Agenda

Find middle element of LL

Merge 2 sorted LL

Sort a LL

Find cycle in LL

Revision Quizzes (1-4)

Quiz 1: Main advantage of using a linked list over an array? It can utilize all free memory without requiring contiguous space.

Quiz 2: TC to search for a value x in LL

O(N) as we traverse LL to search node by node

Quiz 3: Check if LL is palindrome using O(N) time and O(1) space?

4 → 2 → 3 → 2 → 4

4 → 2 → 3

2 → 4

① Find middle, break LL from middle

② Reverse second half of LL

③ Compare nodes

h1: 4 → 2 → 3

h2: 4 → 2

Quiz 4: TC to delete node in a LL

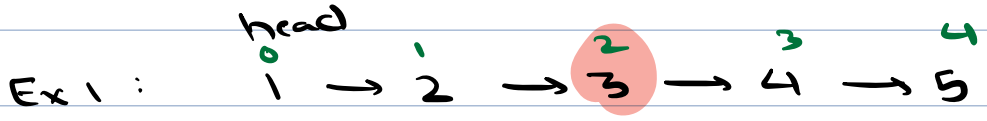
O(N) as we need to traverse to prev of the target node

Quiz 1: Can we do binary search in sorted LL? No

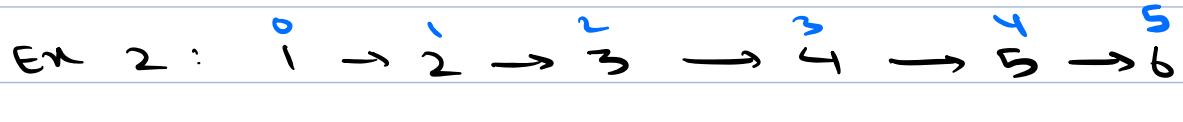
Binary search relies on random access (especially mid) of elements.

Not supported by LL.

1. Given a linked list, find the middle element.

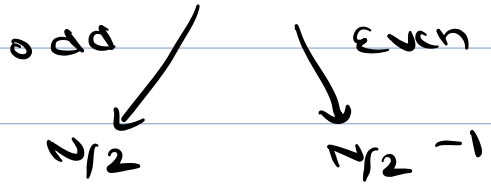


ans → 3



ans → 3 (1st middle)

Approach 1: Get size of LL (traverse & count)
Traverse and reach mid idx



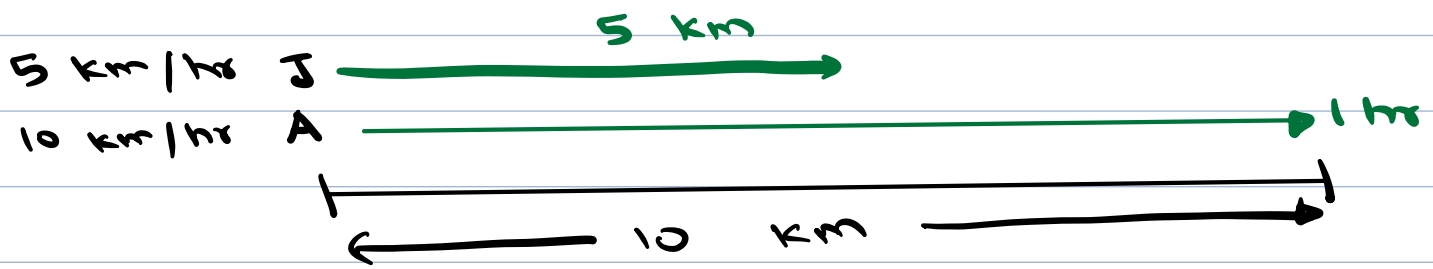

```
Node findMiddle(Node head){

    Node temp = head
    int size = 0

    while(temp != NULL){          // N
        size++
        temp = temp.next
    }

    int midIdx = size/2
    if(size % 2 == 0)
        midIdx--
```

```
temp = head                                   → N/2
for ( j=1 ; j ≤ mid Idx ; j++)
    temp = temp.next

return temp
}
```

TC : O(N + N/2) = O(N) SC : O(1)

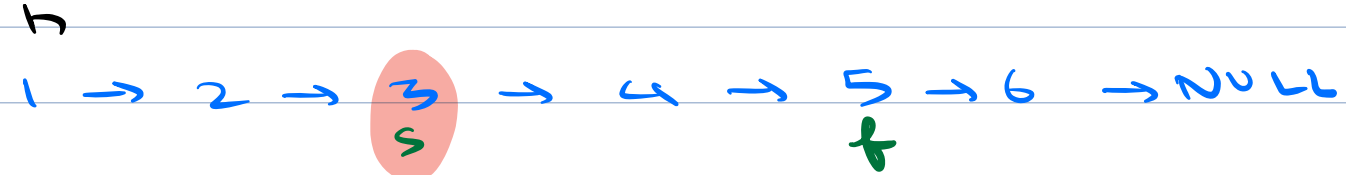
2 passes

Optimized solution : 1 pass of LL



For odd
f → last node

s → 1 step
f → 2 steps



For even
f → 2nd last node

```
Node findMiddle(Node head){
    if(head == NULL) return NULL
    Node slow = head, fast = head
    while(f.next != NULL && f.next.next != NULL)
        slow = slow.next
        fast = fast.next.next

    return slow
}
```

→ N/2 iterations

odd

even

TC: O(N)

SC: O(1)

if (a && b)

STOP if f reaches 2nd last (Even node) or f reaches last node (Odd)

Continue if f != 2nd last node AND f != Last node

2. Given two sorted linked list, merge them into a single sorted linked list.

Ex 1: h1: 1 → 2 → 8 → 10 | h2: 3 → 5 → 9 → 11

ans: newH: 1 → 2 → 3 → 5 → 8 → 9 → 10 → 11 → NULL

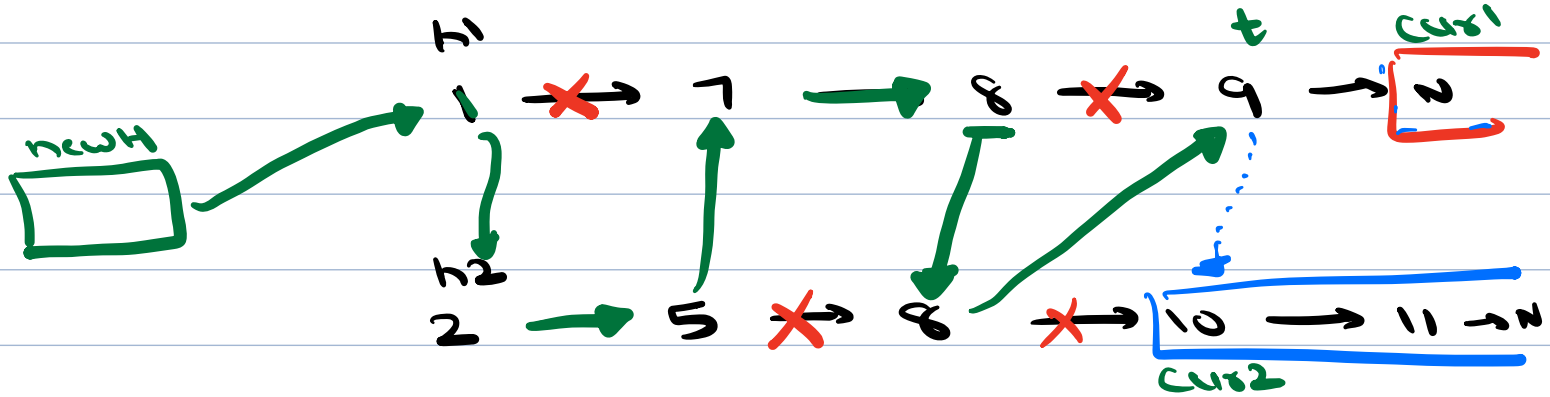
Ex 2: h1: 1 → 7 → 8 → 9 | h2: 2 → 5 → 8 → 10 → 11

ans: newH: 1 → 2 → 5 → 7 → 8 → 8 → 9 → 10 → 11 → NULL

Quiz 5: h1: 2 → 10 → 11 (p1) | h2: 1 → 5 → 12 → 15 (p2)

ans: 1 → 2 → 5 → 10 → 11 → 12 → 15 → NULL

Approach: Take 2 pointers compare nodes and keep adding smaller nodes in final list.



```
if ( cur1.data < cur2.data) {
    t.next = cur1
    cur1 = cur1.next
     t =  t.next
}
```

Final LL → newH → t

LL1 → cur1 → NULL

LL2 → cur2 → NULL ?

```
Node mergeSortedLL (Node h1, Node h2) {
    Node cur1 = h1, cur2 = h2
    Node newH = new Node (-1)
    Node t = newH

while ( cur1 != NULL && cur2 != NULL) {
    if ( cur1.data < cur2.data) {
        t. next  = cur1
        cur1 = cur1. next
         t  =  t. next
    }
    else { // cur2.data ≤ cur1.data
        t.next = cur2
        cur2 = cur2.next
         t =  t.next
    }
}
```

```
if (cur1 != NULL){
    // LL2 empty
    t.next = cur1
}

if (cur2 != NULL){
    t.next = cur2
}

return newH.next
}
```

// LL1 empty

size of LL1

TC : O(N+M)

SC : O(1) size of LL2

Code works for following edge cases

① Both LL empty

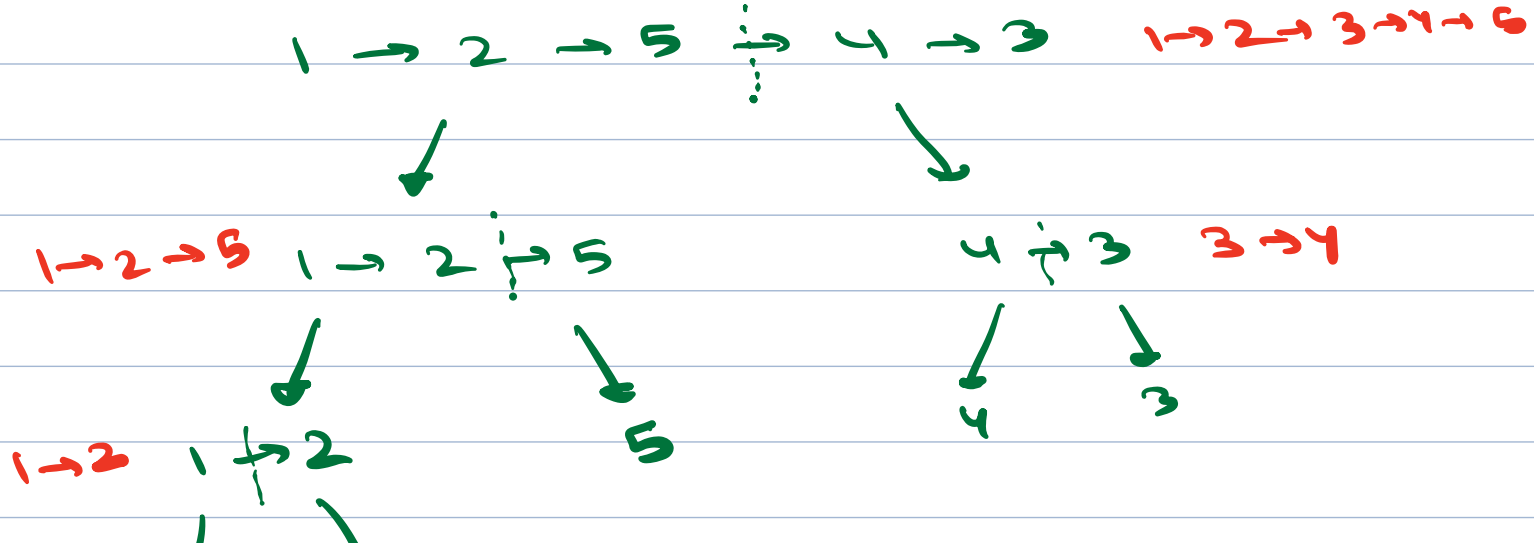
② 1 LL is empty and other is non-empty

10:40

3. Sort the LL using merge sort.

Ex: 1 → 2 → 5 → 4 → 3

ans: 1 → 2 → 3 → 4 → 5



Approach : Apply merge sort algo

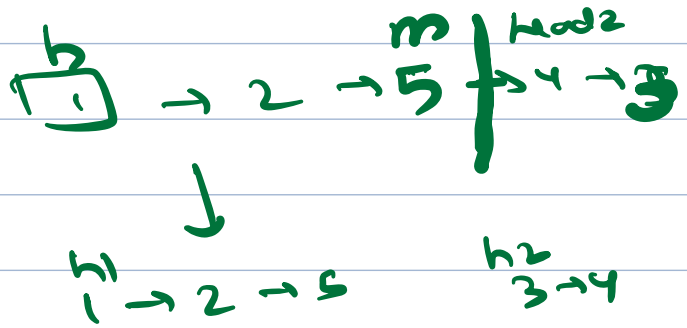
Base case : Single element , return

① Find Mid → Divide LL into 2 piece
② Individually sort both piece
↓
Recursively call merge sort fn
③ Merge both sorted LL
④ Return

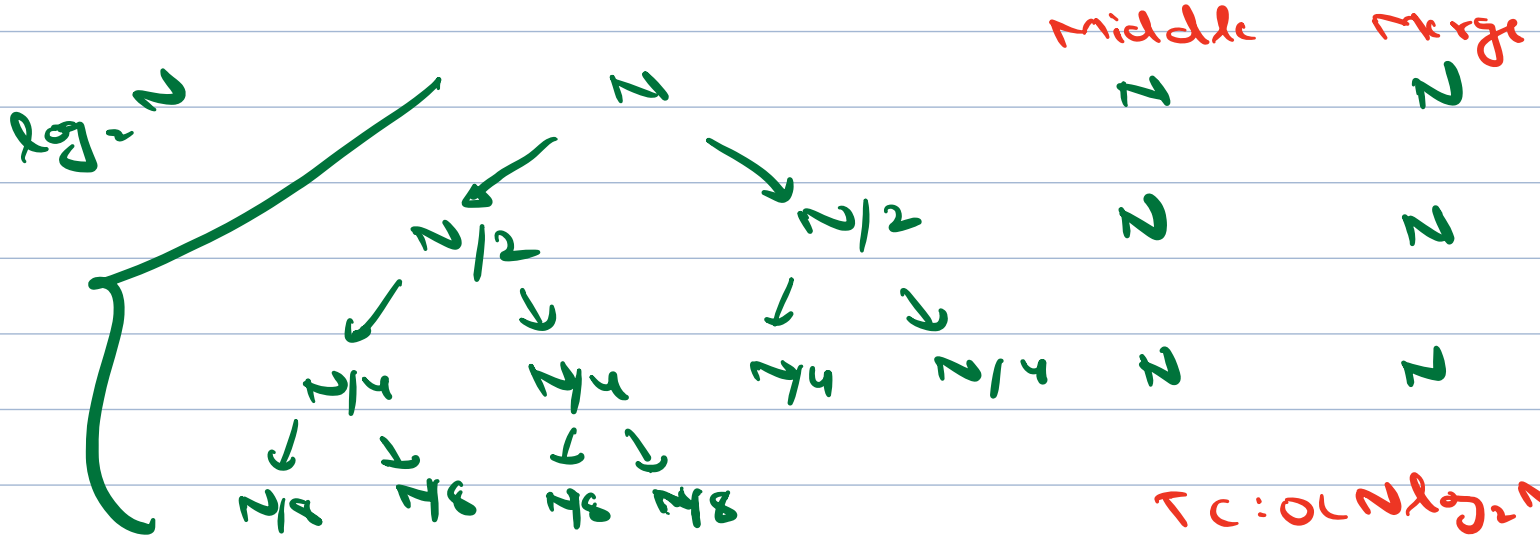
// Given head of LL, fn will sort LL and return of updated head

```
Node mergeSort (Node head) {
    if (head == NULL || head.next == NULL)
        return head

    Node m = findMiddle(head)
    Node head2 = m.next
    m.next = NULL

    Node h1 = mergeSort(head)
    Node h2 = mergeSort(head2)
    return merge(h1, h2)
}
```

Prewritten fn



Recursion

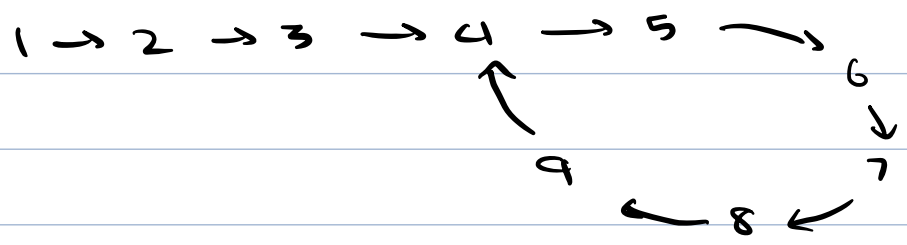


TC: O(NlogN)

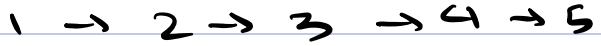
SC: O(logN)

---

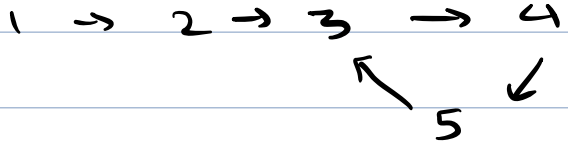
4. Detect cycle in linked list.



ans: true



ans: false

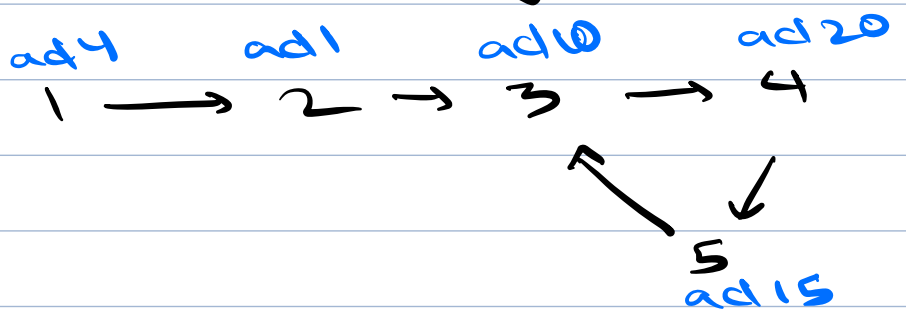


ans: true

Cycle → some nodes are repeating

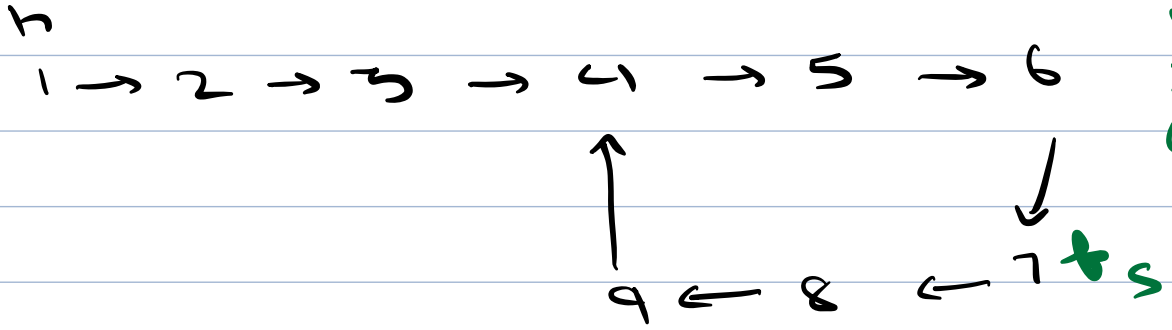
Approach 1: As we traverse LL, we maintain visited nodes in HS. If any node is already in HS, there is a cycle.
Else if you reach NULL, no cycle.

HS → Node Reference



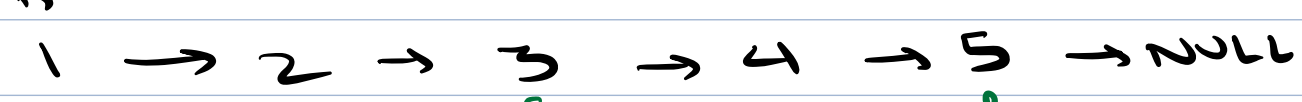
HS <Node>

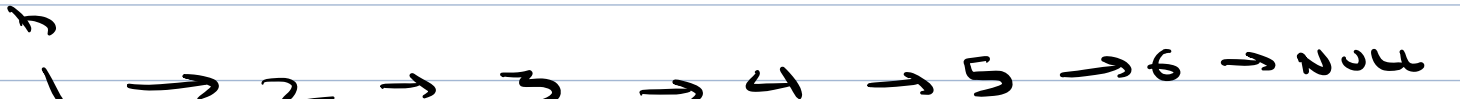
TC: O(N) SC: O(N) hs.add (temp)



		dist
s	f	
4	7	3
5	9	2
6	5	1
7	1	0

Keep slow and fast at head. We move slow by 1 and fast by 2 → if they meet at any point → loop

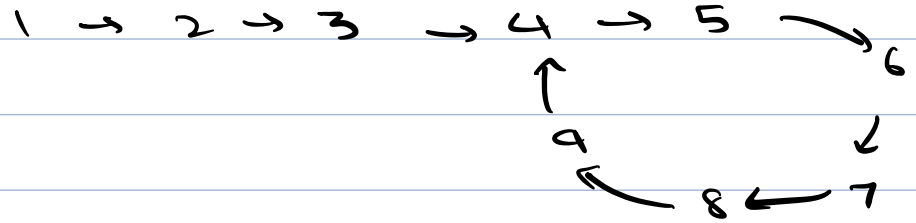




```
bool hasCycle (Node head) {
    Node slow = head, fast = head

    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)

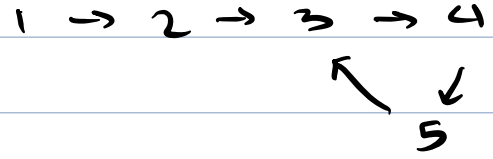
5. Find the ~~meeting~~ start point of the cycle if there is any cycle.



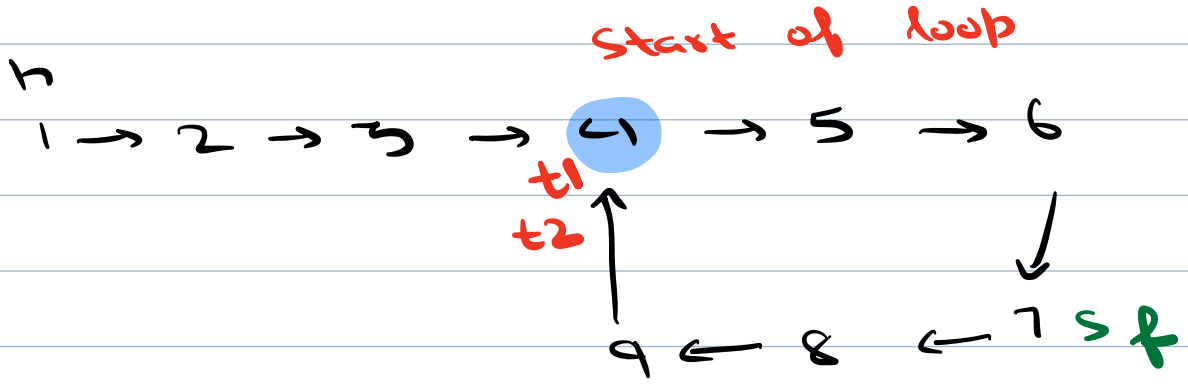
ans : 4

1 → 2 → 3 → 4 → 5

ans : NULL



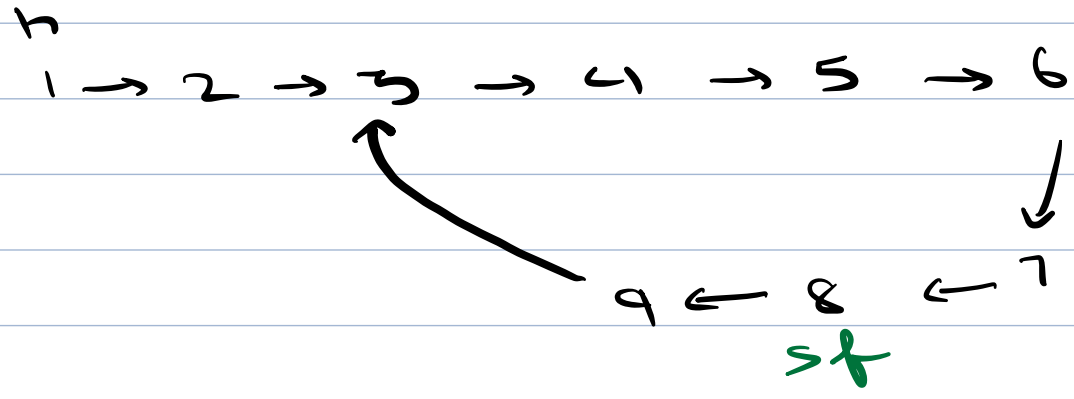
ans : 3



① Find the pt. where s and f meet
↓
whether a cycle exists

② t1 = head , t2 = meeting point
Keep moving both t1 and t2 till they meet

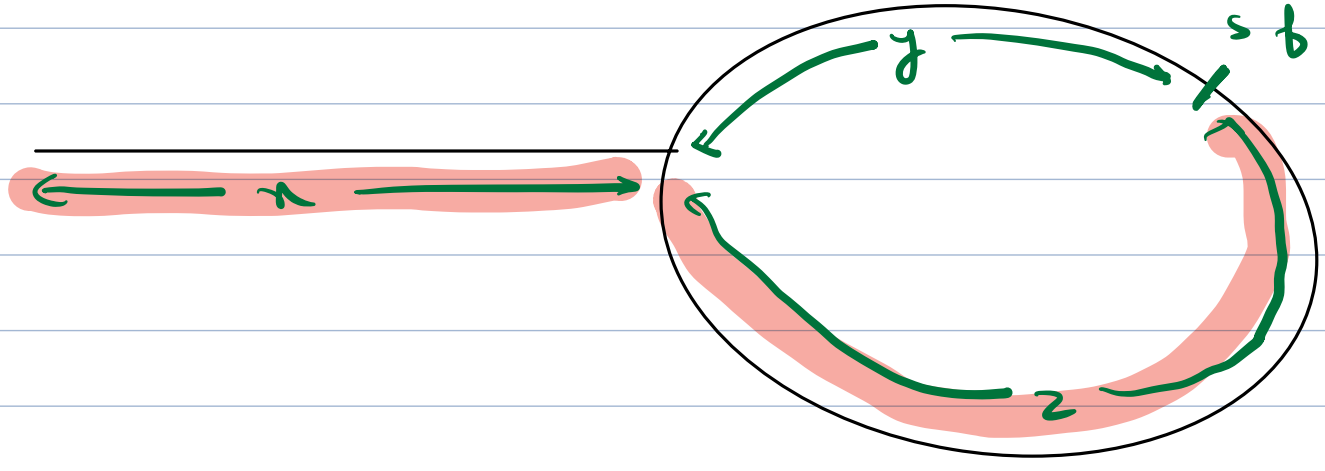
③ t1 and t2's point → start of cycle



```
Node startPointCycle(Node head) {
    Node slow = head, fast = head
    while (fast != NULL && fast.next != NULL) {
        slow = slow.next
        fast = fast.next.next
        if (slow == fast) {
            Node t1 = head, t2 = slow
            while (t1 != t2) {
                t1 = t1.next    t2 = t2.next
            }
            return t1
        }
    }
    return NULL
}
```

TC : O(N)

SC : O(1)



Dist travelled by $s = x + y$

Dist travelled by $f = x + y + y + z$

$= x + 2y + z$

$$Dist_f = 2 \times Dist_s$$

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

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

$$z = 2x - x$$

$$\Rightarrow z = x$$

HW : Break the loop