**Agenda**

1) Find mid of LinkedList

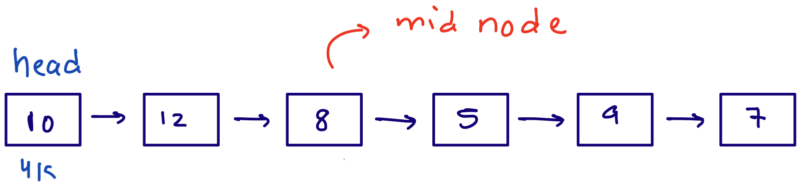
2) Merge two Sorted LL

3) Reorder LL

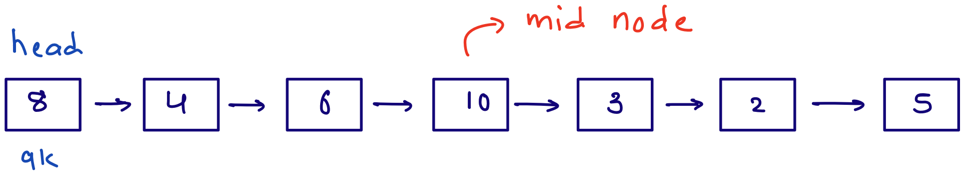
4) Cycle Detection
   - i) Detect cycle
   - ii) Find start of cycle
   - iii) Remove cycle

Q.1 Given a LL, find and return **mid node.**

eg1



eg2

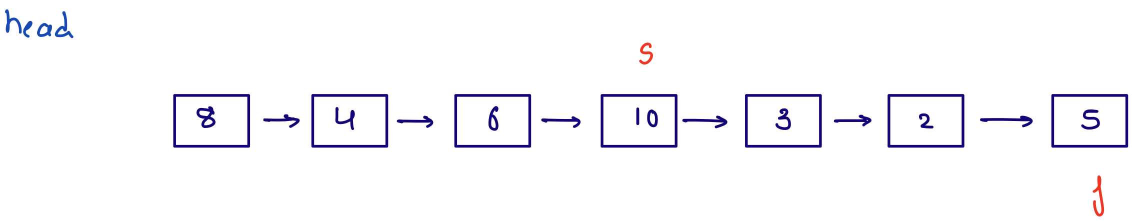


Idea1 : find size of LL and travel size/2 times to find mid.

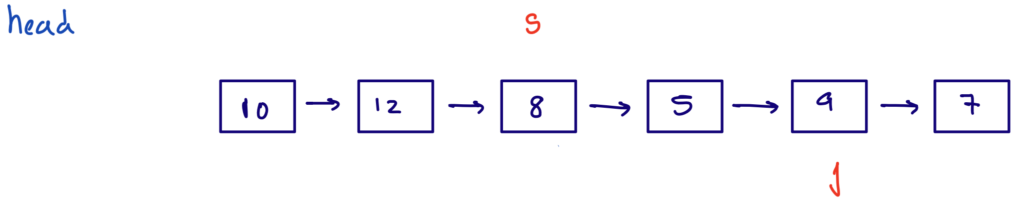
Idea2 : Using slow & fast pointers

→ slow ptr takes 1 step everytime

→ fast ptr takes 2 steps everytimes

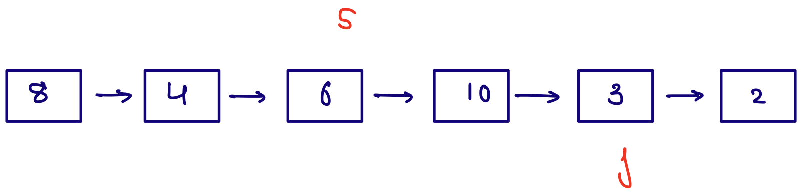


fast.next != null



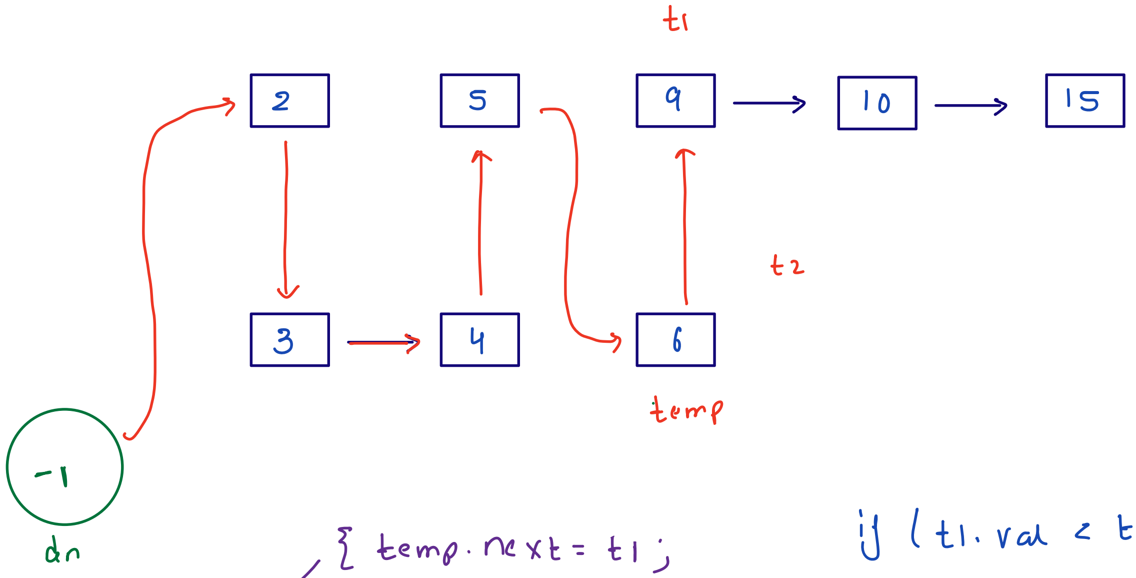
fast.next.next != null

```
Node  midNode( Node head) {
    Node slow = head, fast = head;

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



Q.2 Given 2 sorted Linked list, merge and get final sorted List.

Note: no extra space allowed



t2 exhausted

{ temp.next = t1;

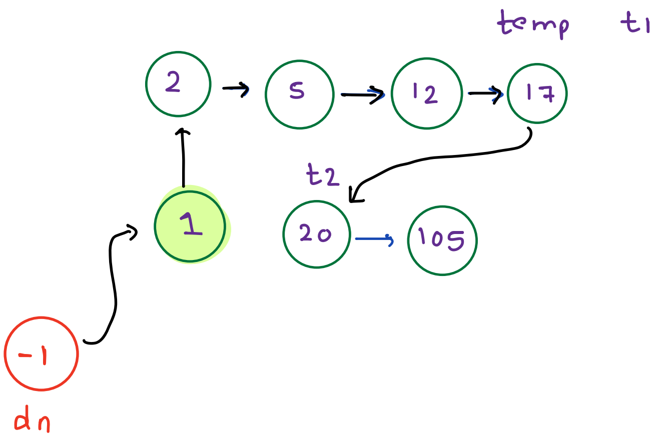
return dn.next;

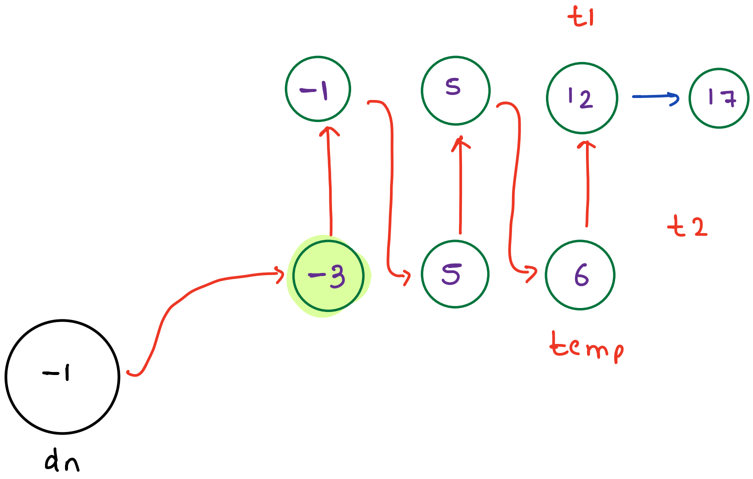
```
if (t1.val < t2.val) {
        temp.next = t1;
        t1 = t1.next;
}
else {
        temp.next = t2;
        t2 = t2.next;
}

temp = temp.next;
```

```
Node   merge2SortedLL ( Node head1, Node head2 ) {
  Node  dn = new Node(-1);
  Node  t1 = head1, t2 = head2, temp = dn;

  while ( t1 != null  &&  t2 != null ) {
        if ( t1.val < t2.val ) {
              temp.next = t1;
              t1 = t1.next;
        }
        else {
              temp.next = t2;
              t2 = t2.next;
        }
        temp = temp.next;
  }

  if( t1 != null ) {
        temp.next = t1;
  }
  if ( t2 != null) {
        temp.next = t2;
  }
  return dn.next;
}
```

SC: O(1)

TC: O(n+m)





```
while ( t1 != null && t2 != null) {
    if ( t1.val < t2.val ) {
        temp.next = t1;
        t1 = t1.next;
    }
    else {
        temp.next = t2;
        t2 = t2.next;
    }
    temp = temp.next;
}
```

Q.3 Rearrange the given Linked List.

$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow \ldots\ldots$

Rearrange the nodes

eg1 1 → 2 → 3 → 4 → 5 → 6

ans : 1 → 6 → 2 → 5 → 3 → 4

eg2 1 → 2 → 3 → 4 → 5 → 6 → 7

ans: 1 → 7 → 2 → 6 → 3 → 5 → 4

first → last → second → 2nd Last → third → 3rd Last → .....

i) find mid node and break LL into two halves.

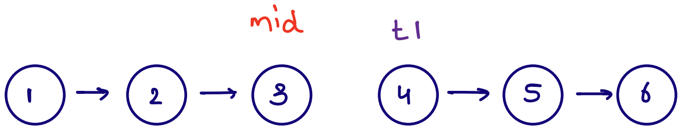
mid.next = null

code : todo

ii) Reverse 2nd half of LL

iii) to create ans join the two LL by taking one node from both LL every time.

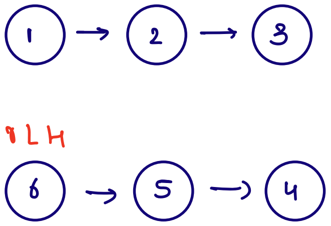
Step 1



```
Node t1 = mid.next;
mid.next = null;
```

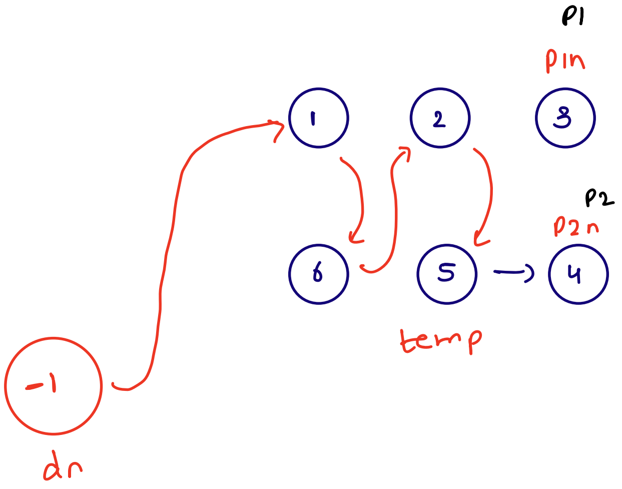
Step 2

```
Node rLH = reverseLL(t1);
```



Step 3

create final ans by taking 1 node from both LL in every step.



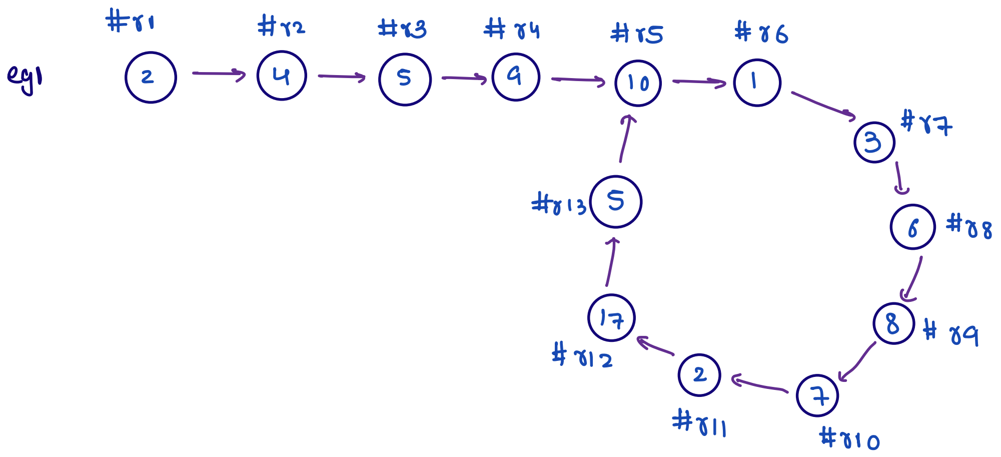
```
Node p1 = head;
Node p2 = rLH;
```

Code : todo?

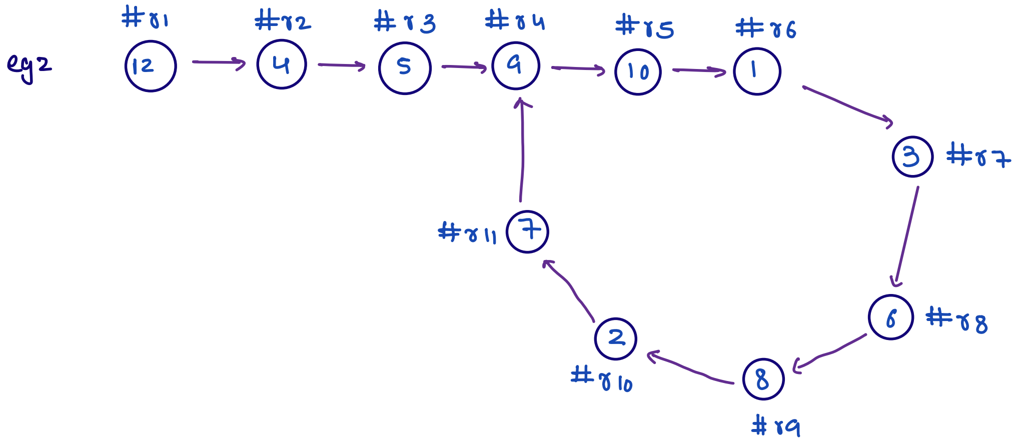
Q.4 Given head node of Linked list, check for cycle detection?

r1, r2, r3 . . .
references of
node.

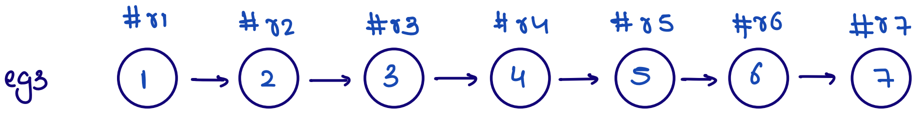
eg1



eg2



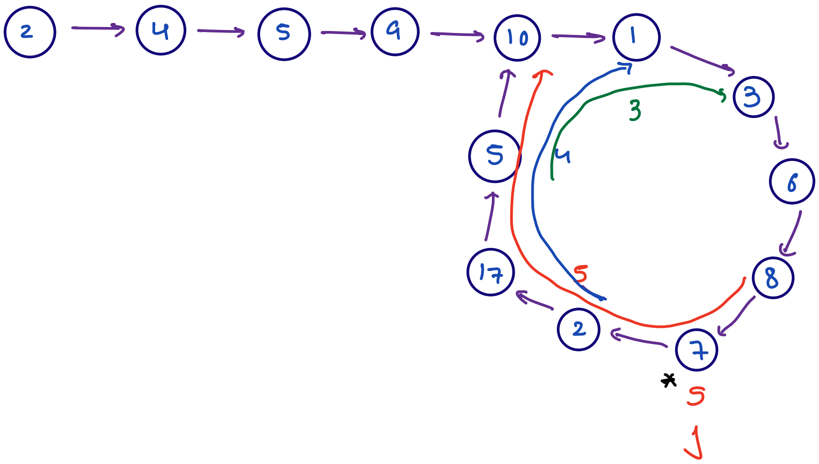
eg3



Idea 1 Using Hashset to check if reference is repeating or not

```
Hashset < Node > hs = new HashSet<> ( );
```

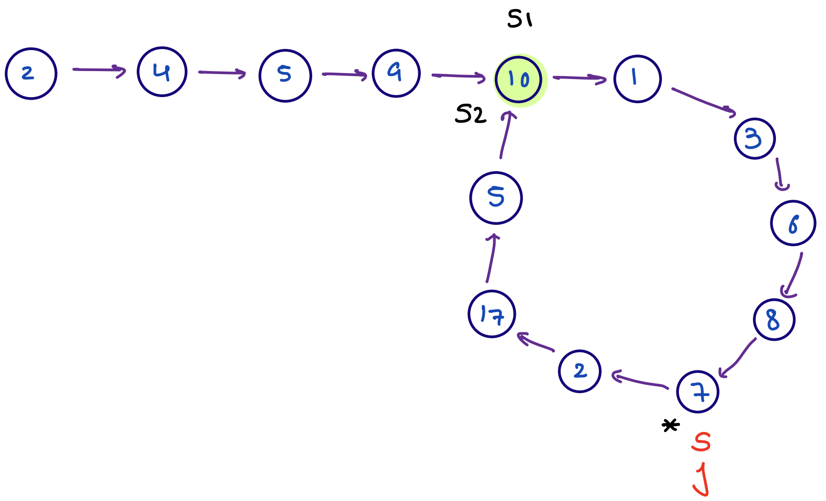
Idea 2 without space { Floyd cycle detection }



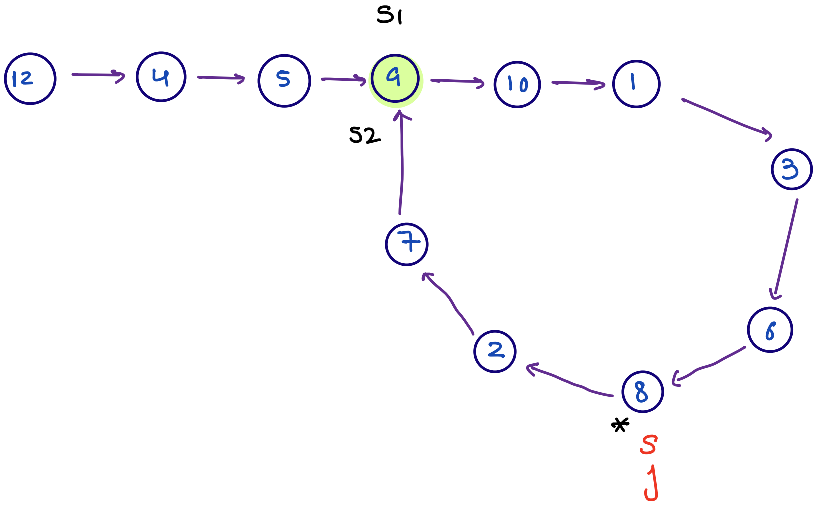
once both slow and fast ptr are inside the cycle, the distance b/w them will decrease at every step and after some time they will certainly meet.

```
boolean cyclePresent (Node head) {
    Node slow = head, 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;
}
```

# Starting point of cycle



eg2



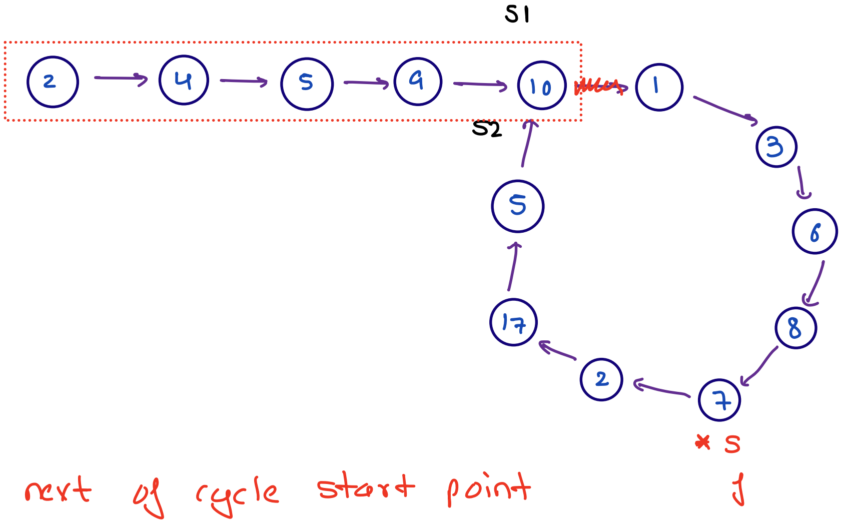
i) Create two slow ptr, put one of them at start of LL and another one at meeting point.

Now move both of them with 1 step every time.

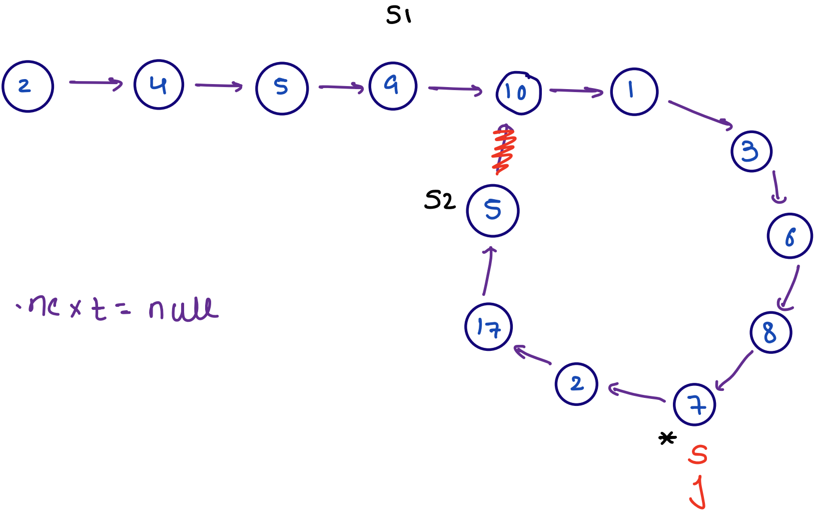
One day they will meet at start of cycle.

```
Node  startPointofCycle (Node head) {
    Node slow = head, 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)   return null;

     Node  s1 = head , s2 = slow;
                            └→ meeting point
     while (s1 != s2) {
            s1 = s1.next;
            s2 = s2.next;
     }

      return s1;   ||starting point

}
```

Remove cycle from LL and return head of LL

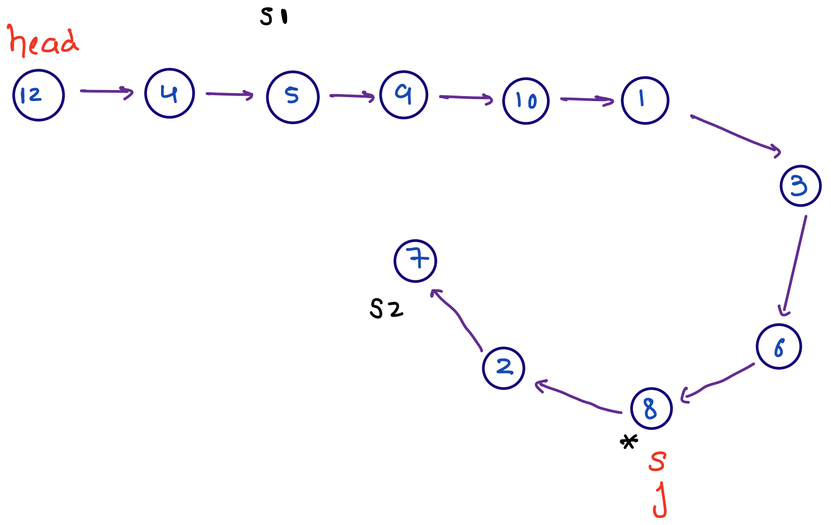


Setting next of cycle start point to null is not correct.

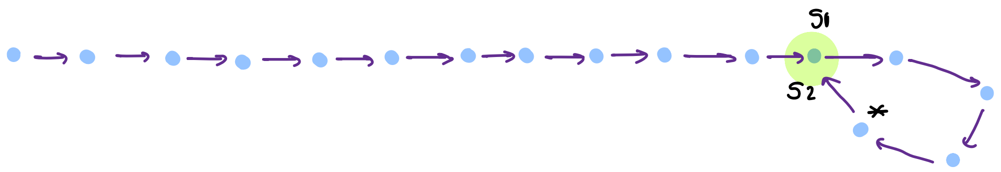


S2.next = null

```
Node   removeCycle (Node head) {

    Node slow = head, 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)   return  head;

     Node  s1 = head , s2 = slow;
                                └──> meeting point
      while ( s1.next != s2.next ) {
              s1 = s1.next;
              s2 = s2.next;
      }

      s2.next = null;
      return head.
}
```


head
S1
12
4
5
9
10
1
3
6
8
2
7
S2
*
s
J

try finding starting point



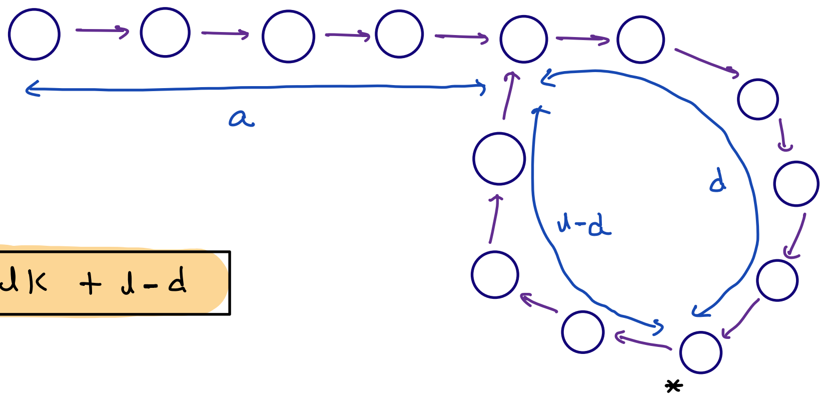
```
while ( fast.next != null && fast.next.next != null) {
        slow = slow.next;

        fast = fast.next.next;

        if (slow == fast) {
                isCycle = true;
                break;
        }
}
```

```
Node s1 = head , s2 = slow;
```

→ meeting point

```
while (s1 != s2) {
        s1 = s1.next;
        s2 = s2.next;
}
```

## proof of starting point logic



$\boxed{a = lk + l - d}$

length from head to start point of cycle = a

length of cycle is = l

distance of meeting point from start point of cycle = d

$d_s = a + lx + d$ (x is rounds travelled by slow in cycle)

$d_f = a + ly + d$ (y is rounds travelled by fast in cycle)

$$d_f = 2d_s$$

$$a + ly + d = 2(a + lx + d)$$

$$\cancel{a} + ly + \cancel{d} = 2a + 2lx + 2d$$

$$ly = a + 2lx + d$$

$$a = \lambda y - 2\lambda x \underbrace{- d + \lambda} - \lambda$$

$$a = \lambda y - 2\lambda x - \lambda + \lambda - d$$

$$a = \lambda(\underbrace{y - 2x - 1}_{K}) + \lambda - d$$

$$\boxed{a = \lambda K + \lambda - d}$$