Recap: Linked List → CRUD

1. Read — while (curr != null): curr = curr.next — print

2. Create — append (head, data):

insert (head, data, position) — stand 1 node behind position

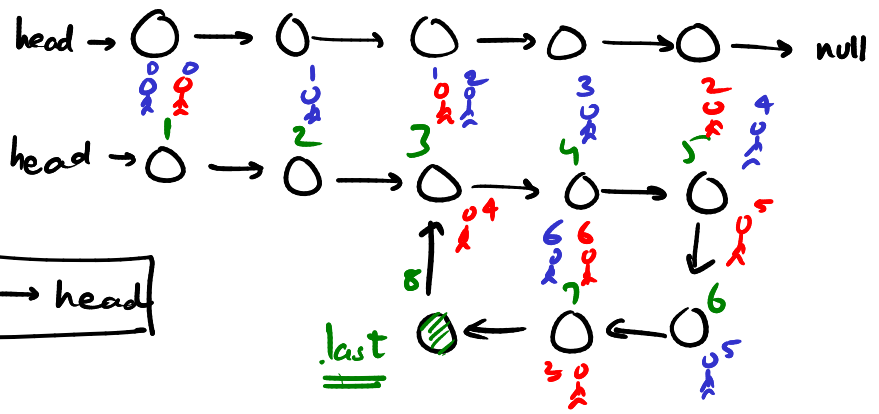
3. Update — update (head, new data, position) — stand on position.

4. Delete — delete (head, position)
   a) deleting the head node
   b) delete in-between node
   c) delete last node

corner cases:
1) What if list is empty?
2) What if it has only 1 node?
3) what if at the start?
4) in between?
5) in last?

4 topics:
① Loop
② Insertion
③ Sorting
④ Reversal.

Looping:



normal LL

loop in LL.

CLL → last → head

① Hashset.

2. a) Detect if a loop exists b) Break the loop. set<Node> s

1 branch | 2 pointer method: slow fast.

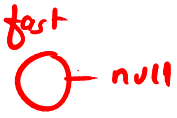
slow = head
fast = head

fast reaches null → no loop

fast catches slow → loop → O(n) time

slow = slow.next
fast = fast.next.next

Nuances: fast 2 steps
fast.next should also exist.



slow = head → if head == null || head.next == null : return false
fast = head

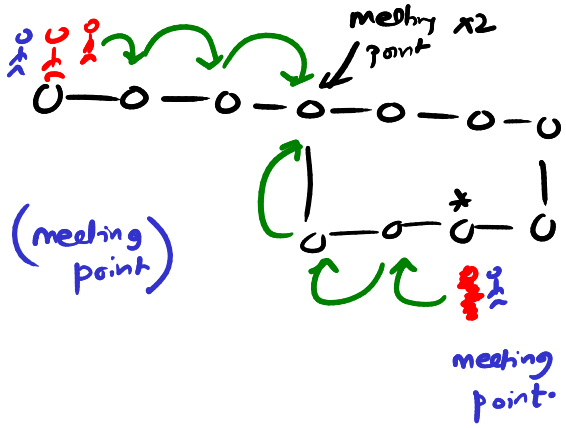
slow = slow.next
fast = fast.next.next

while fast != null and fast.next != null and slow != fast :
    fast = fast.next.next
    slow = slow.next

//do not which caused loop to break

return fast == slow ;

Remove the loop :



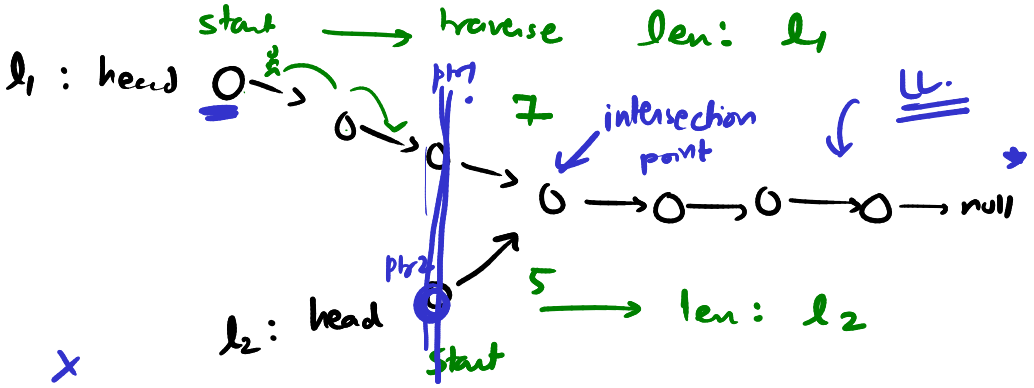
1. Start slow & fast at head.
2. Run the slow & fast till they meet.
3. fast -> head
4. Run slow & fast both at same pace,
   1 step at a time.
5. The next time they meet, they surely meet at looping point.

if fast.next == slow.next : break

slow.next = null ; // break the loop !

Intersection.

if there is a intersection or not?



Logic: ptr1 == ptr2 ✗

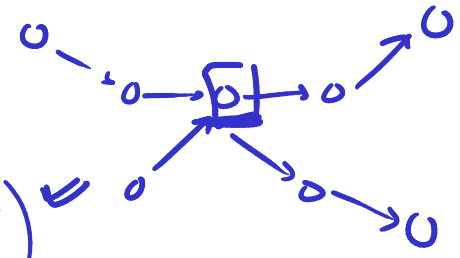
1 Method → → intersection exists.

$|l_1 - l_2|$

l_1 is 2 nodes bigger

$|7 - 5| = 2$

2nd method → set { 7 nodes }



(understand + memorize)

③ Sorting

mergesort

n nodes chain i/p:

n/2 n/2

chains

n/4 n/4 n/4 n/4

. . . .

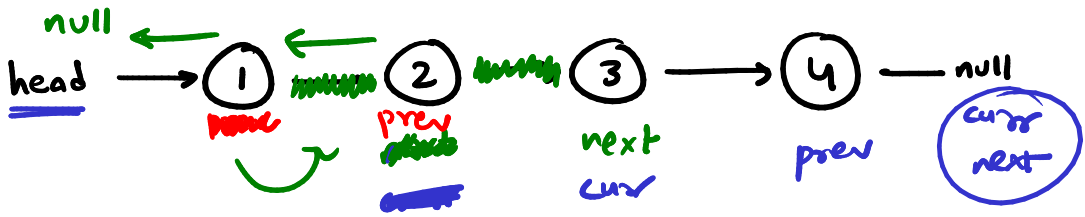
1 node length chains

merge

2 linked lists.

3rd linked list in order

④ Reversal.



4 steps {

curr != null

nextnode = curr.next

curr.next = prev

prev = curr

curr = next

} head = prev

③ Sorting an LL.

✗ bubble sort ✗ quicksort ✓ mergesort. (nlogn)

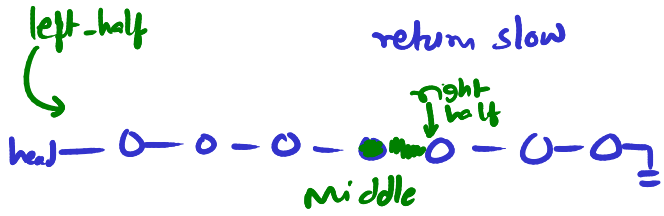
```
merge_sort (head):
    if head == null or head.next == null:
        return head

    middle_node = middle(head)

    right_half = middle_node.next
    middle_node.next = null
    left_half = head

    left = self.merge_sort(left_half)
    right = self.merge_sort(right_half)
    result = merge(left, right)      return result
```

```
middle(head):
    slow = 1 step
    fast = 2 steps
    when fast = null
        slow => middle
    return slow
```




```
merge (left, right):
    result = None
    if not left:  return right
    if not right: return left

    if left.data <= right.data:
        result = left
        result.next = merge(left.next, right)
    else
        result = right
        result.next = merge(left, right.next)
    return result
```