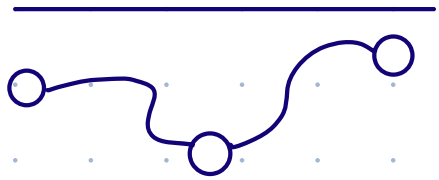
SCALER

LinkedList - 2

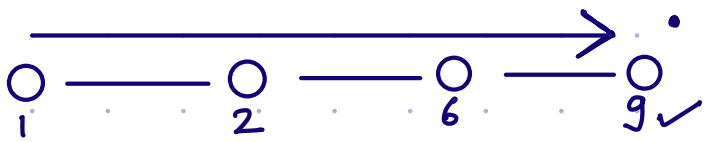
TABLE OF CONTENTS

Revision!

Q1

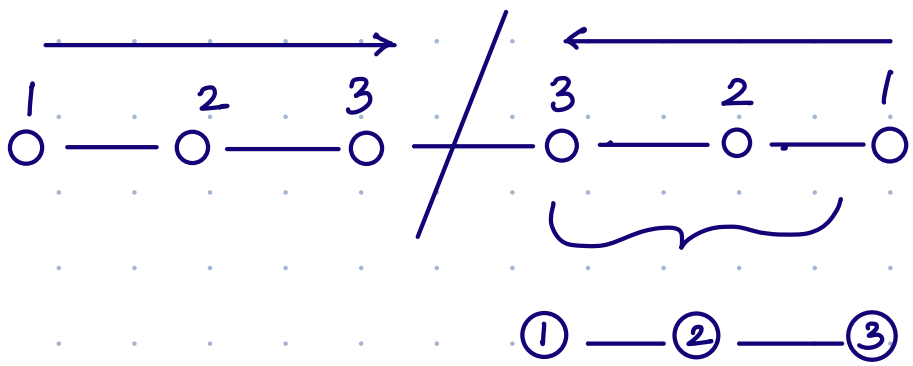


Q2

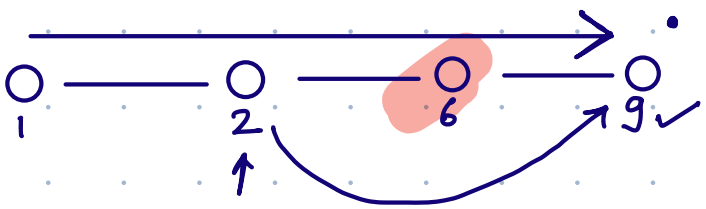


O(N)

Q3

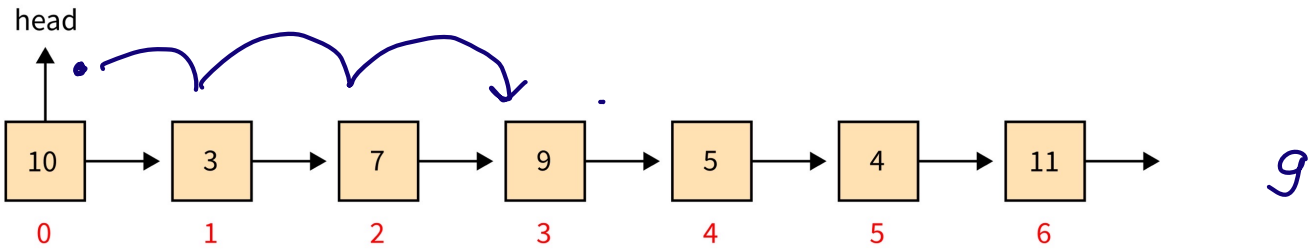


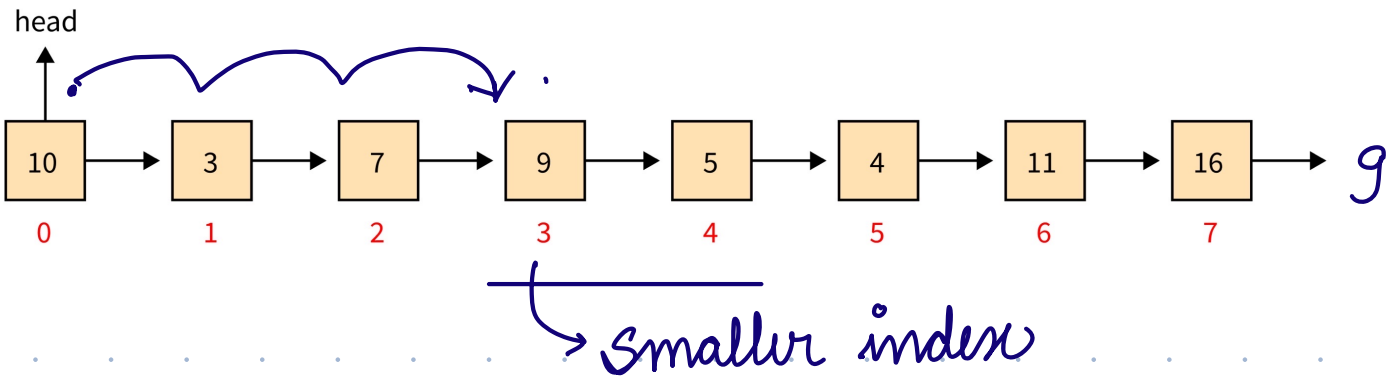
Q4 Delete



O(N)

Middle of a Linked-list





BF Idea → Length of linked list = 7 → $\frac{7}{2} = 3$

= 8 → $\left(\frac{8-1}{2}\right) = \frac{7}{2} = 3$

2 Traversals → Traverse till middle node

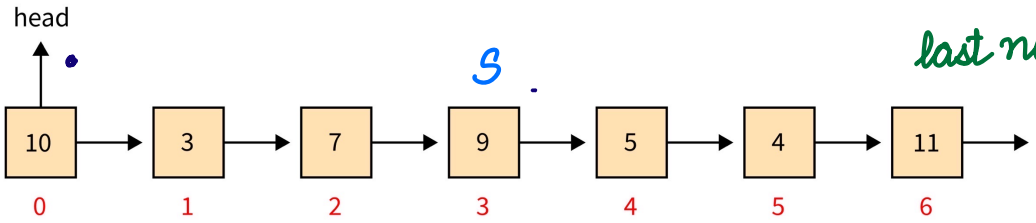
O(N)

Idea -2

Slow and Fast Pointer

{

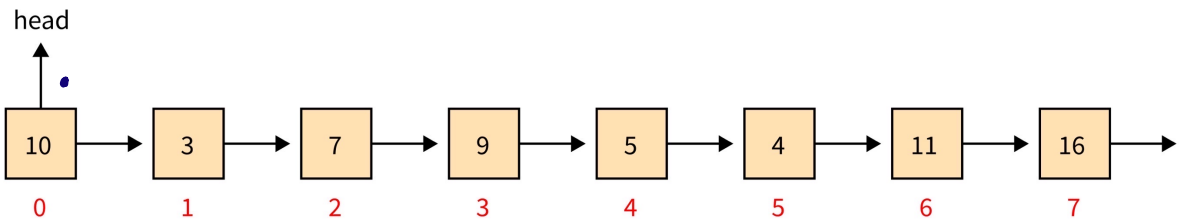
Odd



s

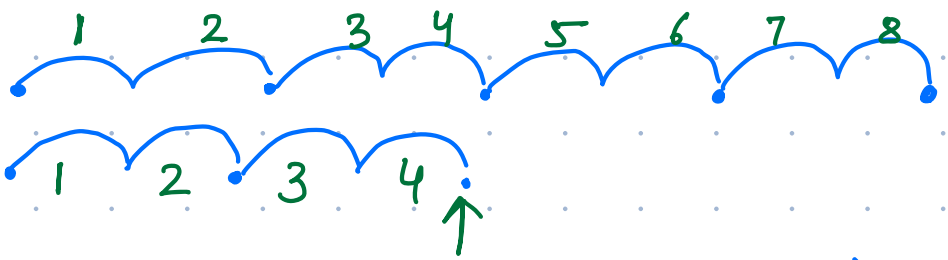
last node

even



s

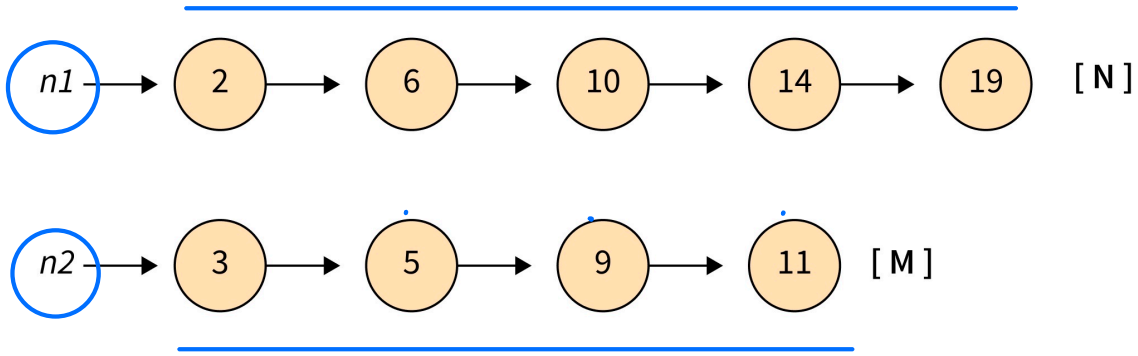
f second last node



N/2 traversal → TC: O(N) → findMid(head)

```
fast = head
slow = head

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

    fast = fast.next.next
    slow = slow.next

}
return slow
```

Merge Two Sorted Linked-list



2 — 3 — 5 — 6 — 9 — 10 — 11 — 14 — 19

n1

head1 ↴
NULL

head2 ↴
12

ans head → 1 — 2 — 3 — 5 — 10 — 11

current

TC : O(N + M)

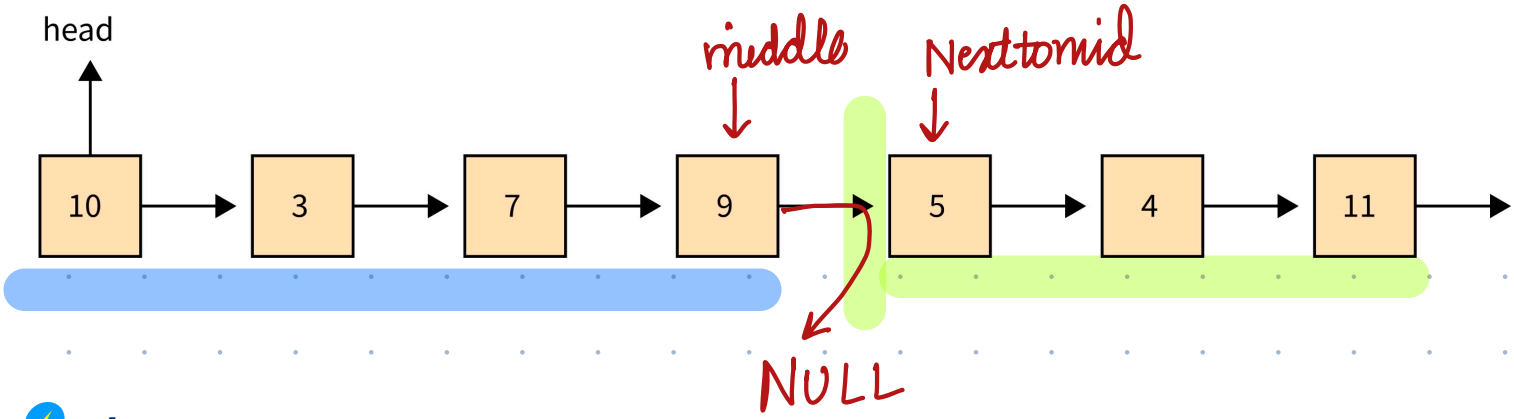
SC : O(1)

< / > Code

```
function Node mergeSortedLists ( h1 , h2 )
    if h1 == null {return h2 }
    if h2 == null {return h1 }

Node anshead = null
Node current = null

    if ( h1.data < h2.data) {
            anshead = h1
            current = h1
            h1 = h1. next
    } else {
            anshead = h2
            current = h2
            h2 = h2. next
    }
```

```
while (h1 != null and h2 != null) {
    if (h1.data < h2.data) {
        current.next = h1
        current = h1
        h1 = h1.next
    } else {
        current.next = h2
        current = h2
        h2 = h2.next
    }
}
if (h1 == null) {
    current.next = h2
} else {
    current.next = h1
}
return anshead
```

Merge Sort a Linked-list



Idea

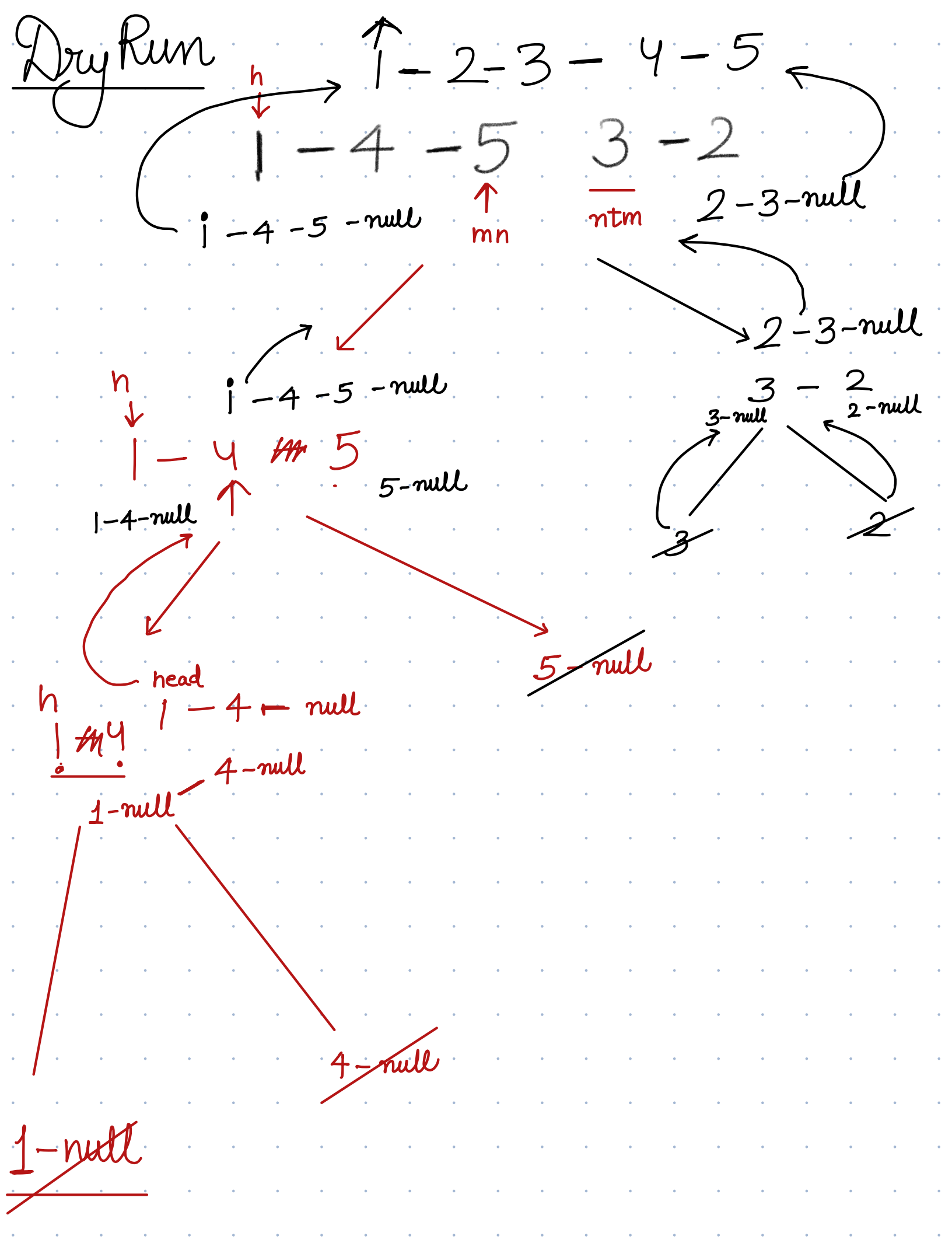
1. Find the middle node
2. Make recursive calls to sort 1st half and 2nd half
3. Finally, merge two sorted linked-list

< / > Code

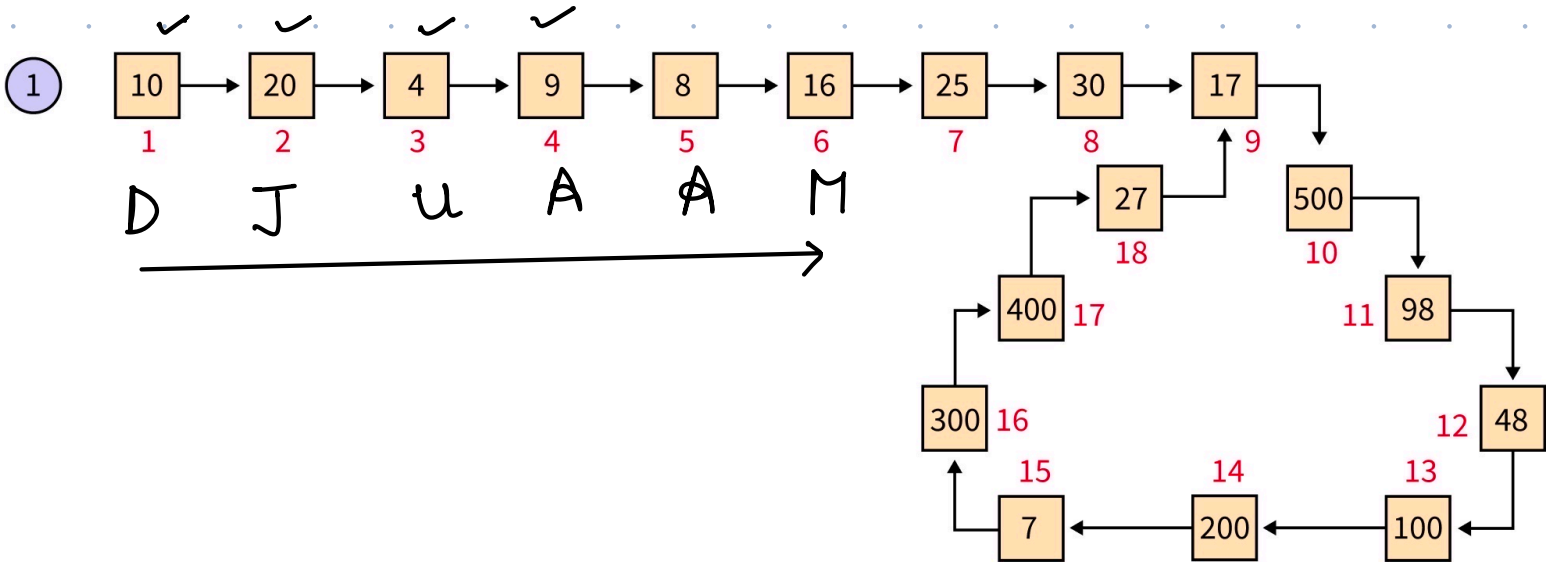
```
function mergeSort ( head ) {
  if ( head == null && head.next == null ) { return; }
    Node middle = findMid ( head )
    nexttoMid = middle.next
    middle.next = null
    left = mergeSort ( head )
    right = mergeSort ( nexttoMid )
```

```
    ans = mergeSortedLists (left, right)

    return ans

}
```

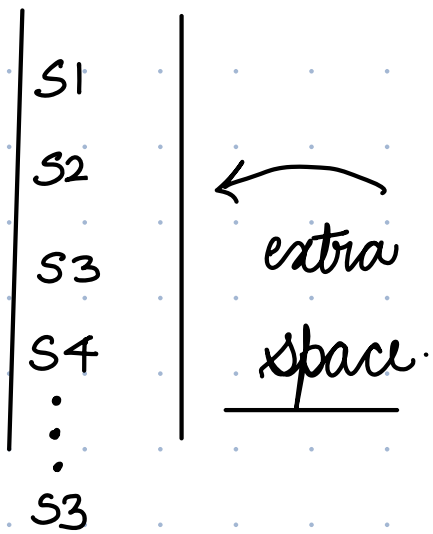
Dry Run


1-2-3-4-5
h
1-4-5
3-2
mn
ntm
1-4-5-null
2-3-null
2-3-null
h
1-4-5-null
3-2
3-null
2-null
1-4 5
5-null
1-4-null
3
2
5-null
head
h
1-4-null
4
4-null
1-null
4-null
1-null

Google Maps

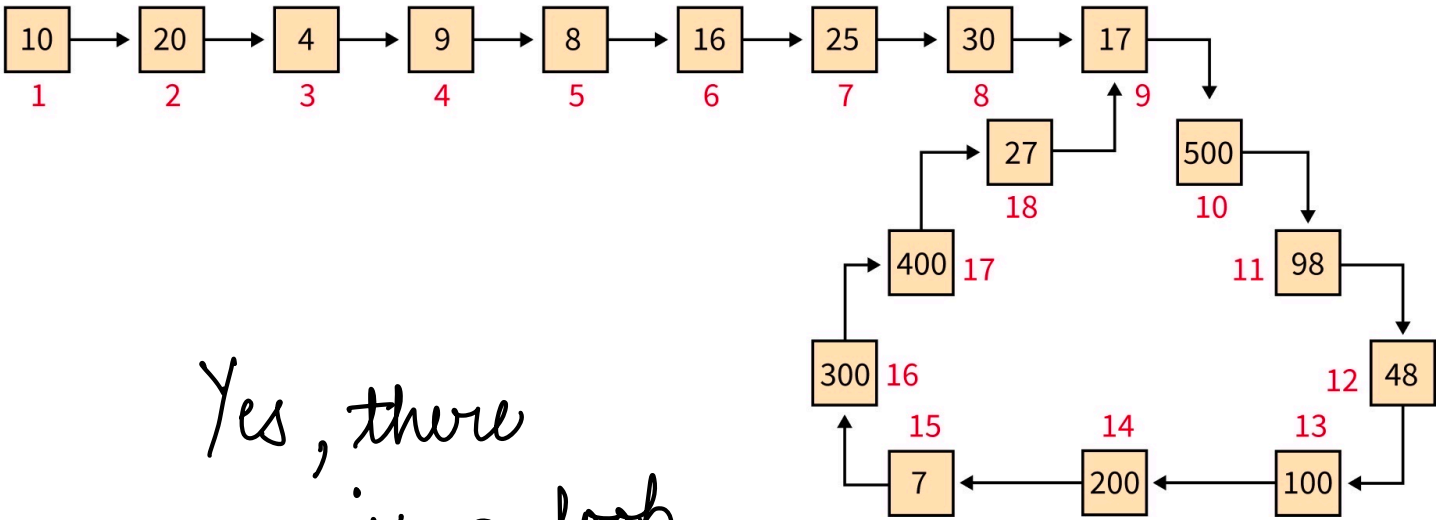


circle ⟶ warm



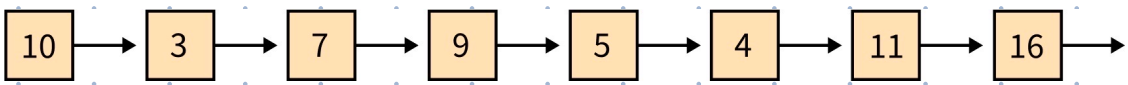
Check if there is a loop

1

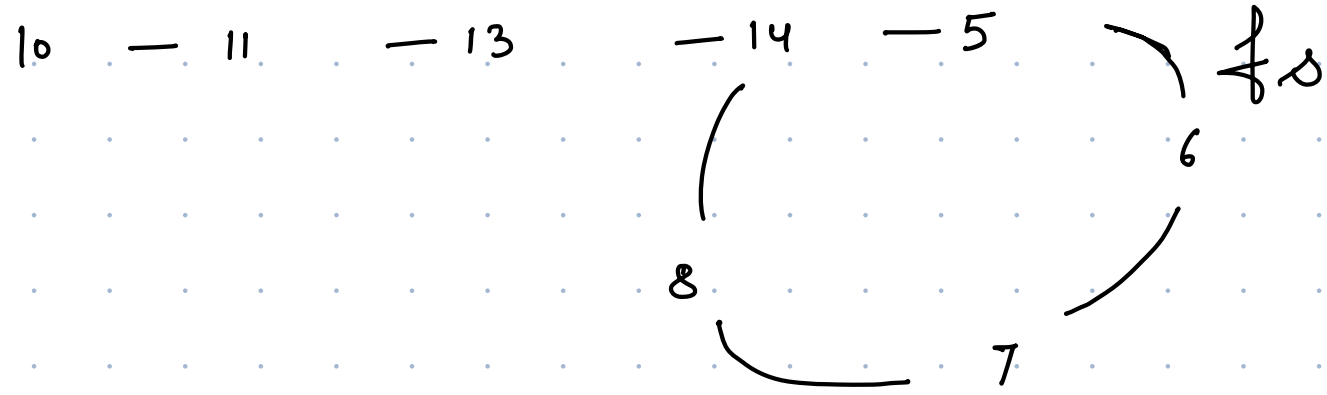


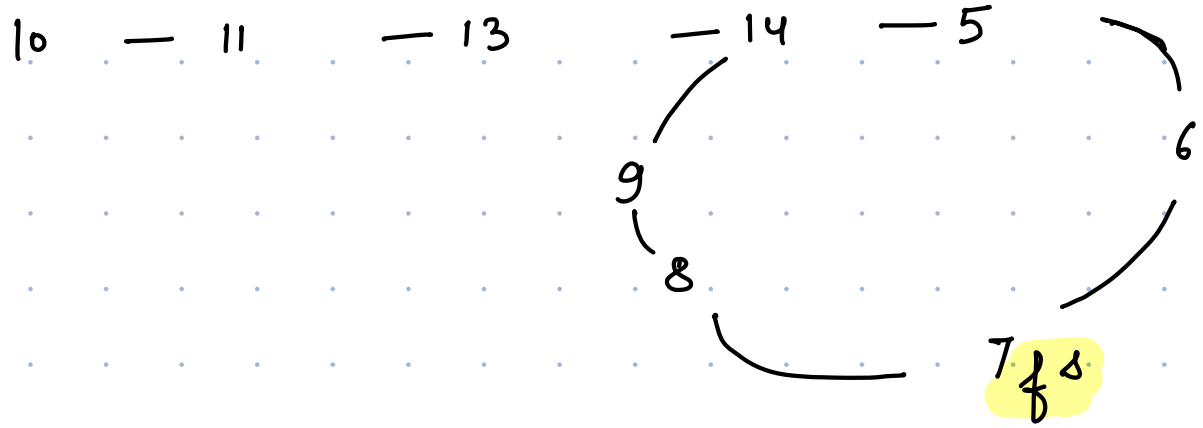
Yes, there is a loop

2



False! No loop



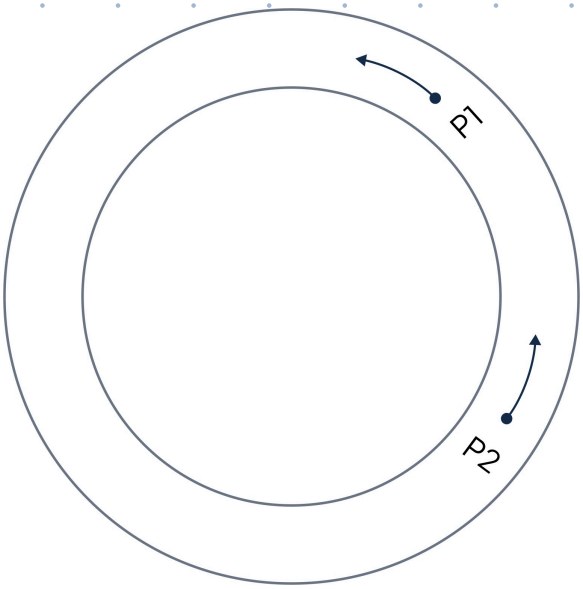


If there is a loop, then they will definitely coincide with each other.

+1 ⟶

+2 ⟶

Idea -2 If two people are running with different speeds on a circular track, they will 100% meet at some point.

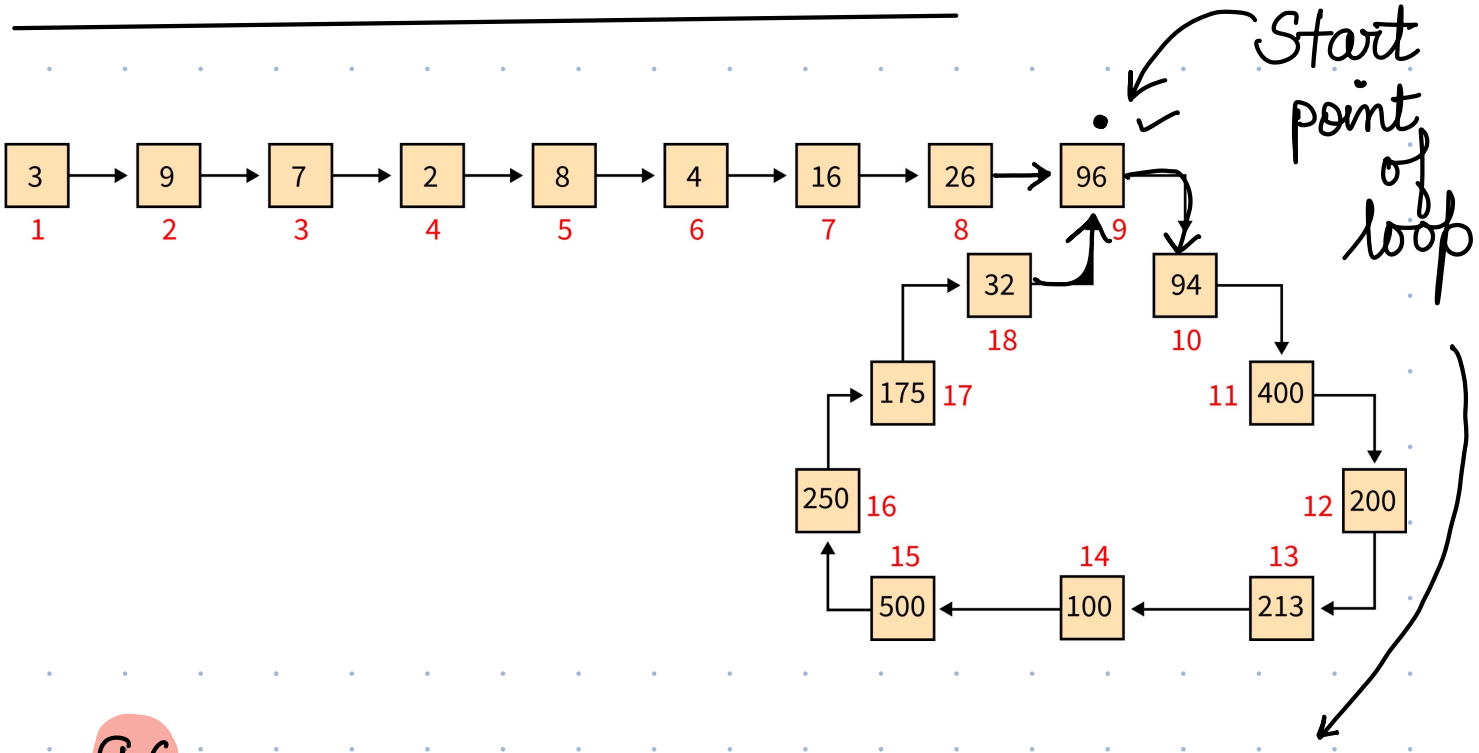


< / > Pseudo-Code

```
function hasCycle ( head ) {

    slow = head
    fast = head

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

        slow = slow.next
        fast = fast.next.next

    }
    return false;
}
```

Next interesting part

Find the start point of the loop



node → which is pointed by 2 previous nodes

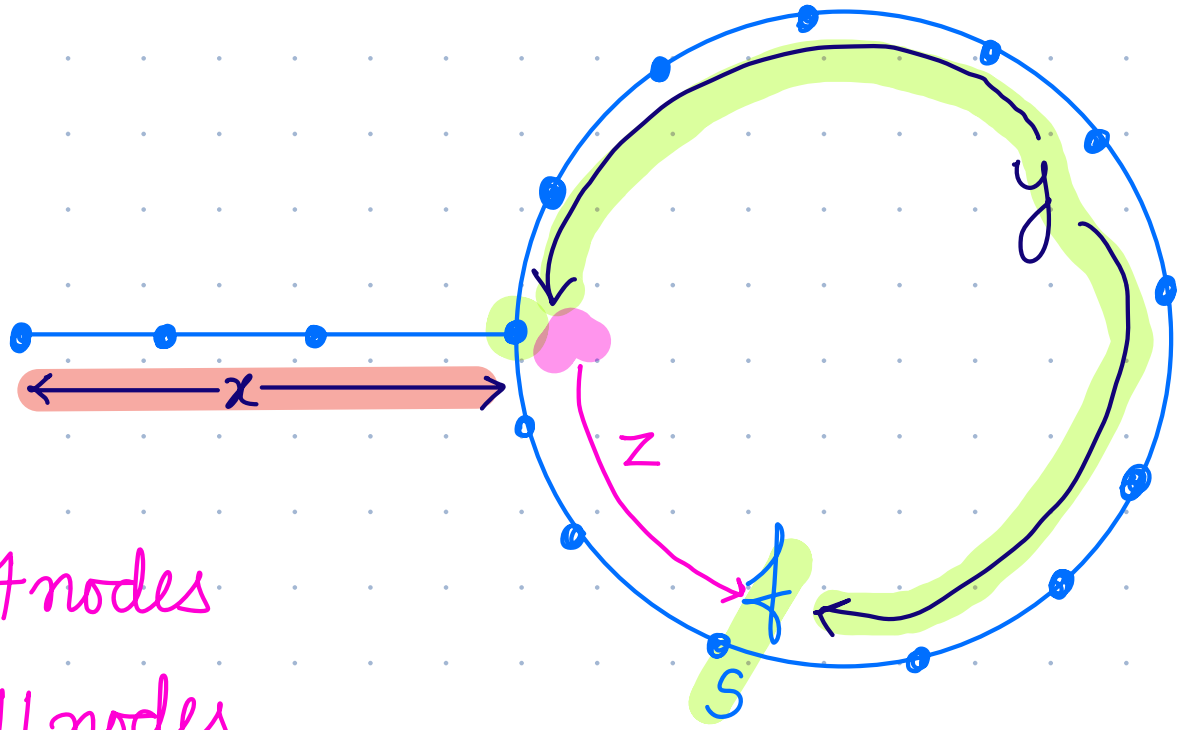
96

Brute force:

3
9
7
2
8
4
16
26
96
94

400	250
200	175
213	32
100	96
500	

Proof



Floyd's Cycle Detection

x = 4 nodes

y = 11 nodes

z = 4 nodes

$$slow = x + y$$

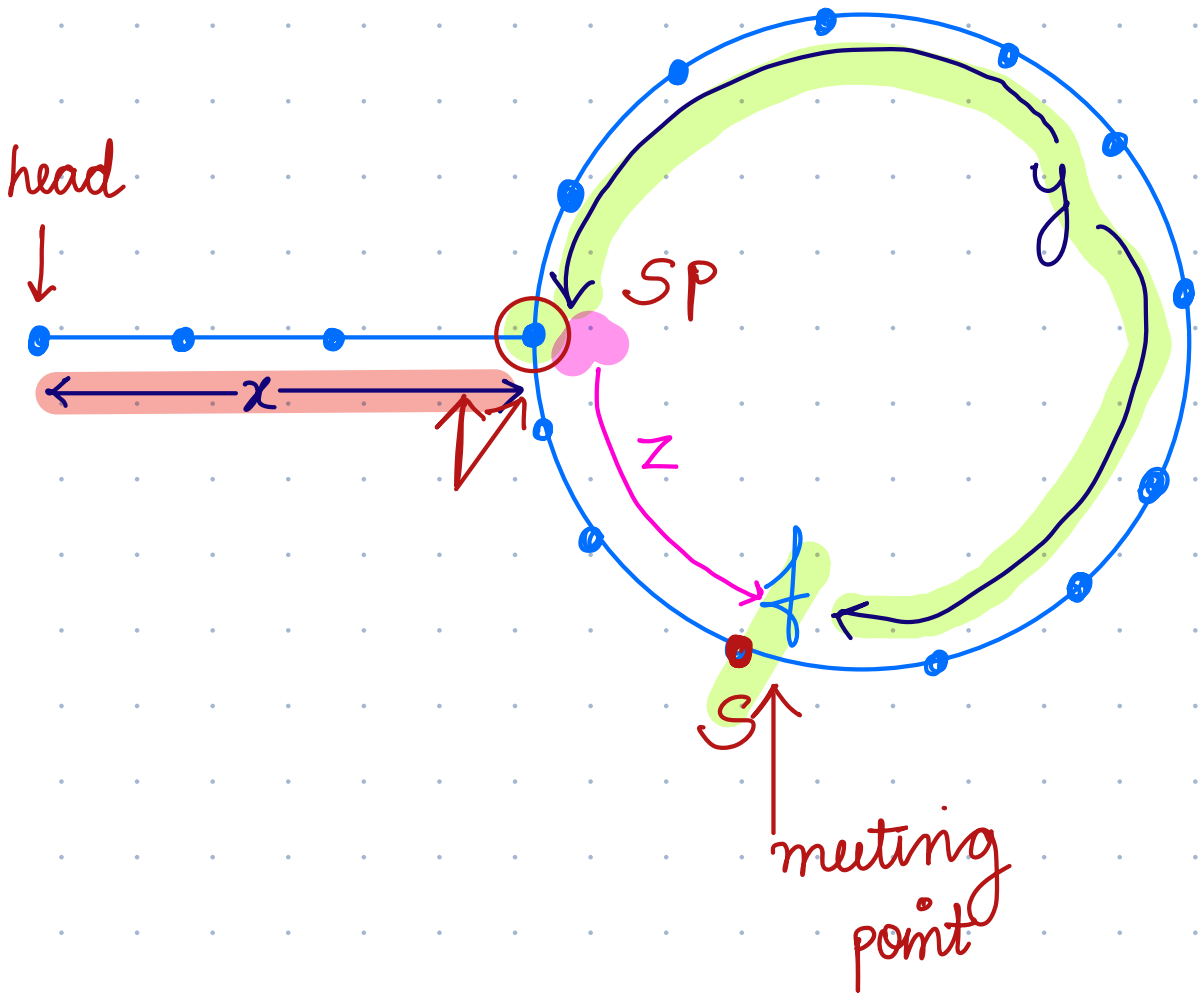
$$fast = x + y + z + y$$

fast has covered twice as many nodes as slow

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

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

$$\boxed{x = z} \leftarrow$$



```
function detectCycleStart (head)
    if (head == null || head.next == null)
        return null

    slow = head
    fast = head
    hasCycle = false

    while (fast.next != null && fast.next.next != null){
        if (slow == fast) {
            hasCycle = true
            break;
        }
        slow = slow.next
        fast = fast.next.next
    }

    if (hasCycle == false) { return null }
```

slow = head

fast ← meeting point

while (slow != fast) {

slow = slow.next

fast = fast.next

}

return slow

TC: O(N)

SC: O(1)