Linked List: Sorting & Detecting Loop

Question

Given a LL, find the middle element.

$1 \rightarrow 2 \rightarrow \underline{3} \rightarrow 4 \rightarrow 5$ ans = 3

0 1 2 3
$1 \rightarrow \underline{2} \rightarrow 3 \rightarrow 4$ ans = 2

Solution :

1. find the length of LL
2. traverse length/2 to find mid.

code

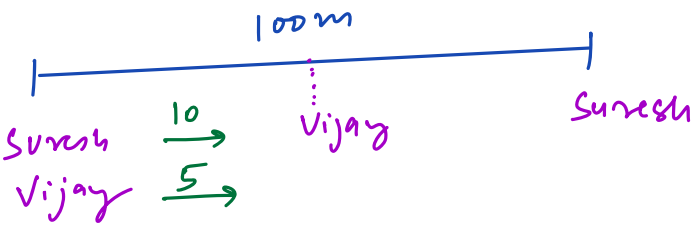
```
Node findMiddle(Head) {
    if(Head == null) return null

    n=0
    curr = Head
    while(curr != null) {
        curr = curr.next
        n++
    }
```

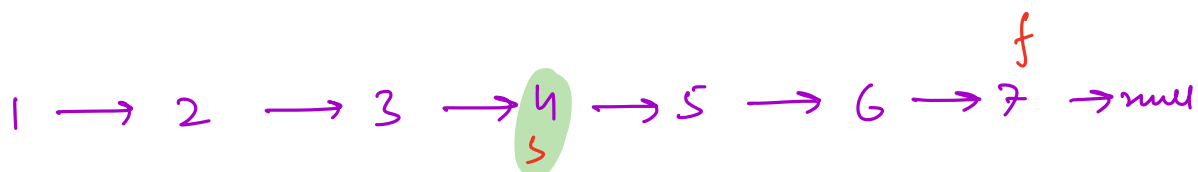
```
mid = n/2
curr = head
for(i=0 to mid-1) {
    curr = curr.next
}
return curr
}
```

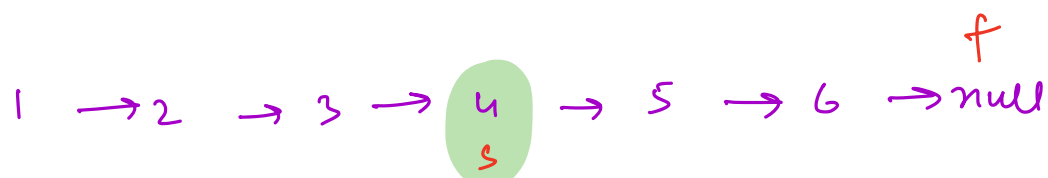
TC = O(N+N) = O(N)

SC = O(1)

Two pointers solution (slow-fast pointer)







Code

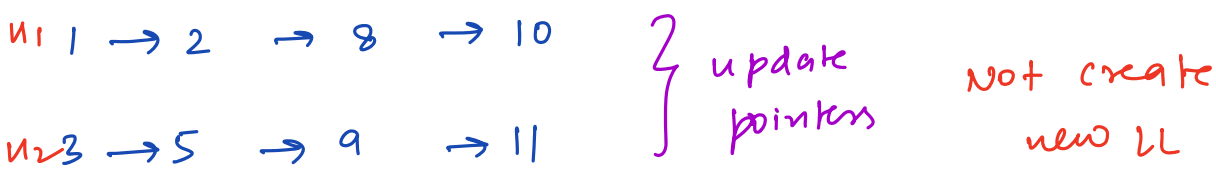
```
Node findMiddle(Head){
    if(Head == null)    return null
    slow = Head
    fast = Head
    while(fast != null && fast.next != null){
        slow = slow.next
        fast = fast.next.next
    }
    return slow
}
```

TC = O(N)

SC = O(1)

Question

Given 2 sorted LL. Merge them into 1 sorted LL.



Head 1 → 2 → 3 → 5 → 8 → 9 → 10 → 11

2 → 10 → 1|

1 → 5 → 12 → 15

⇒ 1 → 2 → 5 → 10 → 11 → 12 → 15

Cases : 1. If any of the list is empty

2. Head ⟶ n_1 if n_1.data <= n_2.data

↳ n_2 else

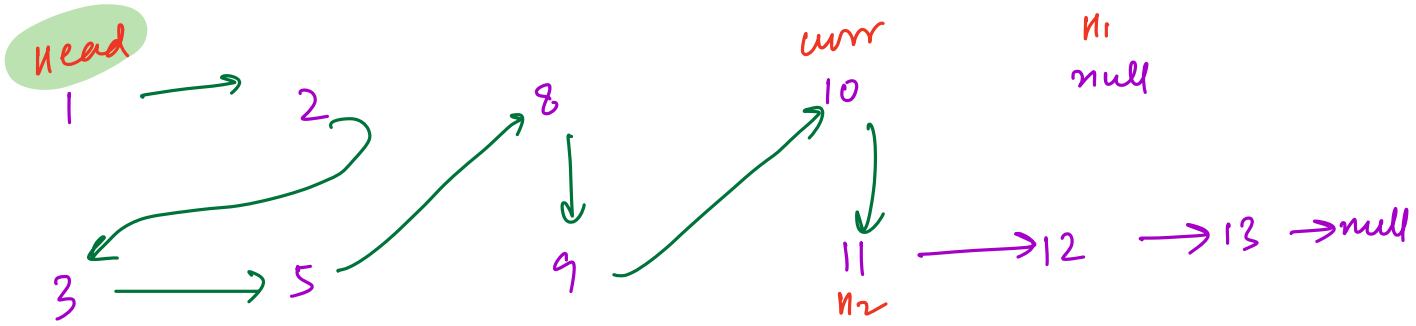
Code

```
Node merge ( n1 , n2 ) {

    if ( n1 == null )    return n2

    if ( n2 == null)     return n1

    head = null

    if ( n1.data <= n2.data ) {

        head = n1

        n1 = n1.next

    }

    else {

        head = n2

        n2 = n2.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)

SC = O(1)



Question

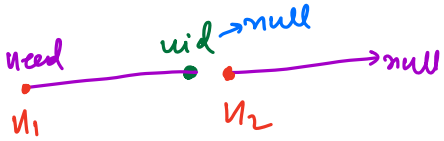
Sort LL using merge sort.

1 → 5 → 2 → 8

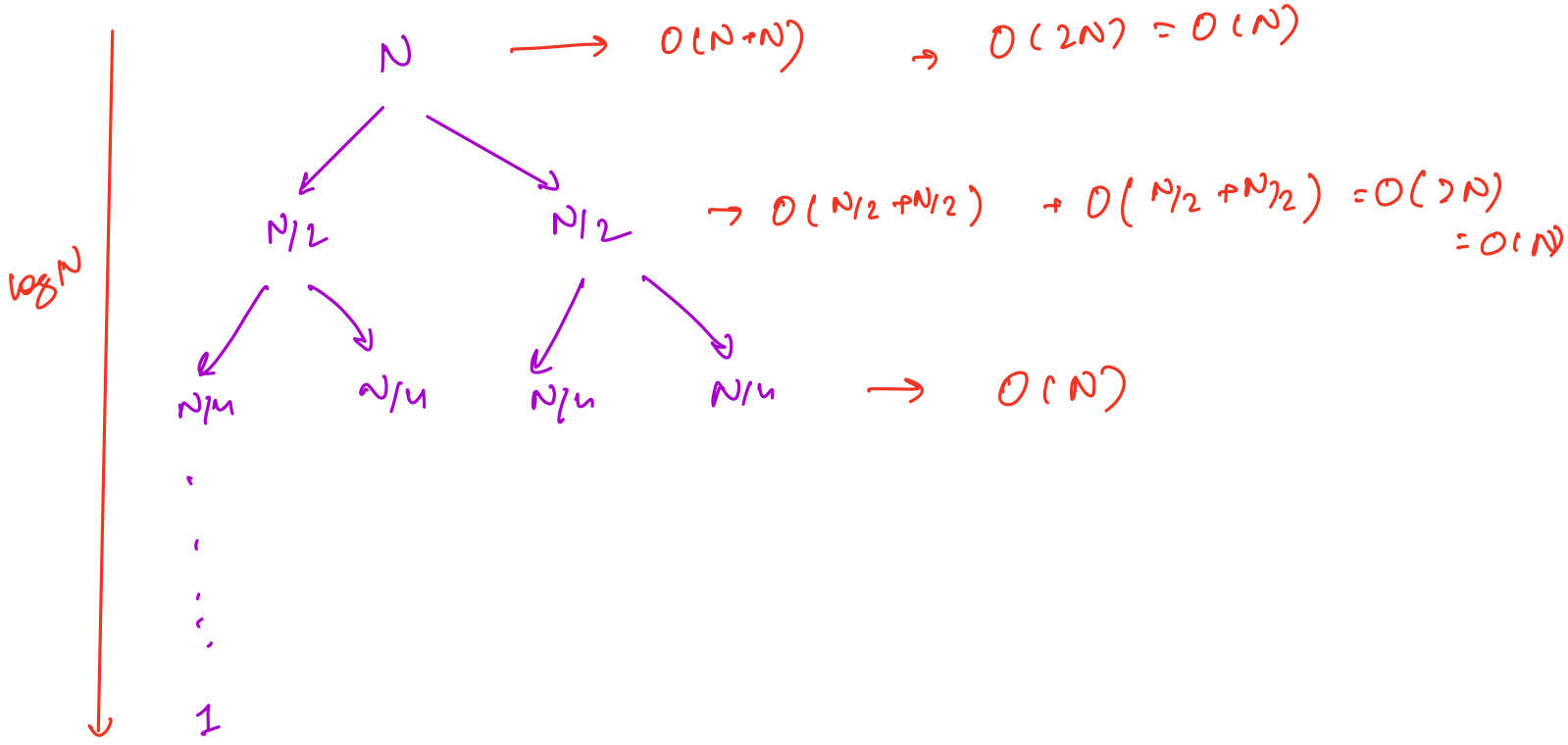
1 → 2 → 5 → 8

Code

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

    midNode = findMiddle(head)        TC = O(N)
    n1 = head
    n2 = mid.next
    mid.next = null
```

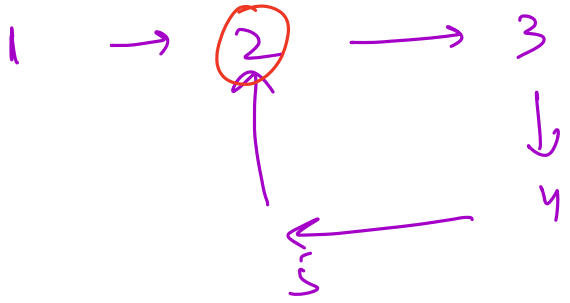


```
    H1 = sort (H1)
    H2 = sort (H2)
    return merge (H1, H2)
                  TC = O(N)
}
```



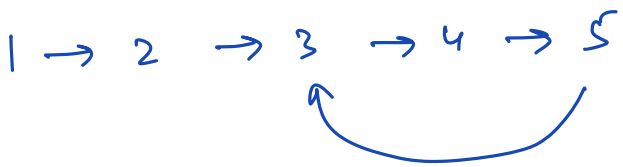
$TC = O(N \log N)$

$SC = O(\log N)$



Question

Given a LL, check if it has a cycle ?



ans = true

1 → 2 → 3 → 4 → null

ans = false

Idea 1 : 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

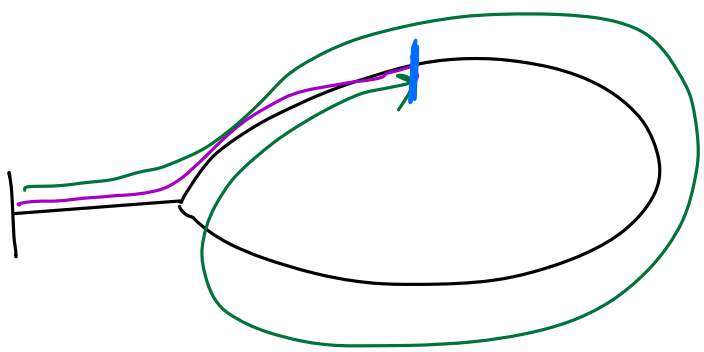
if any node visited twice ⇒ ans = true

else if we got null ⇒ ans : false

use Hashset HashSet<Node> hs

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

Idea 3 :



use slow-fast pointer to detect cycle

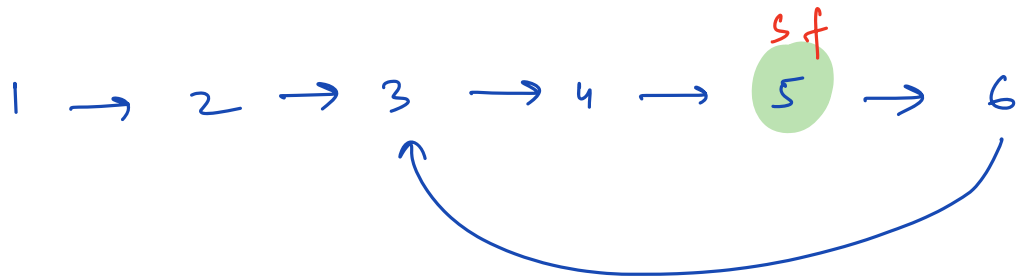
Code

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

```
if (slow == fast)   return true
}
return false
```

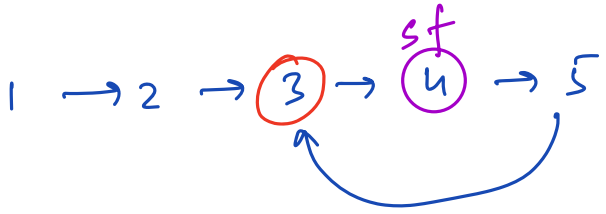
TC = O(N)

SC = O(1)

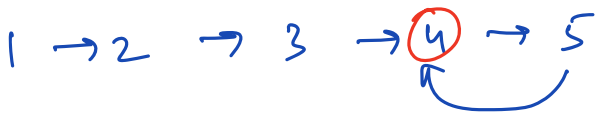


Question

Given LL which contains cycle, find the starting point of cycle.



ans = 3



ans = 4

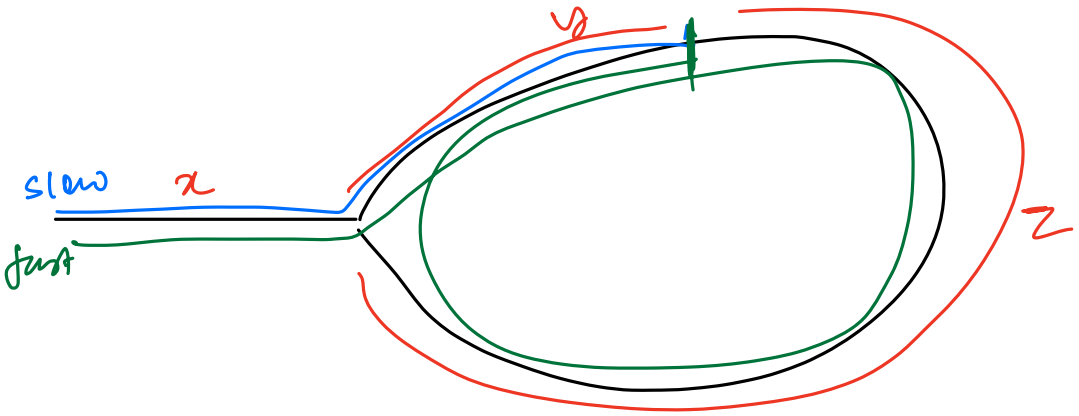
ans = 1

Idea 1 : Iterate & Store node in HashSet

first node visited twice is the answer.

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

Idea 2 :



Distance travelled

slow = $x + y$

fast = $x + y + z + y$

time = $\frac{\text{distance}}{\text{speed}}$

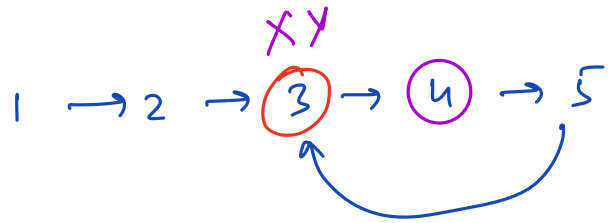
$x + y = \frac{x + y + z + y}{2}$

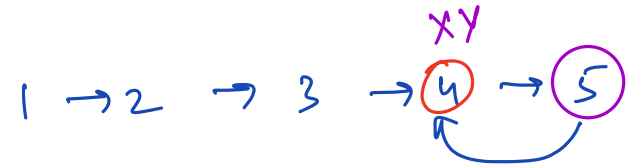
$2x + 2y = x + z + 2y$

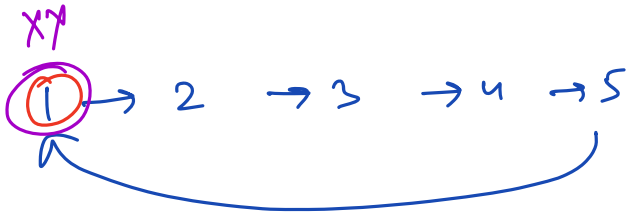
$n = 2$

Code

```
slow = head
fast = head
while ( fast != null && fast.next != null ) {
    slow = slow.next
    fast = fast.next.next
    if ( slow == fast )  break
}
x = head
y = slow  ( or fast)
while ( x != y ){
    x = x.next
    y = y.next
}
return x
```

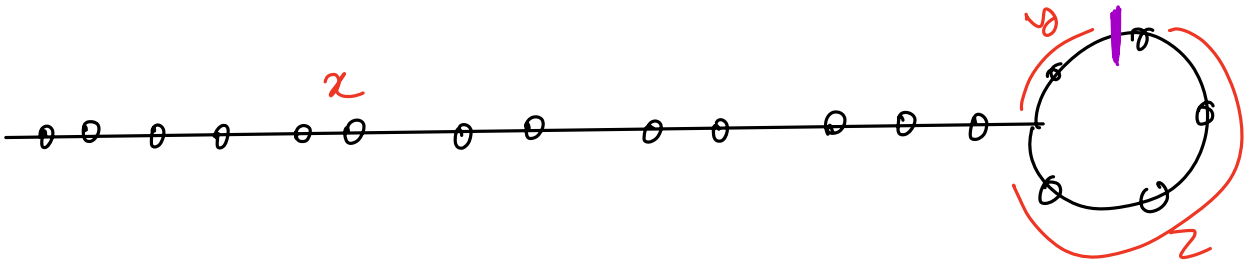






TC = O(N) SC = O(1)

OPTIONAL



slow = $x + y$

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

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

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

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

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