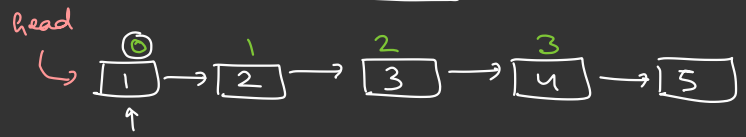
# LINKED LIST




Warm-up

T = 4

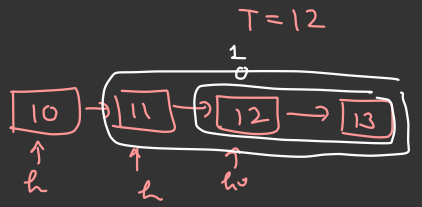
Output → 3

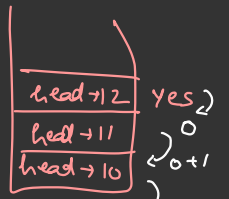
Search

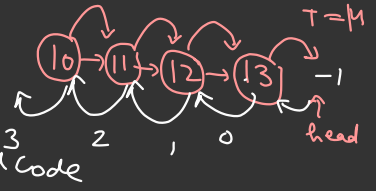
```
int  recSearch (Node head, int target) {
    //Base Case
    if(head == null) {
        return -1;
    }
    //Rec Case
    if( head.data == target){
        return 0;
    }
    else {
        → SubIdx = recSearch ( head.next, Target)
        if(subIdx == -1)
            return -1
        else
            return subIdx + 1
    }
}
```

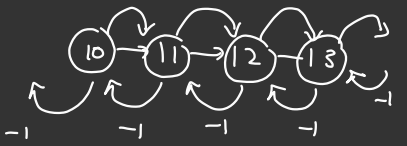
Space → O(N)

Time → O(N)

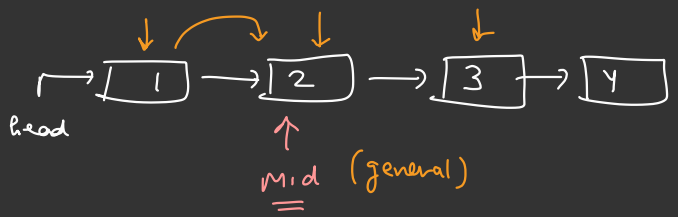


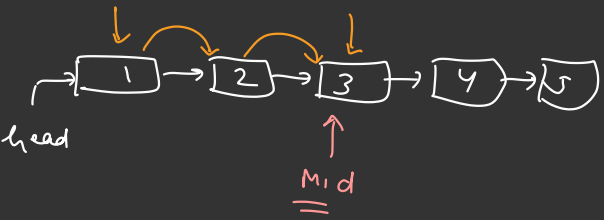






## Q2. Middle Node of Linked List




5 nodes

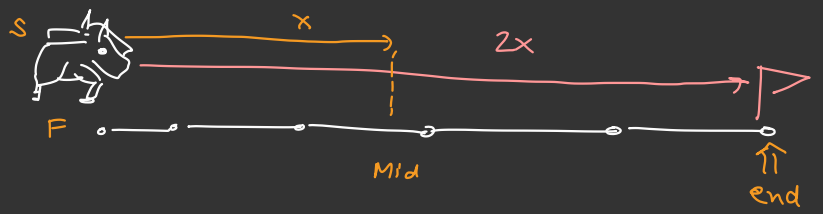
**Algo-1**

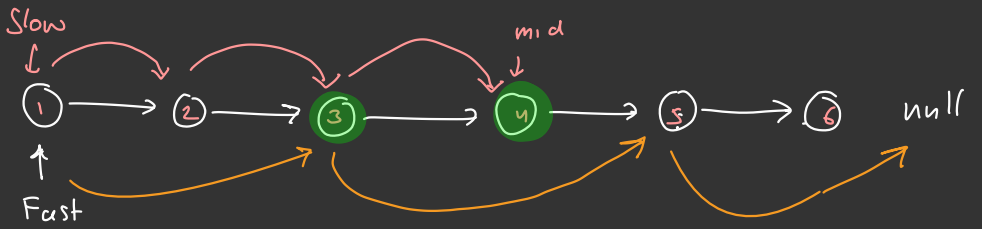
1) Iterate & Count total $n$
2) Iterate $n/2 - 1$ times even len
   $n/2$ times odd len

$N + N/2$

$= O(N)$ with $O(1)$ space

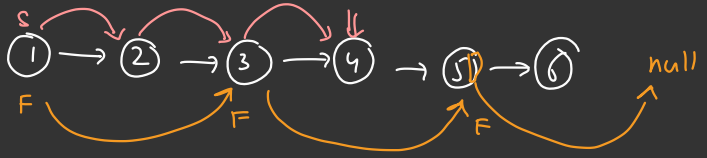
**Algo-2**





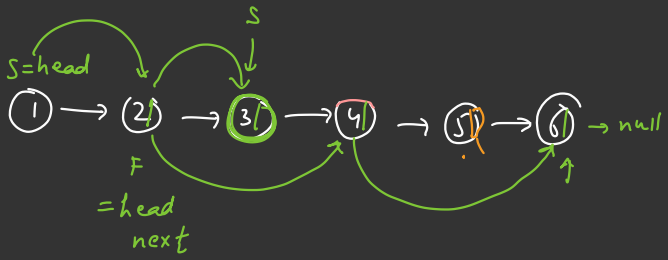

another way - you start fast one step ahead than slow.

Same

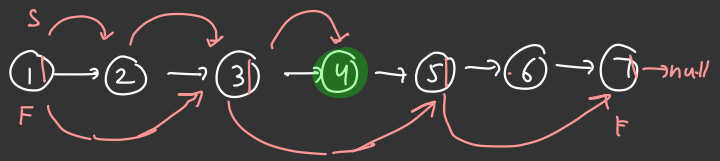



fast = fast next next
slow = slow.next

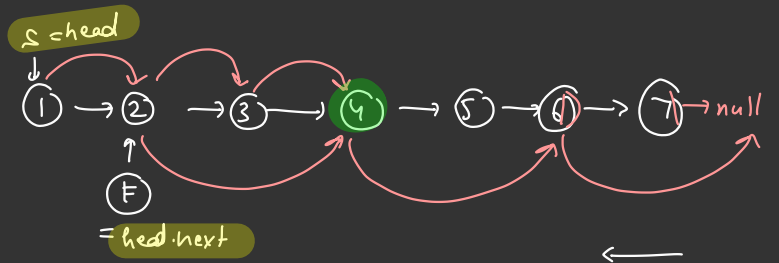
Even Nodes



ODD Nodes

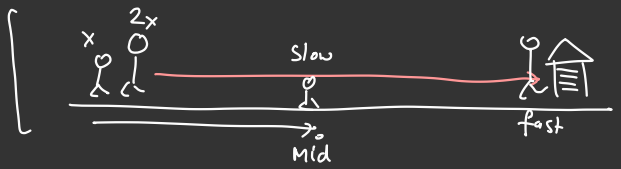



same' algo =>



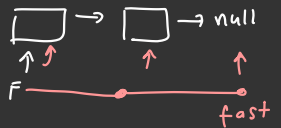


## Runner Technique



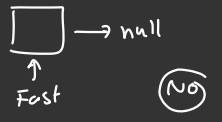

Code -

```
Node    getMid ( Node head ) {

        Node  slow  =  head
        Node  fast  =  head.next
```

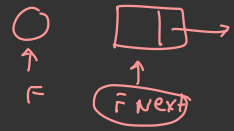



⇒ while( fast != null && fast.next != null )

```
            fast = fast.next.next
            slow = slow.next
        }
        return slow
}
```







a = x.next

b = a.next

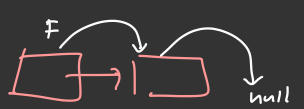
= x.next.next

!=null

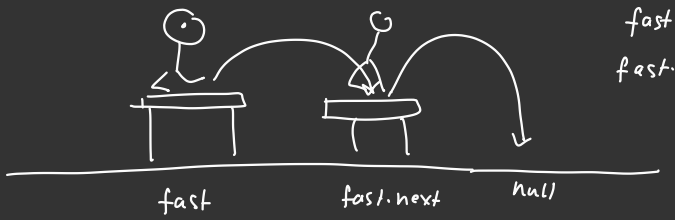
if x is not null

if a is not null

x.next != null



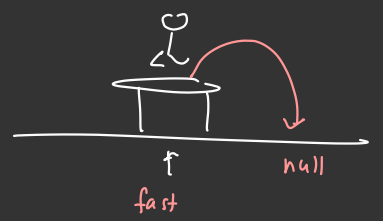





fast != null

fast.next != null

null

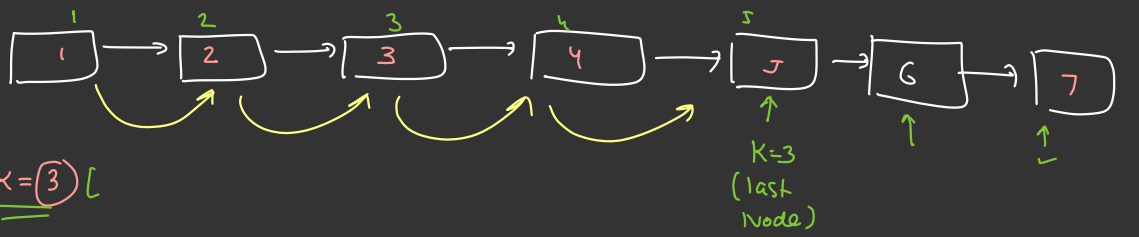



fast != null

fast.next == null

Q $k^{th}$ last Node (from end)

pos




K = (3) [

TODO 5 Mins 9.55

Algo-1 Find len N = (7) O(N)

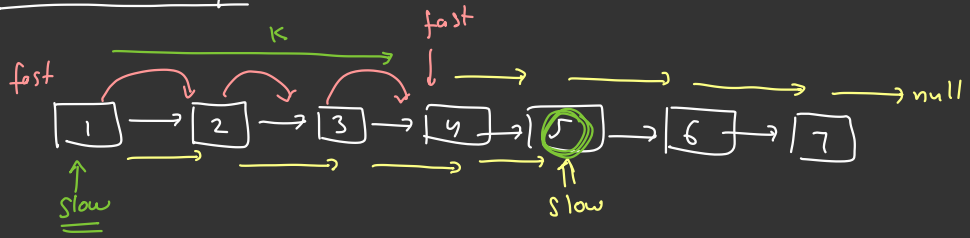
→ N-K+1 Node from beginning 7-3+1 = $5^{th}$ Node

(N-K) + 1-1 jumps

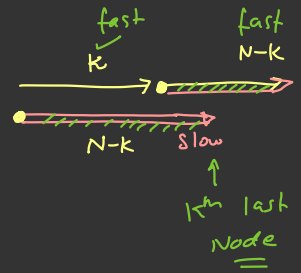
[ loop N-K times:
head = head.next ]

O(N) + O(N-K)
= O(N)

Algo-2 Runner technique

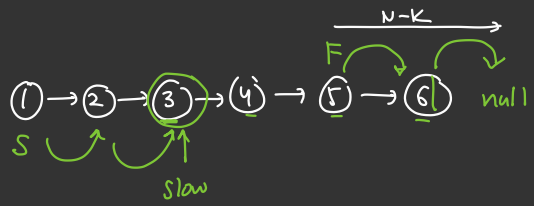



Why :

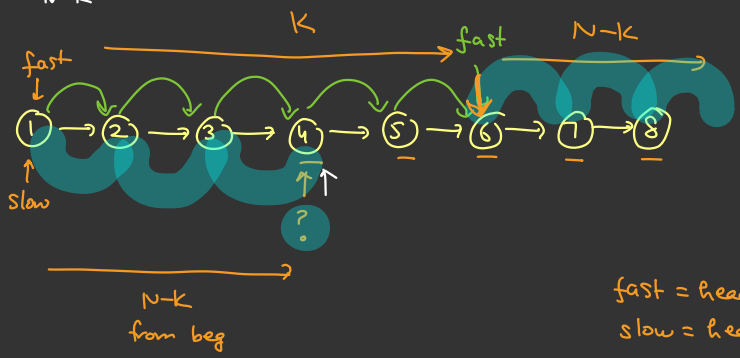



① Move fast by K position

② Move fast & slow together till fast is not null




K = 4th from behind




K = 5

fast = head
slow = head

Q K times

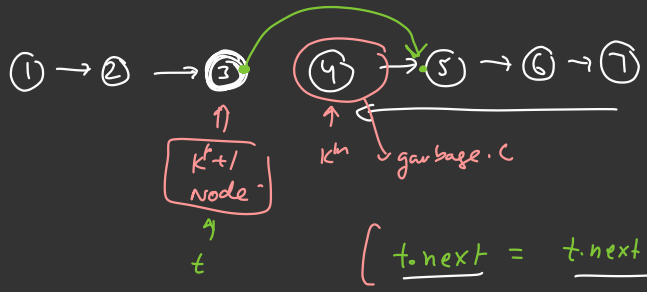
① Take fast K steps ahead. ⇒ fast = fast.next

② move fast = fast.next
slow = slow.next

till fast become null

③ return slow

delete kth from end

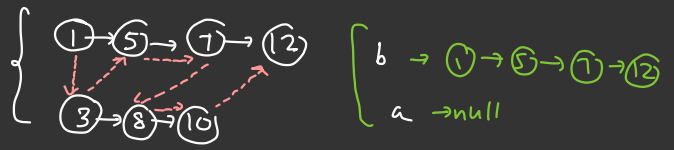


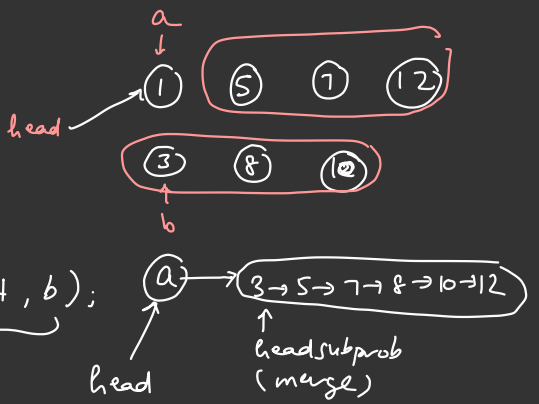

t.next = t.next.next

Welcome Back!

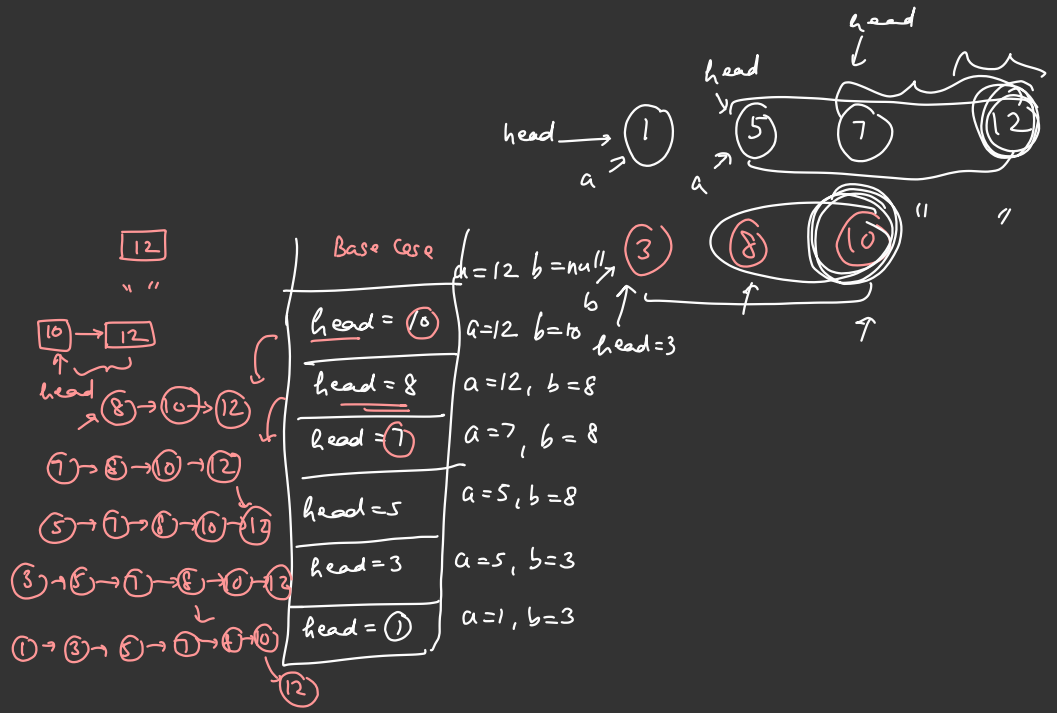

(9) Two Sorted linked list → Merge Them to get a Sorted LL

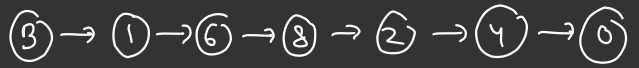
```
Node merge(Node a, Node b){
  // Base Case
  if(b==null)
      return a
  if(a==null)
      return b;

  //Rec Case
  Node head;
  if(a.data <=b.data)
      head = a
      head.next = merge(a.next, b);
  else
      head = b
      head.next = merge(a, b.next)

  return head
```



}


head
head
head
a
a
1
5
7
12
8
10
3
b
head=3
12
Base Case
a=12 b=null
head = 10
a=12 b=10
head = 8
a=12, b=8
head = 7
a=7, b=8
head=5
a=5, b=8
head=3
a=5, b=3
head = 1
a=1, b=3
10 → 12
head
8 → 10 → 12
7 → 8 → 10 → 12
5 → 7 → 8 → 10 → 12
3 → 5 → 7 → 8 → 10 → 12
1 → 3 → 5 → 7 → 8 → 10
12



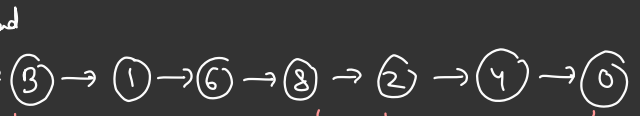
Sort

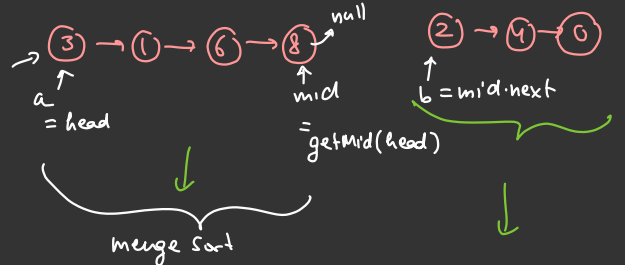
0 → 1 → 2 → 3 → 4 → 6 → 8

Merge Sort

→ Break at Mid
→ Rec Sort 2 part
→ Merge

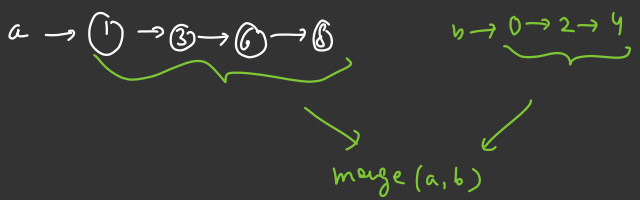
Input:




mergeSort(head)

=



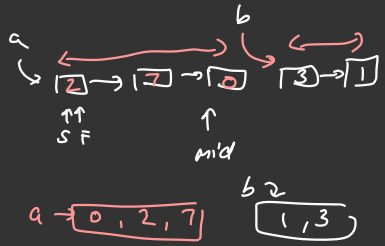
Node

```
mergeSort ( Node head )
// Base Case (0/1 element)
if ( head == null or head.next == null ){
    return head
}
// Rec Case
=> Node mid = getMid(head)
   Node a = head
   Node b = mid.next
   mid.next = null
a = mergeSort (a)
b = mergeSort (b)
return merge(a,b)
}
```

$$T(n) = \underbrace{k}_{\text{Mid}} + 2T\left(\frac{n}{2}\right) + \underbrace{kn}_{\text{Arrays}}$$

$$= kn + 2T(n/2)$$

$$= O(n \log n)$$



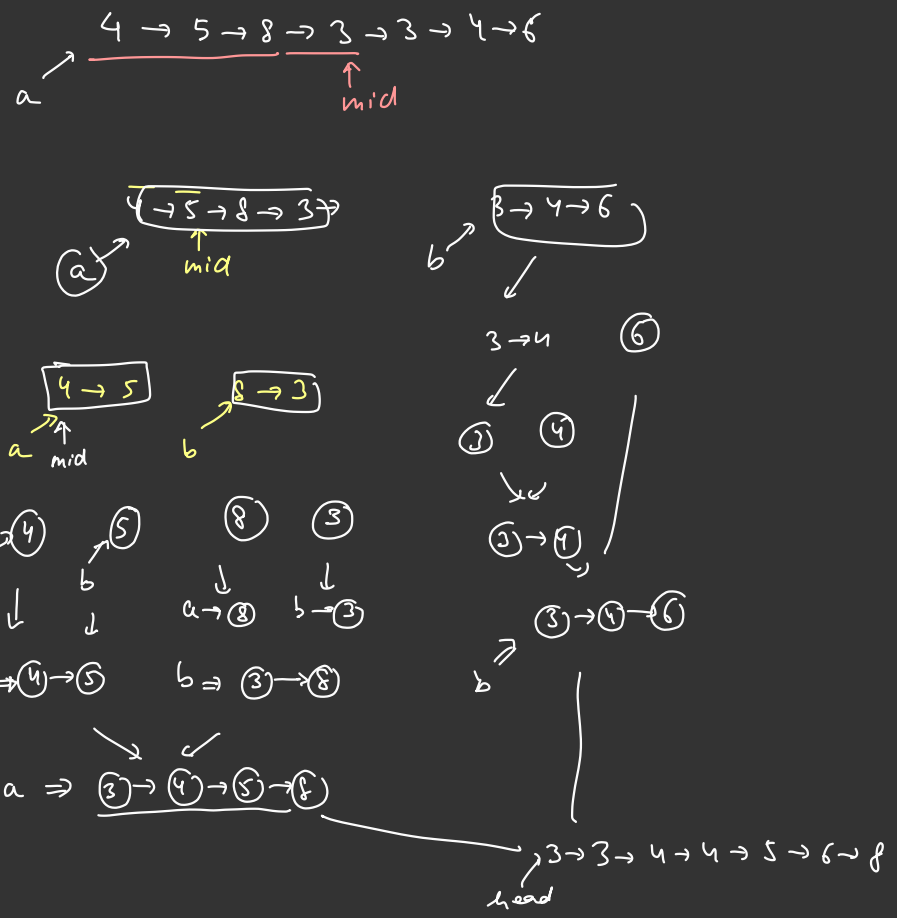

$$T(n) = kn + 2T\left(\frac{n}{2}\right) + kn$$

$$= 2kn + 2T\left(\frac{n}{2}\right)$$

$$= kn + 2T\left(\frac{n}{2}\right)$$

$$= O(n \log n)$$

LL


4 → 5 → 8 → 3 → 3 → 4 → 6
a
mid
4 → 5 → 8 → 3
a
mid
3 → 4 → 6
b
4 → 5
a
mid
8 → 3
b
3 → 4
6
3
4
3 → 4
a
4
5
b
8
3
a → 8
b → 3
a ⇒ 4 → 5
b ⇒ 3 → 8
3 → 4 → 6
b
a ⇒ 3 → 4 → 5 → 6
3 → 3 → 4 → 4 → 5 → 6 → 6
head
Cycle detection ?
next class