```
class ListNode:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next
```

```
def search_list(L: ListNode, key: int) -> ListNode:
    while L and L.data ≠ key:
        L = L.next
    return L
```

Move onto the next node until L exists and the key match w/ data

O(n)

```
def insert_after(node: ListNode, new_node: ListNode) -> None:
    new_node.next = node.next
    node.next = new_node
```

O(1)

```
def delete_after(node: ListNode) -> None:
    node.next = node.next.next
```

O(1)

Reverse a linked list

To reverse a LL, we need to iterate through and LL and reverse one node at a time.
We do this by setting two variables, current and previous ~~and null equal to~~ point to ~~null and~~ head and null respectively.

current, previous = head, ~~null~~ None.

In a stepwise manner, we make current point to previous and previous point to previous.
When current is null, we stop the loop.

```
def reverse(head)
    prev, curr, next = None, head, None
    while curr is not None:
        next = curr.next
        curr.next = prev
        prev = curr
        curr = next
    return prev
```

curr.next, prev, curr = prev, curr, curr.next

Pythonic Tuple Assignment

7.2

Method: Extract sublist, reverse it, splice it back in.

(1) Identify the start of sublist by iteration to get the s^{th} node and it's predecessor.

(2) Reverse the process and keep counting

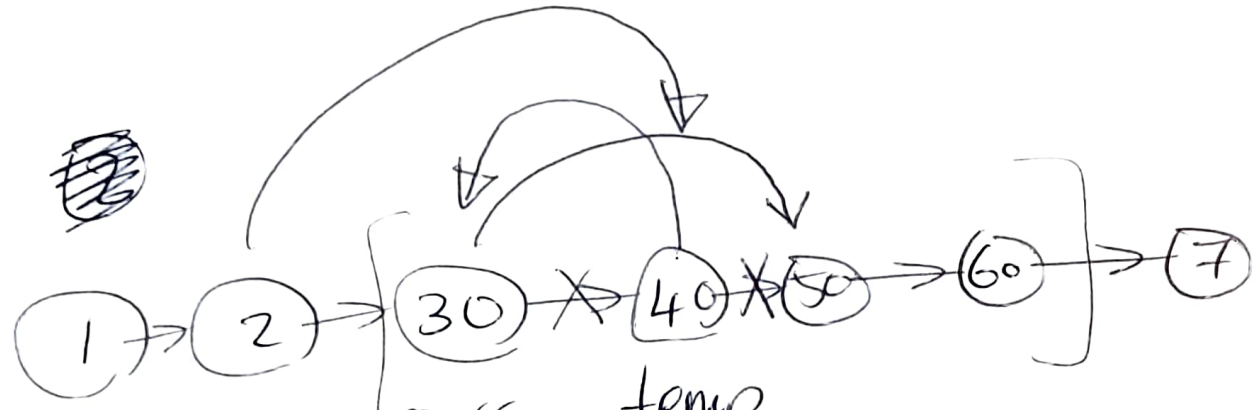
(3) When we reach the f^{th} node, stop reversing and link the reversed section w/ the unreversed sections

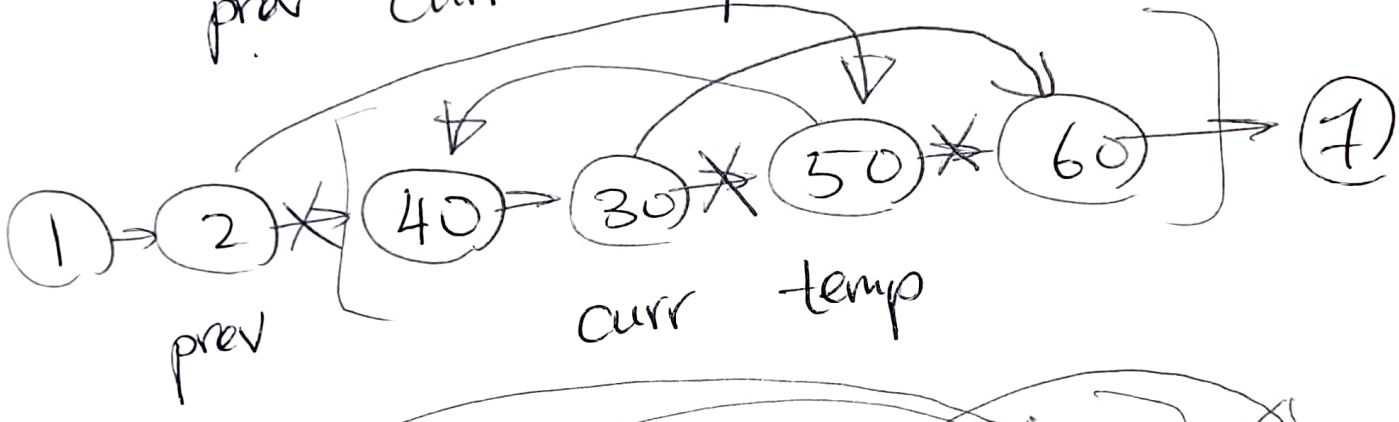
7.2

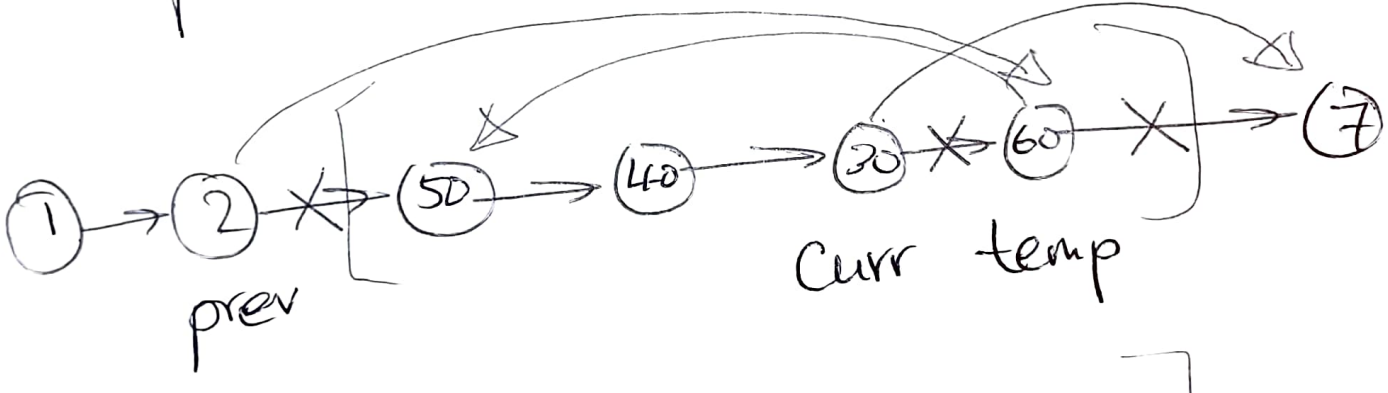
```
def reverse_sublist(L: ListNode, start: int
                    finish: int) -> Optional[ListNode]:
    dummy = sublist = ListNode(0, L)
1   for _ in range(1, start):
        sublist = sublist.next
    sublist_iter = sublist.next
2   for _ in range(finish - start):
        temp = sublist_iter.next
        sublist_iter.next, temp.next, sublist.next
            = (temp.next, sublist.next, temp)
3   return dummy.next
```

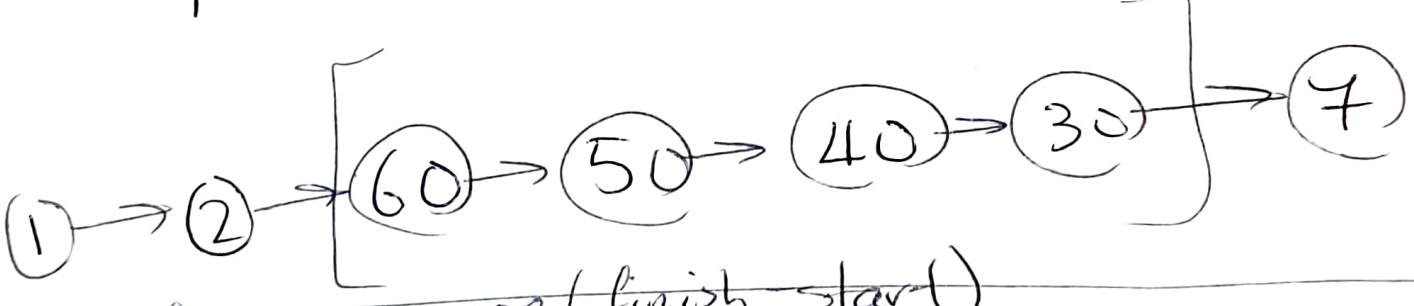
```
def reverse_sublist(L:ListNode, start:int, finish:int)
                -> Optional[ListNode]:
    dummy = prev = ListNode(0, L)
    for _ in range(1, start):
        prev = prev.next
    curr = prev.next
    for _ in range(finish - start):
        temp = curr.next
        curr.next, temp.next, prev.next
            = (temp.next, prev.next, temp)

    return dummy.next
```









~~while~~ for _ in range (finish - start)

temp = curr.next

curr.next, temp.next, prev.next = temp.next, prev.next, temp

7.1

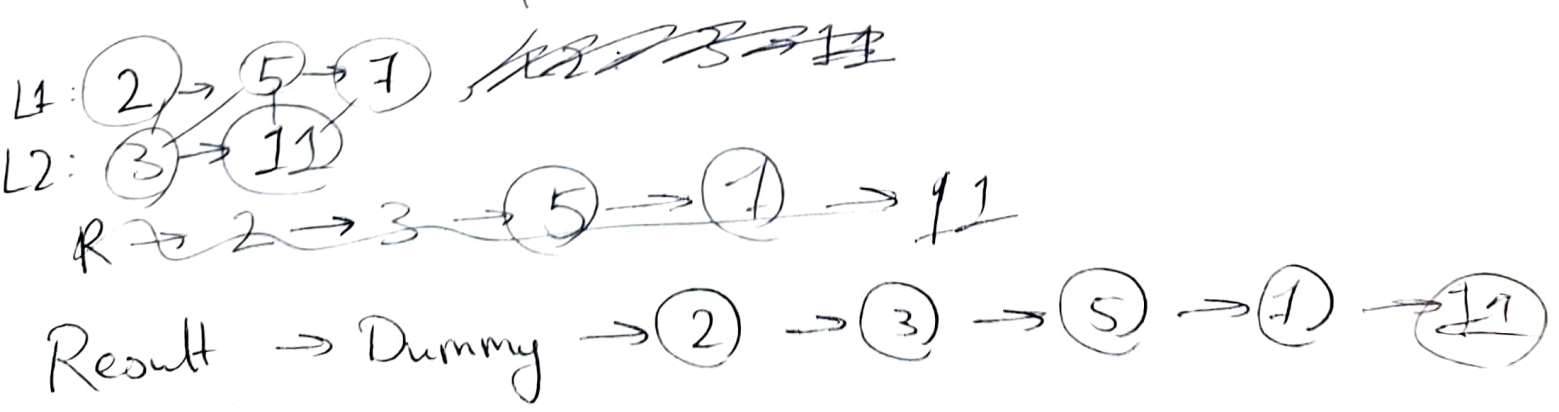
Input: 2 LL sorted
Output: Merged LL sorted

Naive approach is to append one list to the other and sort them ~~while~~ which will take $O(2n \log(2n))$

Another way is to iterate through one list, check each item values and insert from there. This will take $O(2n)$ time and $O(1)$ space.

↳ I don't know how to implement this.

8 min in Quit



Sol'n

- We create a dummy list and append to that list using tail.

  ↳ Starting w/ the dummy head, we insert after if the value from one of the list is less than the other,

- When it is, the tail.next is that value. and we move onto the next value in the list we are observing.

  → tail.next = l1

  → l1 = l1.next

- If we reached end of one list, append all values from the other since we are working w/ sorted lists.

```
def func(L1:, L2) → ListNode:
    dummy = tail = ListNode():

    while L1 and L2:
        if L1.data < L2.data:
            tail.next = L1
            L1 = L1.next

        else:
            tail.next = L2
            L2 = L2.next
        tail = tail.next
    tail.next = L1 or L2

    return dummy.next
```

7.2

Input: LL, two integers
↳ (L, s, f)

Output: LL

Numbering begins at 1 and starts w/ headnode.

→ We are taking a LinkedList and reversing it from the s^{th} to f^{th} node.

The naive approach is to ~~take~~ iterate and take node starting from sth node and swap w/ the end node and increment ~~and decrement~~ the index values.

Something like ~~ll[st] = ll[end]~~

ll[st], ll[end] = ll[end], ll[st]

st += 1

~~end -= 1~~

Using the Naive Approach, we can start from s^{th} index and end the loop ~~before~~ at the f^{th} index.

---

start = s
end = f
~~previous =~~
~~current =~~

How can we access Linked List value at a certain index?
↳ Search?

We can do something like the search method for linked list to ~~find~~ locate at s^{th} index.

```
while L and L.data ≠ S :
    L.next
previous = L
current = L.next
```

Now we can reverse

```
while start < end:
    current.next, previous, current
        = previous, current, current.next
    start += 1
```

4min