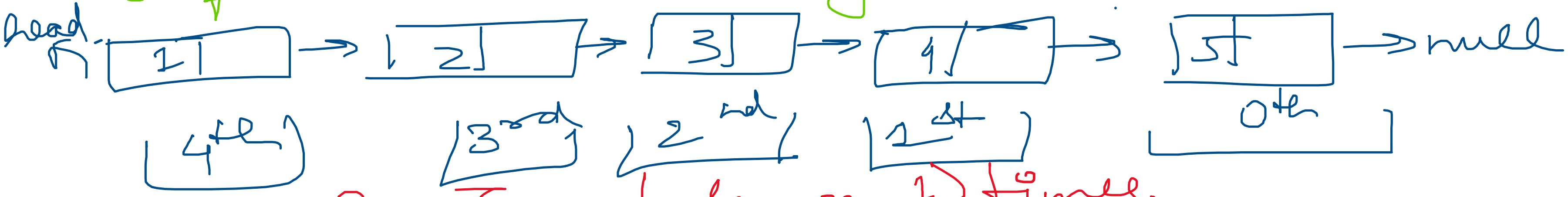
Linked Lists, July 21

21 July 2020 16:58

$\frac{\text{from end}}{n}$ $\frac{\text{from Start}}{(len - n - 1)}$

Get the Node

Loop1 → Calculate length (O(len))

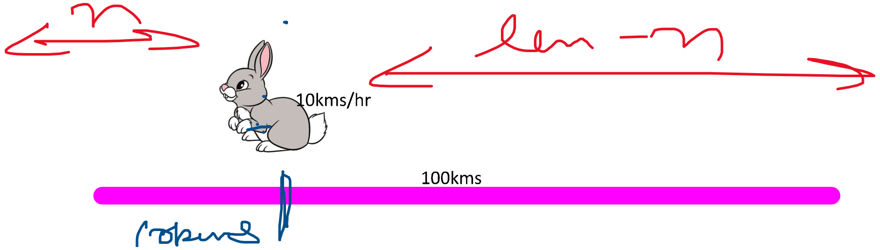


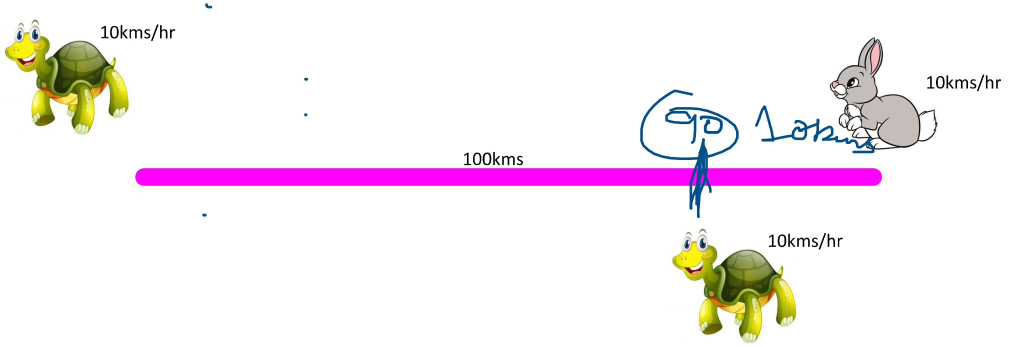
Loop2 → Jump (len-n-1) times.

$$\boxed{0 \le n < \text{length}}$$

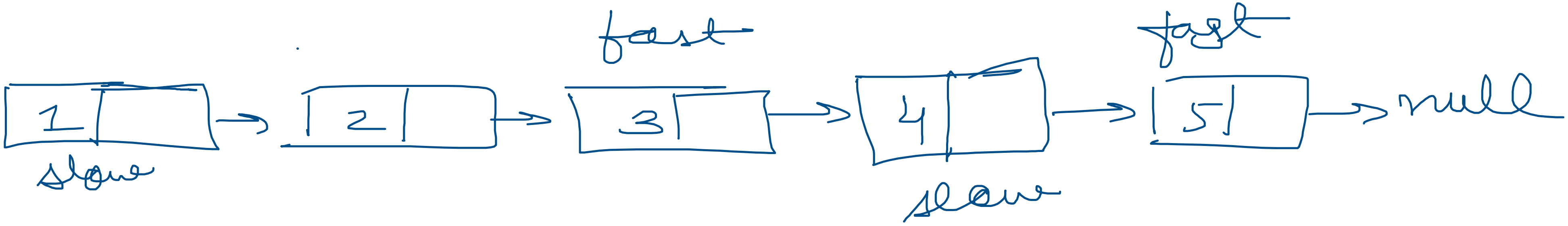
$(\text{length} - n - 1)^{th}$ → from Start

$(5 - 1 - 1) \Rightarrow 3$





[find 1st node from end]



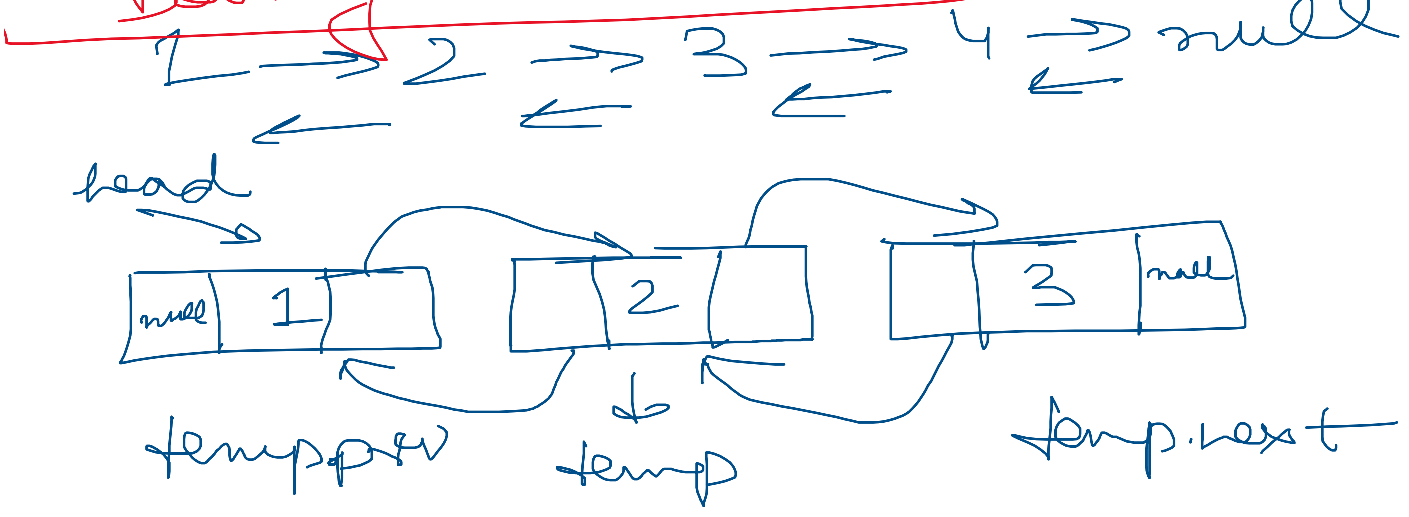
Steps

1. Initialize
   slow = head
   fast = head
2. Take fast n-elements Ahead
3. Fast & slow move together till fast becomes last element.

Slow will be at nth

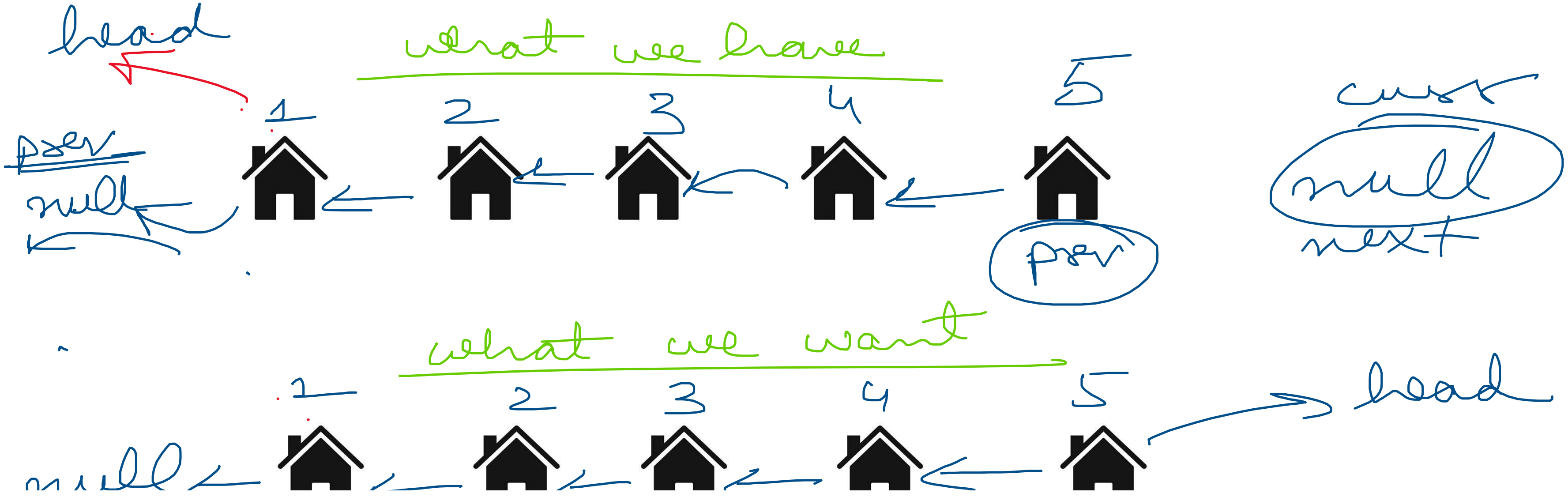
element at end.

Doubly linked list



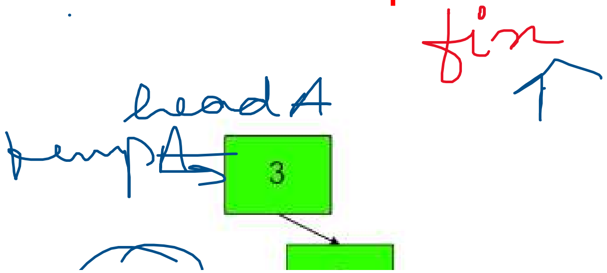
```
class Node
{   int data;
    Node next;
    Node prev;
}
```

Reverse a linked list



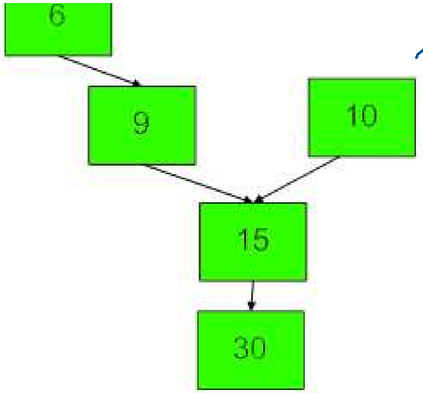
```
{
Node curr = head;
Node prev = null;
Node next = null;
while( curr != null)
  { next = curr.next;   <- Store the next value
    curr.next = prev;
    prev = curr;
  } curr = next;
return prev
}
```

Intersection Point in Y Shaped Linked Lists



```
while(tempA != null)
{
  while(tempB != null)
  {
    if (tempA ==tempB)
```

m



n

→ headB

```
if (tempA == tempB)
{ return tempA.data;
}
tempB = tempB.next;
}
tempB = headB;
tempA = tempA.next;
}
return -1;
```

→ o(m×n)

1 → m

2 → n

|

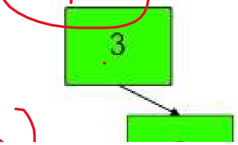
|

m → n

Efficient Way

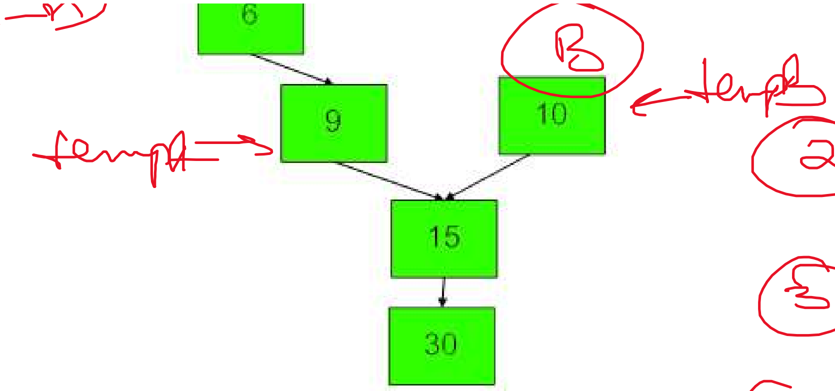
Clear?

A



1> Calculate length of

(m-n)

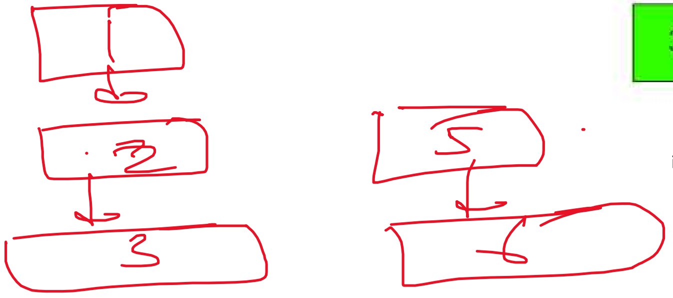


Both lists

② Call the longer LL→A

③ Take (m-n) steps in A

④ traverse together until equal or any becomes null.

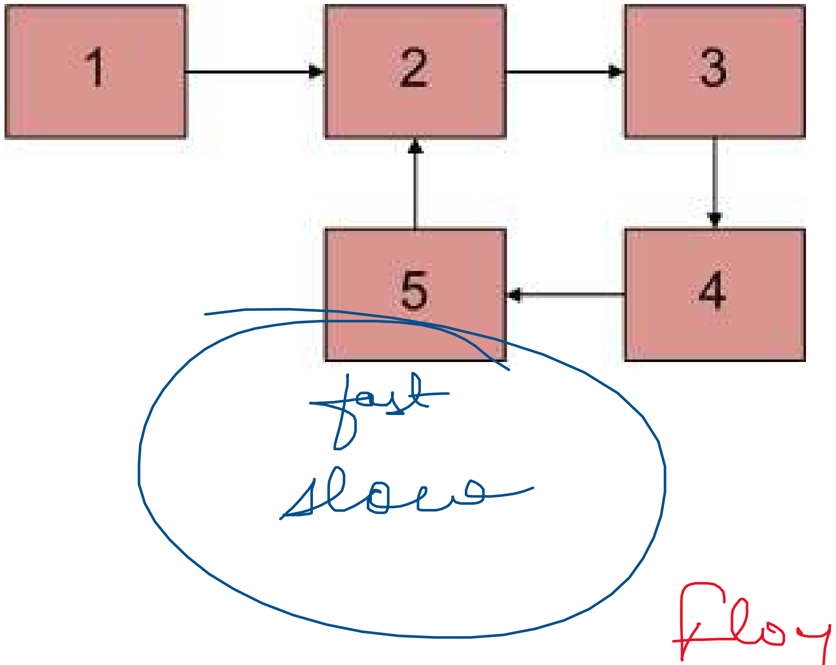


```
int intersectPoint(Node headA, Node headB)
     {
     int lenA = 0;
     int lenB = 0;
     Node tempA = headA;
     while(tempA!=null)
     {
        tempA=tempA.next;
        lenA++;
     }
     Node tempB = headB;
     while(tempB!=null)
     {
        tempB=tempB.next;
        lenB++;
     }
     //Length of both LL calculated in lenA, lenB
     int diff=0;
     if(lenA>lenB)
     {
        tempA=headA;//Call the longer one A
        tempB=headB;//Call the shorter one B
        diff=lenA-lenB;//Should be positive always
     }
     else{
        tempA = headB;//Call the longer one A
        tempB = headA;//Call the shorter one B
        diff = lenB-lenA;//Diff should be positive always
     }
     //Move the longer one diff steps ahead
     for(int i=0;i<diff;i++)
     {
        tempA=tempA.next;
     }
     //Reached the require state now compare
     while(tempA!=null && tempB!=null)
     {
        if(tempA==tempB)
        {
           return tempA.data;
        }
```

```
    tempA=tempA.next;
    tempB=tempB.next;
  }
  //Equality not found.
  return -1;
   }
```

Detect a Loop in a LL

1 → 2 → 3 → 4 → 5 → null



```
public static boolean detectLoop(Node head)
{
      Node slow = head;
      Node fast = head;
      while(fast!=null && fast.next!=null)
      {
         slow=slow.next;
         fast=fast.next.next;
         if(slow==fast)
         {
            return true;
         }
      }
      return false;
   }
```

Floyd's Cycle Detection Algorithm

emove a Loop in a
-

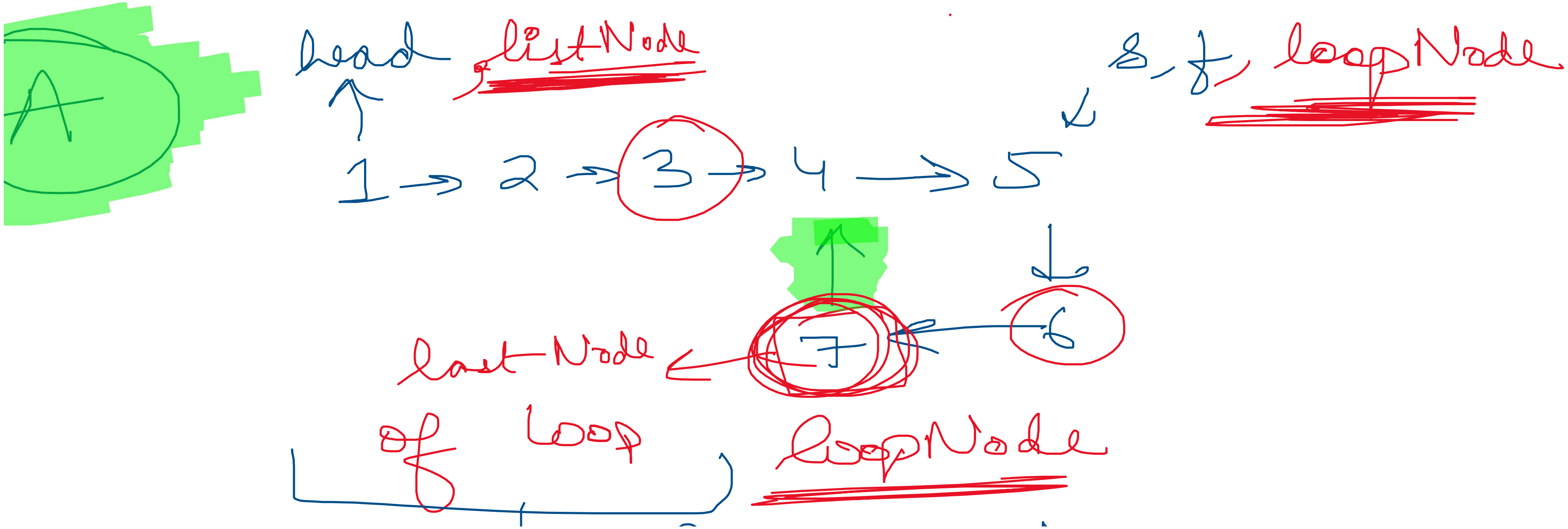
1 → 2 → 3 → 4 → 5

s,f

~~slow.next = null~~

1 → 2 → 3 → 4 → 5 → null

6, 7 → ud gaye



↳ leafNode has a next
lies on main list