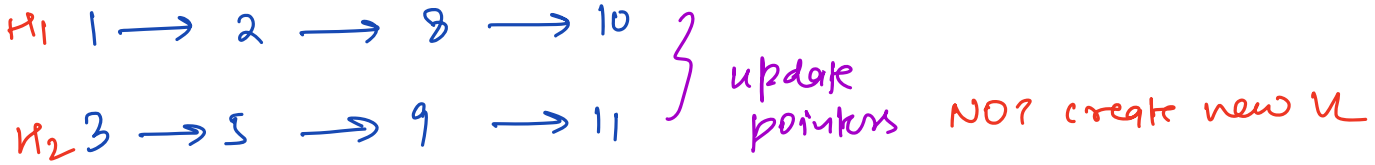
Linked List 2 : Sorting & Detecting Loop

Ques → Given two sorted linked lists.
Merge them into 1 sorted LL.

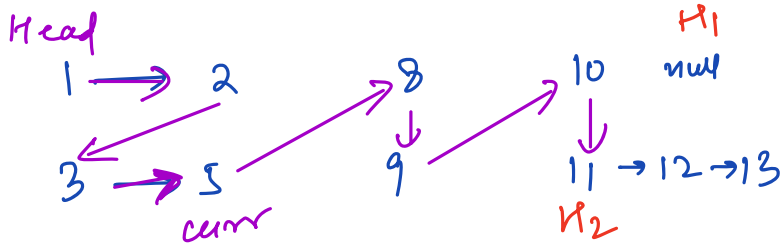


Cases → 1. If any of the list is empty ✓

2. Head → H_1 if H_1.data <= H_2.data
→ H_2 else

```
Node merge(H1, H2) {
    if (H1 == null)  return H2
    if (H2 == null)  return H1

    Head = null
    if (H1.data <= H2.data) {
        Head = H1
        H1 = H1.next
    }
}
```

```
else {
    Head = H2
    H2 = H2.next
}

curr = Head

while (H1 != null && H2 != null) {
    if (H1.data <= H2.data) {
        curr.next = H1
        H1 = H1.next
    }
    else {
        curr.next = H2
        H2 = H2.next
    }
    curr = curr.next
}

if (H1 == null)
    curr.next = H2

else
    curr.next = H1

return Head
}
```



TC = O(N+M) or O(N)

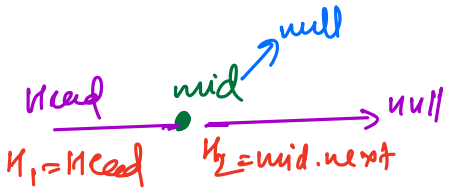
SC = O(1)

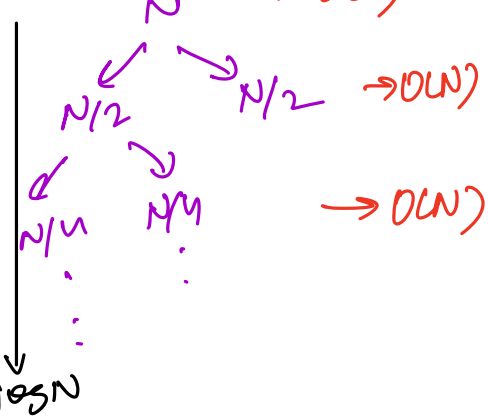
Ques → Merge sort on linked list

Eg 1 → 5 → 2 → 8

o/p : 1 → 2 → 5 → 8

```
Node sort ( Head ) {
    if (Head == null || Head.next == null)
        return Head

    mid = getMiddle (Head)        <- TC=O(N)
    H1 = Head
    H2 = mid.next
    mid.next = null
    H1 = sort(H1)
    H2 = sort(H2)
    return merge (H1, H2)         <- TC=O(N)
}
```





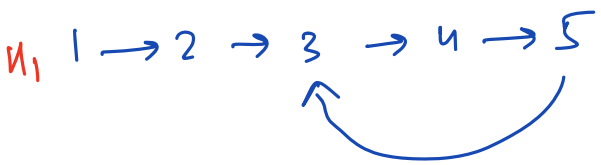
TC = O(NlogN)

SC = O(logN)

Circular linked list

1. No null pointer
2. Head can be updated but not ideal to do so.

Ques → Given a LL, check if it has a cycle.



ans = true

H_2 1 → 2 → 3 → 4 → 5 → null

ans = false

Ideal : search for null node, if present → ans = false
~~else~~ → ans = true

this will not stop in case we have a cycle (TLE)

Idea 2 : Iterate & store each node,

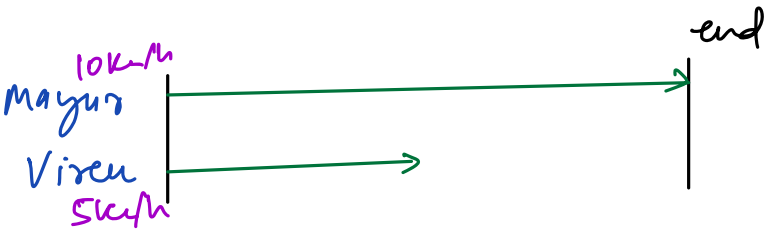
if any node is visited twice ⇒ ans = true

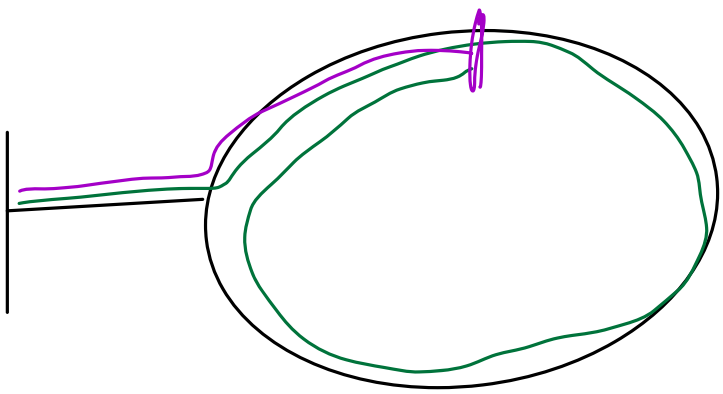
else we will get null ⇒ ans = false

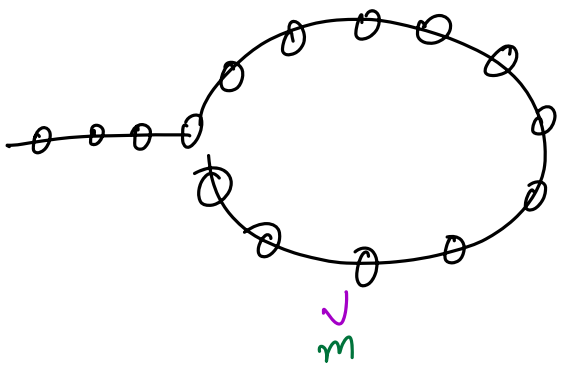
use HashSet HashSet<Node> hs.

TC = O(N) SC = O(N)

Idea 3 :

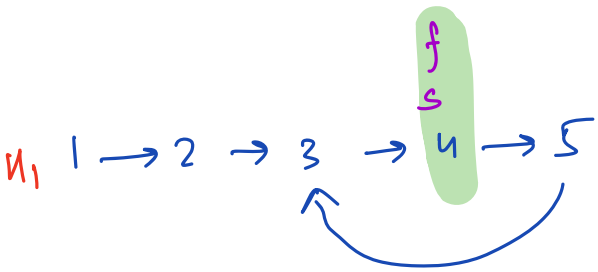




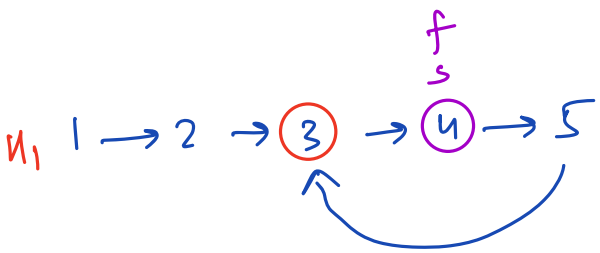


```
slow = Head
fast = Head
while ( fast != null && fast.next != null ) {
    fast = fast.next.next
    slow = slow.next
    if ( slow == fast ) return true
}
return false
```

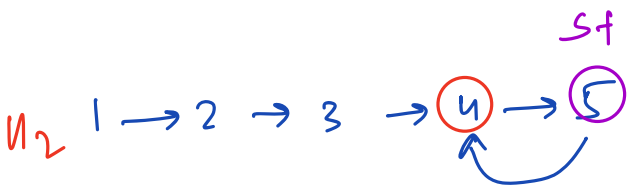
TC = O(N)
SC = O(1)



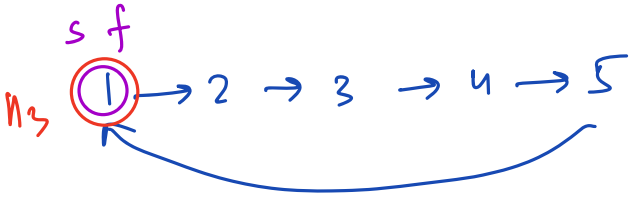
Qnn → Given a LL which contains a cycle.
find the start point of the cycle.



ans = 3



ans = 4

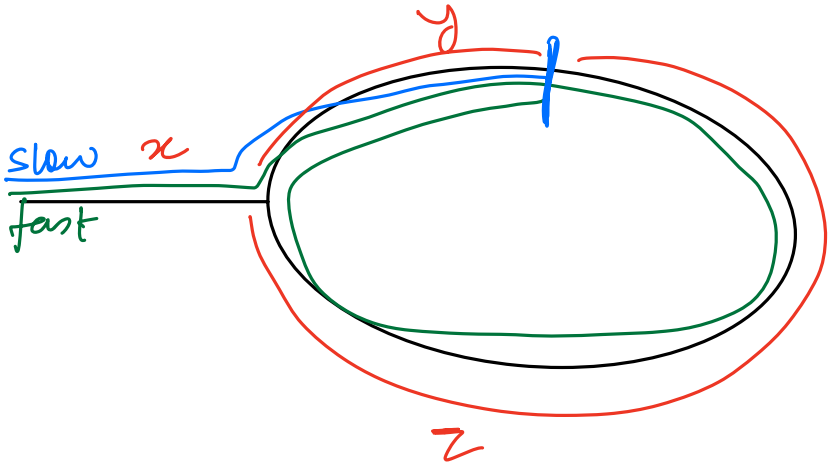


ans = 1

Idea 1 : Iterate & store each node in Hashset, the first node visited twice is the answer.

TC = O(N) SC = O(N)

Idea 2 :



Distance travelled

slow = $x+y$

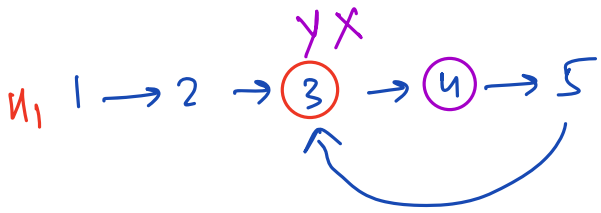
fast = $x+y+z+y$

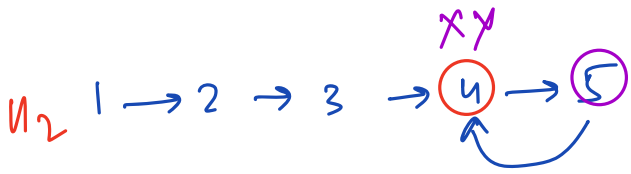
$2(x+y) = x+y+z+y$

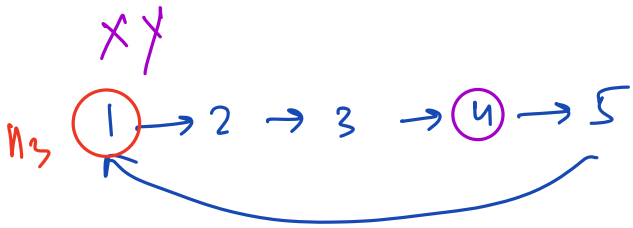
$x = z$

```
slow = Head
fast = Head
while ( fast != null && fast.next != null ){
    fast = fast.next.next
    slow = slow.next
    if ( slow == fast )  break
```

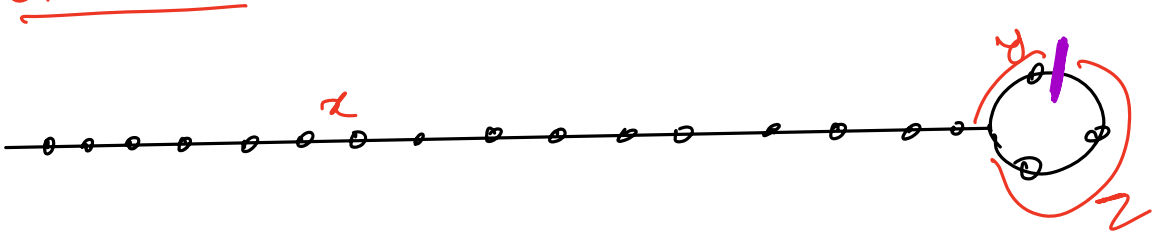
```
}
X = Head
Y = slow
while (X != Y){
    X = X.next
    Y = Y.next
}
return X
```







OPTIONAL



slow : $x+y$

fast : $x + K(y+z) + y$

$$2(x+y) = x + K(y+z) + y$$

$$x+y = K(y+z)$$

$$x+y = (K-1)(y+z) + y + z$$

$$\boxed{x = z + (K-1)(y+z)}$$