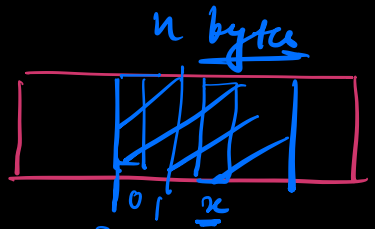
Linked Lists Basics :

Arrays

- Contiguous memory
- Sequential
- Linear

n bytes



→ constant time access

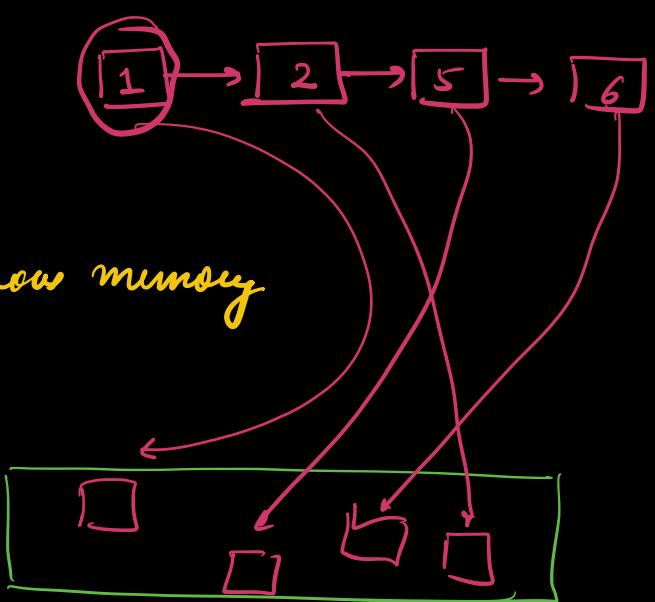
O(1)

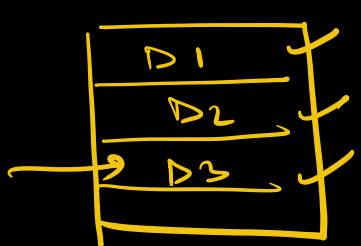
Problems

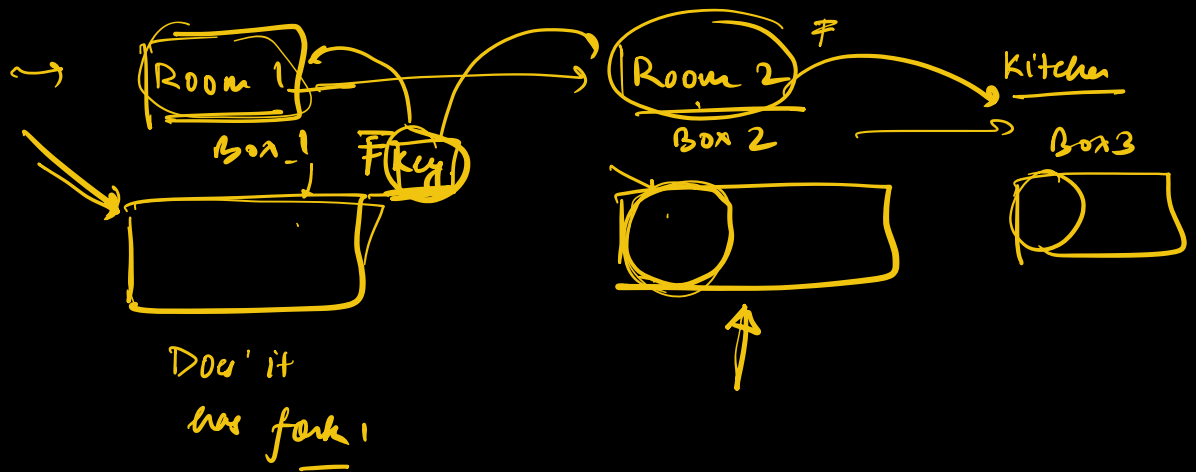
- Subsequence Question.
- Bit Manipulation
- Recursion
- Hashing

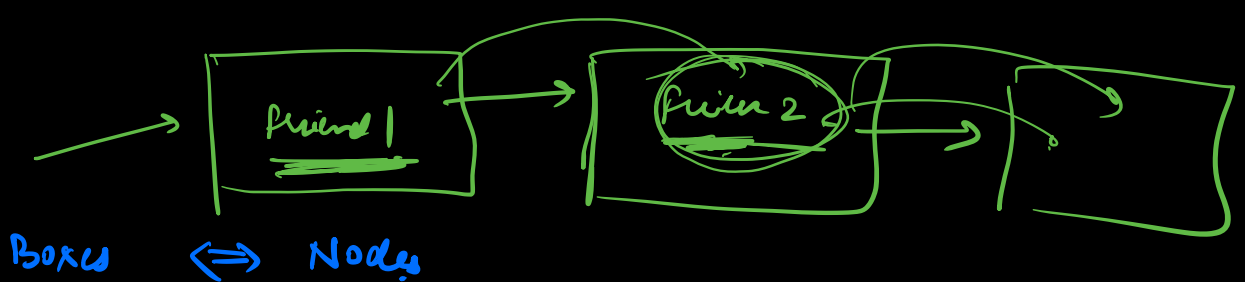
Linked List

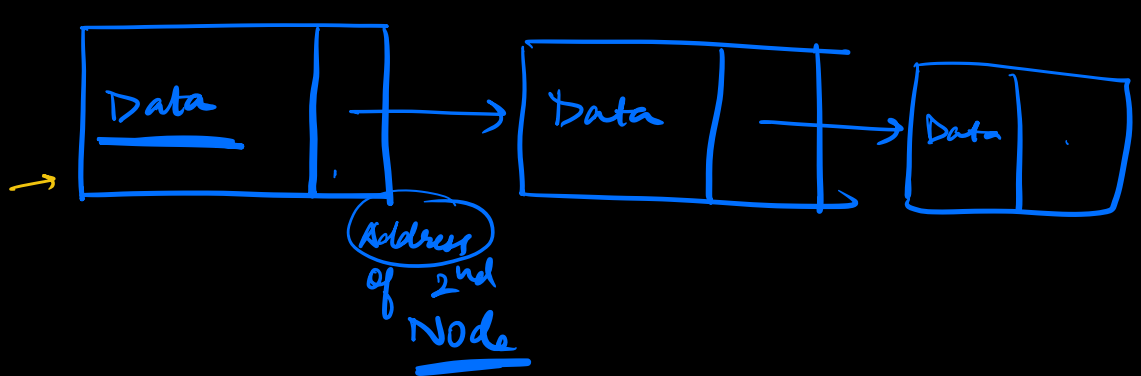
- Linear
- Sequential
- not contiguous memory allocation.



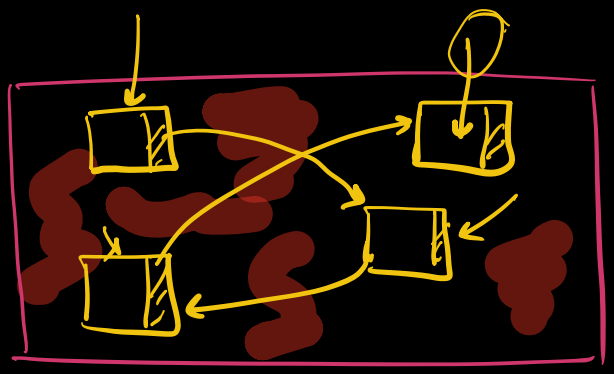








Singly Linked List



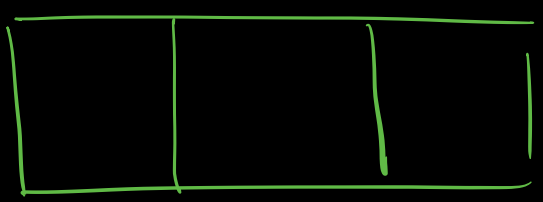
Elastic Search

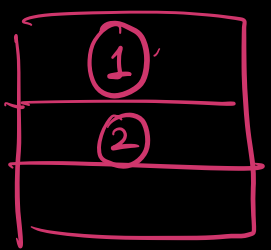
① LL are the building block of many data structures ⇒ Stacks / Queue / Trees / Tries

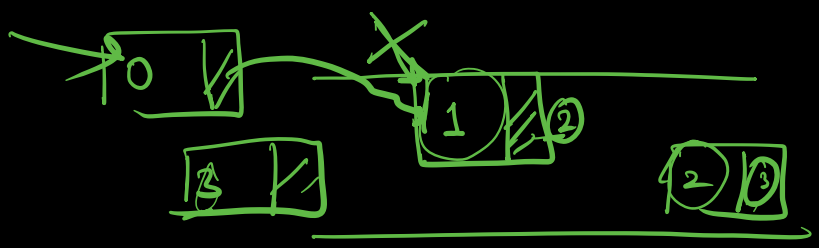
Collections are handled by LL in Hash Maps

LL + Hashmaps

Array

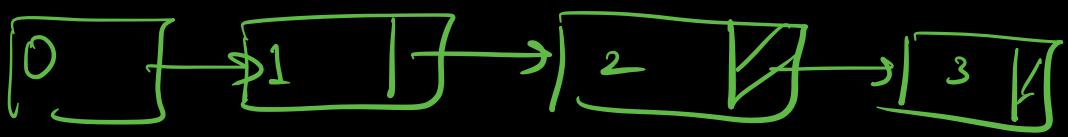


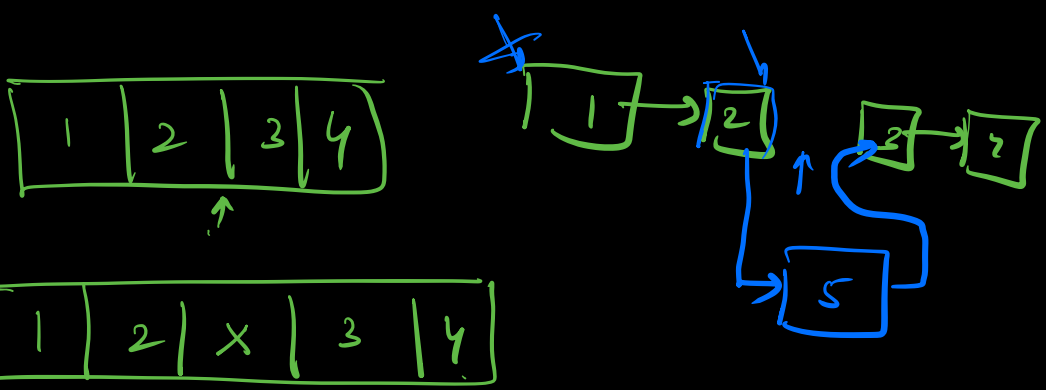




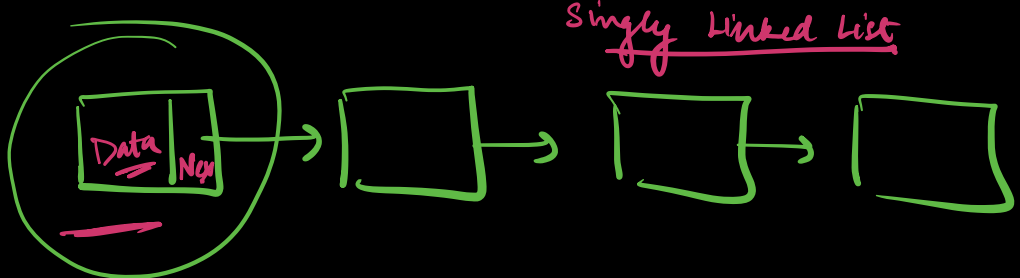
O(N)
Time

Insert an element in front





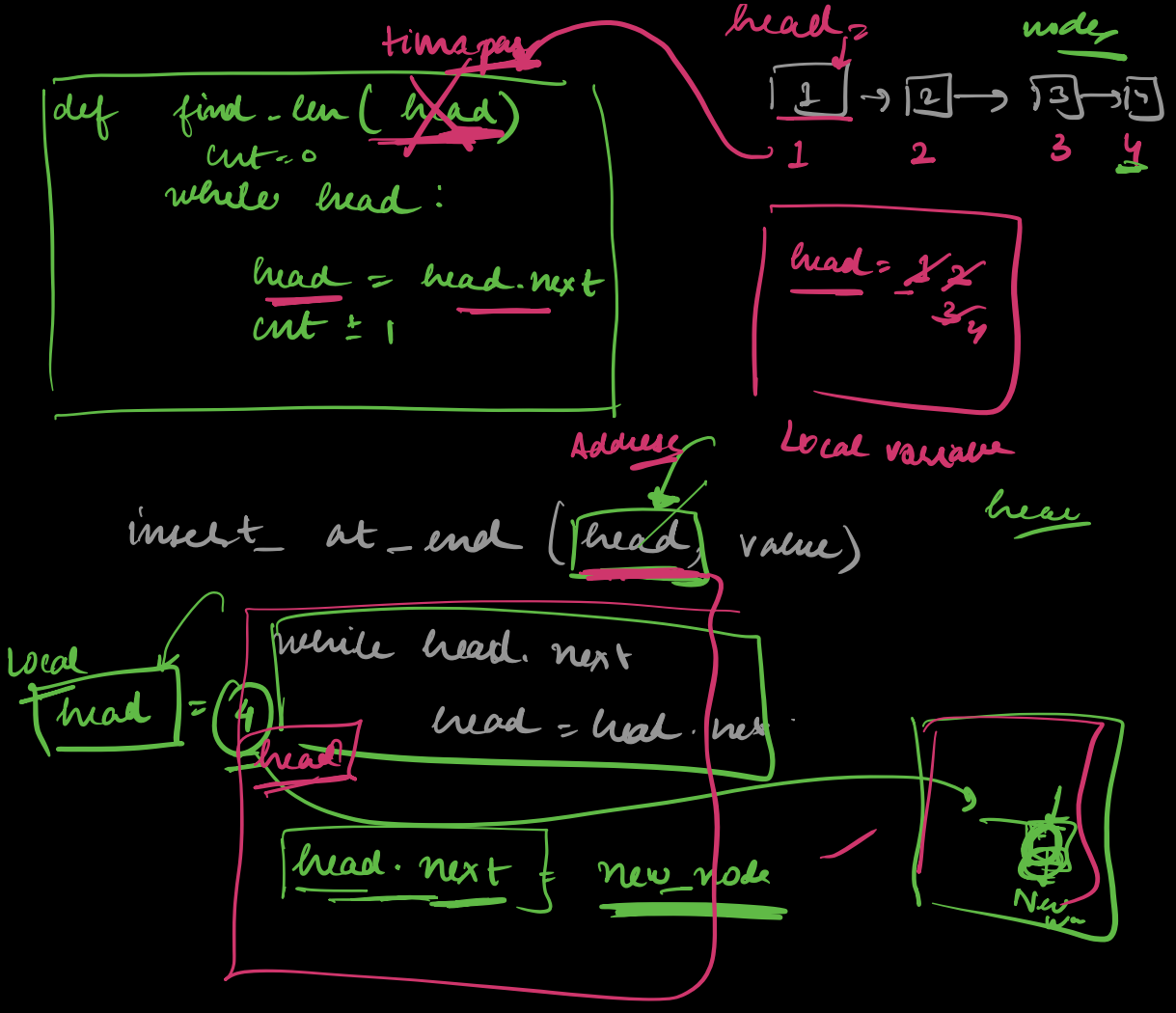
Singly Linked List

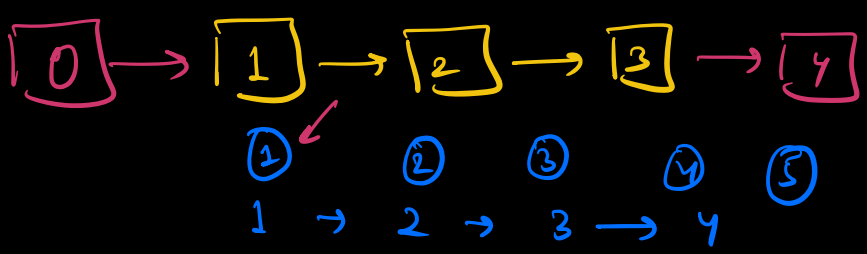


Structure of Node of a LL

```
class Node:
    def __init__(self, data):
        self.data = data;
        self.next = None
```

→ Address of next

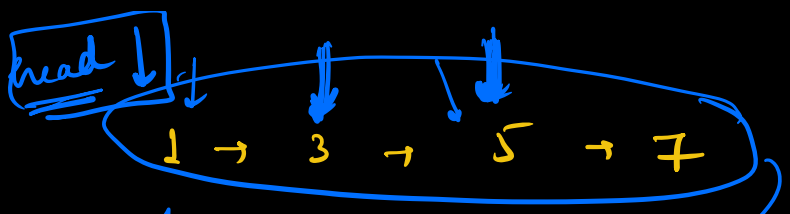




Insert a node at K^{th} position

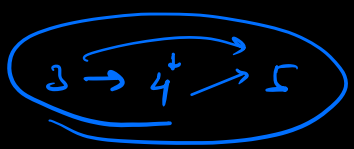
head, value, K

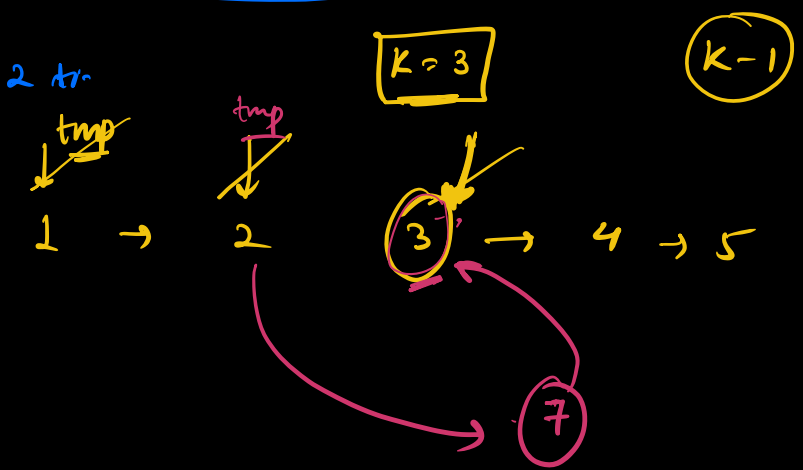
↳ return the new head

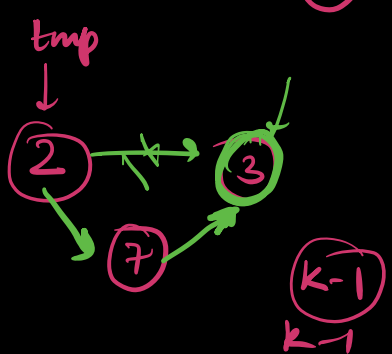


new LL 1 → 3 → 4 → 5 → 7

insert 4 at 3rd position







(1) Move head ptr $\underline{\quad\quad}$ times.

$k-2$

$k-2$

$k-2$

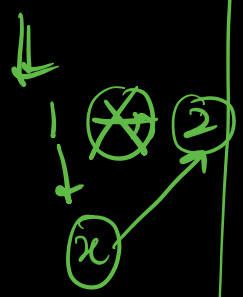
(2)

new_node.next = tmp.next

or tmp.next = new_node

```
def insert at k ( value, head, k):
    tmp = head
    for i in range(k-2):
        tmp = tmp.next

    new_node = Node(value)
    new_node.next = tmp.next
    tmp.next = new_node
```



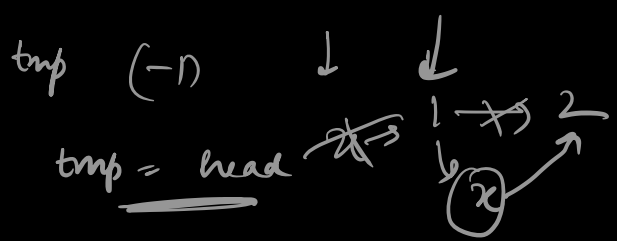
k=2

k=1

(1) k^{th} position

(2) Last position (in the end)

(3) Insert at beginning → k=1

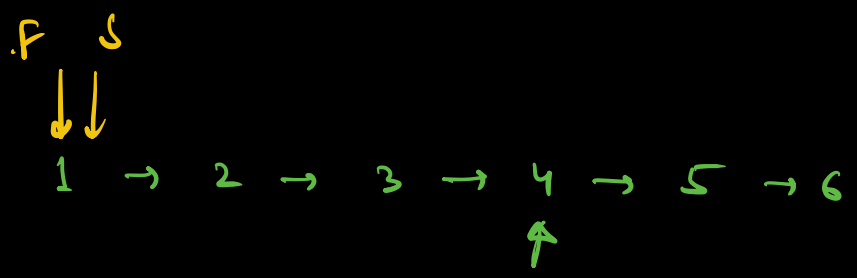


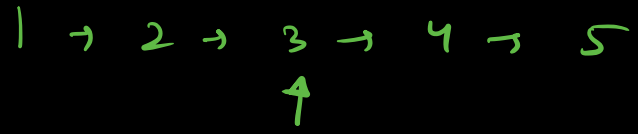
Edge case

```
if k==1:
    insert at begin
```

if head = None
new_node
return new_node

```
new_node.next = head
return new_node
```

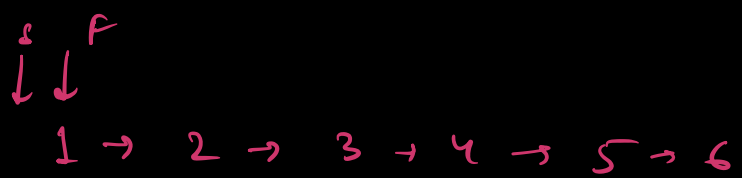




① Get len

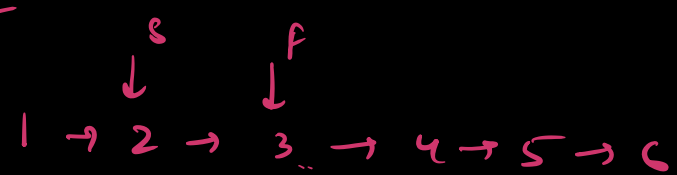
② Go to middle Print it.

Fast and slow ptrs.
Hare and Tortoise

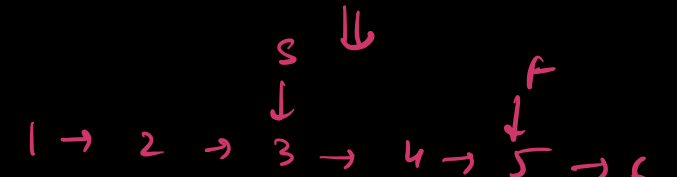


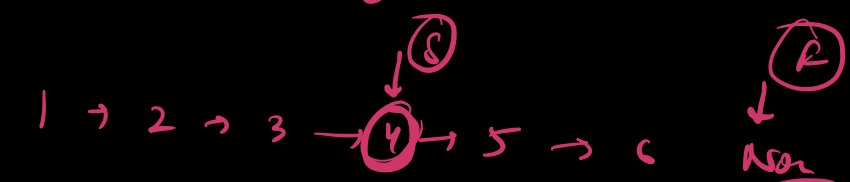
F = F.next.next

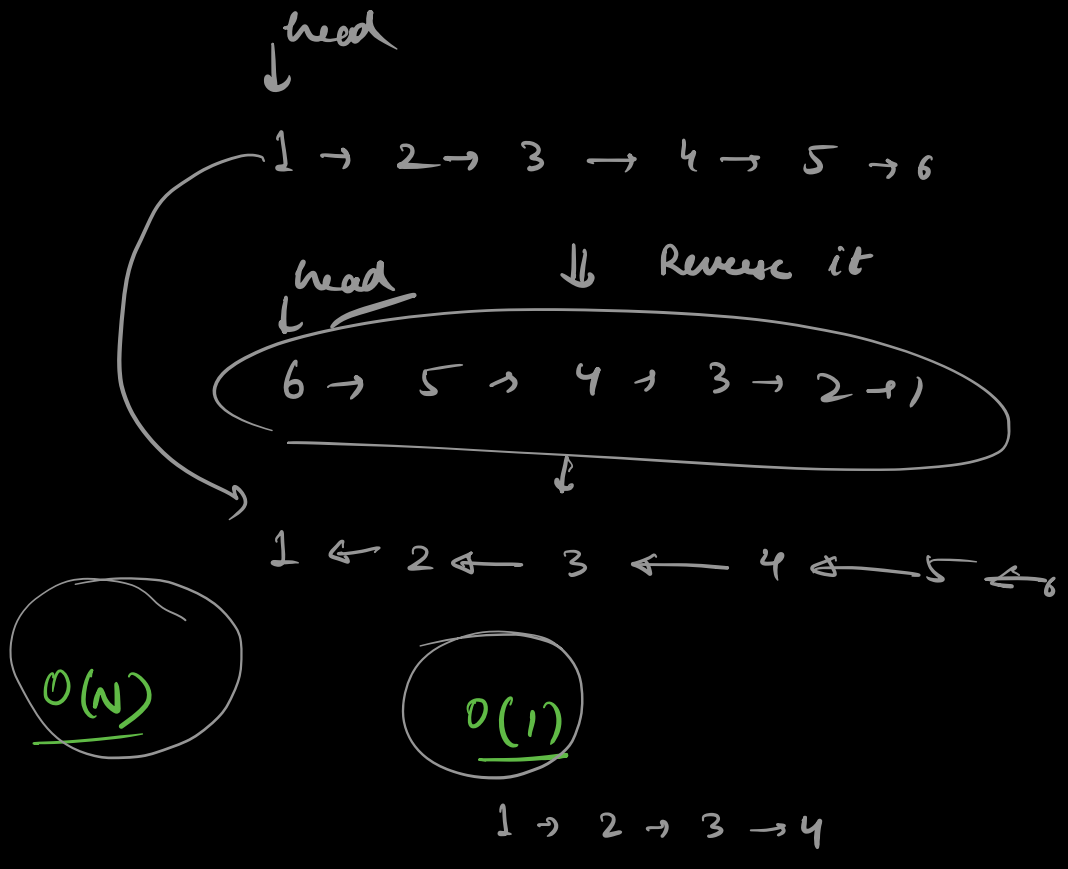
S = S.next



middle

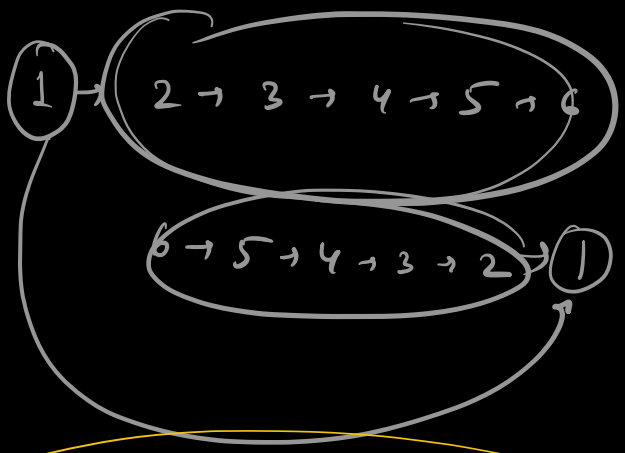






Finding length → Recursion

```
def size(head):
    if head == None:
        return 0
    return 1 + size(head.next)
```



O(1)
space