Linked List 2

I will not erase all my hard work because it is the weekend!

Good Evening

Today's content

→ Mid of Linkedlist

a) Merge two sorted Linkedlist

b) Merge Sort

c) Cycle detection

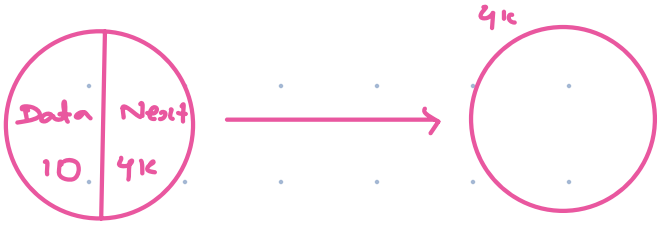
(i) Detect cycle

(ii) Find start of cycle

(iii) Remove cycle

Main advantage of using linked list

→ It doesn't require continous memory just like arrays.



Time complexity to search a value x in a linkedlist

TC: O(n)

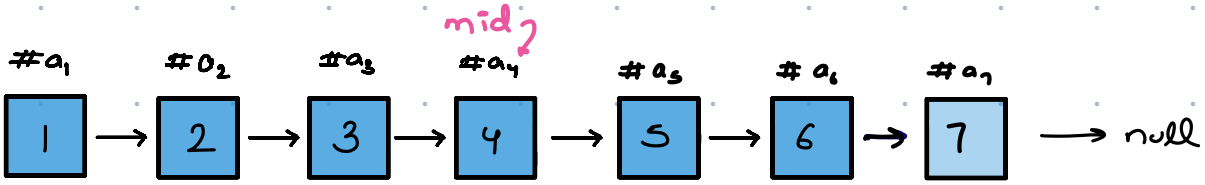
Check for a palindrome using O(n) time & O(1) space

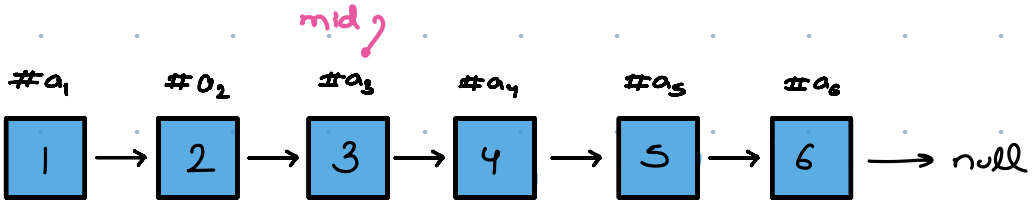
↳ Middle of LL, reverse second half & the compare values.

Time complexity for deleting a node → TC: O(n)

Binary search on sorted Linkedlist → TC: O(n)

* Given head of Linkedlist, find middle node of Linkedlist.





} return 1st mid

Brute force → Count all the nodes in a LL

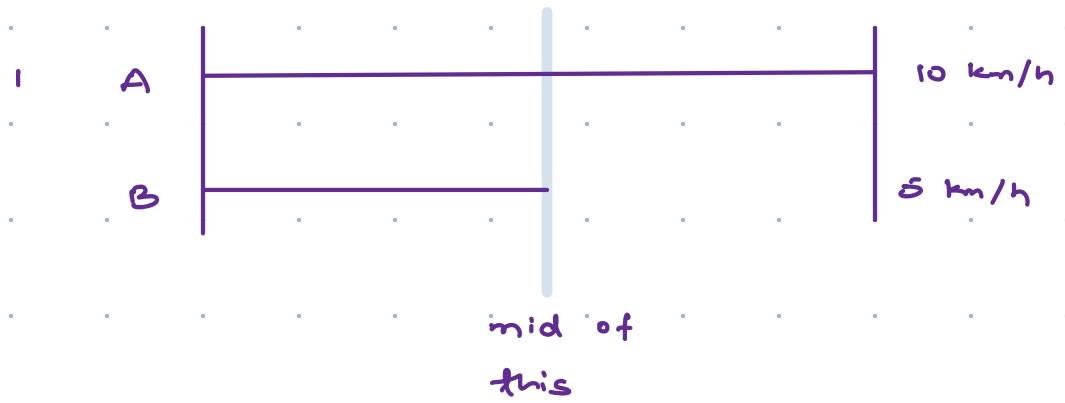
if (count % 2 == 0) return $\frac{count}{2}^{th}$ node

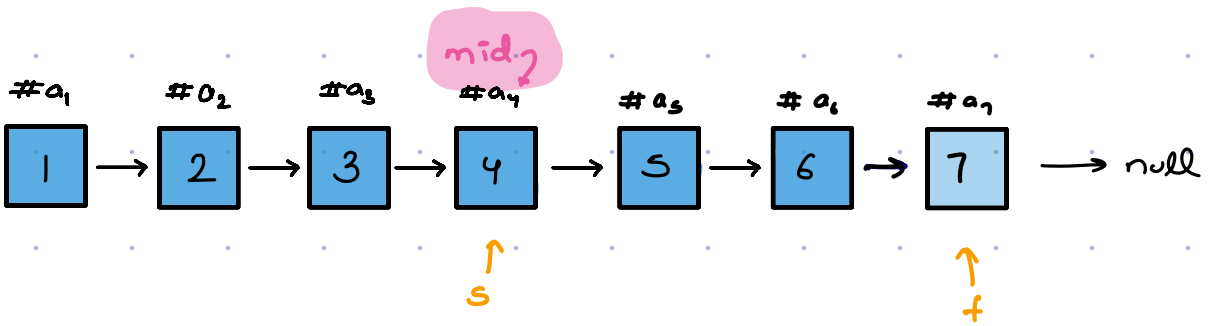
else traverse till $\left(\frac{count}{2}+1\right)^{th}$ node.

TC : O(n)

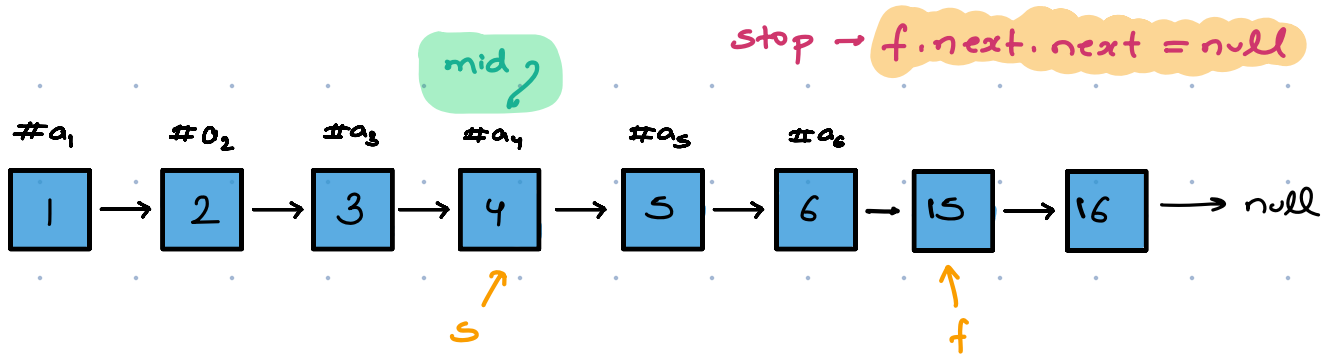
SC : O(1)

Idea 2





stop → f.next = null



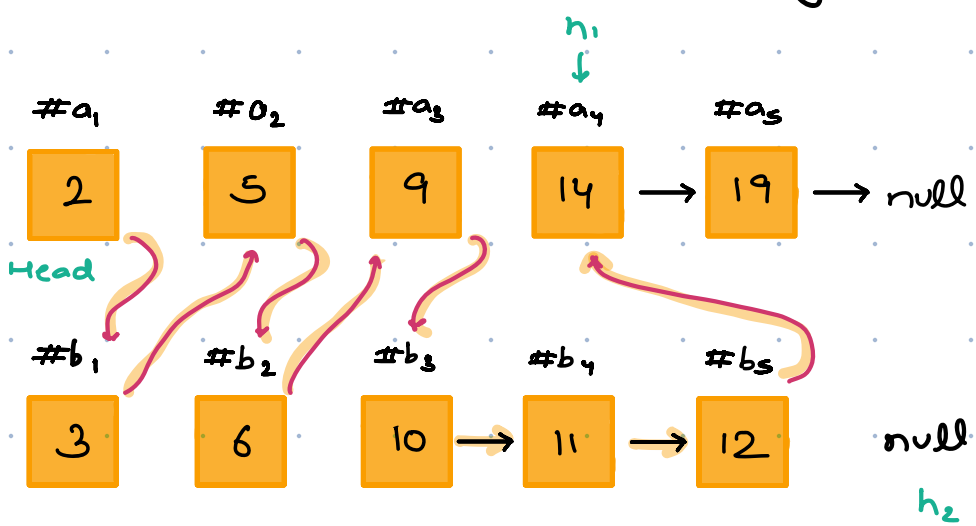

```
Node middle (Node head)
{
    if (head == null) return null;
    Node slow = fast = head;

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

TC : O(n)

SC : O(1)

Q Given 2 sorted linkedlist, merge & get final sorted list



```
Node merge (Node h1, Node h2)
    if (h1 == null) return h2;
    if (h2 == null) return h1
    Node head = null;
    if (h1.data ≤ h2.data) {
        head = h1;
        h1 = h1.next;
    }
    else {
        head = h2
        h2 = h2.next;
    }
    Node temp = head;
    while (h1 != null && h2 != null) {
        if (h1.data < h2.data) {
            temp.next = h1
            h1 = h1.next;
        }
        else {
            temp.next = h2
            h2 = h2.next;
        }
        temp = temp.next;
    }
```

TC : O(m+n)

SC : O(1)

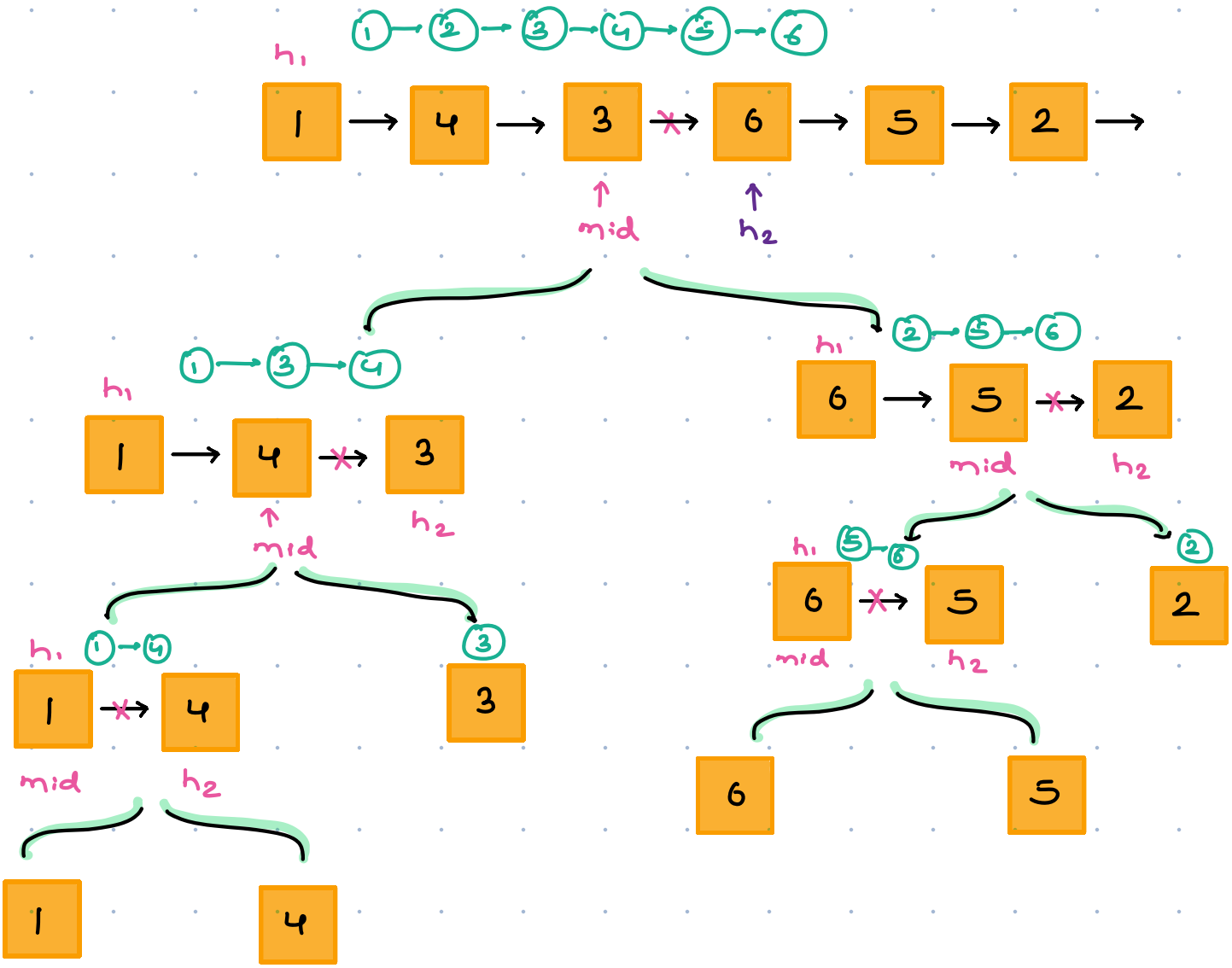
```
        if (h1 == null)  temp.next = h2;

        if (h2 == null)  temp.next = h1

        return head;
    }
```

* Sort the Linked list

Merge Sort (Divide & Conquer Technique)



```
Node mergeSort (Node h1){
    if (h1 == null || h1.next == null) return h1;
    Node mid = middle(h1)
    Node h2 = mid.next;
    mid.next = null;
    Node t1 = mergesort(h1)
    Node t2 = mergesort(h2)
    return merge(t1, t2)
}
```

TC : O(nlogn)

SC : O(logn)

3

10:12pm → 10:22 pm

You are using **Google Maps** to help you find your way around a new place. But, you get lost and end up walking in a circle. **Google Maps** has a way to keep track of where you've been with the help of special **sensors**.

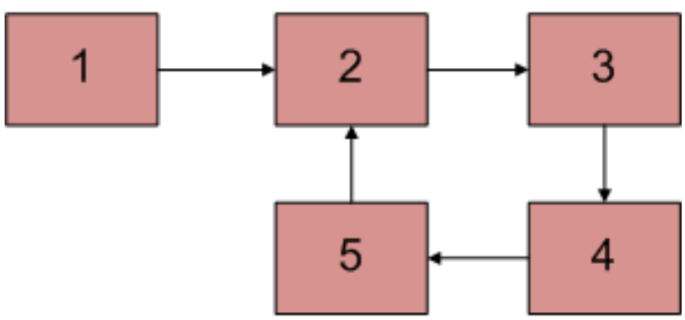
These sensors notice that you're **walking in a loop**, and now, **Google** wants to create a new feature to help you figure out exactly where you started going in circles.

Problem

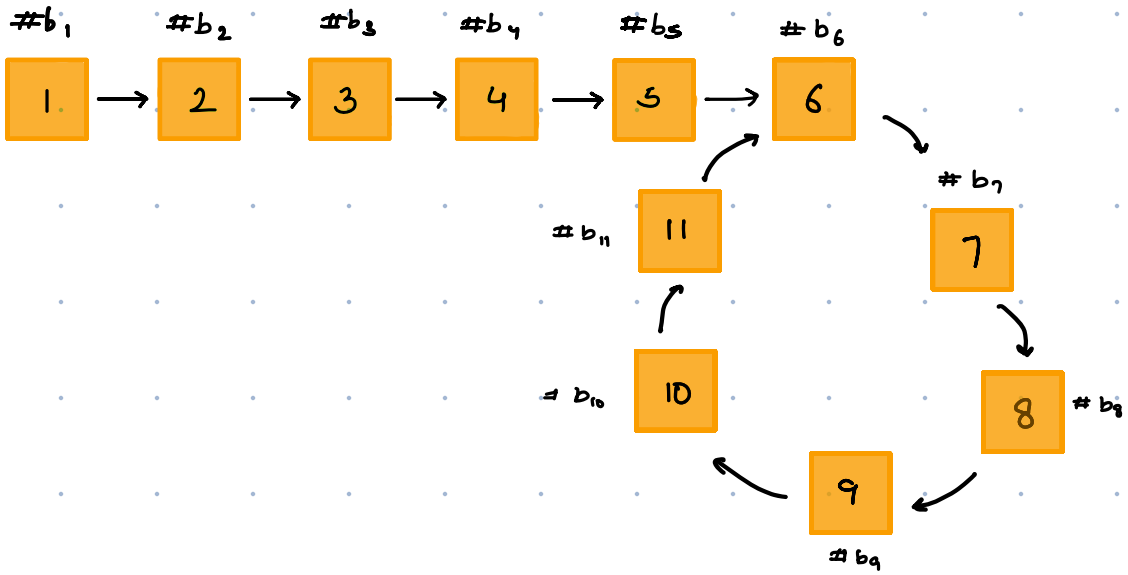
You have a **linked list** that shows each **step** of your **journey**, like a chain of events. Some of these steps have accidentally led you back to a place you've already been, making you **walk in a loop**. The goal is to find out the exact point where you first started walking in this loop.

Example

Input:



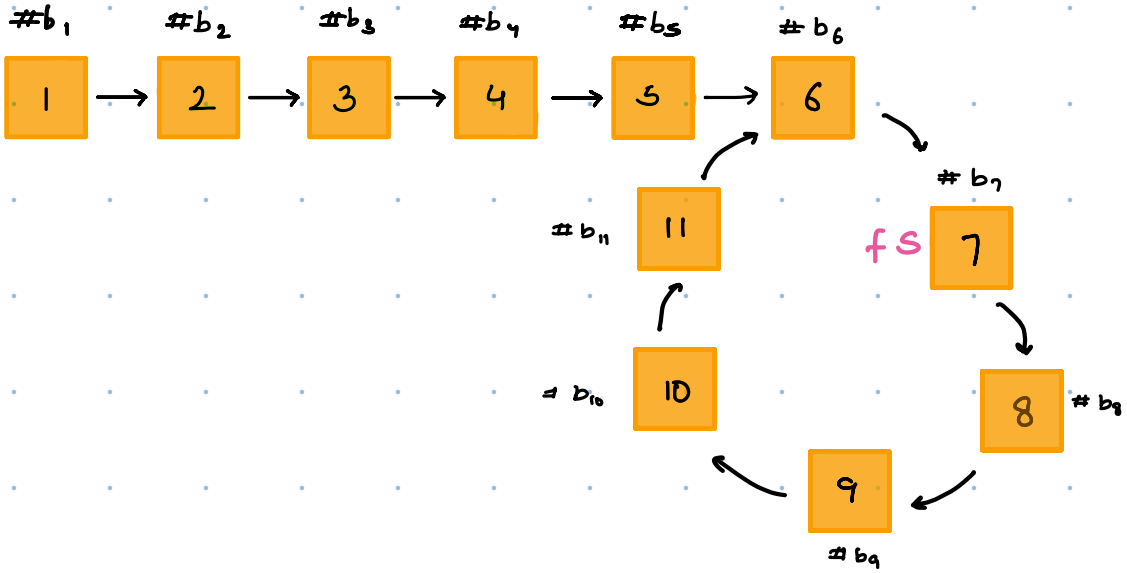
03. Given a head node of Linkedlist, check for cycle detection?

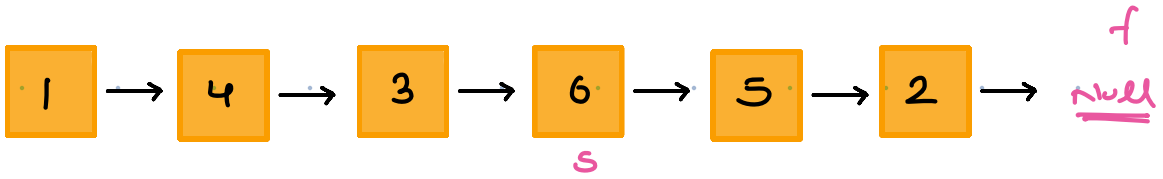


Brute force → Create a hashset & store node addresses, if any address is already present? → cycle present

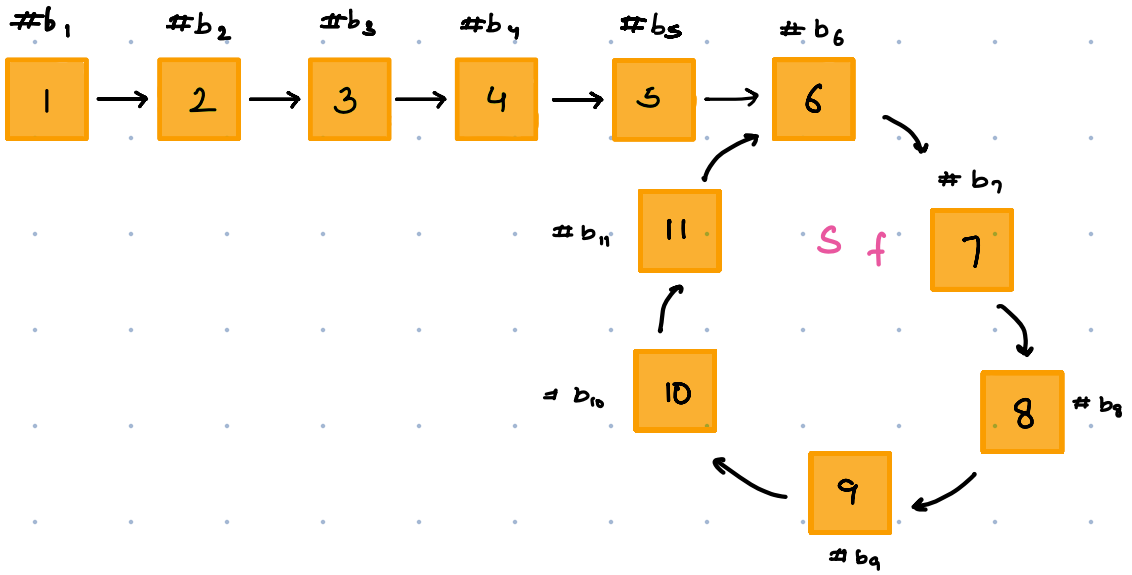
TC: O(n)
SC: O(n)

* No extra space is allowed



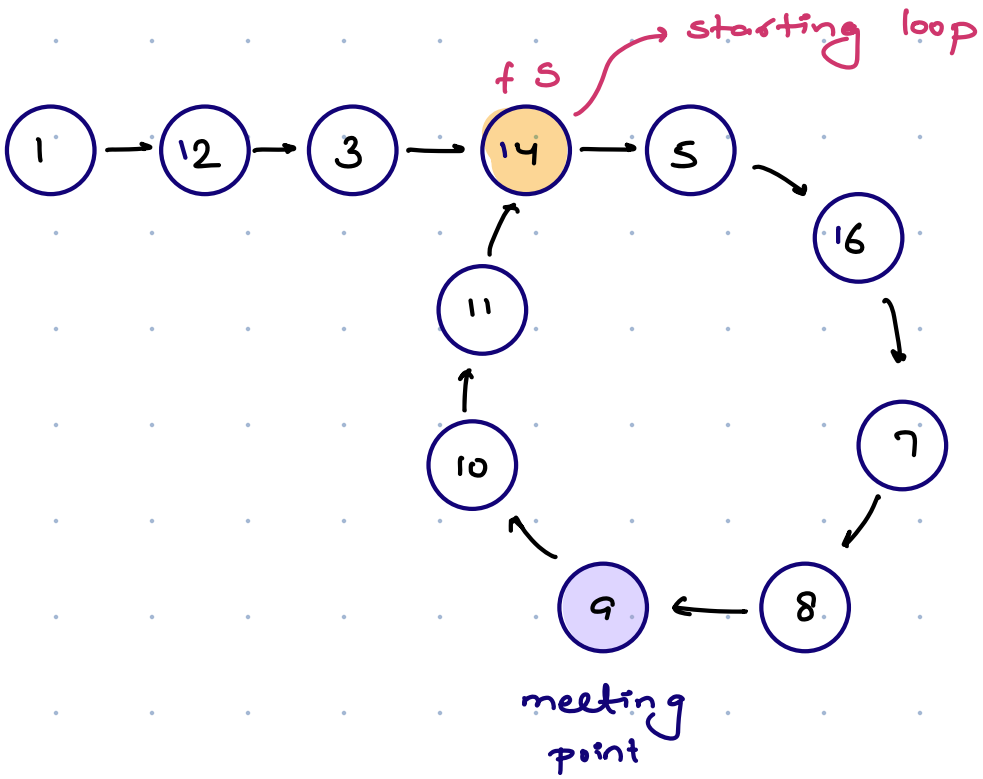


Note → We can go & move fast pointer with speed of 3x, 4x, 5x .... but in this case fast ptr can bypass slow ptr & thus more no. cycles.




```
boolean detectCycle (Node head){
    Node s = f = head;
    while (f != null && f.next != null){
        s = s.next
        f = f.next.next;
        if (s == f) return true;
    }
    return false;
}
```

TC : O(n)
SC : O(1)



Steps

01. Find if cycle is present.
02. Move slow ptr to head.
03. Move slow & fast ptr by 1 step until they meet.

Starting point of cycle

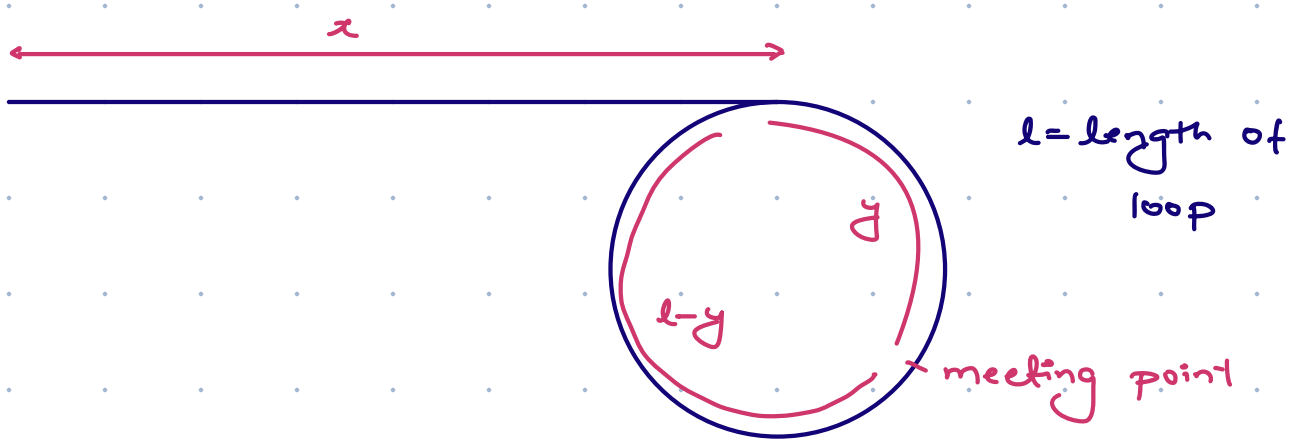
```
Node p1 = fast
Node p2 = head

while ( p1 != p2 ) {
    p1 = p1.next;
    p2 = p2.next;
}
return p1
```

Remove cycle

On your own

* Mathematical proof for P_1 & P_2 meeting at loop starting



Speed of fast ptr = 2 * speed of slow ptr

No. of rotations taken by fast ptr = R_1

No. of rotations taken by slow ptr = R_2

Distance covered by fast ptr = $x + R_1 \cdot l + y$

Distance covered by slow ptr = $x + R_2 \cdot l + y$

$$\text{Time} = \frac{\text{Distance}}{\text{speed}}$$

$$\frac{D_f}{S_f} = \frac{D_s}{S_s}$$ } Meeting time will be same

$$\frac{D_f}{2S_s} = \frac{D_s}{S_s}$$

$$D_f = 2D_s$$

$$x + R_1 l + y = 2 * (x + R_2 l + y)$$

$$x + R_1 l + y = 2x + 2R_2 l + 2y$$

$$R_1 l - 2R_2 l = 2x + 2y - x - y$$

$$R_1 l - 2R_2 l = x + y$$

$$\underbrace{(R_1 - 2R_2)}\, l = x + y$$

$$R_3 l = x + y$$

$$x = R_3 l - y$$

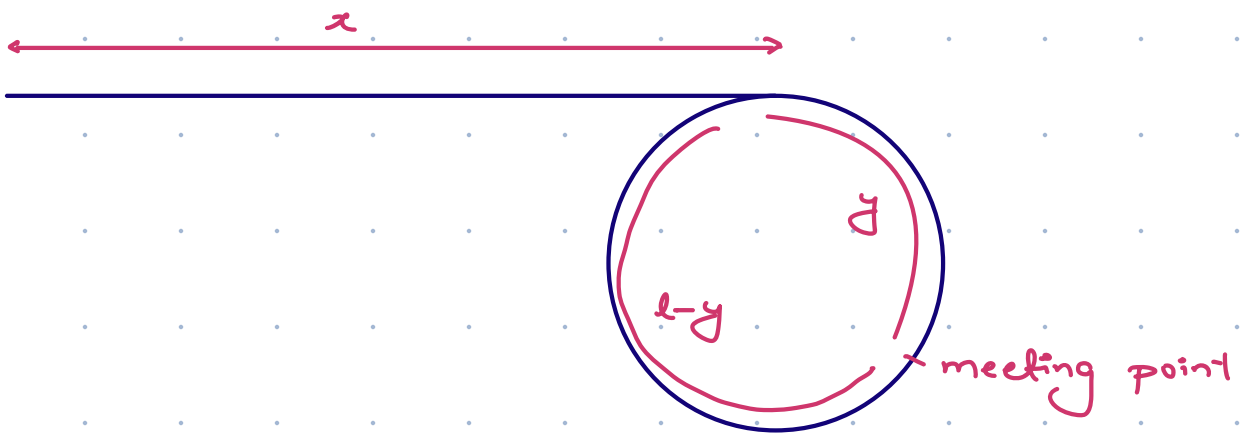
$R_3 = 1$ $\quad x = l - y$

$R_3 = 2$ $\quad x = 2l - y$

$= l + (l - y)$

$R_3 = 3$ $\quad x = 3l - y$

$x = 2l + (l - y)$


x
y
l-y
meeting point