Sorting and Detecting Loop

Middle of the Linked List

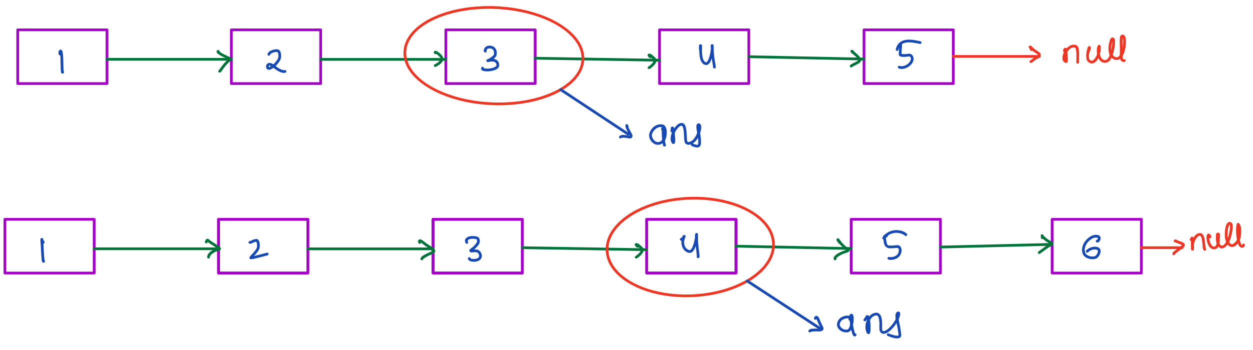
Merge two sorted linked lists

Merge Sort

Circular Linked List

Middle of the Linked List

Find the middle element in the linked list



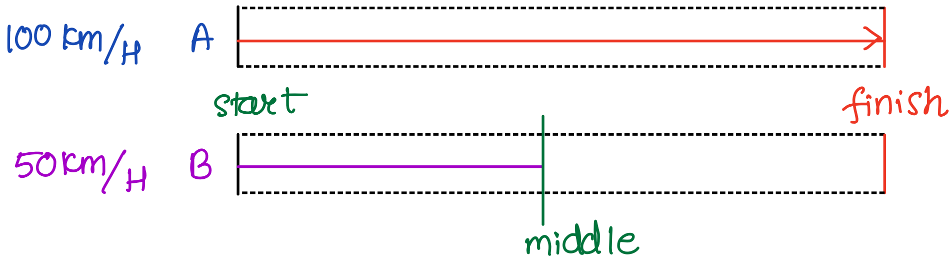
Idea from previous class

step 1> Get total size of linked list

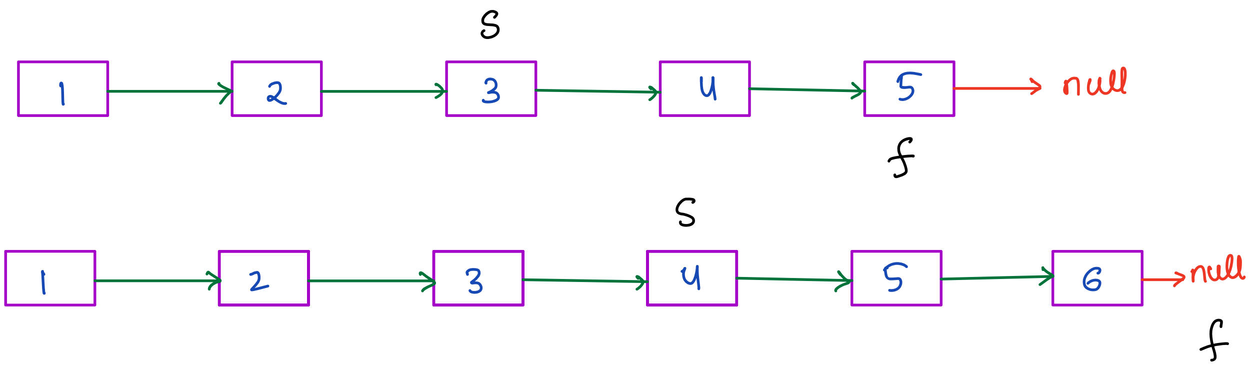
step 2> Traverse to size/2 index.

Idea 2

car



slow and fast pointers



Pseudocode

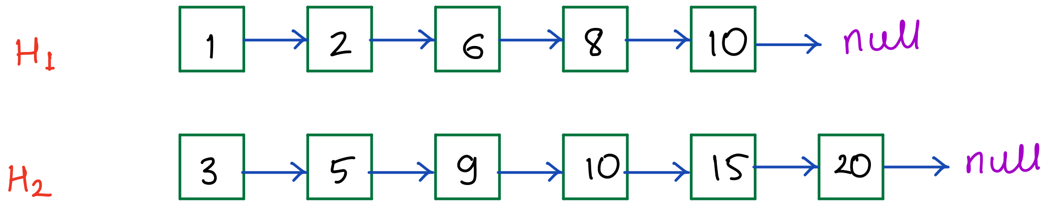
```
Node   getMiddle ( Node  head ) {
      slow  =  head
      fast   =  head

      while ( fast != null  && fast.next  != null ) {
            slow  =  slow.next        // jump 1
            fast  =  fast.next.next   // jump 2
      }

      return   slow
}
```

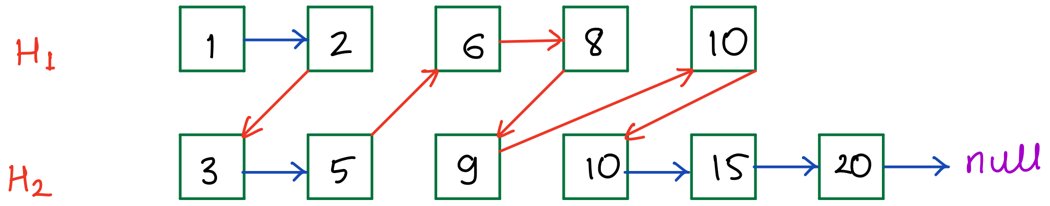
TC : O(N)

SC : O(1)

Merge two sorted linked lists

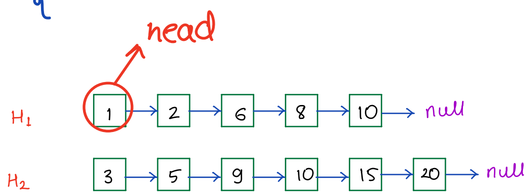
Q → Merge two sorted lists into one sorted list.



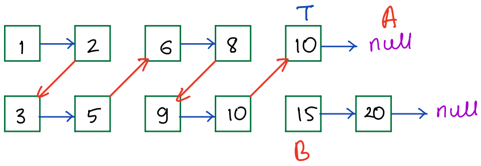


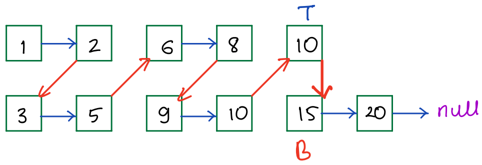
Pseudo code

```
Node merge ( A , B ) {
        if ( A == null ) return B
        if (B == null ) return A

        head = A
      if ( A.val < B.val ) {
              head = A
              A= A.next
      }
      else {
              head = B
```



```
        B = B.next
    }

    temp = head

    while ( A != null && B != null) {
        if ( A.val < B.val ) {
            temp.next = A
            A = A.next
            temp = temp.next.
        }
        else {
            temp.next = B
            B = B.next
            temp = temp.next.
        }
    }

    if( A== null) {
        temp.next = B
    }
    if( B == null ) {
        temp.next = A
    }
}
    return head
```

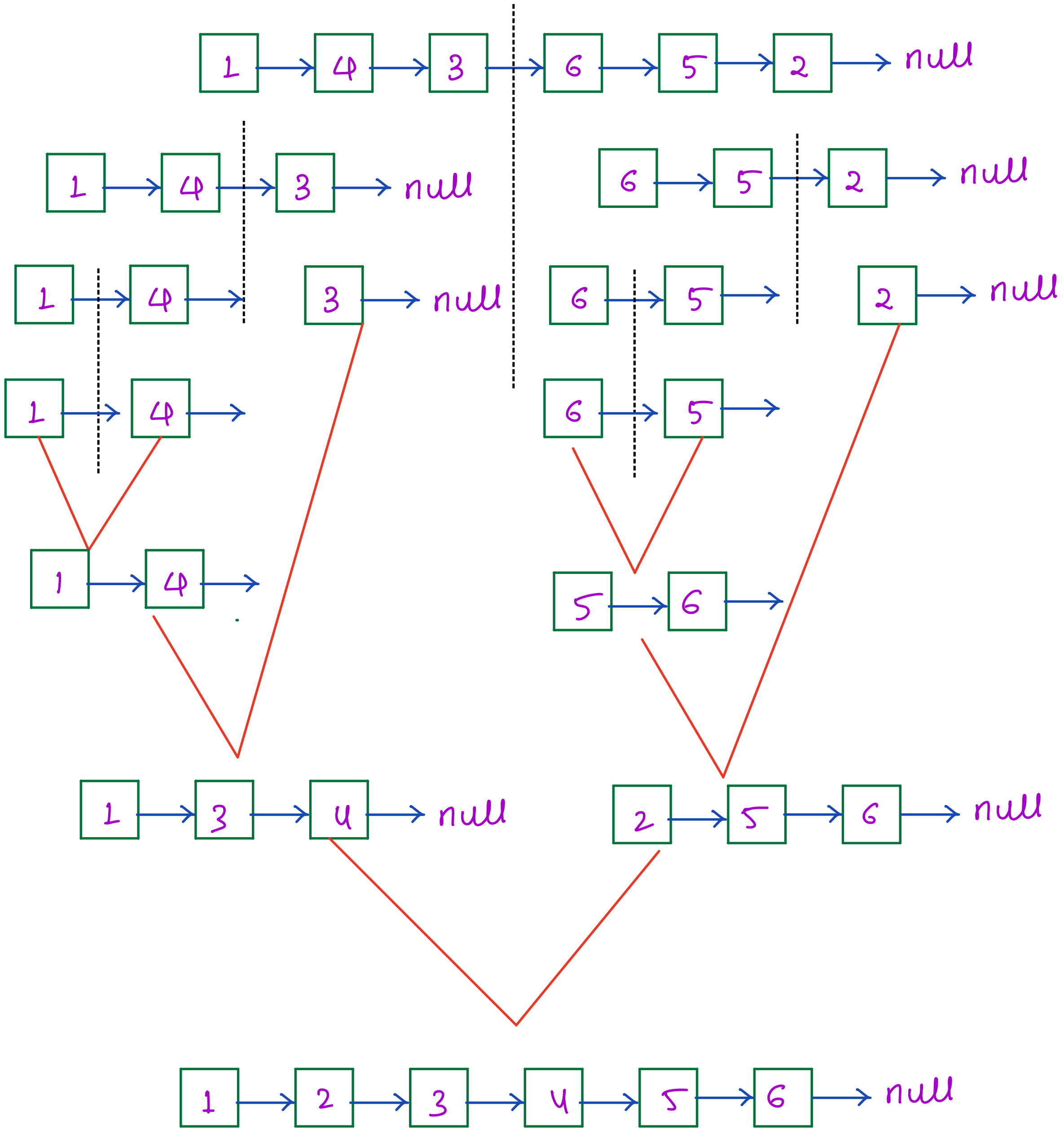




TC: O(N+M)

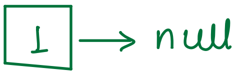
SC: O(1)

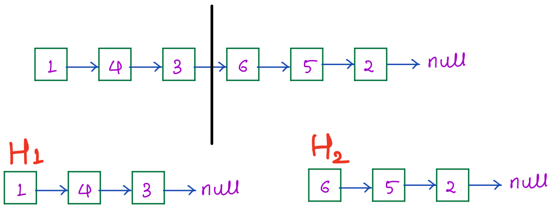
Merge sort { Divide & conquer }



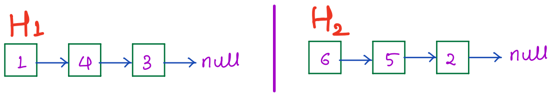
Pseudocode

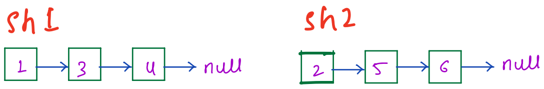
```
Node    mergeSort ( head) {
        // Base condition
        if(head == null) {
                return head
        }

        if(head.next == null){
                return head
        }

        mid = getMid (head)

        h1 = head
        h2 = mid.next

        mid.next = null

        sortedh1 = mergeSort (h1)
        sorted h2 = mergeSort (h2)

        return merge(sortedh1 , sortedh2)
}
```





TODO Modify getMid to return 3 instead of 6 in case of even



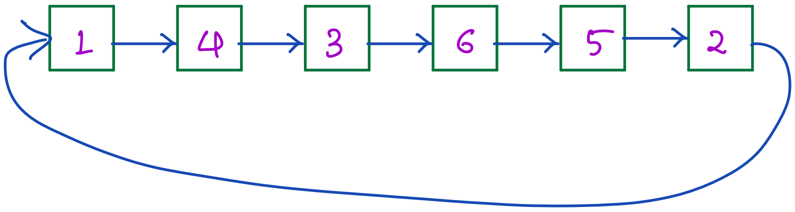


TC: O(N log N)

Recursive Stack Space SC : O(log N)

Circular Linked List

8:40 am



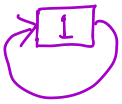
1) which node should be the head ?

Any

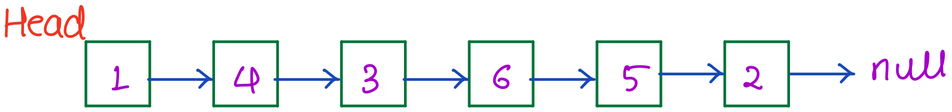
2) How to travel if there is no null node ?

⟶ Count of all nodes in circular linked list

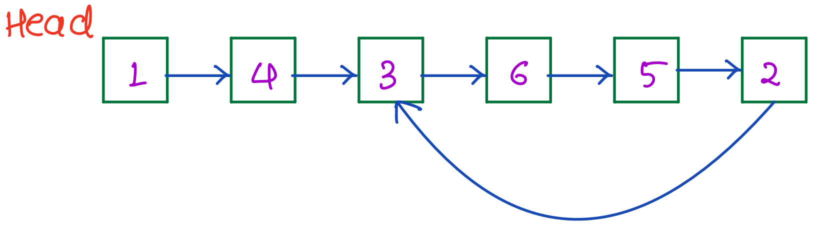
```
if (head == null) print(0) return

temp = head  // given as input
count = 1

while ( temp.next != head ) {
        count += 1
        temp = temp.next
}

print(count)
```



Q)* Check if the given linked list has a cycle *



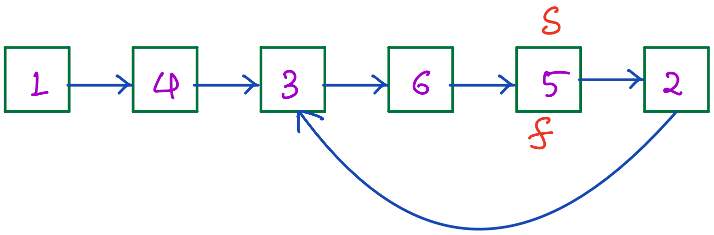
False



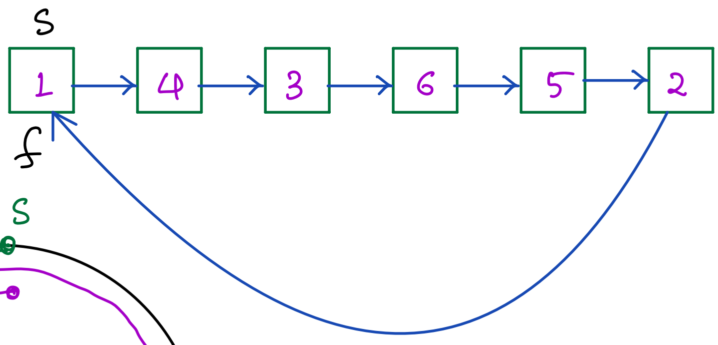
True

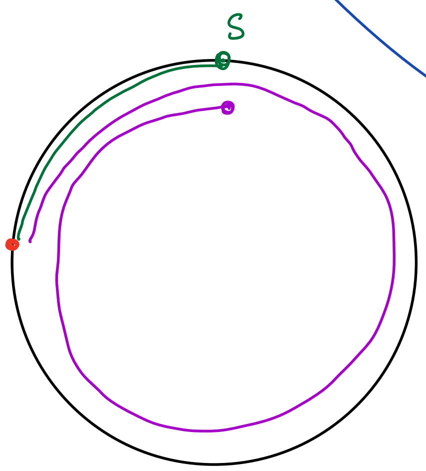
Bruteforce : Use hashset, if we get to the node return true.

Idea : Slow and Fast pointer



```
Boolean findCycle ( Node head ) {
        slow = head
        fast = head

        while ( fast != null 88 fast.next != null ) {
              slow = slow.next          // jump 1
              fast = fast.next.next     // jump 2

              if (slow == fast)  return  true
        }
}       return  false
```





TC : O(N)

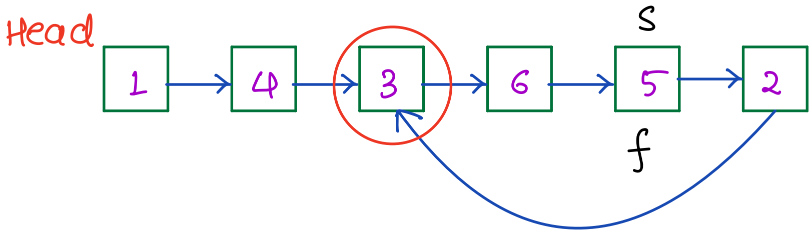
SC : O(1)

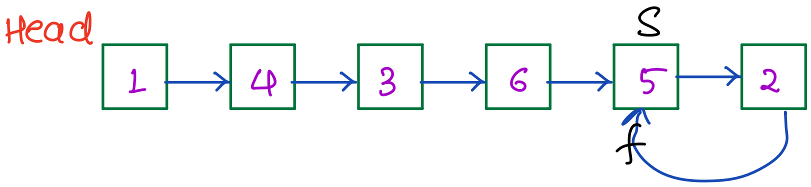
frame of reference

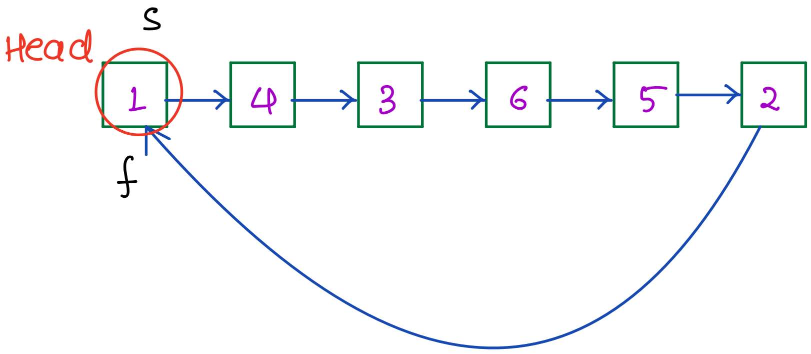
Time

s = 1 km/h	0	1	2	3
f = 2 km/h	0	1	2	3
	0	2	4	6
Dist b/w s/f →	0	1	2	3

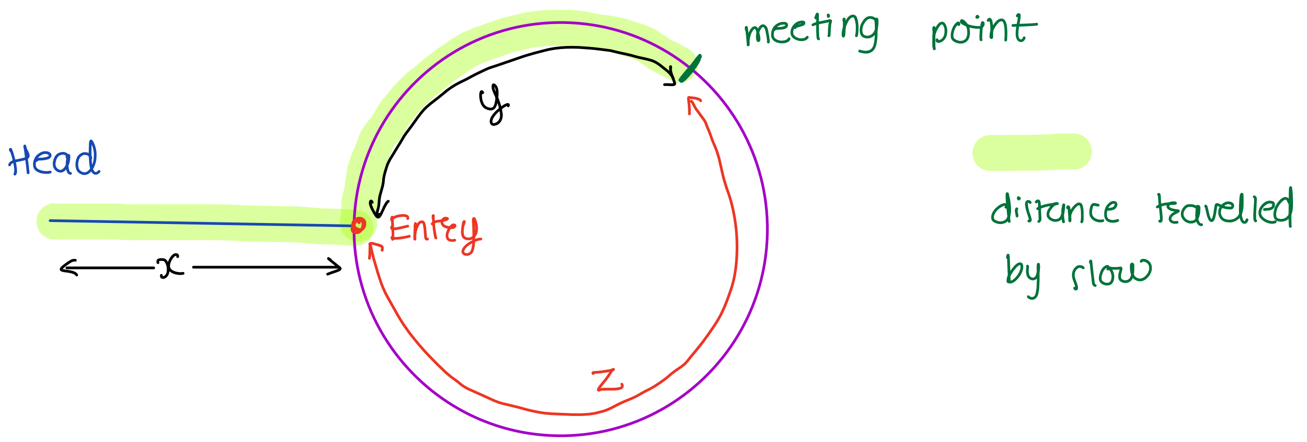
Q ⟶ Given a linked list with cycle, find the start of the cycle ?





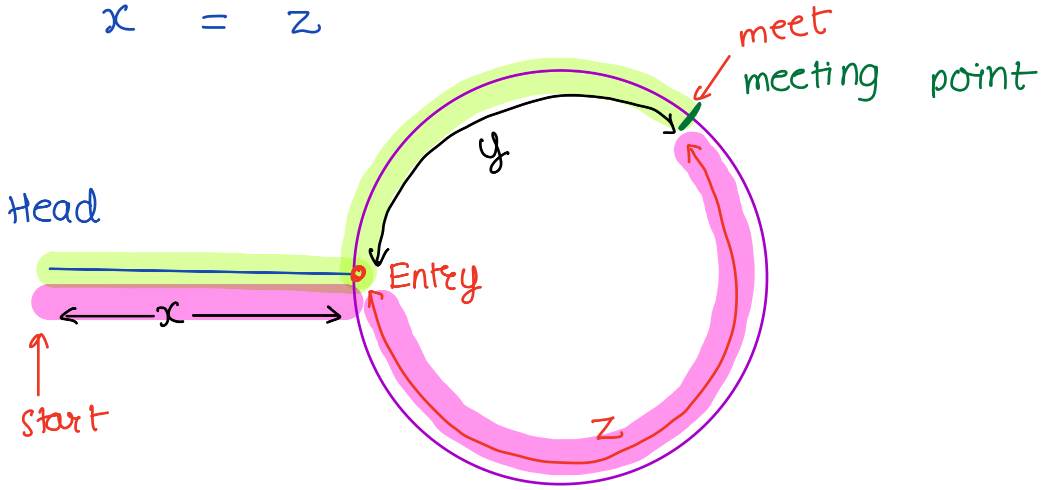


Idea – slow & fast pointers



$$slow = x + y$$
$$fast = x + y + z + y$$

$$2 * \{x + y\} = x + y + z + y$$
$$2x + \cancel{2y} = x + z + \cancel{2y}$$
$$x = z$$



from the meeting point traverse 1 step at a time
and start again from the head
Once both meet thats my entry point

```
Node  EntryPoint( Node  head ) {
    slow  =  head
    fast  =  head

    while ( fast != null && fast.next != null ) {
        slow  =  slow.next          // jump 1
        fast  =  fast.next.next     // jump 2

        if (slow == fast)   break
    }

    // slow and fast are at meeting
    meet  =  slow
    start  =  head

    while ( meet != start ) {
        start = start.next
        meet  =  meet.next
    }

    return  start
}
```

TC : O(N)

SC : O(1)

Doubt