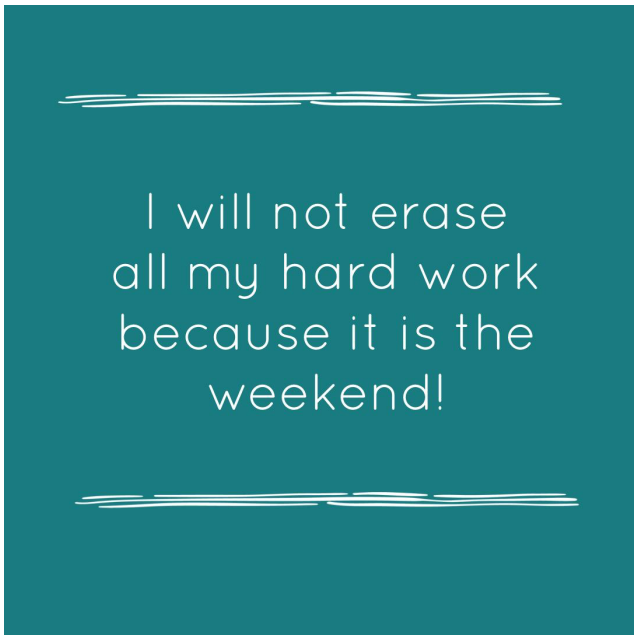
Linked List 2



Good Evening

Today's content

→ Mid of Linkedlist

a) Merge two sorted Linkedlist

b) Merge Sort

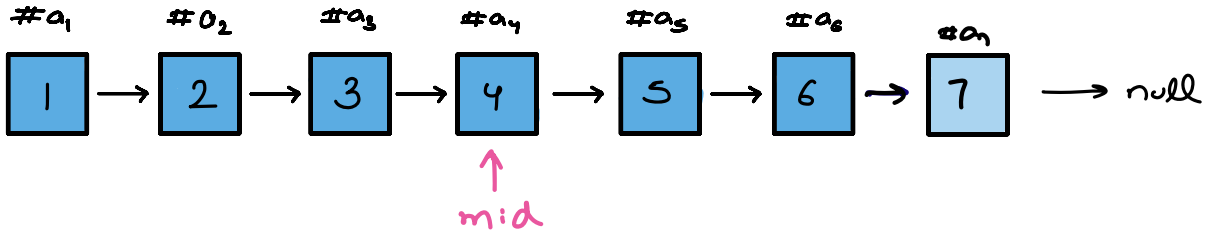
c) Cycle detection

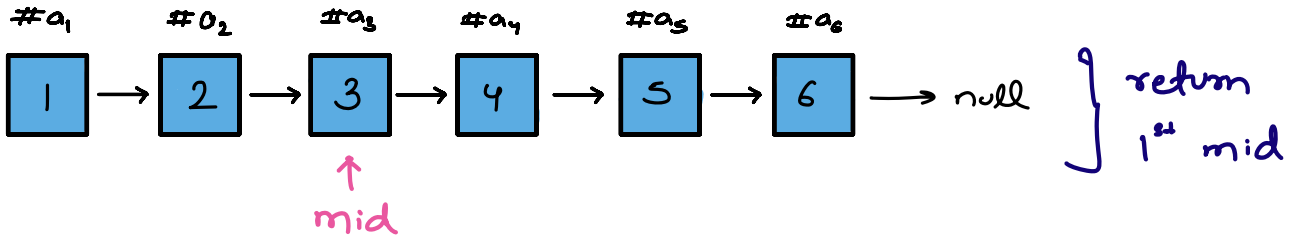
(i) Detect cycle

(ii) Find start of cycle

(iii) Remove cycle

01. Given head node, find middle of linkedlist.





BF → Count the no. of nodes

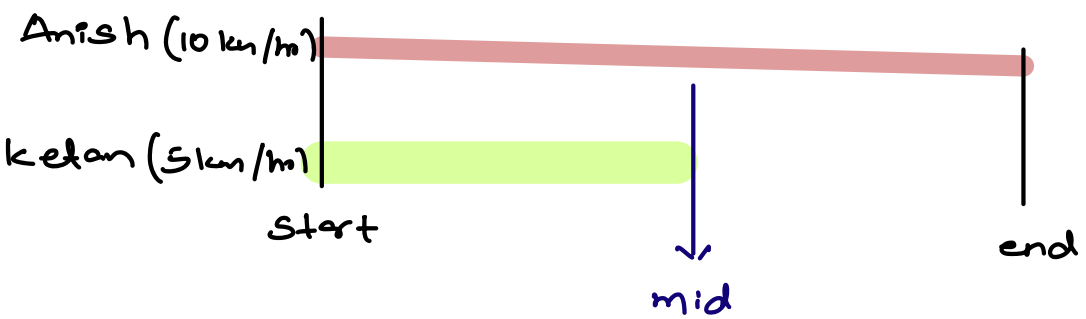
if (count % 2 == 0) iterate for count/2

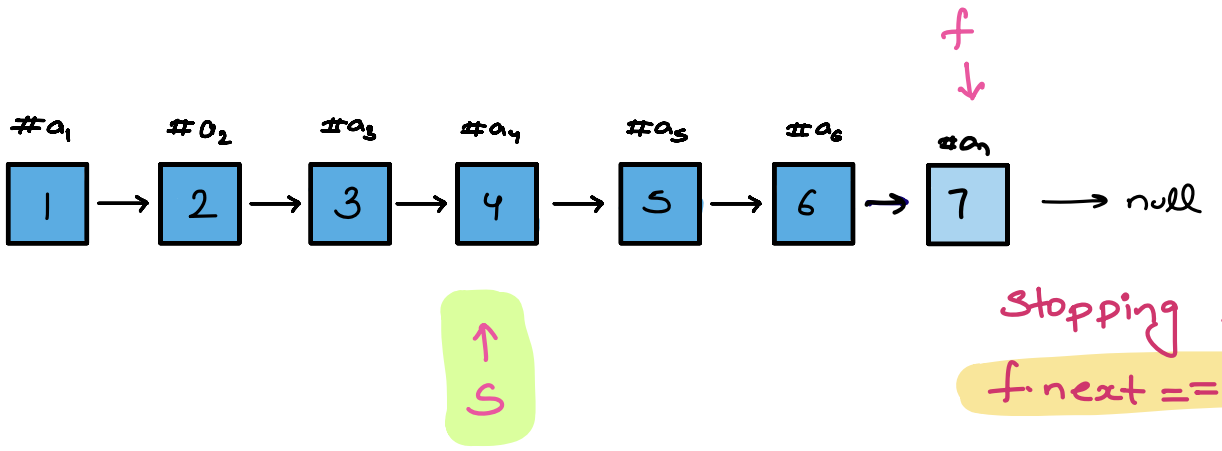
else traverse from head to count/2 + 1

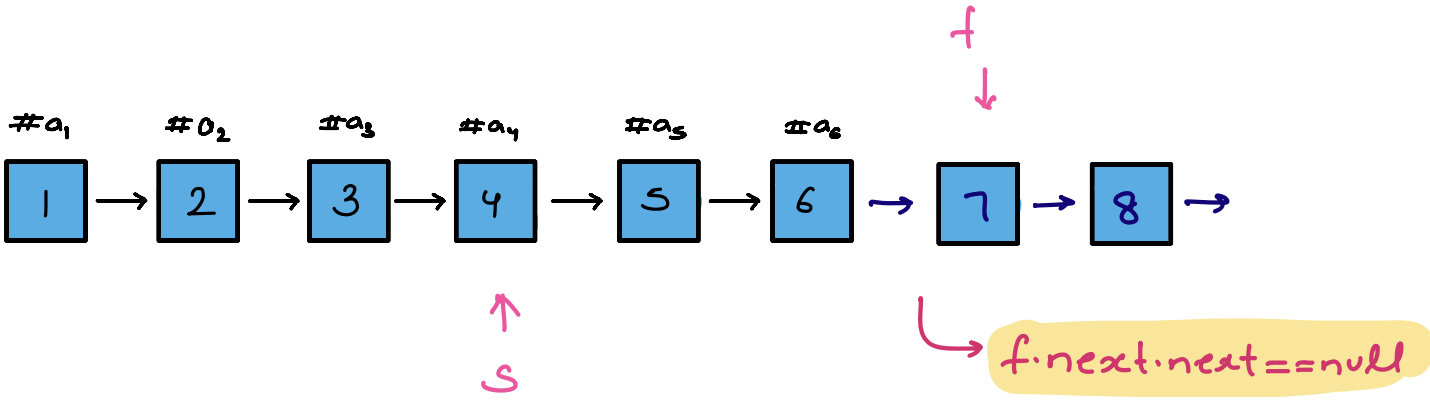
TC: O(n)

SC: O(1)

Idea 2 → slow & fast ptr





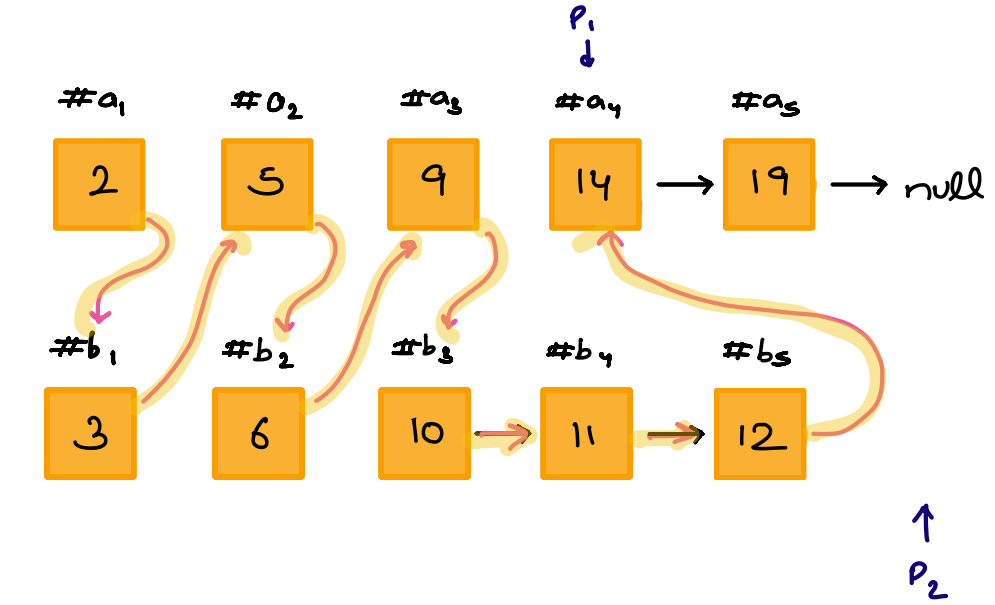



```
Node middle (Node head)
    if (head == null) return head;
    Node slow = head, fast = head;
    while (f.next != null && f.next.next != null)
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

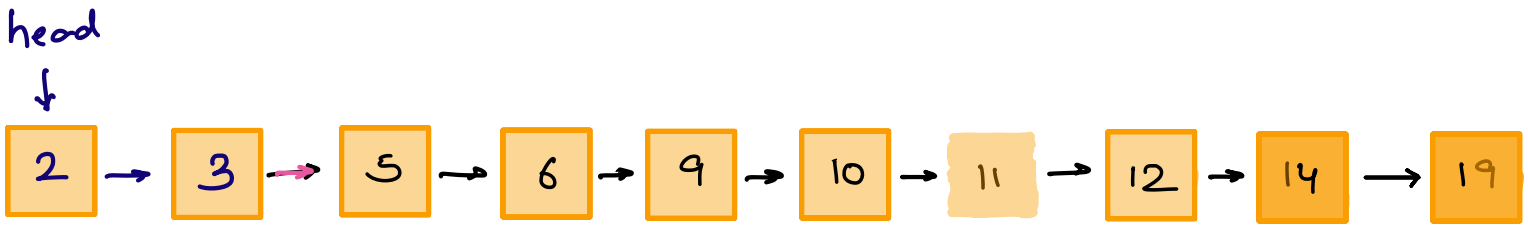
TC: O(n)

SC: O(1)

Given 2 sorted linkedlist, merge & get final sorted list

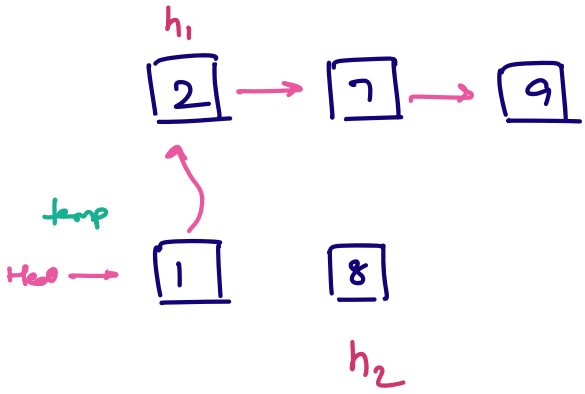


SC: O(1)

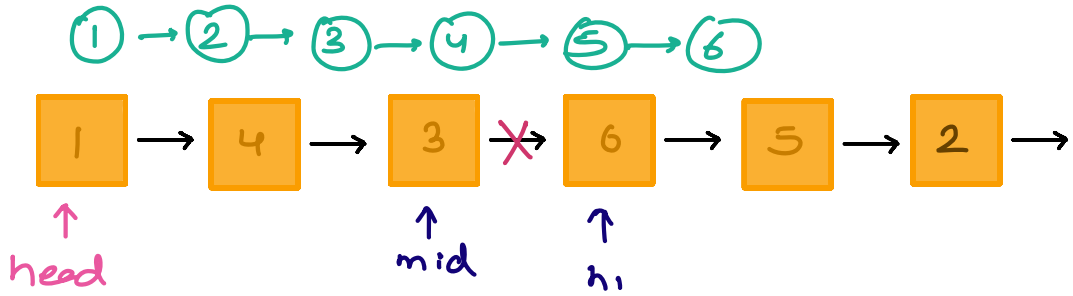


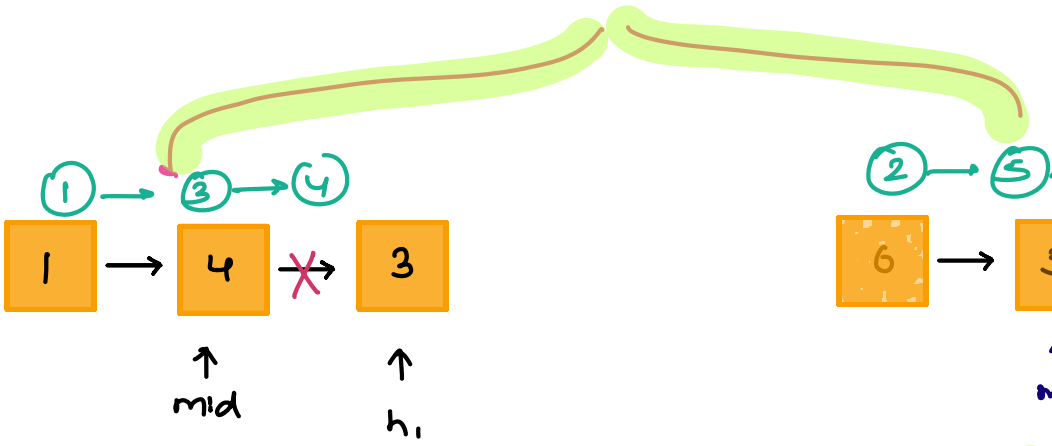

```
Node merge (Node h1, Node h2)
    if (h1 == null) return h2
    if (h2 == null) return h1
  Node head = null;
  if (h1.data ≤ h2.data){
  |   head = h1;
  |   h1 = h1.next;
  }
  else {
  |   head = h2
  |   h2 = h2.next;
  }
```

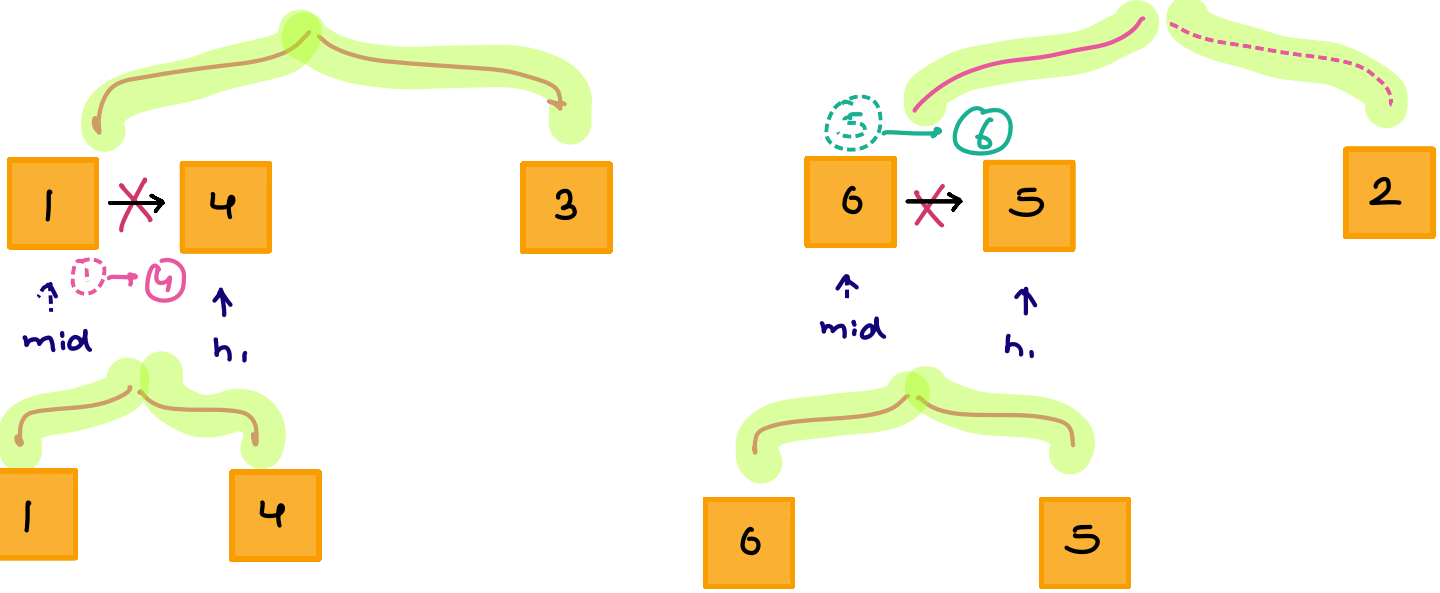
```
Node temp= head:

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;

}

if (h1 != null){

    temp.next = h1;

}

if (h2 != null){

    temp. next = h2;

}

return head:
```



Merge Sort (Divide & Conquer)







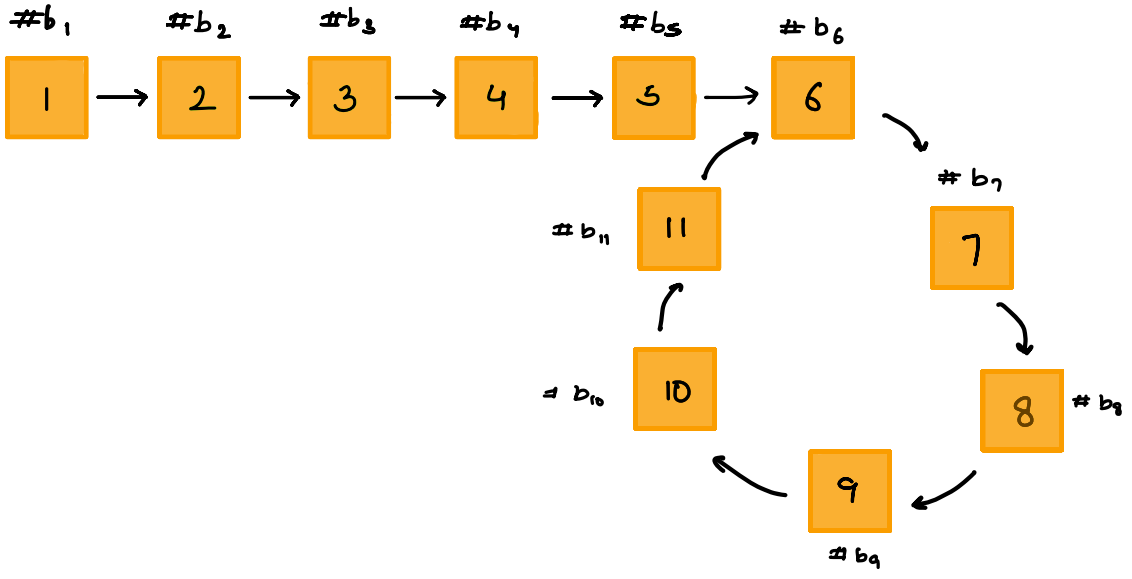
```
Node mergeSort(Node head)
{
    if (head == null || head.next == null) return head;

    Node mid = middle(head);  → O(n)

    Node h1 = mid.next;
    mid.next = null;

    Node t1 = mergeSort(head);

    Node t2 = mergeSort(h1)

    return merge(t1, t2);
}
```

TC: O(nlogn)

SC: O(logn)

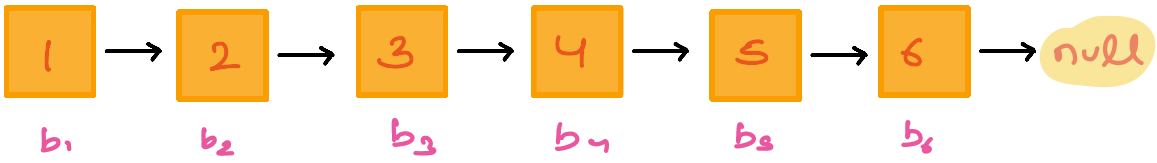
10:03 pm → 10:13 pm

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

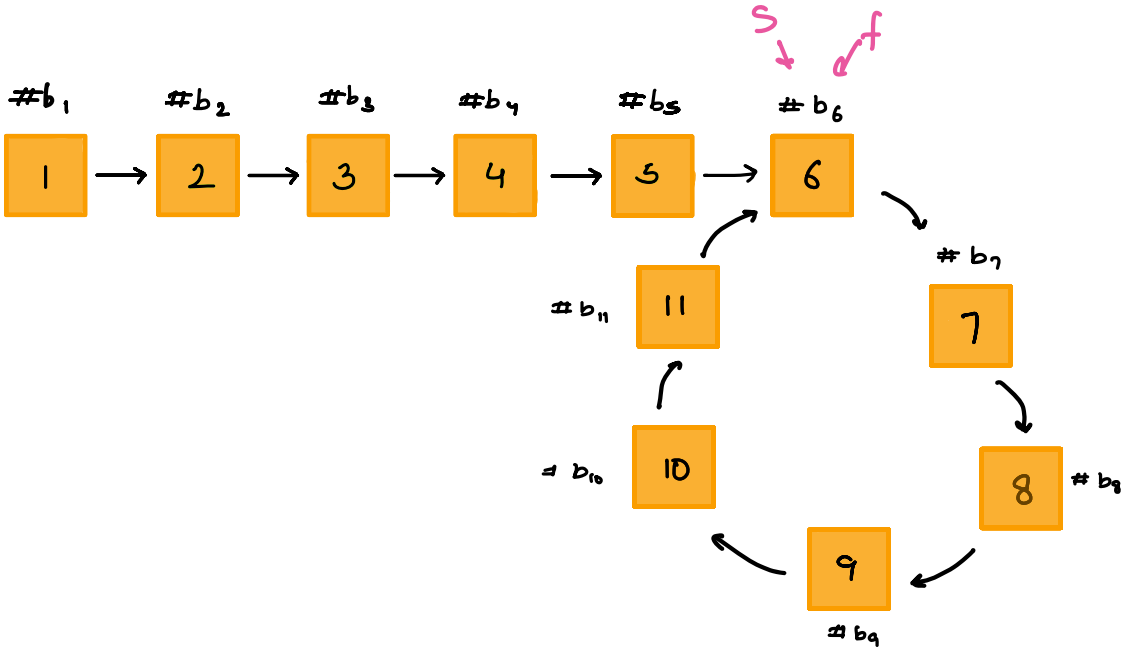


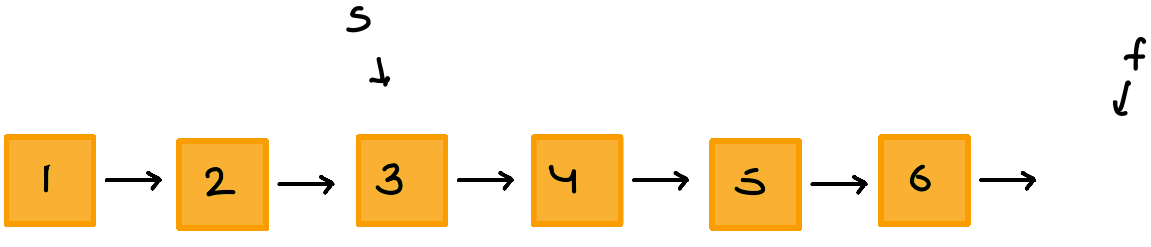
Brute force → Create a hashset & store addresses comparing if it is already present → a cycle is present

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

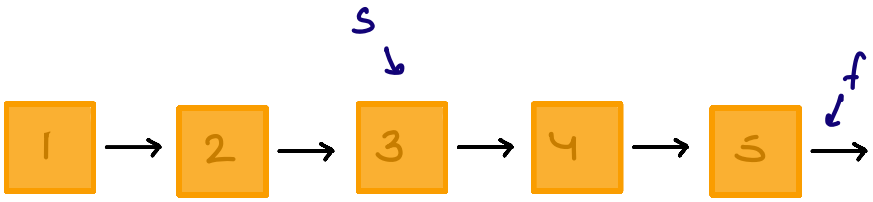


* No extra space is allowed





if (f == null) No cycle



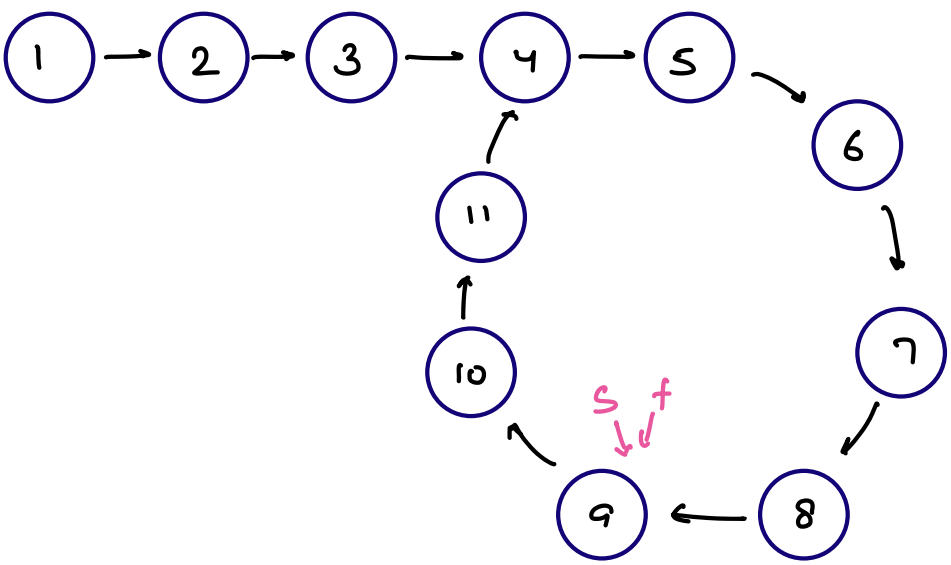
if (f.next == null)

```
boolean detectCycle(Node head)
{
    Node slow = head, fast = head

    while(fast != null && f.next != null){
        s = s.next;
        f = f.next.next;
        if(s == f){ return true; }
    }
    return false;
}
```

TC:

SC:

* Find start of cycle

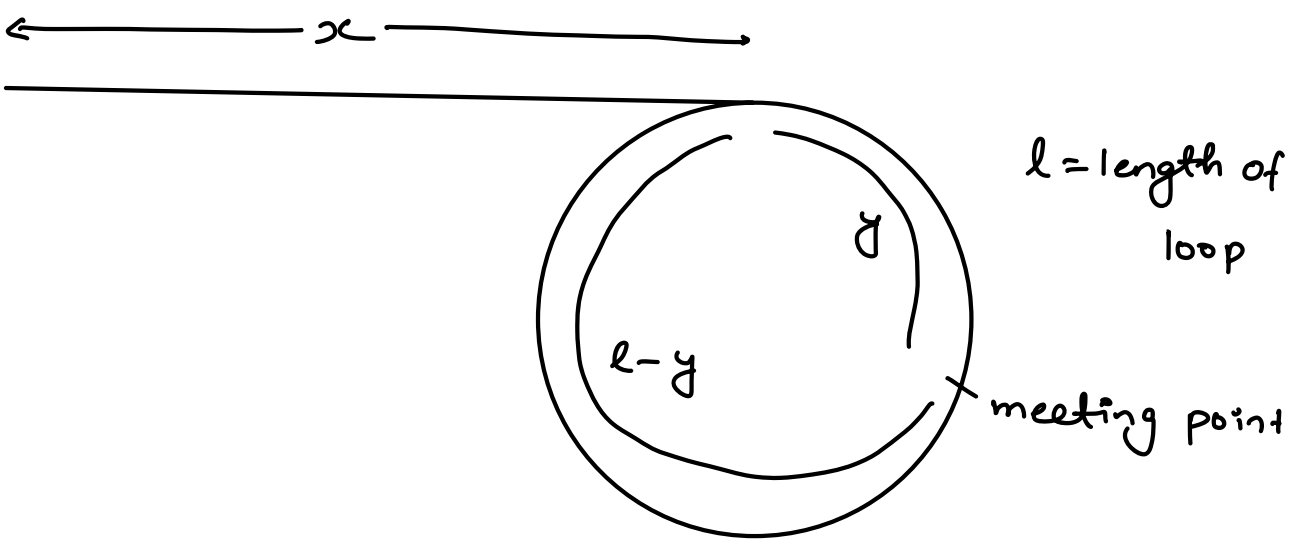


Node $P_1 = f$ Node $P_2 = head$

```
while (P1 != P2) {
    P1 = P1.next;
    P2 = P2.next;
}
return P1;
```

Remove cycle

```
Node temp = P1
while (temp.next != P1) {
    temp = temp.next;
}
temp.next = null;
```



Speed of fast ptr = 2 * speed of slow ptr

No. of rotations for fast ptr = R_1

No. of rotations for slow ptr = R_2

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

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

$$\text{Distance} = s * t$$

$$\text{time} = \frac{D}{S}$$

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

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

$$D_f = 2 * D_s$$

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

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

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

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

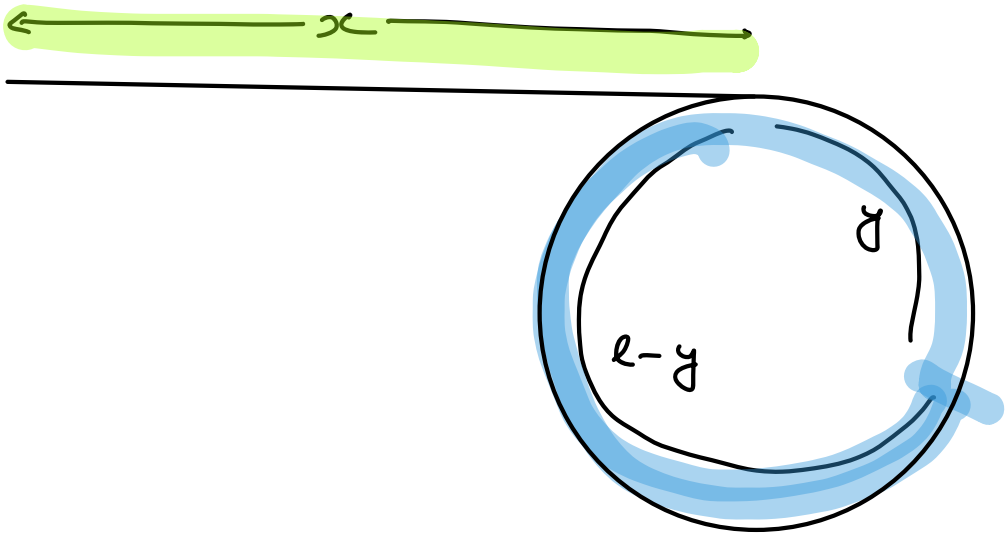
$$R_3 \cdot l = x + y$$

$$x = R_3 \cdot l - y$$

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

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

$x = l + (l - y)$



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

$= 2l + (l - y)$