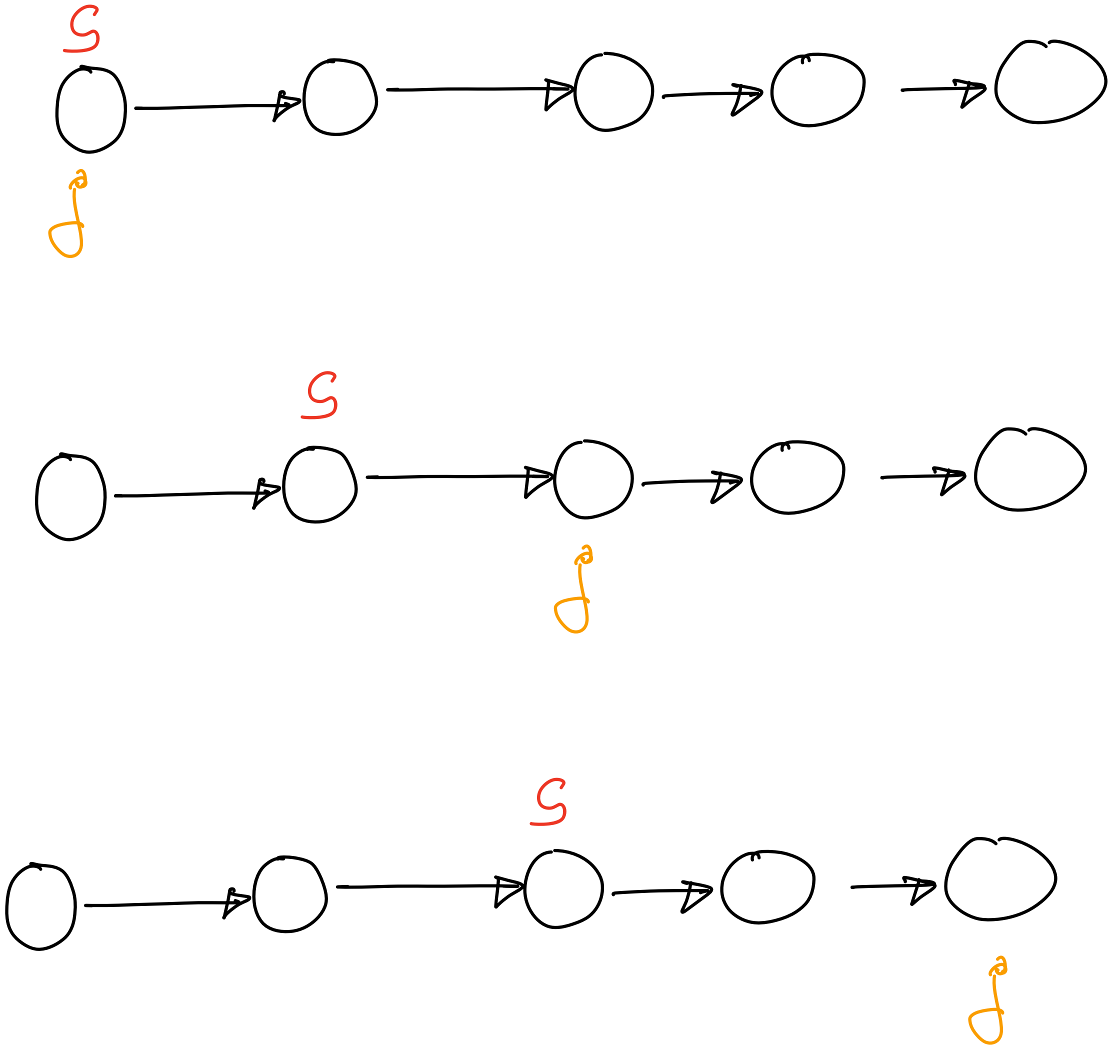
$$SC : O(1)$$

Approach 2 : Using Fast & Slow pointers



When yellow reaches the end, red is in middle.

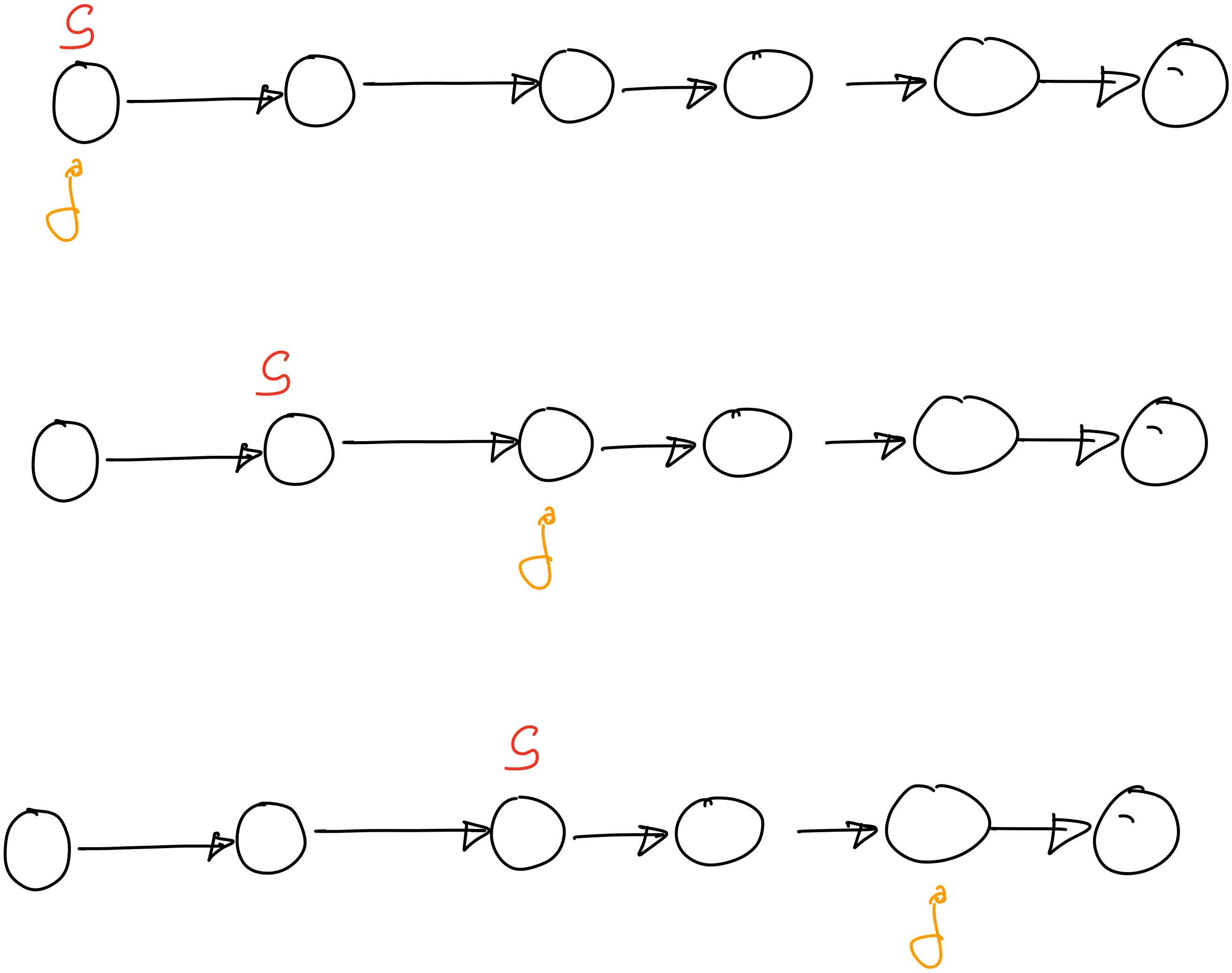
Odd length



Exit condition

f.next == null

Even length



Exit Condition

f.next.next == null

Pseudo Code !

```
if (head == null)
        return null

Node slow = head;
Node fast = head;

while (fast.next != null && fast.next.next != null){
    fast = fast.next.next;
    slow = slow.next;
}

return slow;
```

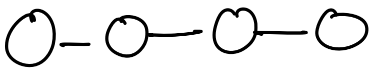
Tc : O(n)

Sc : O(1)

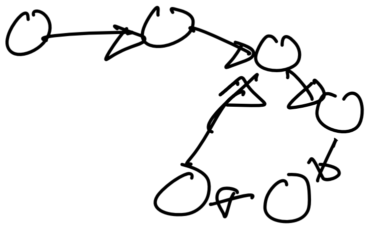
Q_2 Find if a cycle exists in a linked list.

Ex2

Ex1 :



false



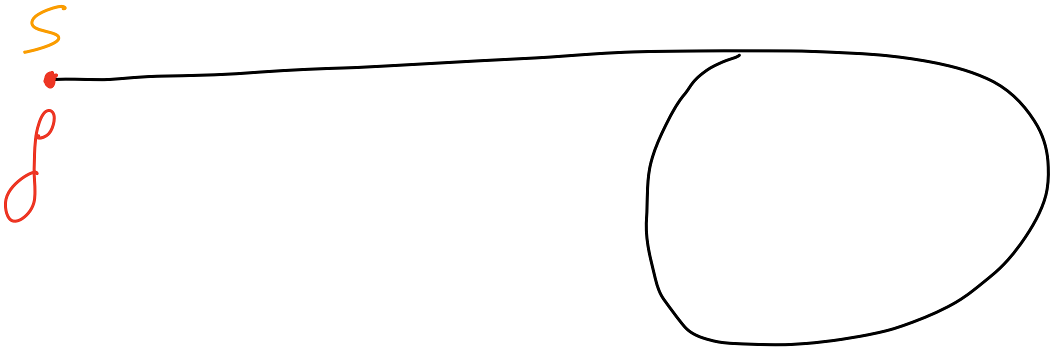
True.

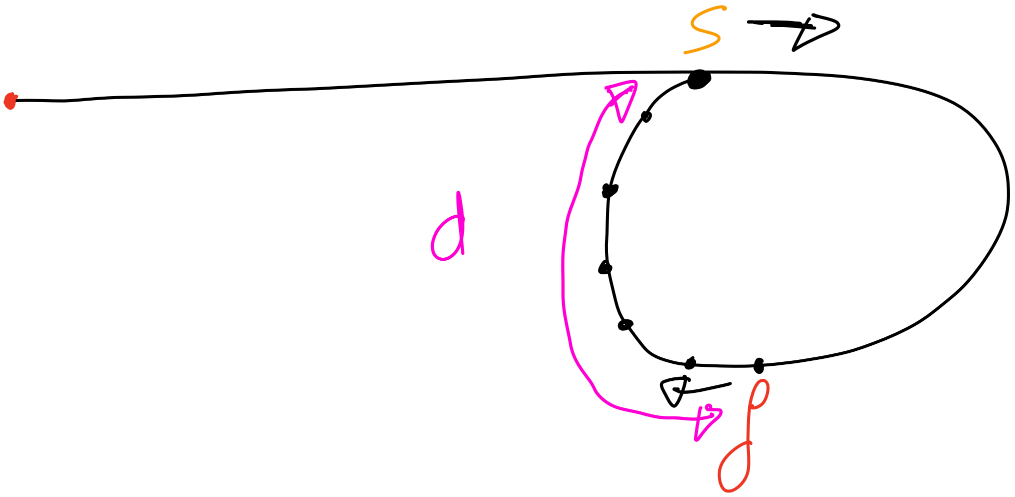
Approach 1: Traverse and store each Node in a hashset. If you reach the some node again. This means a cycle is present.

TC: O(n)

SC: O(n)

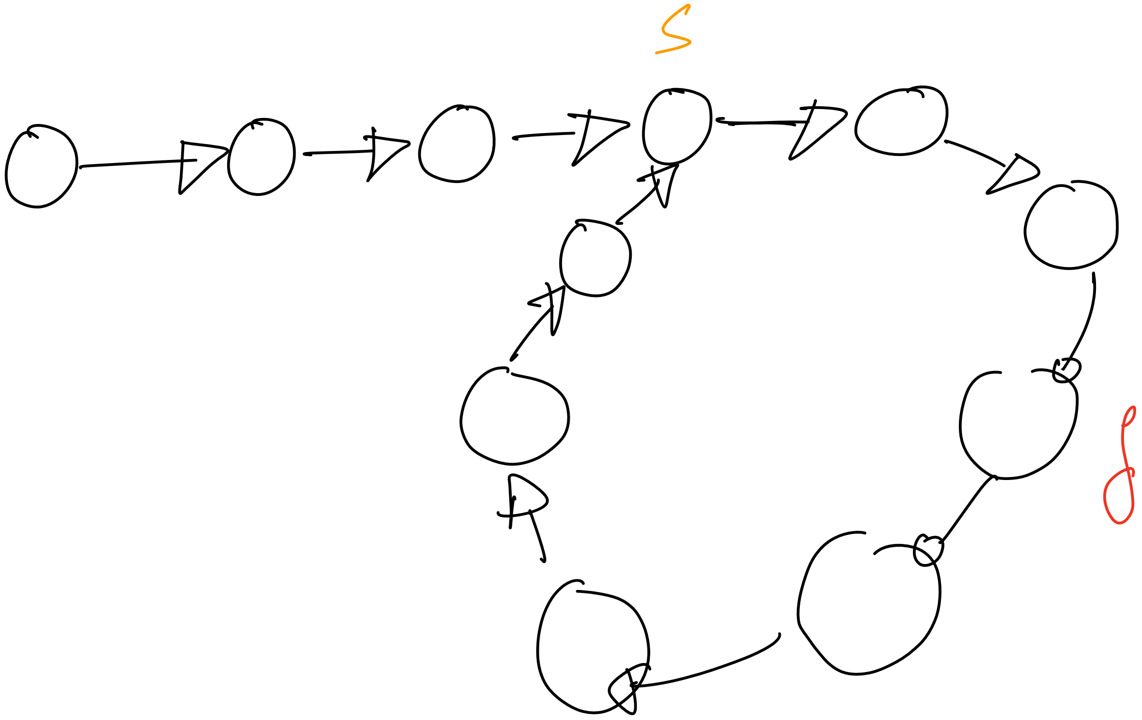
Approach 2 : Using fast & slow

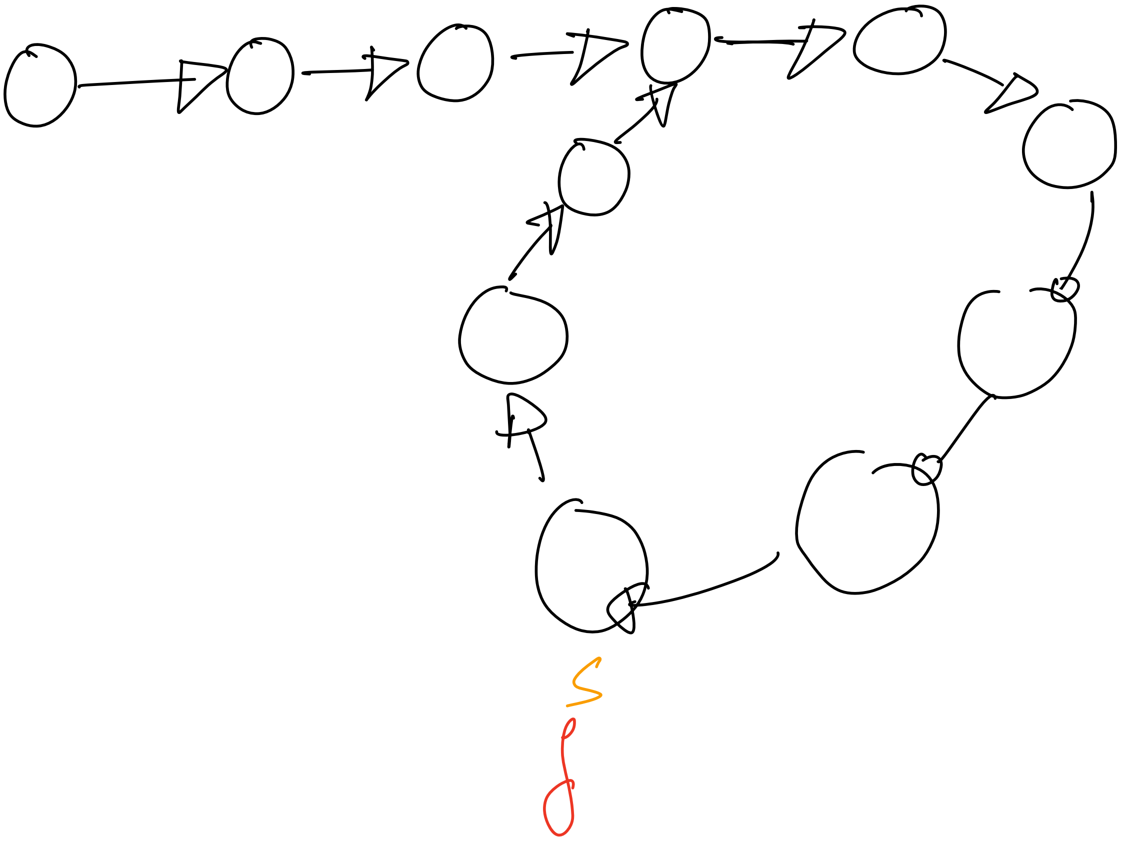




Time	Distane b/w s & f
0	d
1	d - 1
2	d - 2
3	d - 3
d	0

Dry run





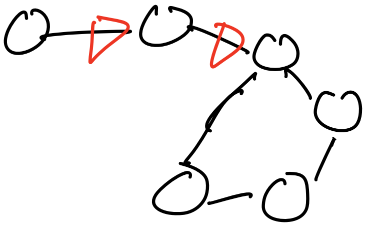
```
Node f = head
Node s = head

while ( f.next != null && f.next.next != null){
        s => s.next
        f => f.next.next
        if (s == f)
            return true
}
return false;
```

Tc: O(n)

Sc: O(1)

Q3 If a loop exists, find starting point of the loop.

Ex 1

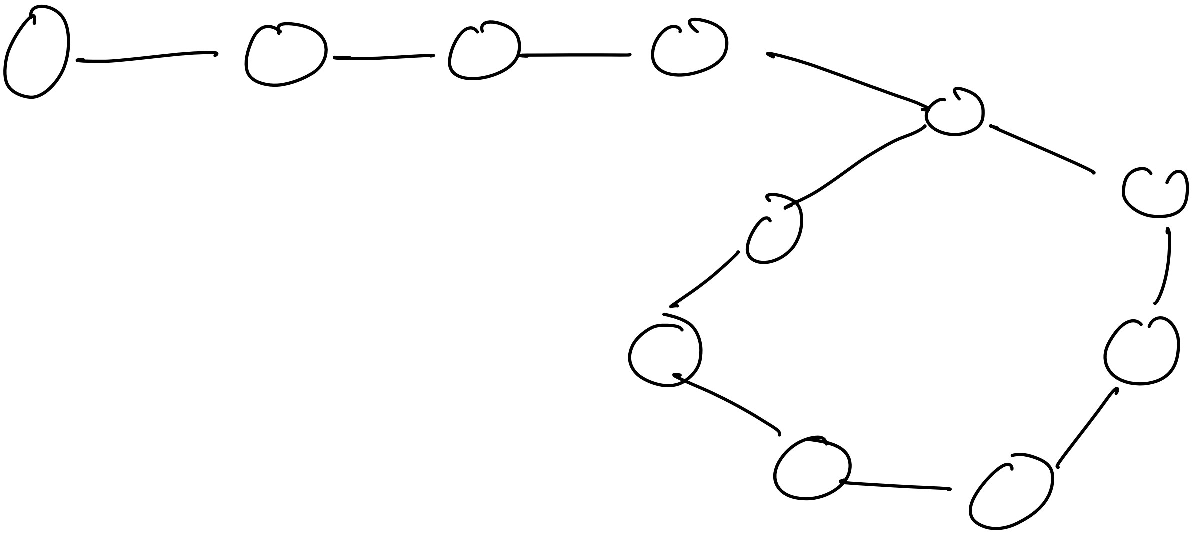


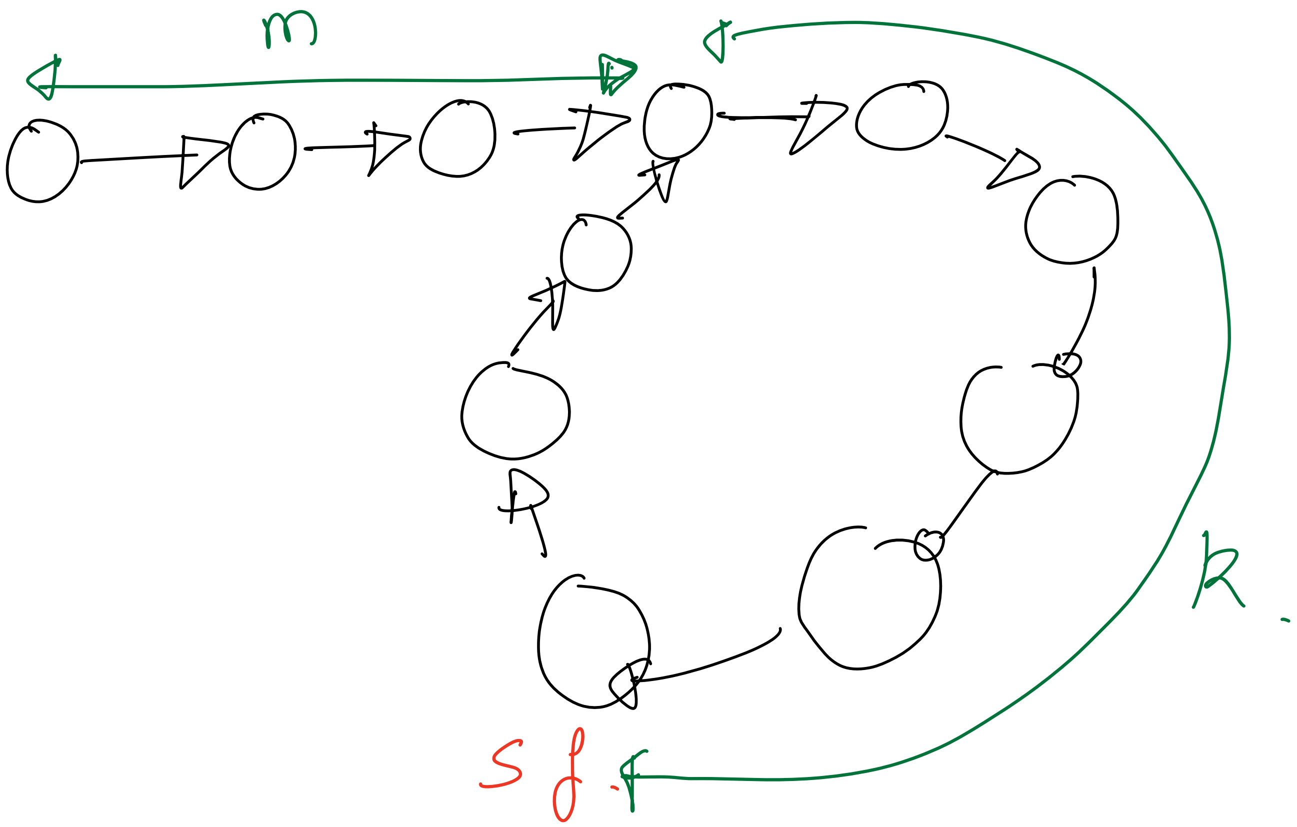
Approach 1:

Use hashset

TC: O(n)

SC: O(n)





Length of loop ⇒ n

Total distance travelled by s ⇒ $m + yn + k$

$y \geq 0$

Total distance travelled by f ⇒ $m + xn + k$

$x > 0$

$$\text{Distance (fast)} \Rightarrow 2\ \text{Distance(slow)}$$

$$m + xn + k \Rightarrow 2(m + yn + k)$$

$$m + xn + k \Rightarrow 2m + 2yn + 2k$$

$$xn - 2yn \Rightarrow m + k$$

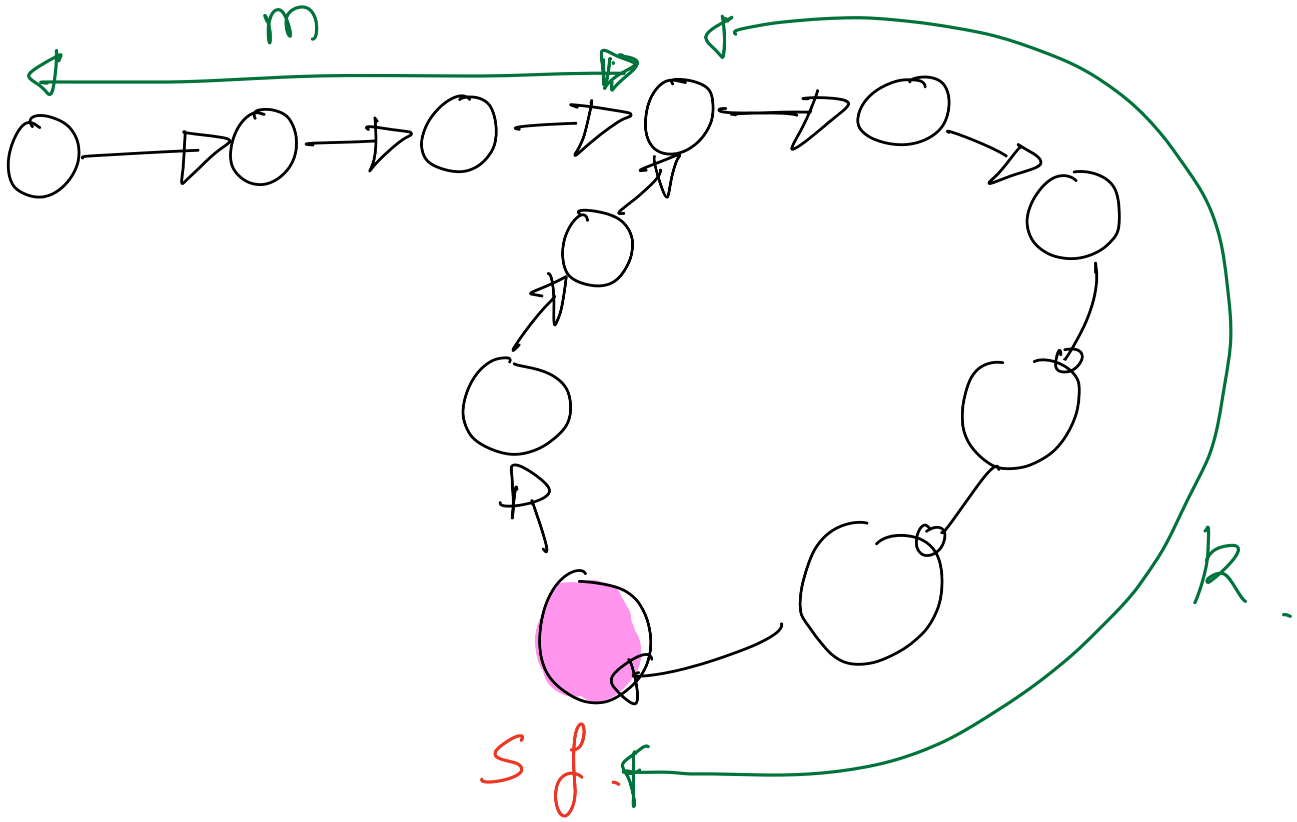
$$m+k \Rightarrow xn - 2yn.$$

$$m+k \Rightarrow n(x-2y)$$

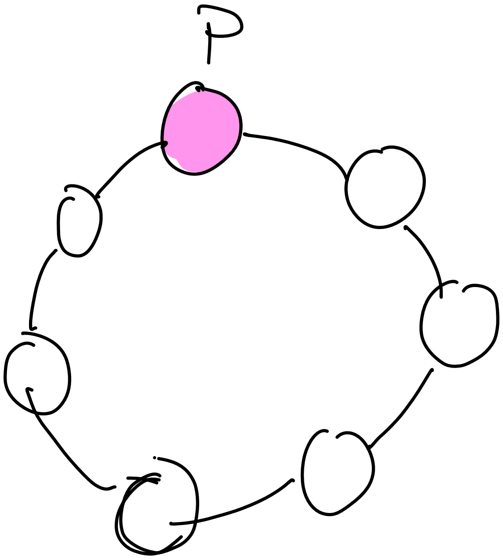
$$\Downarrow$$

$$m+k \Rightarrow n \times \{\text{Something}\}$$

m+k is a multiple of n



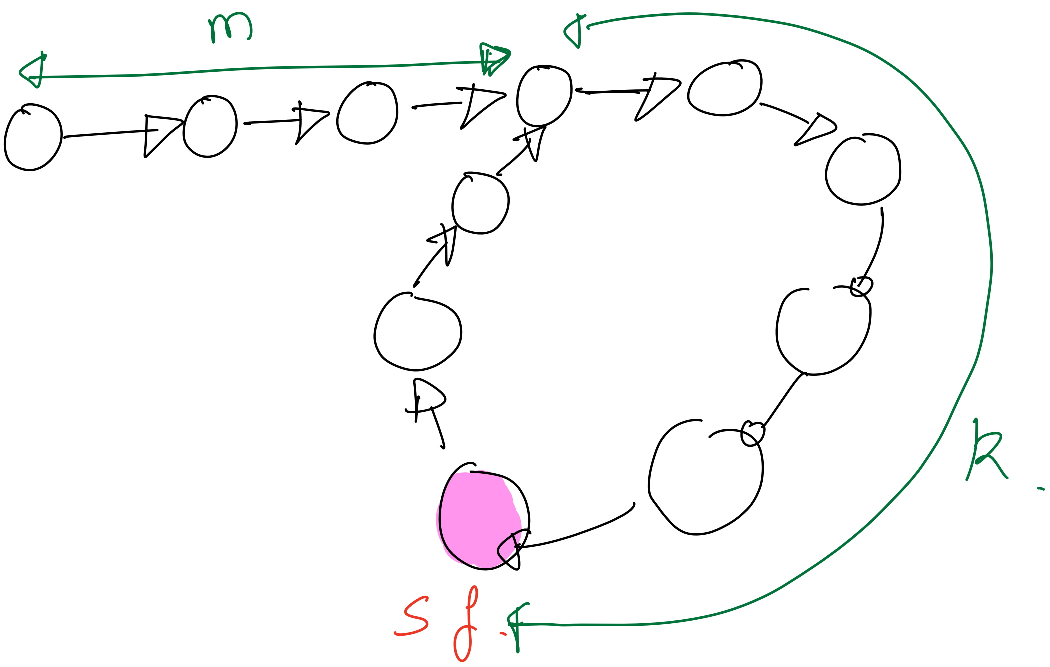
m+k is a multiple of n



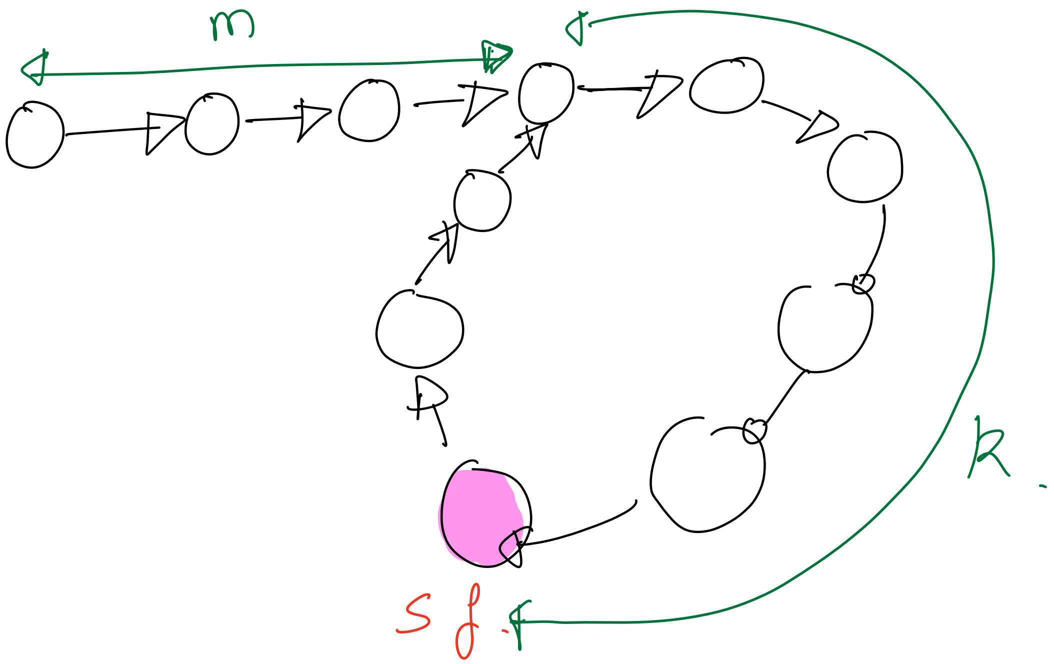
n = 7

If P travels (m+k) times

7 14 21 28



If you travel m steps ahead of the meeting you will reach starting point of loop.



→ You can start traversing from head & the meeting point of sf. Eventually wherever your pointers is the answer.

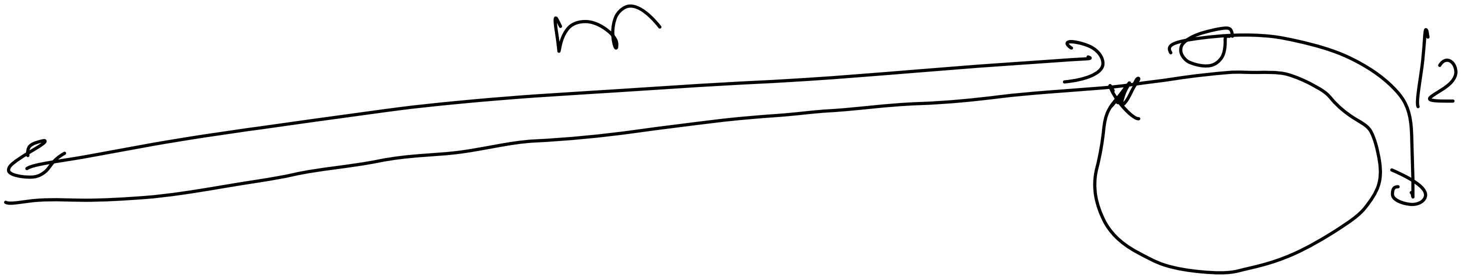
```
Node checkLoop ( Node head) {.

    if (head = null)
        return null:

    Node slow, fast = head;

  while ( fast.next != null && fast.next.next != null) {

        slow = slow.next
       fast = fast.next.next;
       if (slow == fast) break;
  }
  if (slow == fast) {.
     Node P1 => head.
     Node P2 => slow.
     while (P1 != P2) {.
         P1 = P1.next
         P2 = P2.next
     }
     return P1;
  }
```

no of nodes.

↑

Tc: O(n)

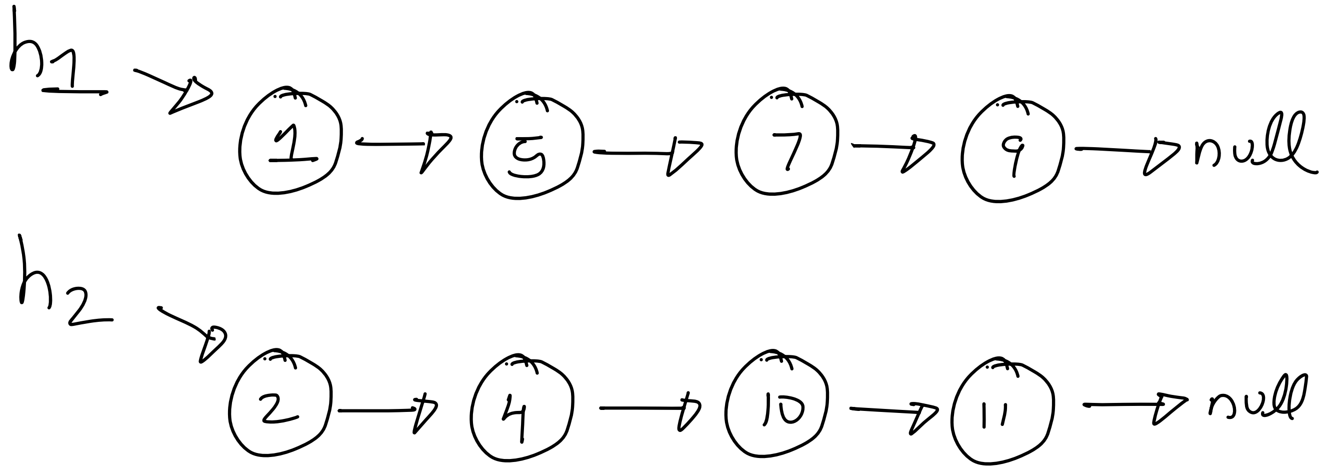
Sc: O(1).

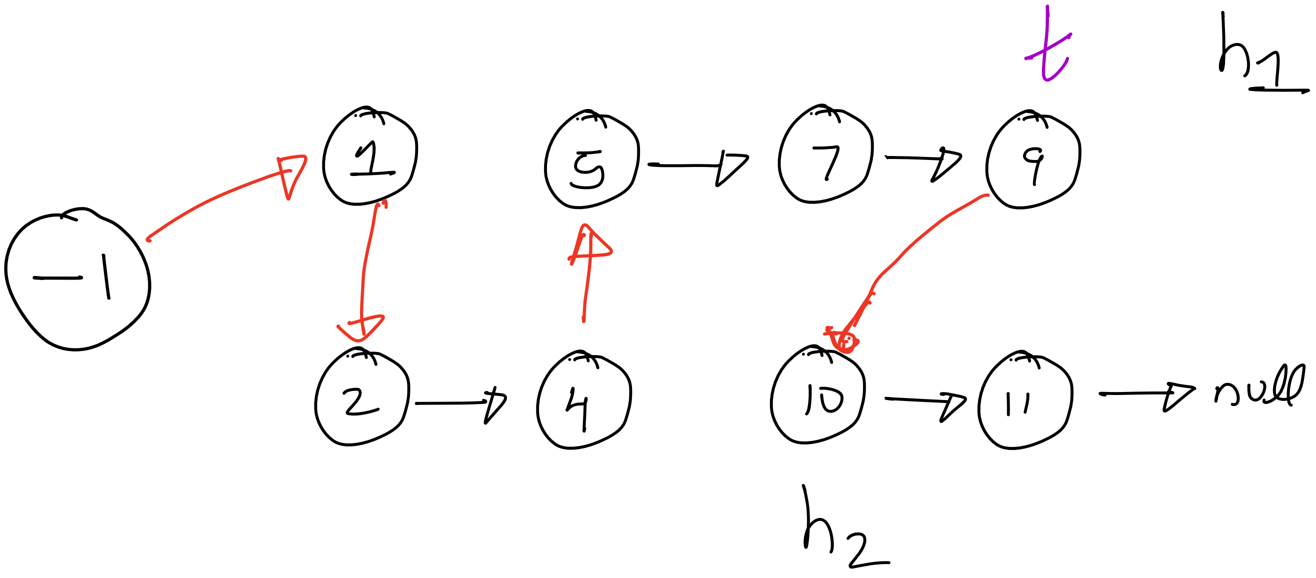


m 7 n

m+k =7

Q4 Given 2 sorted linked list.
Merge them into a single sorted LL.



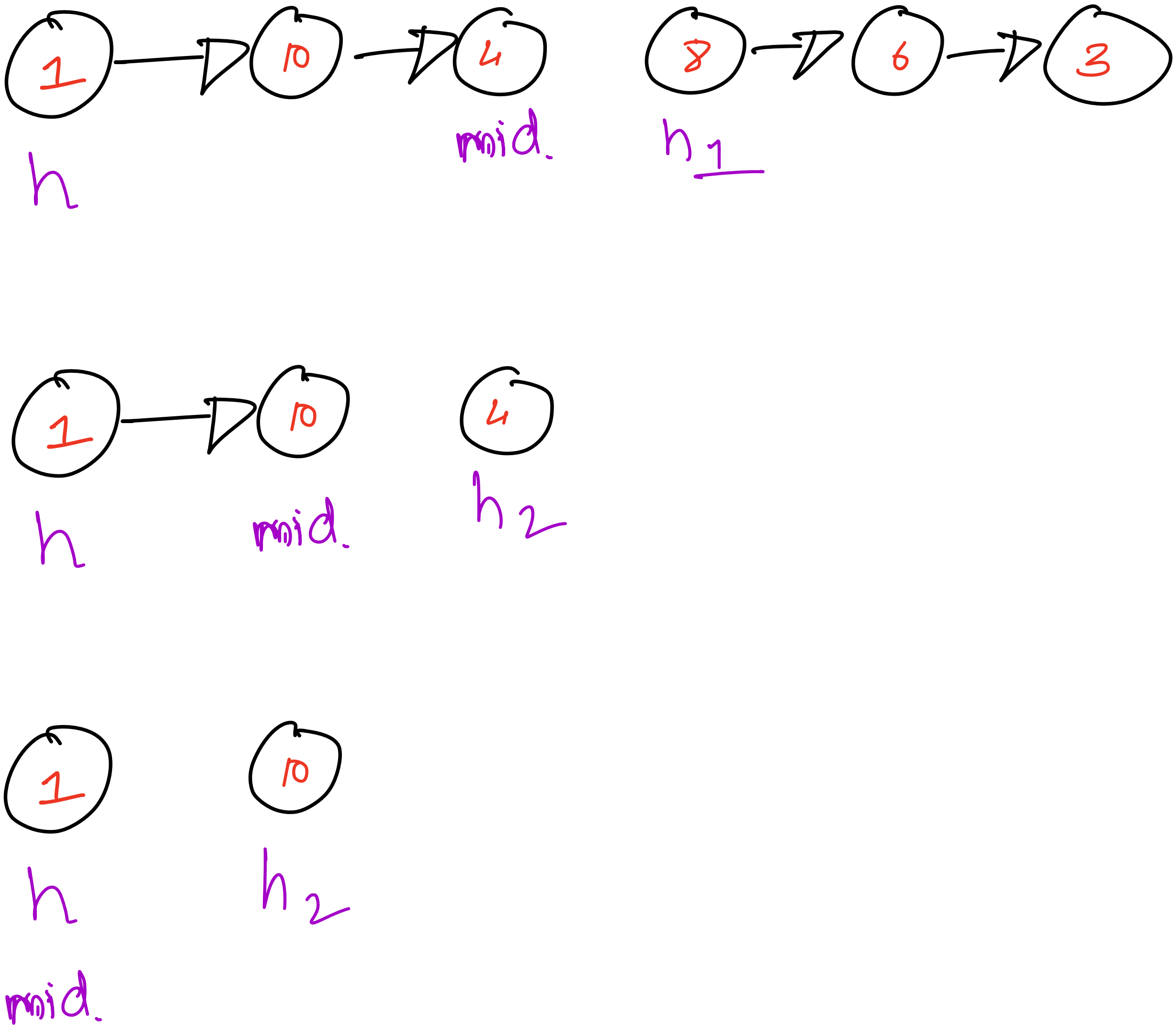


```
Node merge (Node h1 , Node h2) {

   Node dummy = new Node(-1);
   Node curr = dummy:

   while (h1 != null && h2 != null) {
        if (h1.val ≤ h2.val) {
           curr.next = h1;
            h1 = h1.next,
           curr = curr.next;
        } else {
           curr.next = h2
            h2 = h2.next;
           curr. = curr.next;

        }
   }
   if (h1 == null)
        curr.next = h2
   else
      curr.next = h1
```

```
    Node head = dummy.next;
    dummy.next = null;
    del (dummy); free (dummy)
    return head;
}
```

TC: O(n)

SC: O(1)


1
10
4
8
6
3
h
mid
h1
1
10
4
h
mid
h2
1
10
h
h2
mid

```
Node mergeSort (Node head) {

    if (head == null || head.next == null)
                return head;

   Node mid = findMid (head);

   Node h2 = mid.next;

   mid.next = null

   Node head1 = mergeSort (head);
   Node head2 mergeSort (h2);

   head = merge (head1, head2);

   return head;
}
```

$T(n) \Rightarrow n + 2T(n/2)$

$T(n) \Rightarrow n \log n$

$SC : O(\log n)$