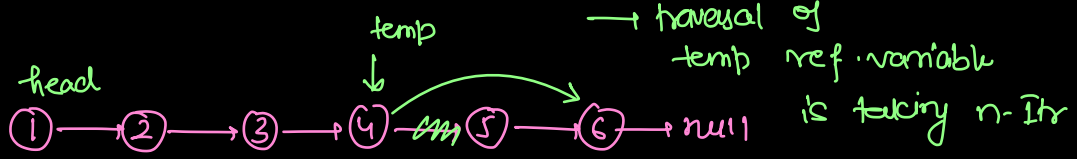
Linked List 2

Agenda

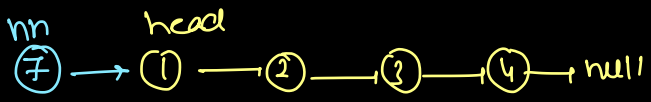
1. Revision Quizzes
2. Find Middle of linkedlist
3. Merge Two Sorted linkedlist
4. Sort the given linkedlist (MergeSort)
5. Google Maps got your back
6. Detect Cycle in the linkedlist
7. Starting point of cycle

Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start session
from 9:06 PM

Revision Quizzes

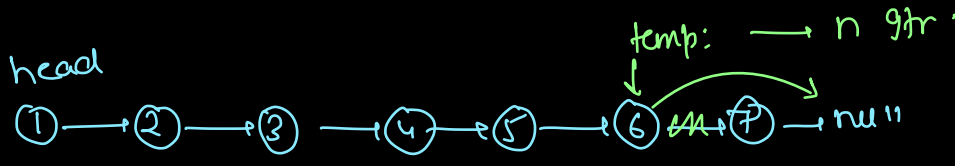


Quiz 1:
What is the time complexity needed to delete a node from a linked list?
[ ] O(1) [✓] O(N)
[ ] O(log(N)) [ ] O(N^2)



nn.next = head
head = nn;

Quiz 2:
What is the time complexity needed to insert a node as the head of a linked list?
[✓] O(1) [ ] O(N)
[ ] O(log(N)) [ ] O(N^2)



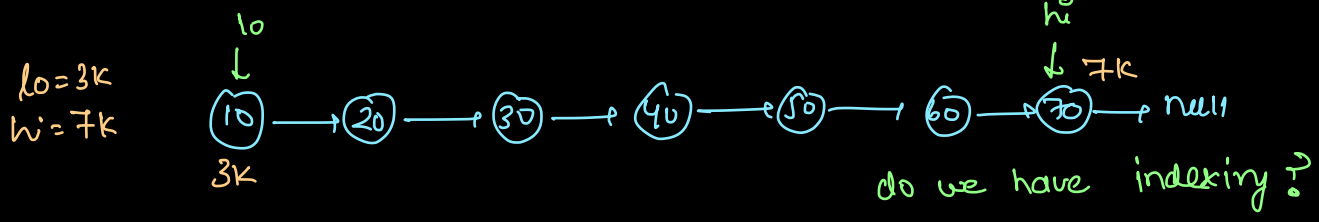
Quiz 3:
What is the time complexity needed to delete the last node from a linked list?
[ ] O(1) [✓] O(N)
[ ] O(log(N)) [ ] O(N^2)

Quiz 4:

Can we do Binary Search in a sorted Linked List?

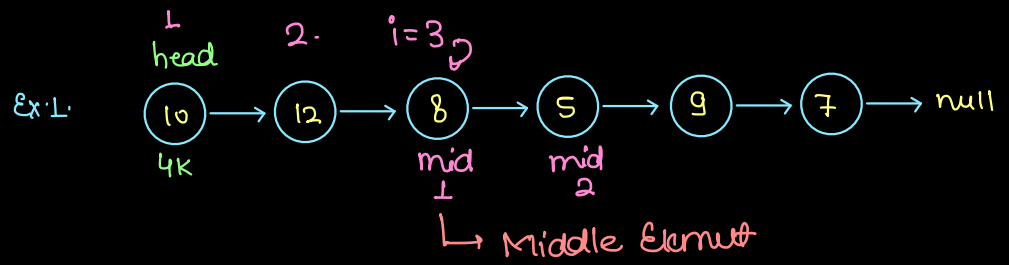
[ ] Yes [ ] No

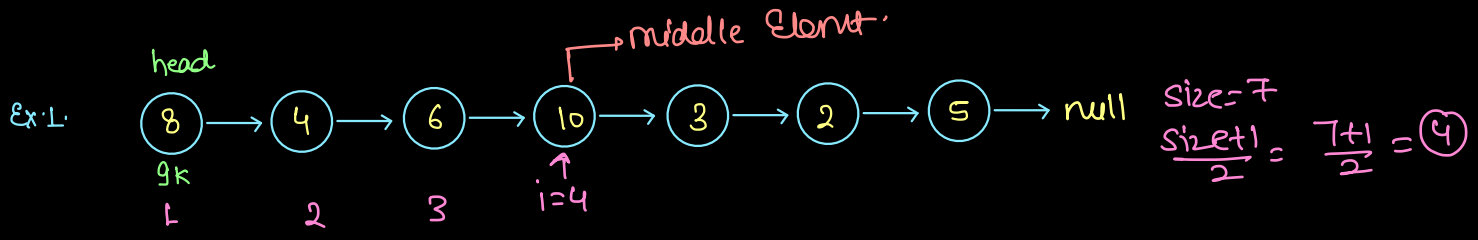
mid = $\frac{lo+hi}{2}$



Find Middle of linkedlist

Given a Linked List, Find the middle element.





Idea 0: Brute force Approach:

1. Calculate size of the linkedlist from head pointer.
2. Iterate from 1 to $\frac{size+1}{2}$ to find middle of LL.

pseudo:

```
Node middleElement( Node head) {
    // calculate size  ⟶ n - iterate

    // iterate from i=1 to i < size+1/2 to find midde ⟶ n/2 itr

}
```

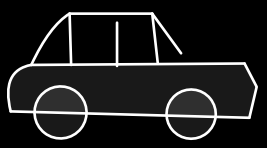
Total itr = $n + \frac{n}{2} = \frac{3n}{2}$

T.C: O(n)
S.C: O(1)

Idea 1: slow and fast pointers:

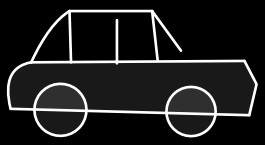
Speed = $\frac{dist}{time}$ ⇒ dist = speed x Time

1. slow
2. fast



Speed of car = x (slow)

$d_{slow} = x * T = xT$

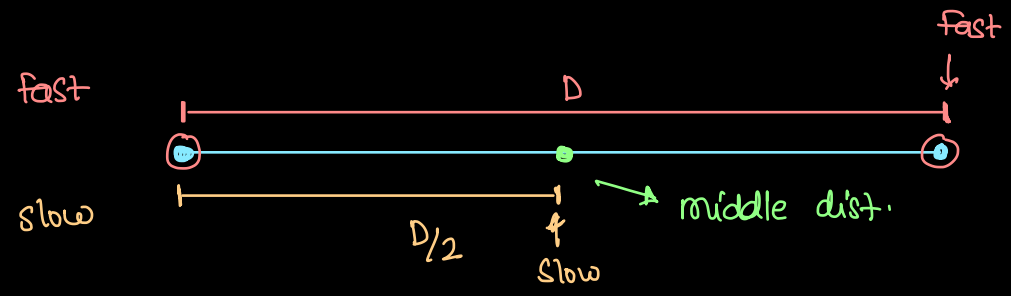


Speed of car = 2x (fast)

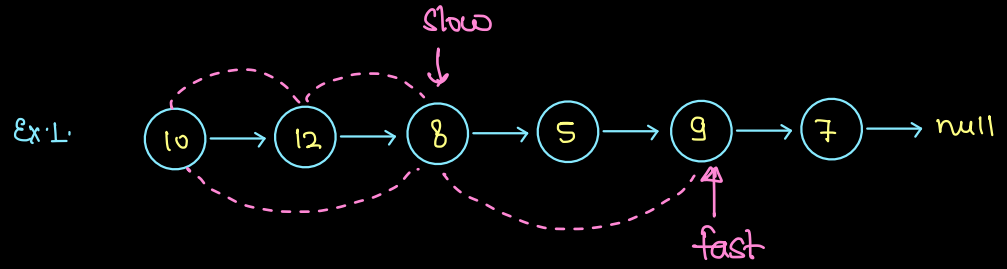
$d_{fast} = 2x * T = 2xT$

$dist(fast) = 2 * dist(slow)$

$\Rightarrow dist(slow) = \frac{dist(fast)}{2}$

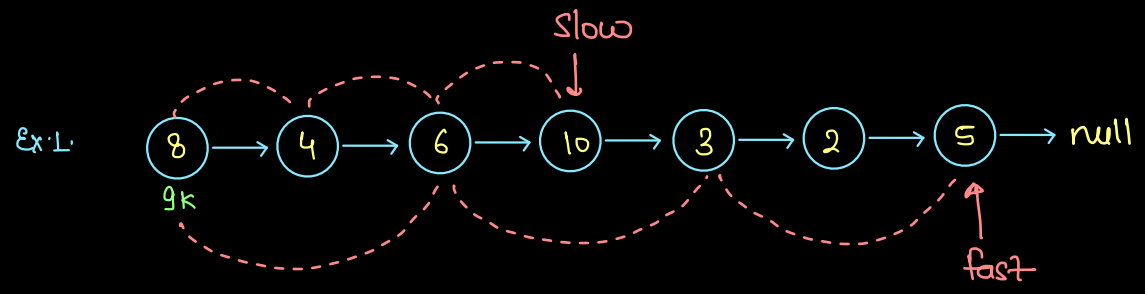


Conclusion:
* Start slow & fast from same point
* When fast reaches at end position of slow is mid point.



Stop condition ⇒ fast.next.next == null

Running condition ⇒ fast.next.next != null → keep go



Stop condition → fast.next == null

Running cond. → fast.next != null → keep gtr ②

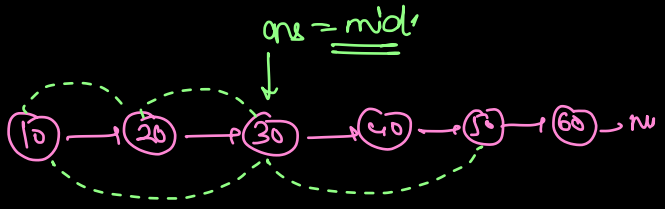
How to write both condition together →

Condition 1 → fast.next.next != null

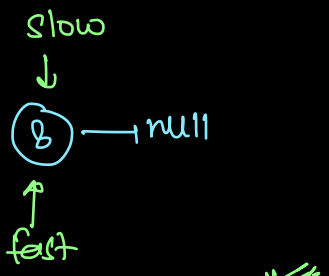
Condition 2 → fast.next != null

Final cond. → Condition 2 && Condition 1

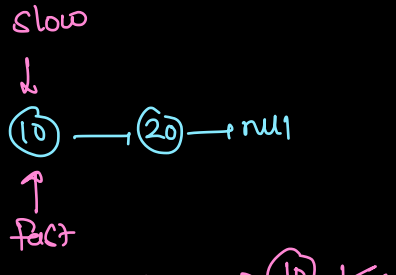
pseudo code:




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



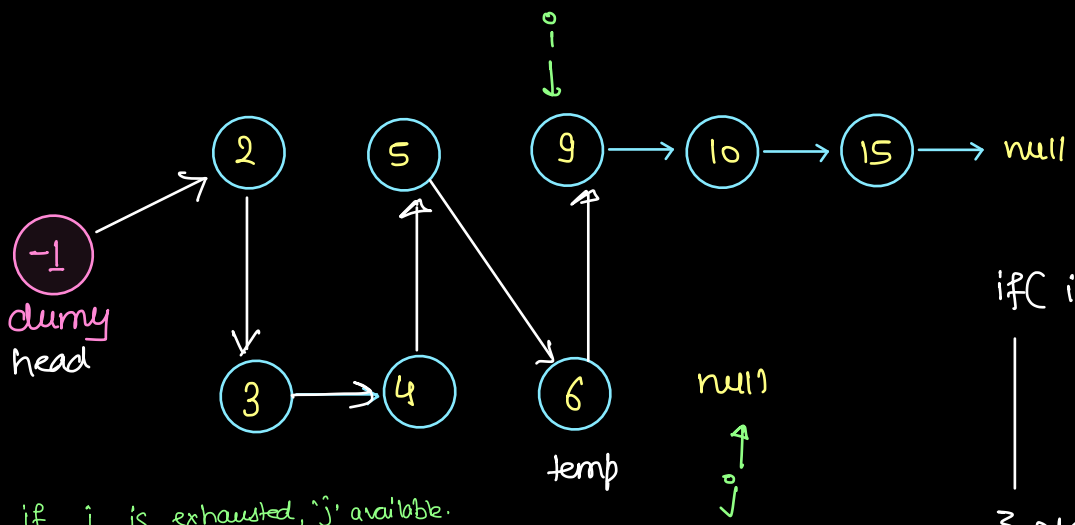
return slow



gtr = $\frac{n}{2}$. TC: O(n)

Merge Two Sorted linkedlist

Given two sorted Linked Lists, Merge them into a single sorted linked list.



if i is exhausted, 'j' available.
temp.next = j;

if j is exhausted, 'i' is avl.
temp.next = i;

```
if( i.data < j.data) {
    temp.next = i
    temp = temp.next.
    i = i.next.
} else {
    temp.next = j
    temp = temp.next
    j = j.next;
}
```

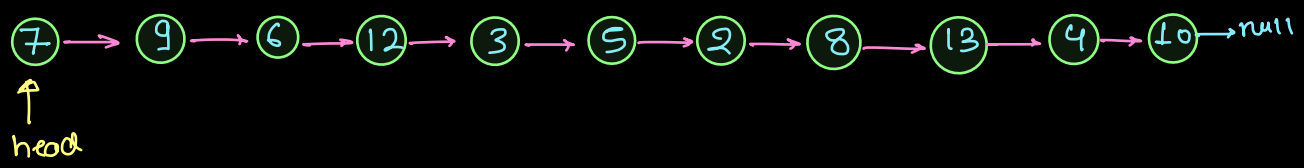
ans → dummyhead.next;

pseudo code:

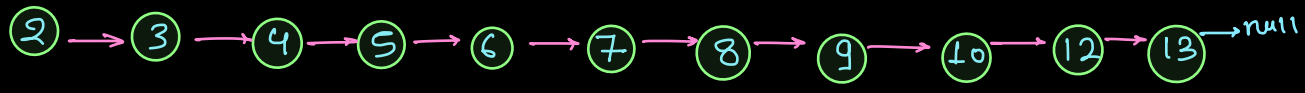
```
Node mergeTwoSorted ( Node head1, Node head2) {
    Node dummyHead = new Node(-1);
    temp = dummyhead,  i = head1,  j = head2;
    while( i != null && j != null) {
        if( i.data <= j.data) {
            temp.next = i;
            i = i.next;
        } else {
            temp.next = j;
            j = j.next;
        }
        temp = temp.next;
    }
    if( i != null) temp.next = i;
    if( j != null) temp.next = j;
    return dummyhead.next;
}
```

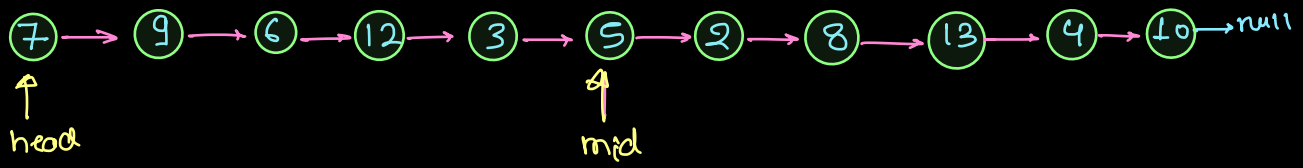
Sort the given linkedlist (MergeSort)

A Linked List is given, Sort the Linked list using merge sort.



o/p:



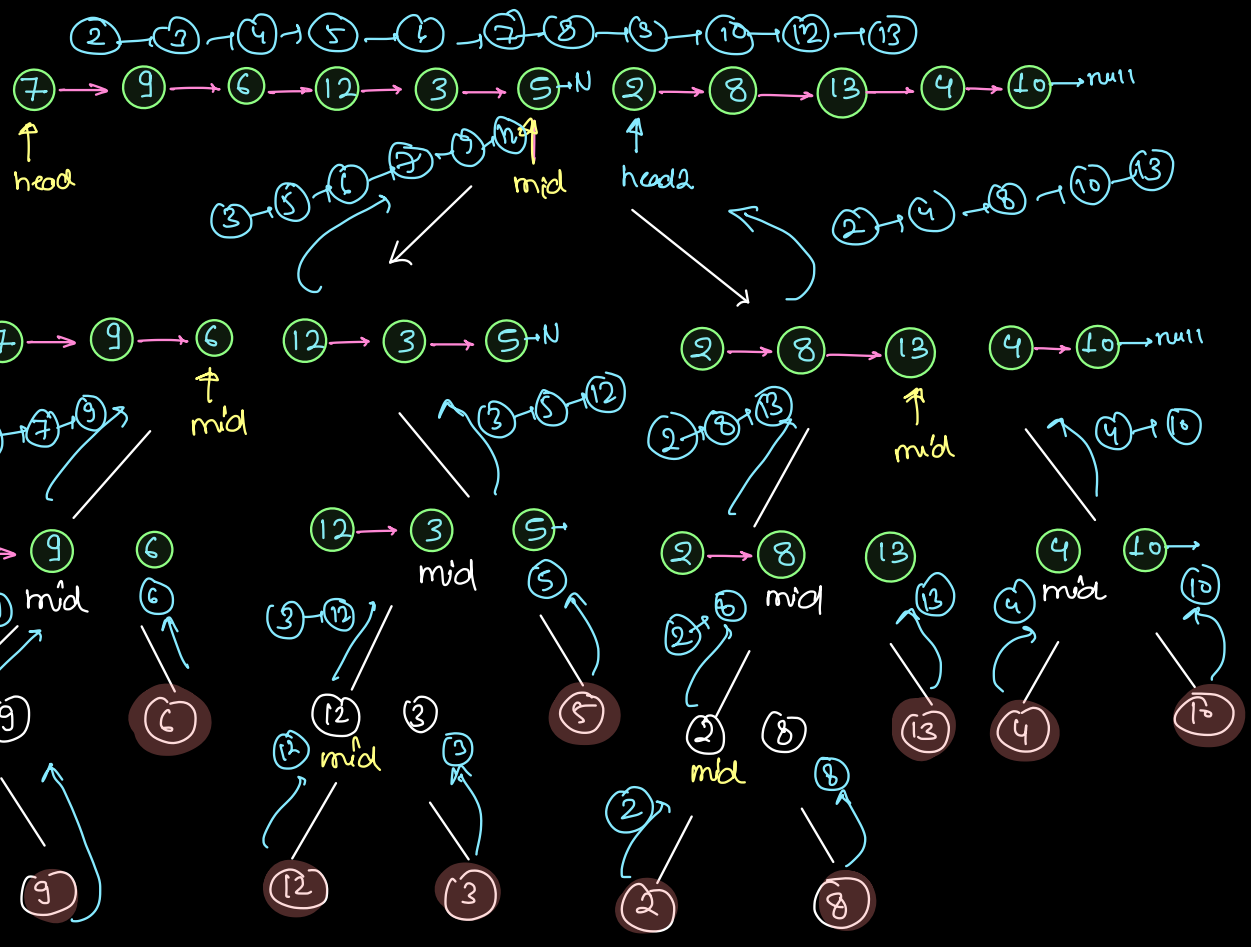


① Given head

② find middle node.
→ split the LL in two parts.
head1 → head to mid
head2 → mid.next to last

③ Ask from Recursion to sort both of the LL

④ Merge two sorted LinkedList. & return final head;



pseudo code :

```
Node mergeSort(Node head) {                                  ⟶ T(n)
    if(head.next == null) {
        return head;
    }
    Node mid = middleNode(head);                             ⟶ n/2
    Node head2 = mid.next;
    mid.next = null;
    Node left = mergeSort(head);                             ⟶ T(n/2)
    Node right = mergeSort(head2);                           ⟶ T(n/2)
    Node ans = mergeTwoSortedLL(left, right);                ⟶ n
    return ans;
}

Node middleNode(Node head) {
    TODO
}

Node mergeTwoSortedLL(Node head1, Node head2) {
    TODO
}
```

$$T(n) = 2T\left(\frac{n}{2}\right) + kn + k_2$$ ⟶ Solve it

$\Rightarrow$ T.C : $O(n \log n)$

S.C : $O(\log n)$

Google Maps got your back

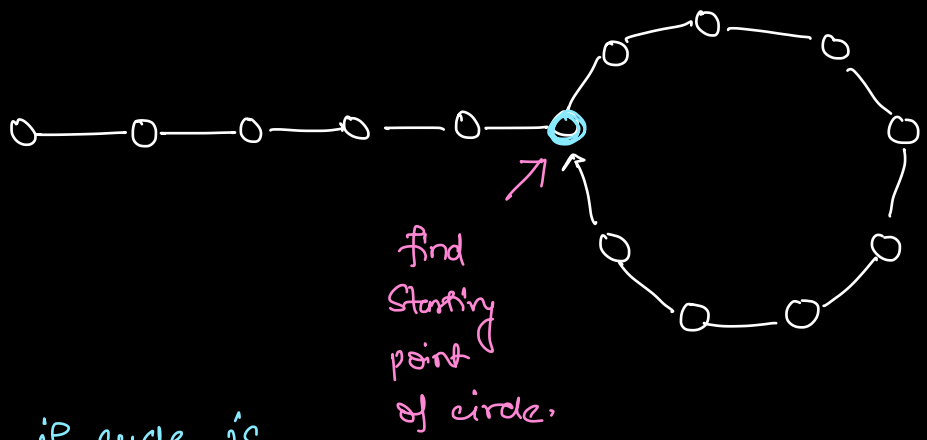
Scenerio

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.



* Check if cycle is actually present.
* find starting point of cycle
* Remove cycle from the Linked List

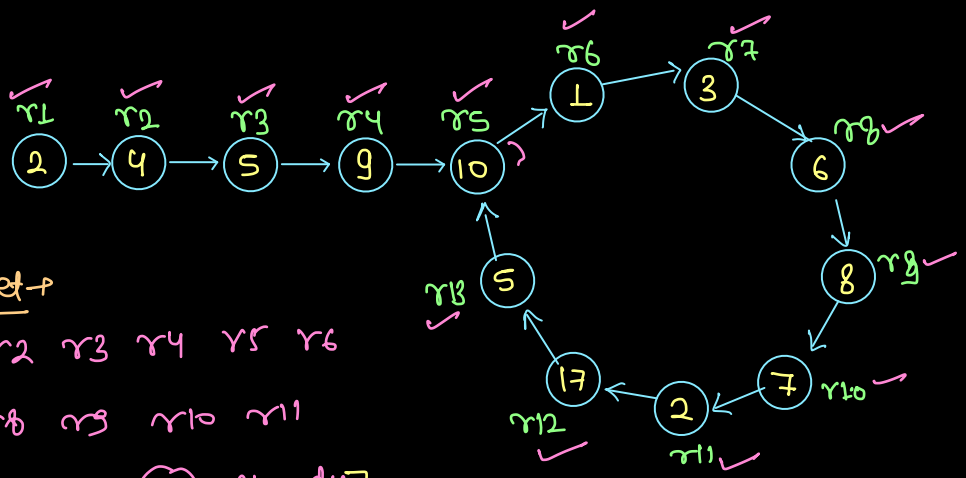
10:36 – 10:46 PM

Break

Detect Cycle in the linkedlist

Given a Linked List, Find whether it contains a cycle.

Eg 1.



HashSet →

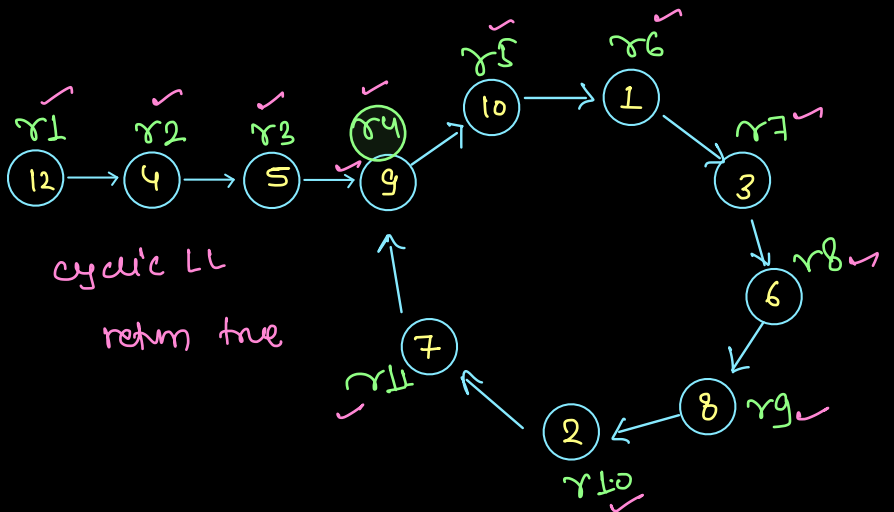
r1 r2 r3 r4 r5 r6

r7 r8 r9 r10 r11

r12 r13 (r5) Already added] Cycle available ↳ return true

T.C.: O(n)] using HashSet
S.C.: O(n)

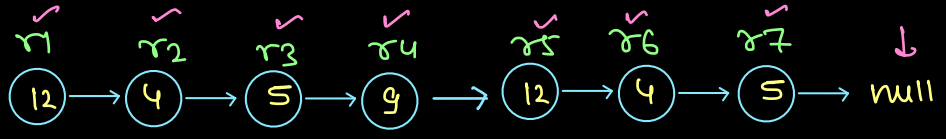
Eg 2.



cyclic LL

return true

Eg 3.



Nothing repeated

→ No cycle false

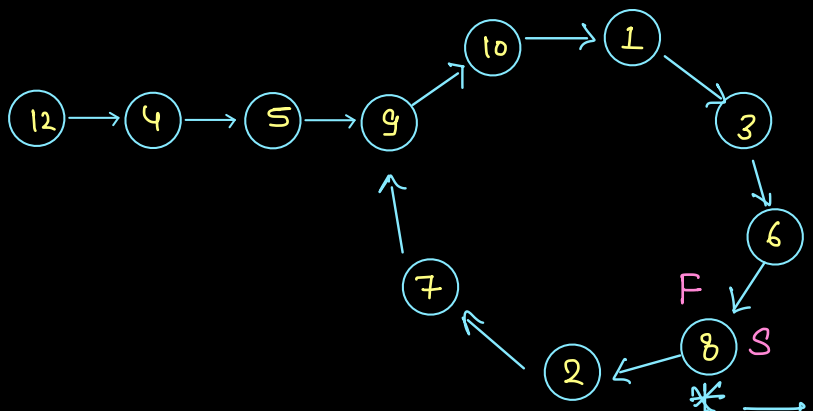
Idea 0: Using HashSet →

HashSet < Node >

T.C.: O(n)
S.C.: O(n)

Query 1: Without using extra space

Floyd Cycle Detection technique



* → if they meets
LL is cyclic

Steps:

① slow = head, fast = head;

② slow = slow.next at 1x speed
fast = fast.next.next at 2x speed.

③ if cycle present, slow & fast will definitely enter in cycle.
& if they enter in cycle, they will cross path again

④ If they meet at any point, that means LL is cyclic.

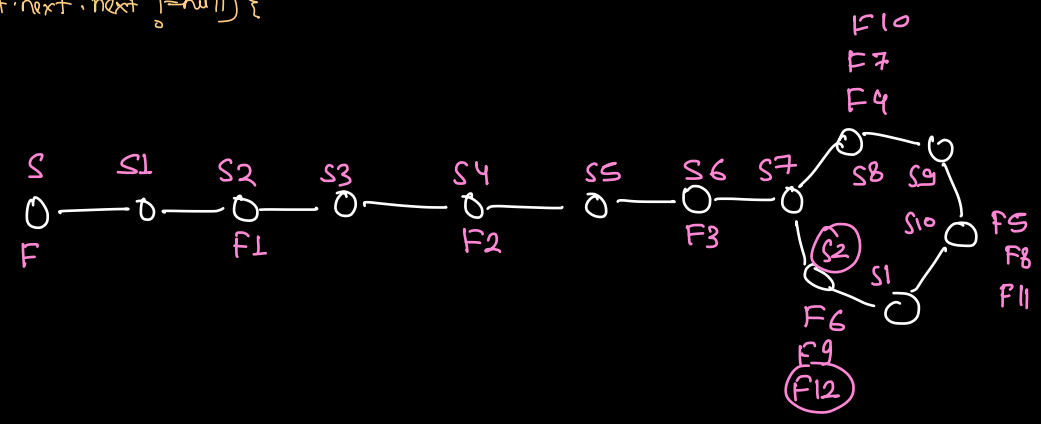
pseudo code:

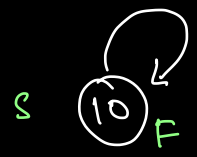
```
boolean isCyclic(Node head) {
    Node slow = head;
    Node fast = head;
    boolean isCycle = false;
    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if(slow == fast) {
            isCycle = true;
            break;
        }
    }
    return isCycle;
}
```

```
boolean isCyclic(Node head) {
    Node slow = head;
    Node fast = head;
    boolean isCycle = false;
    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if(slow == fast) {
            isCycle = true;
            break;
        }
    }
    return isCycle;
}
```

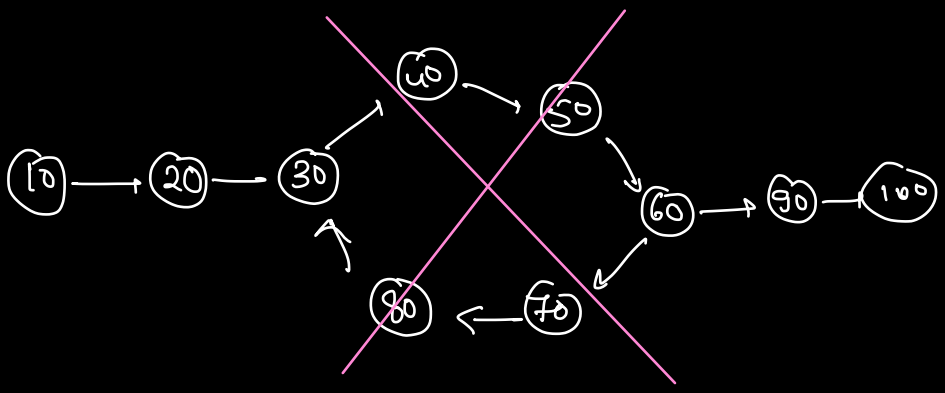
true.

is cycle = ~~false~~ true





T.C: O(n)
S.C: O(1)
→ floyd cycle Detection Algorithm.

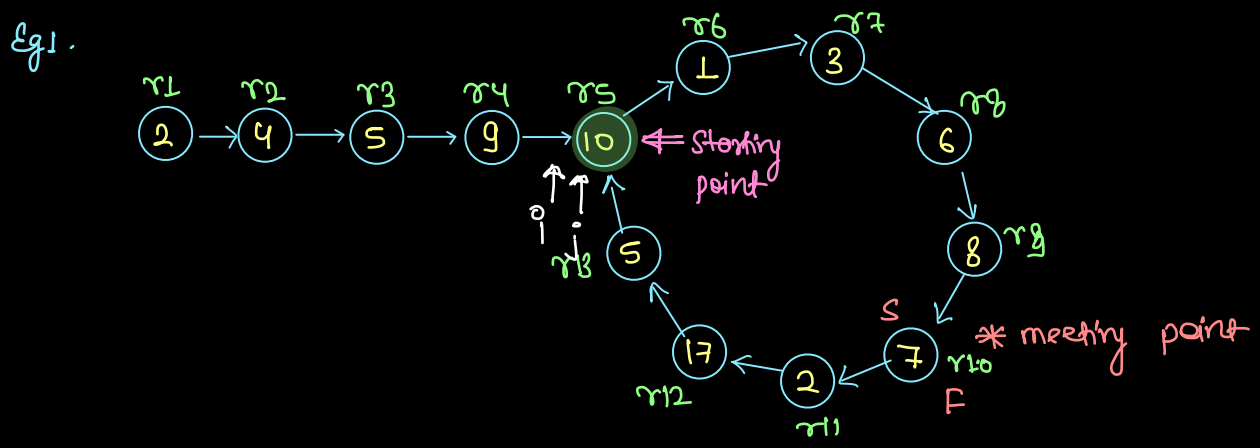


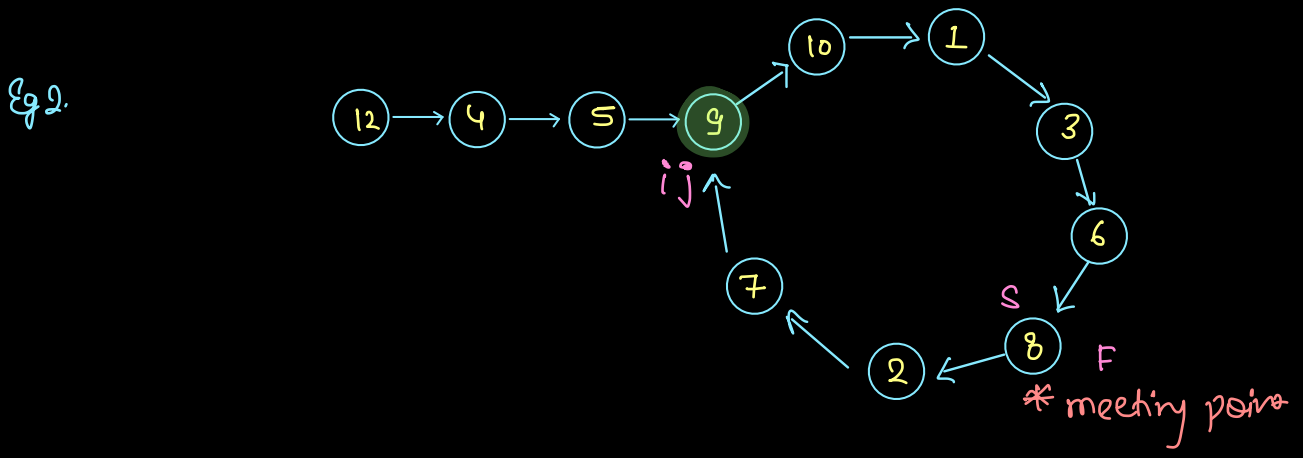
~~~~~~~~~~~~~~~~~~~~~~~~

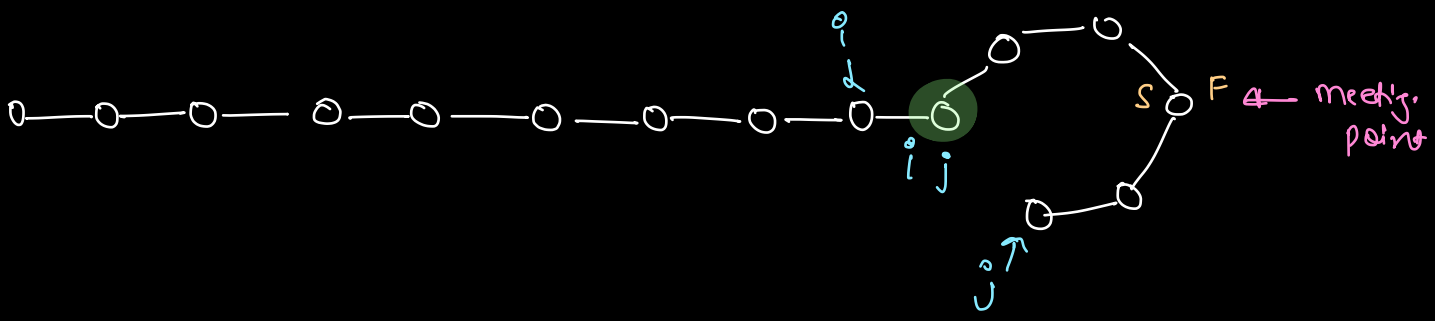
Starting point of cycle

~~~~~~~~~~~~~~~~~~~~~~~~

Given a Linked List which contains a cycle , Find the start point of the cycle.







pseudo code :

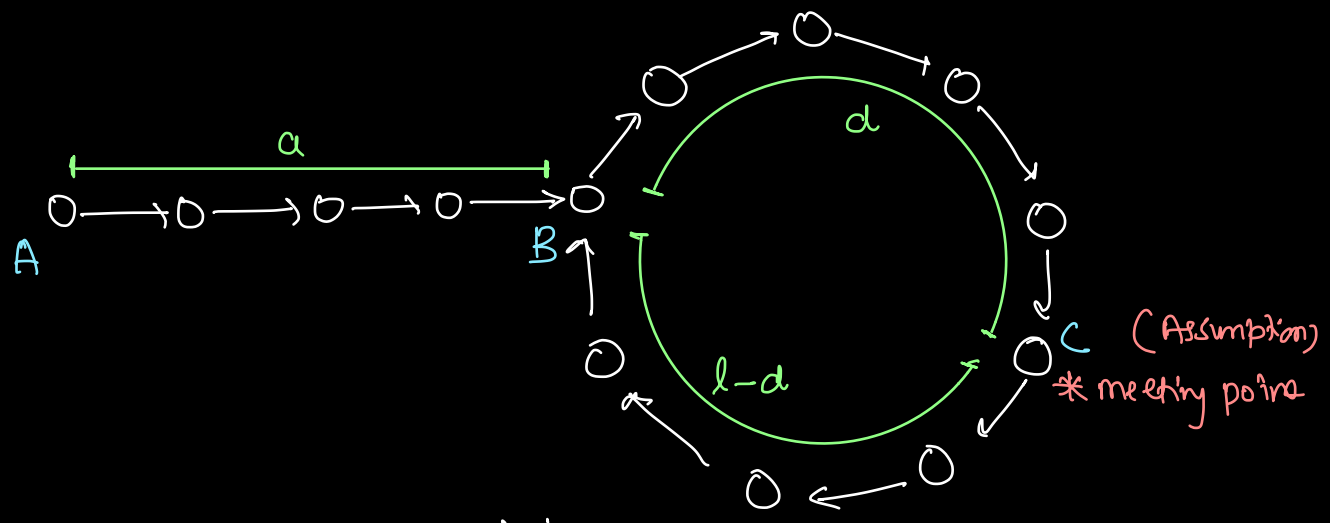
```
Node startingOfCycle(Node head) {
    Node slow = head;
    Node fast = head;
    boolean isCycle = false;
    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if(slow == fast) {
            isCycle = true;
            break;
        }
    }
    if(isCycle == false){
        // Do it according to problem statement
        // for now, we are assuming, we have to return null
        return null;
    }
    Node i = head;
    Node j = slow; // meeting point
    while(i != j) {
        i = i.next;
        j = j.next;
    }
    return i;
}
```

T.C : O(n)
S.C : O(1)

think about it ?? TODO

```
if you have to break the cycle
    while(i.next != j.next) {
        keep iterating
    }
    After loop.
    j.next = null
```

Proof of Starting point:



Distance b/w A to B = 'a'

length of complete cycle is = 'l'

Dist. b/w $\overset{\frown}{BC}$ = 'd'

Dist b/w $\underset{\smile}{BC}$ = 'l-d'

To prove ⇒ $\boxed{a = K * l + (l-d)}$

dist. travelled by slow ptr = 'a + d'

dist. travelled by fast ptr = "a + x * l + d" [x → is rotation taken by fast]

$d_S = D$

$d_F = 2D$

⇒ $d_F = 2 * d_S$, put the value.

$a + x*l + d = 2(a+d)$

$a + xl + d = 2a + 2d$

$xl = a + d$

$$a = xl - d$$

Add and subtract l in RHS:

$$\Rightarrow \quad a = xl - d + l - l$$

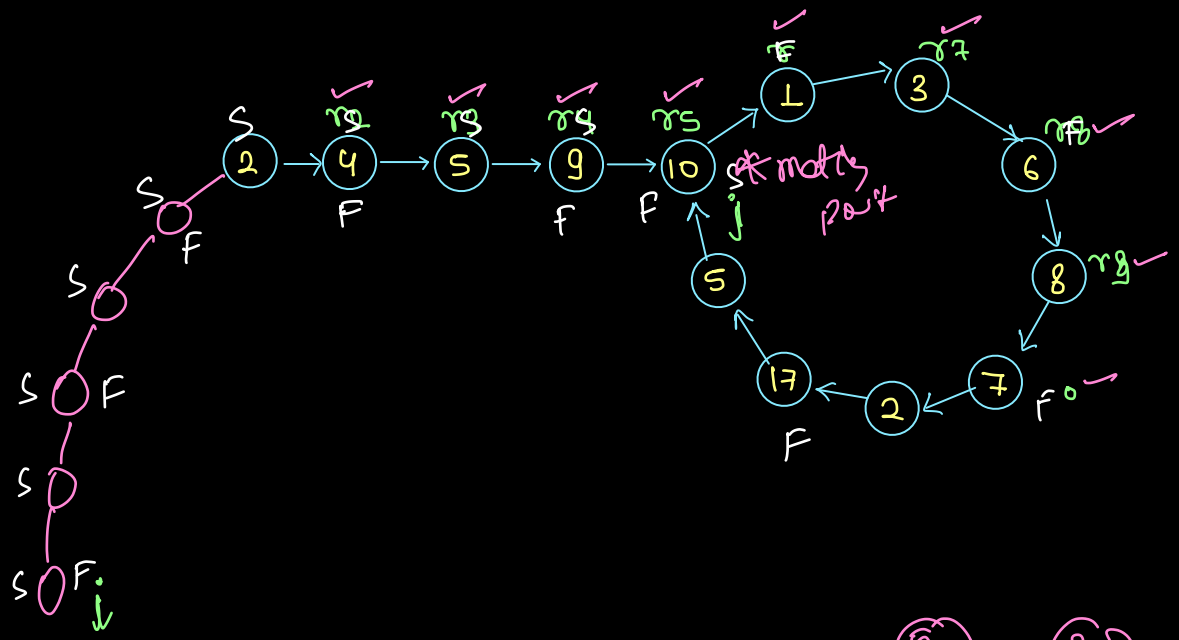
$$\Rightarrow \quad a = xl - l + l - d$$

$$\Rightarrow \quad a = l[x-1] + l - d$$

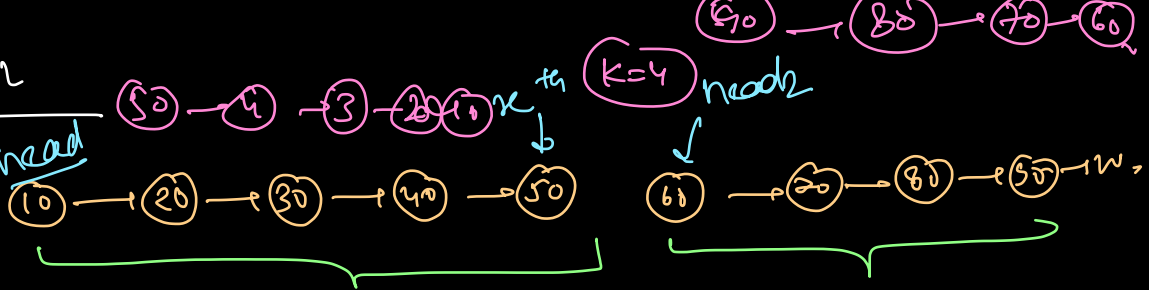
$$\Rightarrow \quad \boxed{a = kl + l - d}$$

proved.

doubt session:



K-Rotate in LL



(*) calculate total size, n=9

(*) find $(n-k-1)^{th}$ node (indexing is 0 based) → x^{th} node

(*) head2 = x.next; x.next = null