Agenda

1) find mid of LinkedList

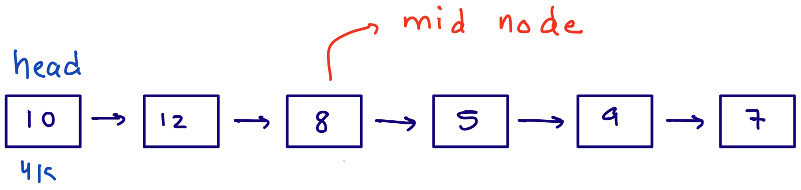
2) Merge two Sorted LL

3) Reorder LL

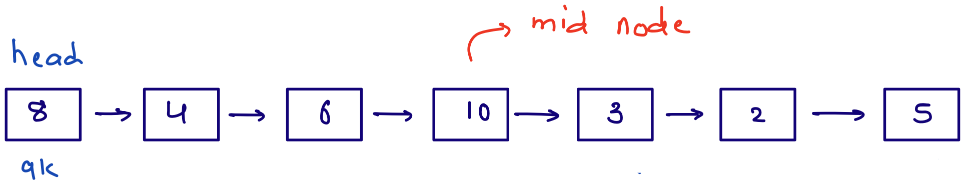
4) Cycle Detection
   - i) Detect cycle
   - ii) Find start of cycle
   - iii) Remove cycle

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

eg1



eg2

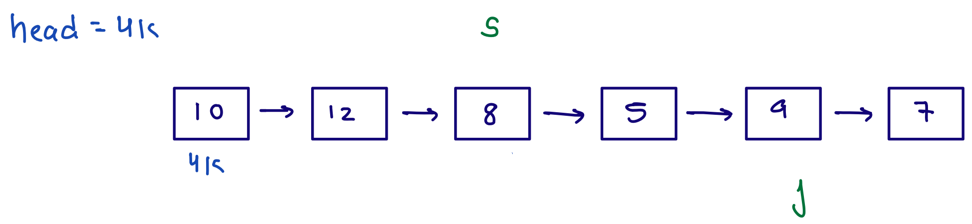


Idea1: find size of LL and then travel size/2 to find mid node.

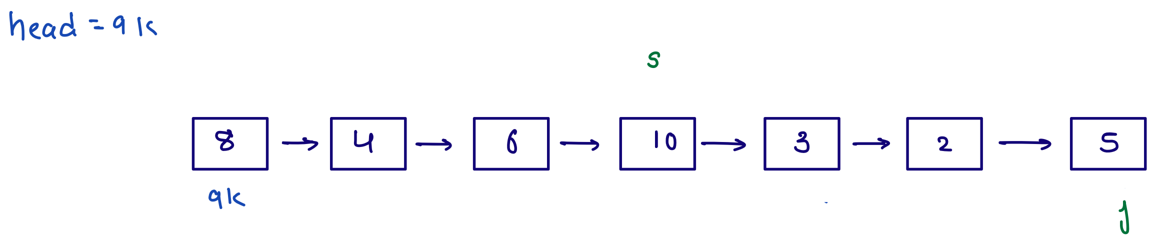
Idea2: using slow and fast pointer

slow → take one step every time

fast → take two steps every time



fast.next.next != null

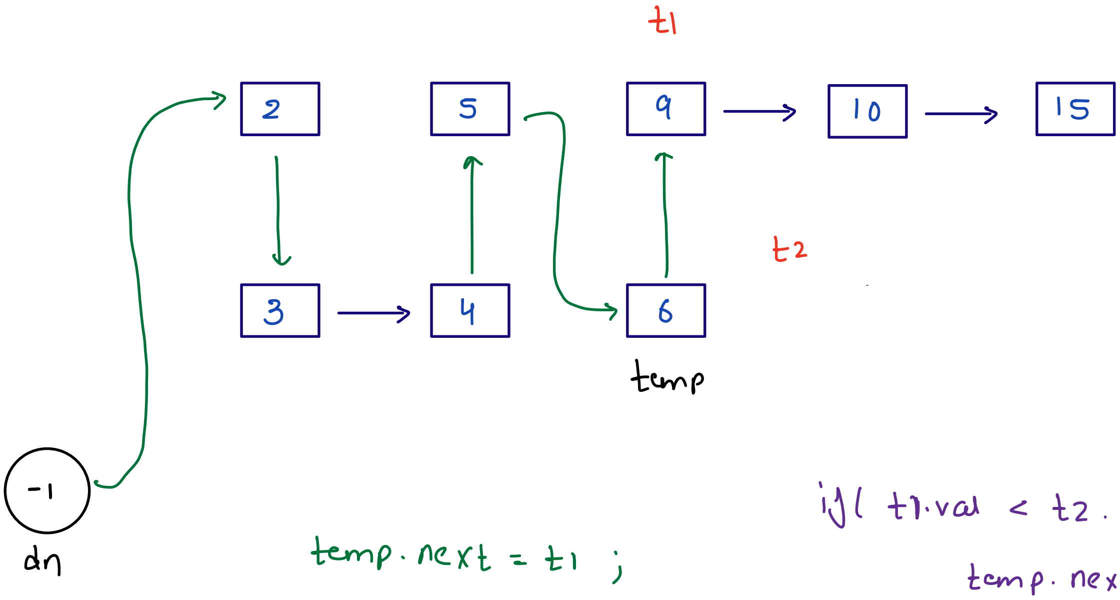


fast.next != null

```
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;
}
```

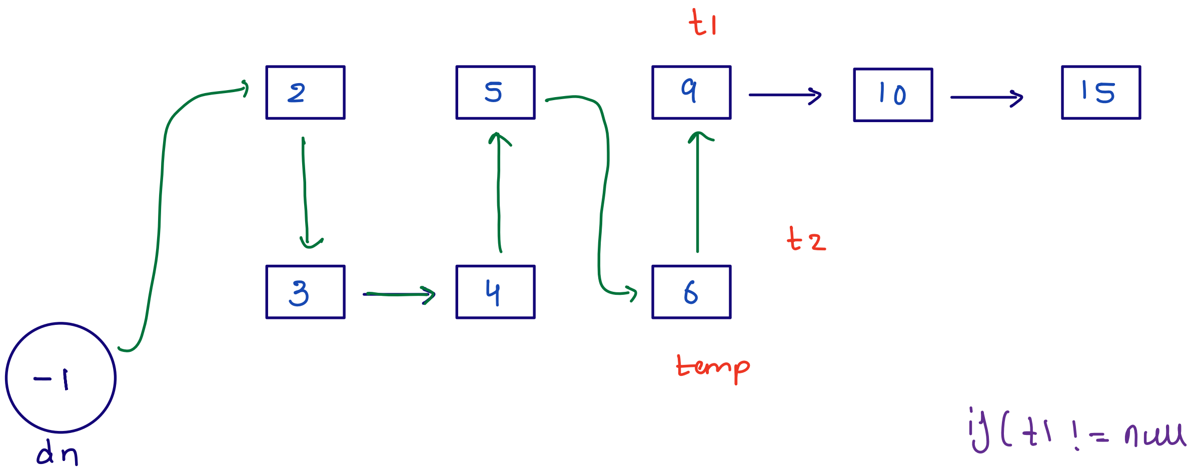
Q-2 Given 2 sorted Linked List, merge and get final sorted List.

Note: no extra space allowed



```
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  dn= new  Node(-1);

→  temp= dn,  t1=head1,  t2 = head2.;

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;

}

return  dn. next;
```

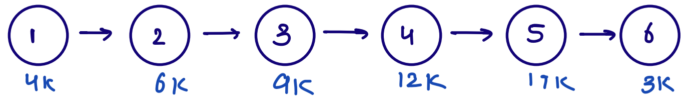
```
Node   merge2SortedLL ( Node head1, Node head2) {
      Node dn= new Node(-1);
      Node temp= dn;
      Node t1 = head1, t2 = head2;
      while (t1 != null && t2 != null) {
          if (t1.val < t2.val) {
                temp.next= t1;
                t1 = t1.next;
          }
          else {
                 temp.next= t2;
                 t2= t2.next;
           }
           temp= temp.next;
      }

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

TC: O(n+m)

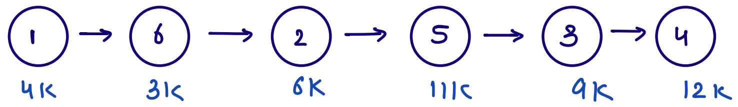
SC: O(1)

Q.3 Rearrange the given Linked List. TC: O(n) SC: O(1)

eg1

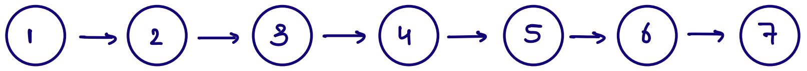


ans :

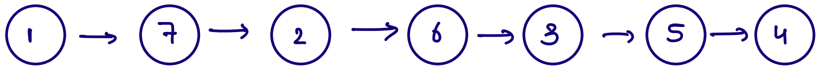


Rearrange the nodes

eg2



ans:



first → last → second → second Last → third → third last . . . .

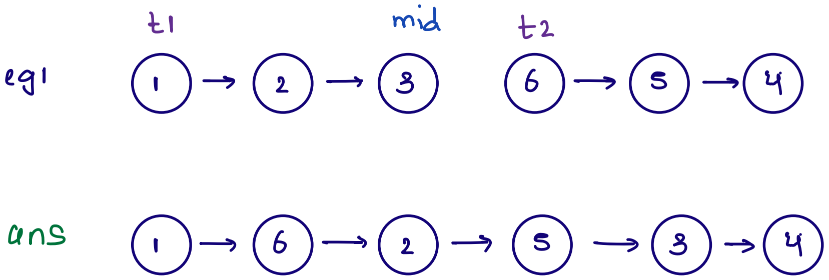
i) find mid of LL and break it into two halfs.

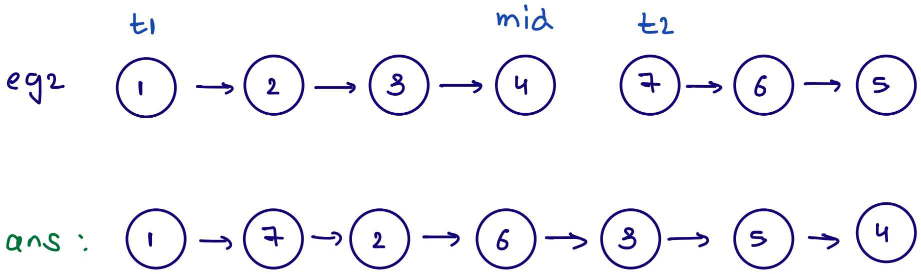
mid.next = null

ii) reverse the second half

iii) get final ans by picking one node every time from first & second LL.

code : todo

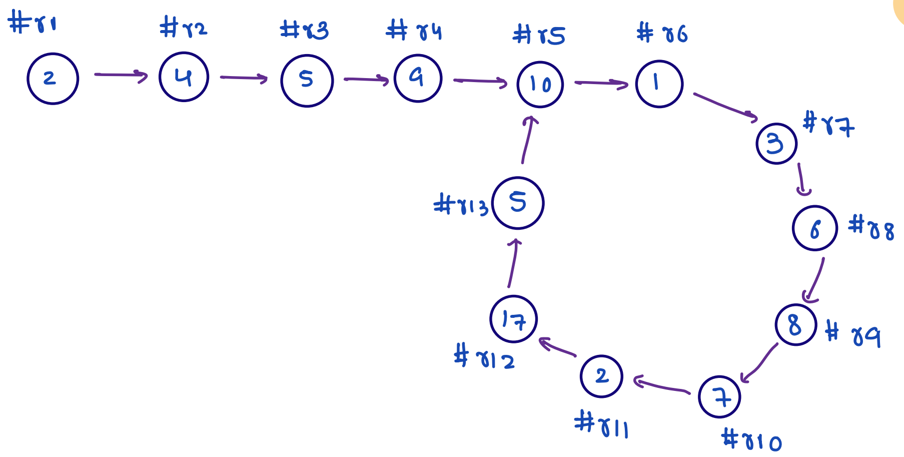

t1
mid
t2
eg1
1 → 2 → 3
6 → 5 → 4
ans
1 → 6 → 2 → 5 → 3 → 4


t1
mid
t2
eg2
1 → 2 → 3 → 4
7 → 6 → 5
ans :
1 → 7 → 2 → 6 → 3 → 5 → 4

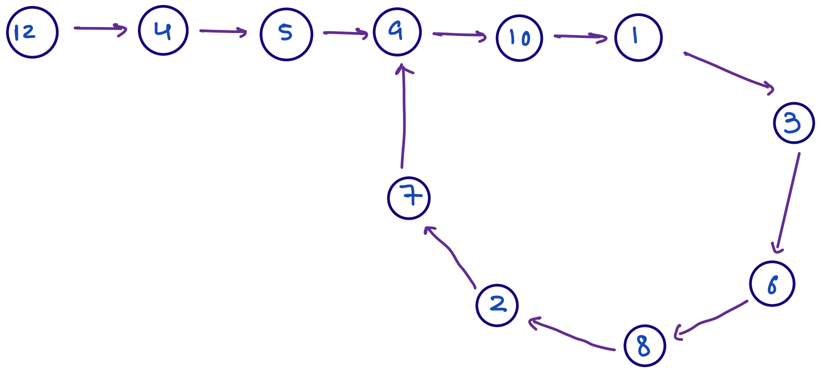
Q.4 Given head node of Linked list, check for cycle detection?

#r1 , #r2 etc.
are reference/
address
of node

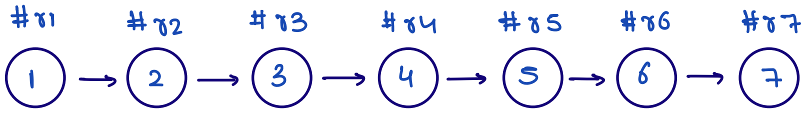
eg1



eg2



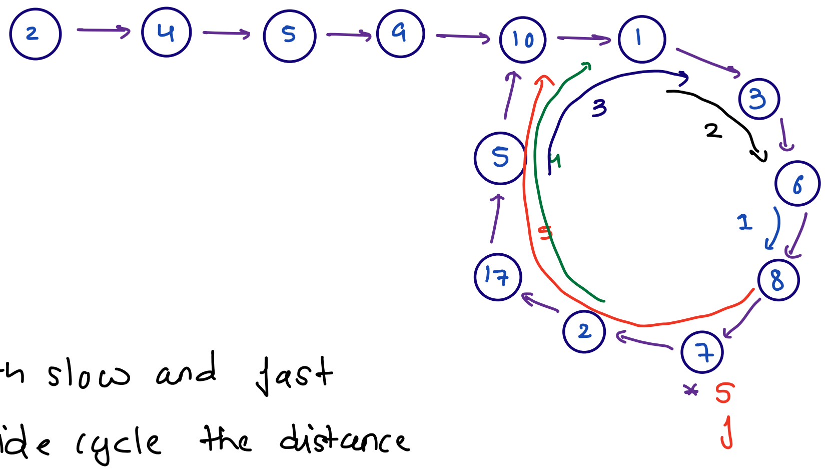
eg3



Idea1 : Using Hashset

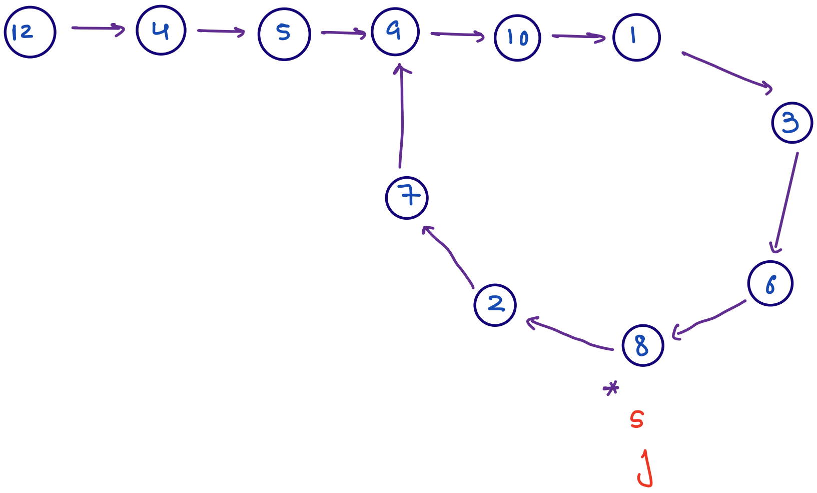
```
Hashset <Node> hs = new Hashset<>();
```

Idea2 : without space {Floyd cycle detection algo}

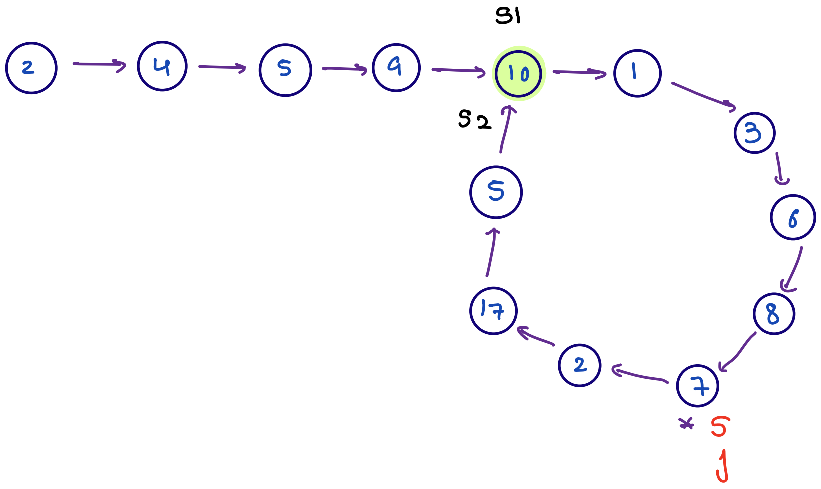


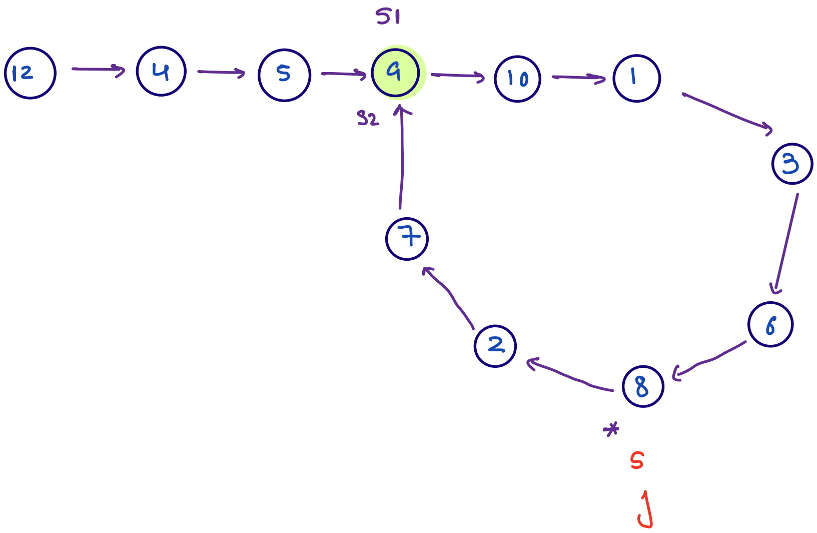
Once both slow and fast are inside cycle the distance b/w them keep on dec. by 1 every time. After some time this dist will become 0 and slow & fast will meet.

```
boolean isCycle(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) {
            //cycle is present
            isCycle = true;
            break;
        }
    }
    return isCycle;
}
```



find start point of cycle



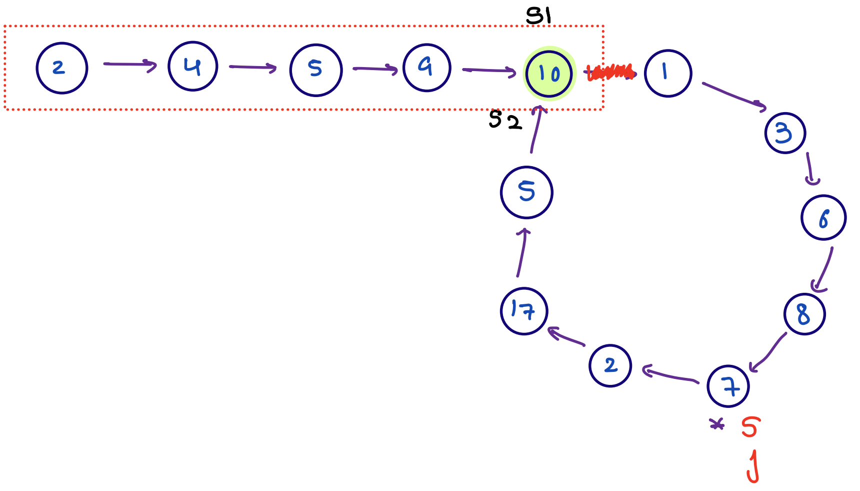


Once slow & fast meet make 2 pointers, put the first ptr at start of LL and second pointer at meeting point. move both ptr by one step every time, one day they will meet and i.e the start point of cycle.

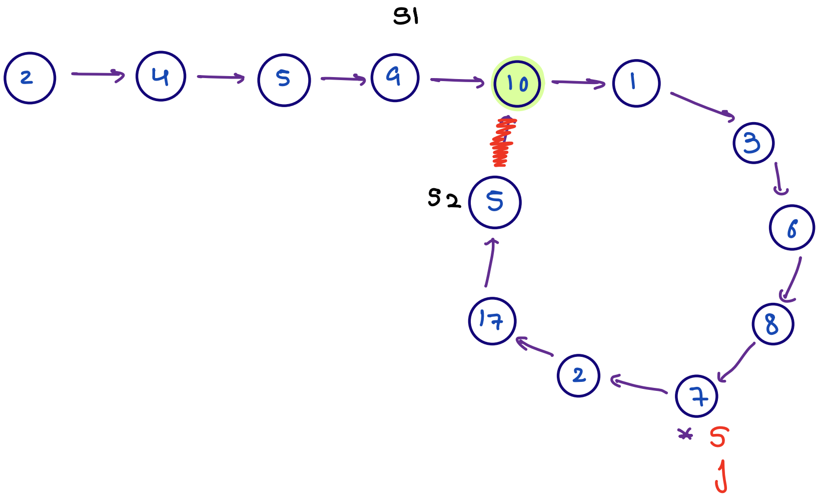
proof: Doubts

```
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) {
            //cycle is present
            isCycle = true;
            break;
        }
    }

    if( isCycle == false) return null;

    Node s1 = head, s2 = slow;

    while (s1 != s2) {

        s1 = s1.next;

        s2 = s2. next;

    }

    return s1;     // starting point

}
```

Remove cycle and head of LL

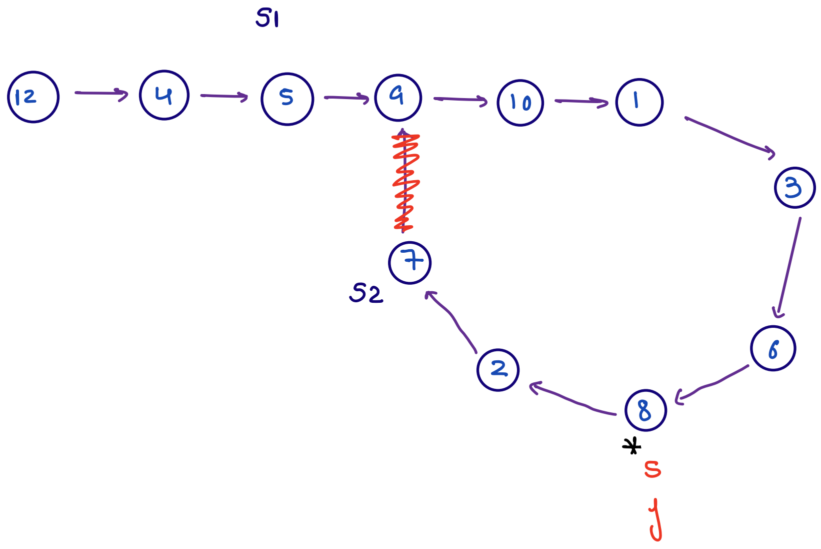


setting next of start point cycle to null is incorrect. (X)

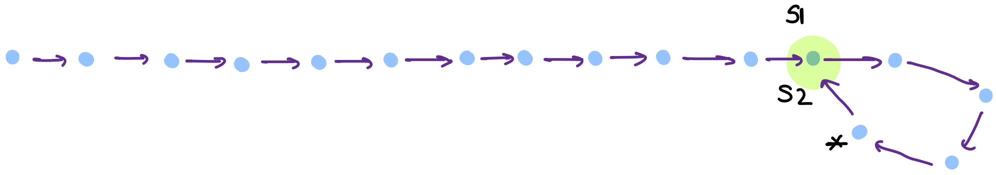


S2.next = null

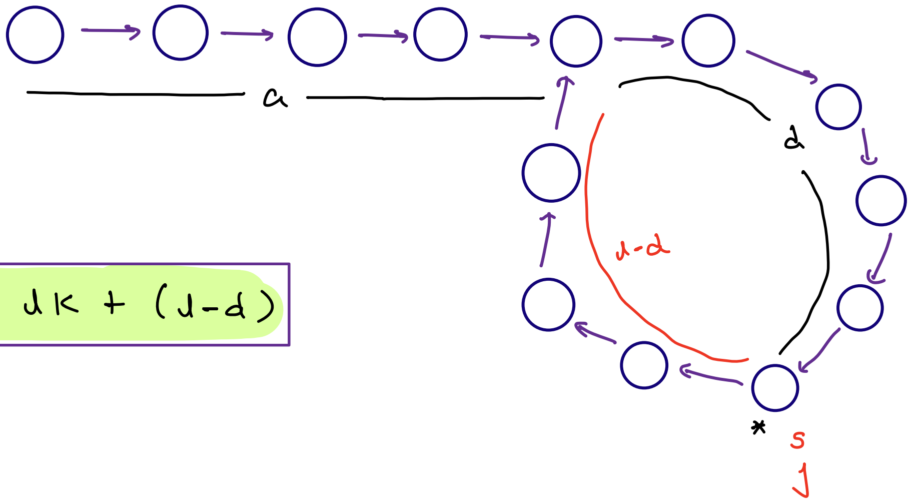
```
Node removeCycle (Node head) {
    Node slow = head, fast = head;
    boolean isCycle = false;
    while( fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if(slow == fast) {
            //cycle is present
            isCycle = true;
            break;
        }
    }
    if( isCycle == false) return head;
    Node s1 = head, s2 = slow;
    while (s1.next != s2.next) {
        s1 = s1.next;
        s2 = s2.next;
    }
    s2.next = null;
    return head;
}
```



Doucbts



*→ meeting point slow & fast



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

length of cycle = l

distance from starting pt. to meeting pt. is d.

Cs: no. of rounds taken by slow in cycle.

$d_s = a + lC_s + d$

cf: no. of rounds taken by fast in cycle.

$d_f = a + lc_f + d$

$d_f = 2d_s$

$a + lc_f + d = 2(a + lC_s + d)$

$\cancel{a} + lc_f + \cancel{d} = 2a + 2lC_s + 2d$

$lc_f - 2lC_s = a + d$

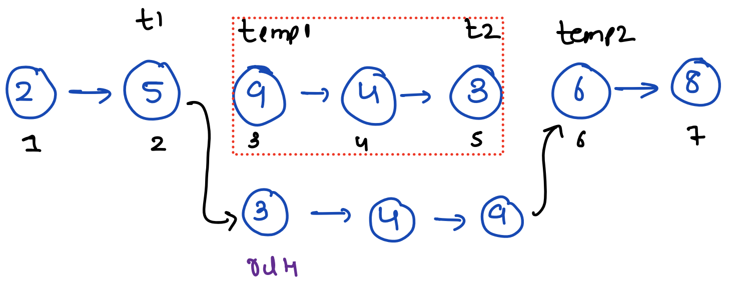
$lc_f - 2lC_s - d = a$

$a = dC_j - 2dC_s - \underbrace{d + d}_{} - d$

$a = d\left(C_j - 2C_s - 1\right) + d - d$

$\boxed{a = dK + (d - d)}$

Reverse in range



s = 3

e = 5

t1 → node at s−1

t2 → node at e

temp1 = t1.next

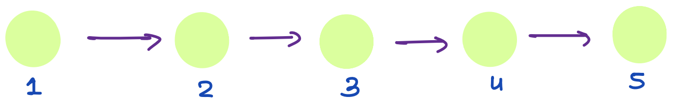
temp2 = t2.next

t1.next = t2.next = null

rLH = reverseLL(temp1);

t1.next = rLH ;

temp1.next = temp2 ;



K = 2

{1 based}

K^{th} from last = N − K + 1 from first / left

remove K^{th} from last means
removing $(N-K+1)^{th}$ from front / left } 1-based

N → length of LL