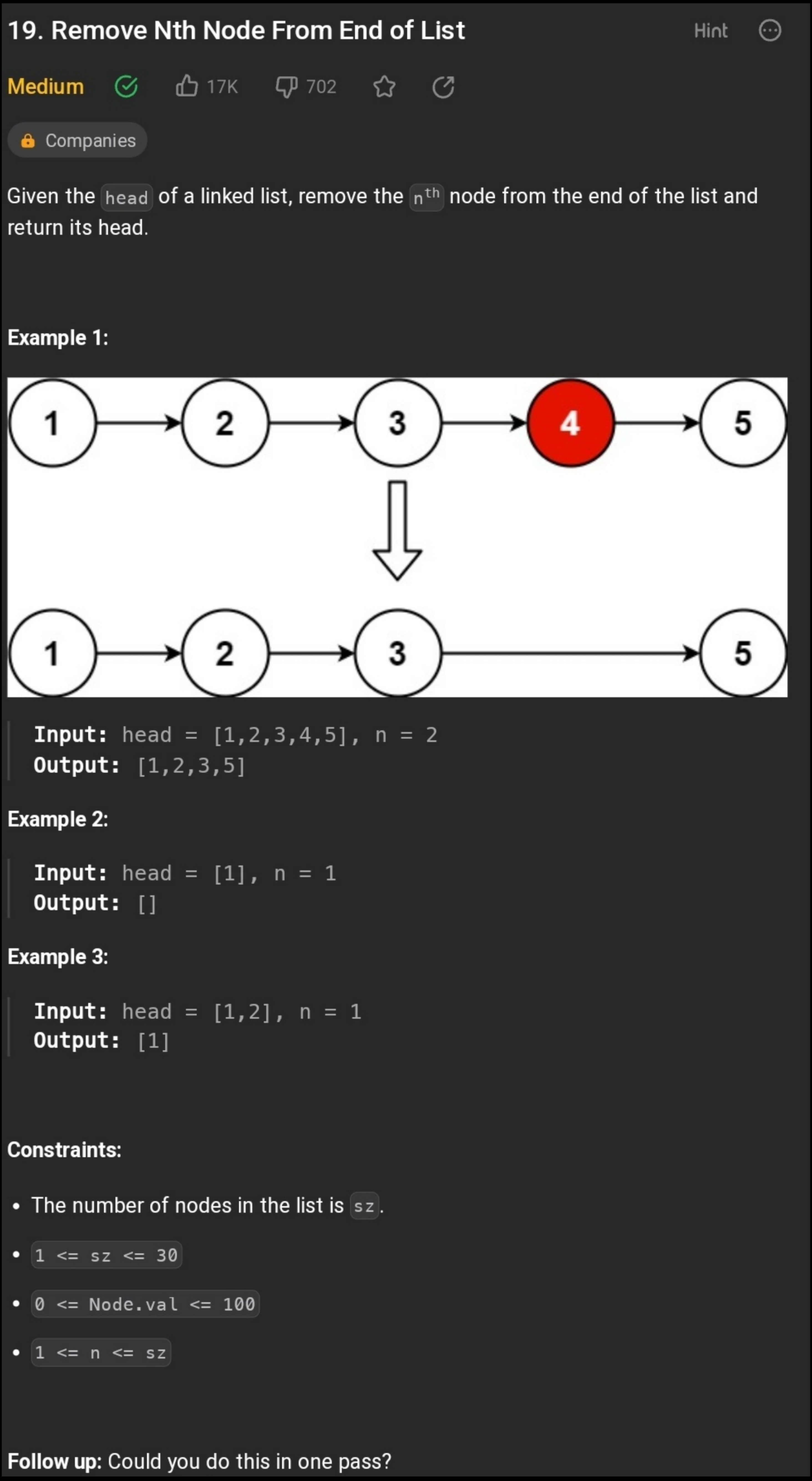


19. Remove Nth Node From End of List

Given the `head` of a linked list, remove the `n`th node from the end of the list and return its head.

Example 1:

```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

Example 2:

```
Input: head = [1], n = 1
Output: []
```

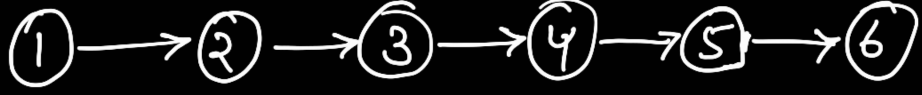
Example 3:

```
Input: head = [1,2], n = 1
Output: [1]
```

Constraints:

- The number of nodes in the list is `sz`.
- `1 <= sz <= 30`
- `0 <= Node.val <= 100`
- `1 <= n <= sz`

Follow up: Could you do this in one pass?



$k = 2$

Approach 1:

→ find length n of linked list.
→ if $k == n$ return head → next.

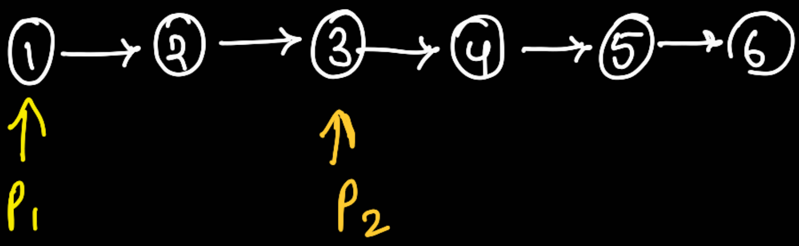
→ $x = n - k - 1$
→ start from head and make x steps. with a pointer p. and then make
$p \rightarrow next = p \rightarrow next \rightarrow next$

This is a trick to use instead of using two pointers. we can get the job done just by using a single pointer.

The above algo makes two passes over the linked list.

$T(n) : O(n) + O(n)$
$S(n) : O(1)$

Approach 2: In single pass



$k = 2$

from p_1 the   from P_2 the

no. of nodes that
we need to move
is $n-k$
i.e. $n-2$

no. of nodes we
can move before
p_2 becomes NULL
is $n-k$
$n-2$

Hence if we start moving both p_1 and p_2 at same time, by the time p_2 becomes NULL the p_1 would be exactly on the required node.

Q: But how can we keep p_2 on node ③?
make k no. of steps

$T(n) : O(k) + O(n-k)$
i.e. $O(n)$
a single pass

$S(n) : O(1)$