Linked Lists

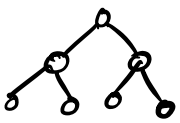
May 13, 2022

AGENDA:

- What are Linked Lists?
- Why Linked Lists?
- Creating a Linked List
- Basic operations
  - Traversal
  - Insertion
  - Deletion

Arrays.

* Contiguous.
* Homogenous.
* Sequential / Linear



TC of accessing an element (at i^{th} element.) → O(1)

TC of insertion / deletion of element in array. → O(N)

Linked List

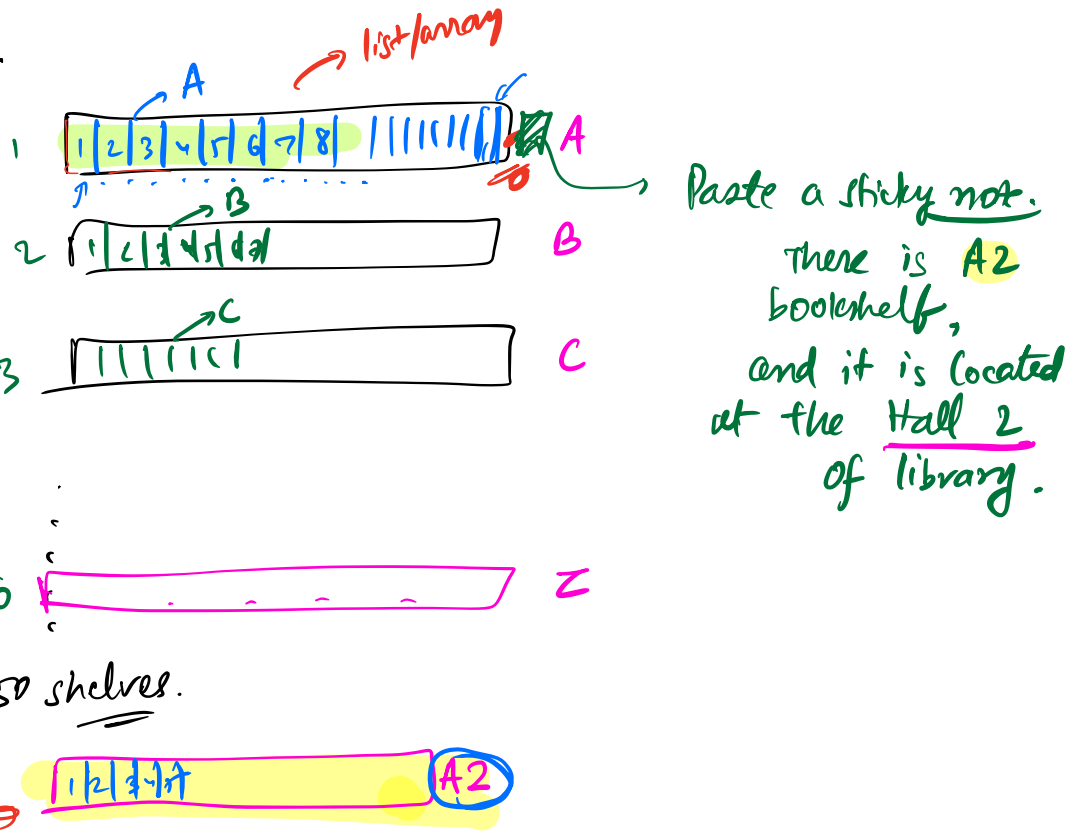
* Linear / sequential (Same as arrays)

* Not contigous.

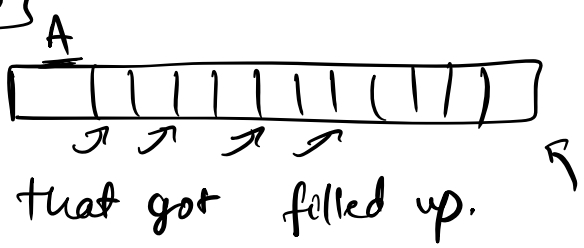
(**) TC of accessing an element → O(N)

TC of insertion / deletion → O(1).

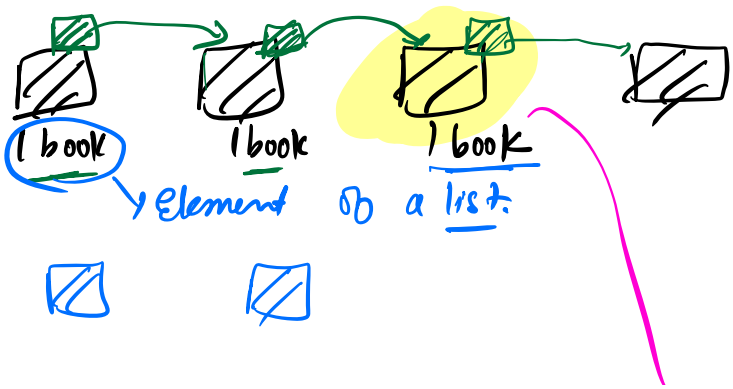
Library.



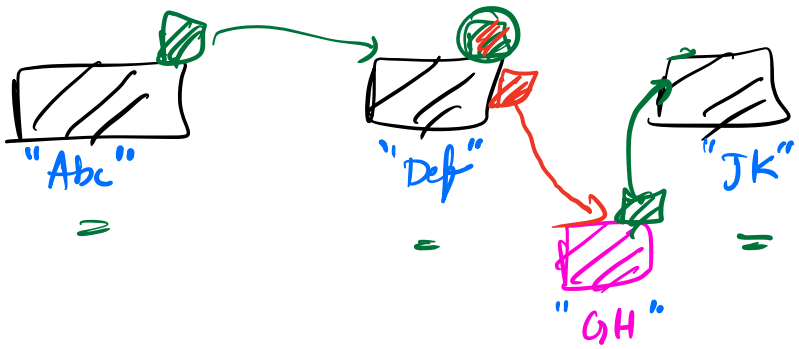
Linked list



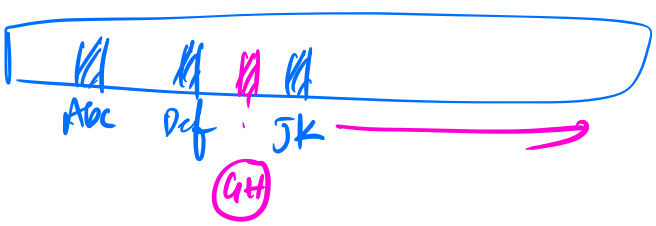
O(N) → shift elements in an array.



① Book
Sticky note
↓ direction
Address.



Very min effort. O(1) time.



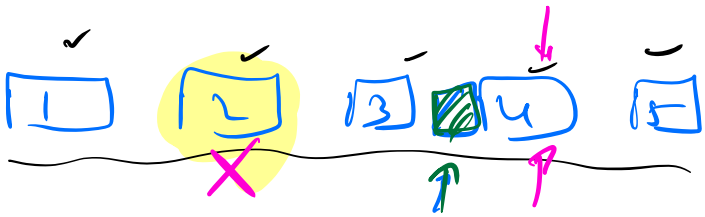
More effort → O(N)

Linked list doesnot mean linking different list.

means linking elements in form of a list.

Why Linked List ?

Search engine

"iphone" → 

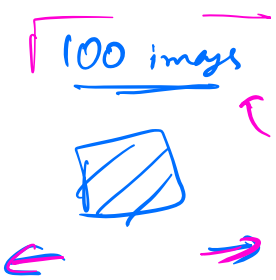
"Sears" 

Insertion / Deletion needs to be efficient

Random Access with O(1) time is not needed.

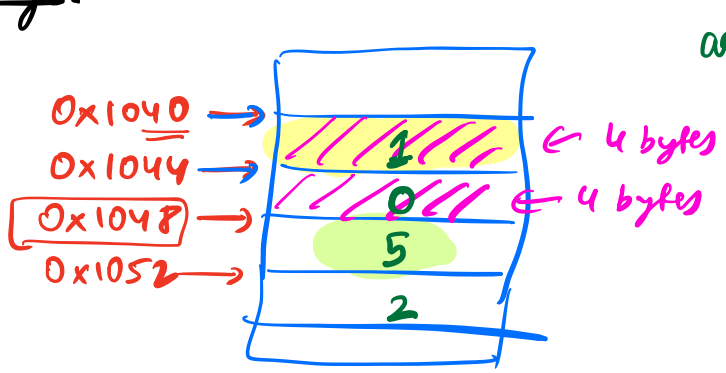
Access. O(N) is justified.

** Image Viewer



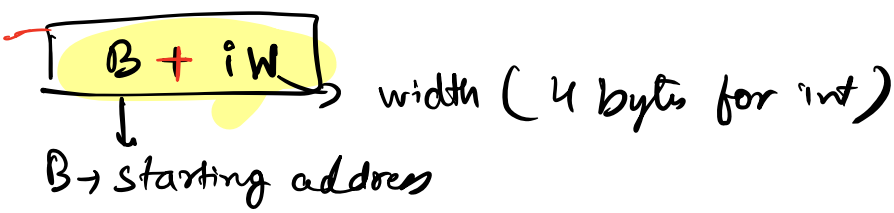
What happens inside the memory ?

Arrays.



ar[2] = ?

ar[0] will be stored at 0x1040
ar[2] " " 0x1048



B + W → 1st element
B + 2W → 2nd element
B + 3W → 3rd element.

Given the address of memory,
TC to get the value stored in that memory ?

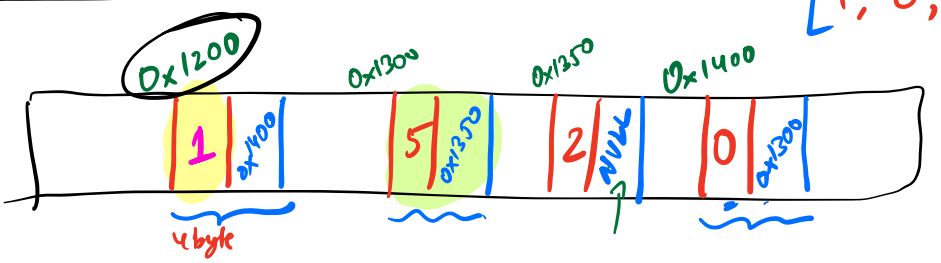
↳ O(1) in case of RAM.

RAM → Random Access Memory

You can randomly access any memory location in constant time.

Linked Lists.

[1, 0, 5, 2]



→ 0x1200 ?

head

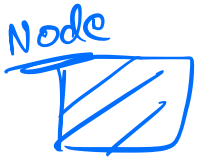
arr = [1, 0, 5, 2]

↓

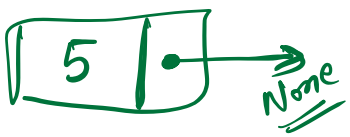
variable itself takes care of starting address.

Node

(Val
Address of next node.)




```
class Node :

    def __init__(self, val):
        self.data = val
        self.next = None
```

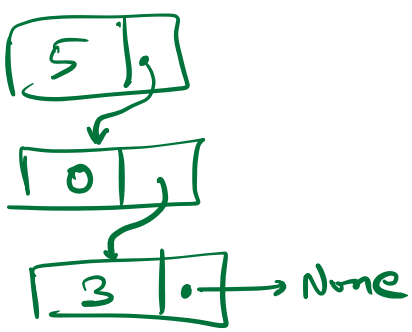


Create 3 Nodes and link all of it.

[5, 0, 3]

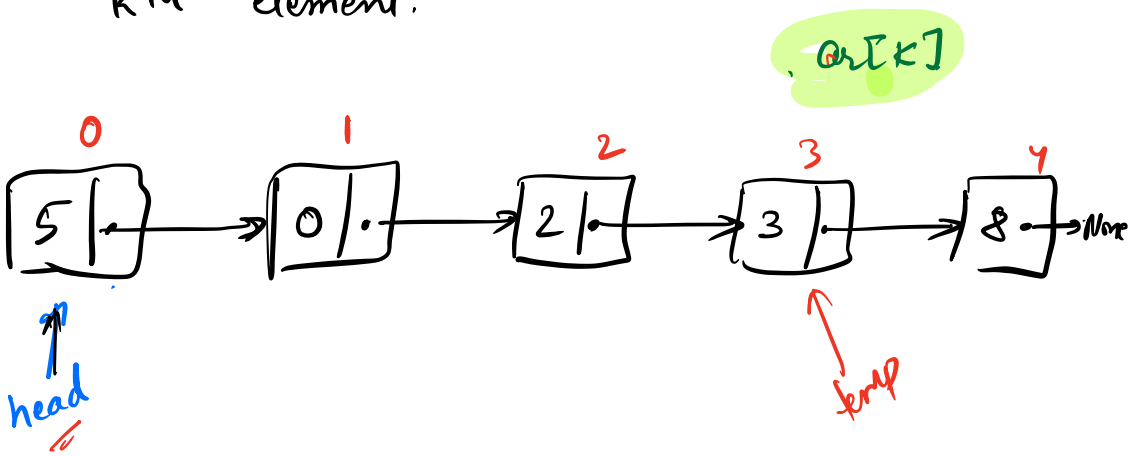
```
head = Node(5)
second_node = Node(0)
third_node = Node(3)
```

```
head.next = second_node
second_node.next = third_node
head = second
```



* You will be given the 'head' node.

Q. Given a LL, (given the head), find the Kth element.



temp = temp.next

```
def findKthNode(head, k):

    temp = head
    for i in range(k):
        temp = temp.next

    return temp.data
```

None.next

None has no attribute named 'next'

Base cases.

K = 10

head is of a linked list.

```
def findKthNode(head, k):
    temp = head
    for i in range(k):
        if temp.next == None:
            return None
        temp = temp.next
    if temp == None:
        return None
    return temp.data
```

TC: O(N)
SC: O(1)

```
print(head)
print(type(head))
```

class Node

Break till 10:22

→ ( ) { } [ ] ←

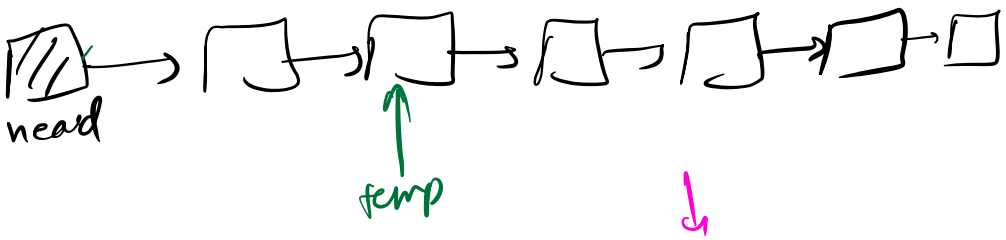
inbuilt data structures

Sorted

O(N) space.

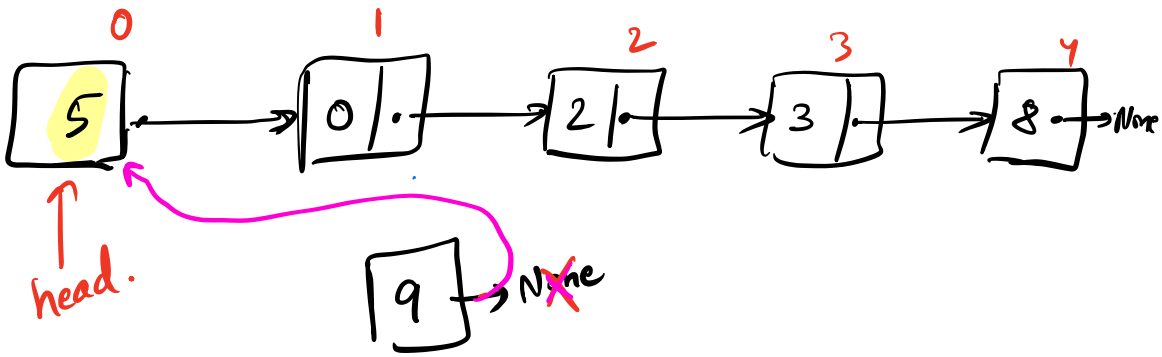
Array → O(N) space.

Find the length of linked list.



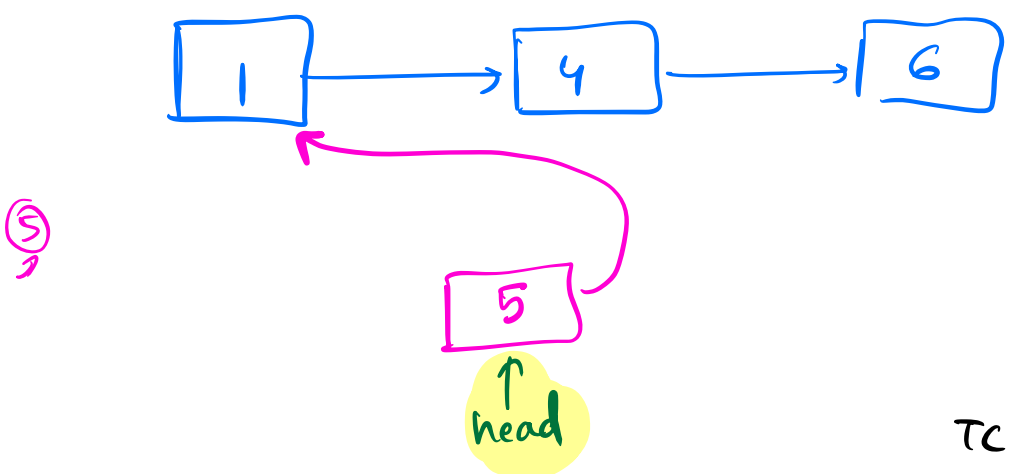
O(N)

Insertion in a linked list.



* Insertion at the beginning. a value.

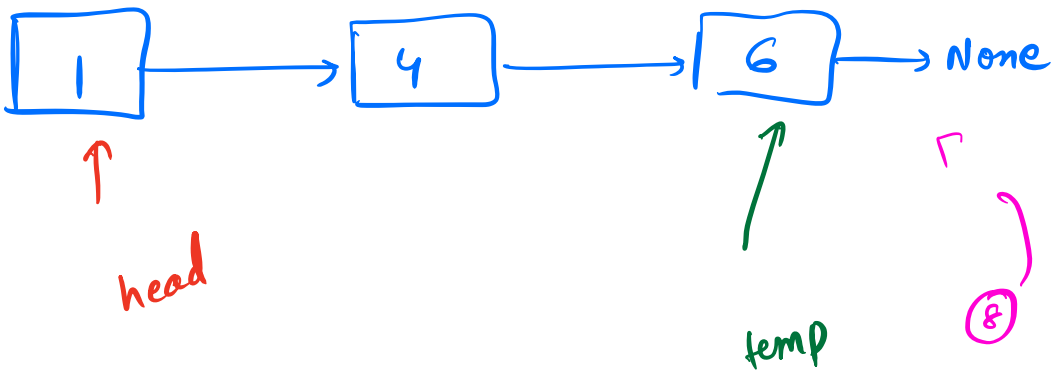
```
def insert_at_beginning( head, val) :

    temp = Node(val)
    temp. next = head
     head = temp
```



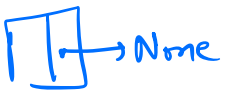
TC : O(1)

temp. next = head

** Insert at end.




```
def insert_at_end(head, val):
```




```
    x = Node(8)
    temp = head
    if head is None:
        head = x  return
    while (temp.next is not None):
        temp = temp.next
```

temp.next = x

x.next = None ✗ ✗ ← Not needed.

Edge-case

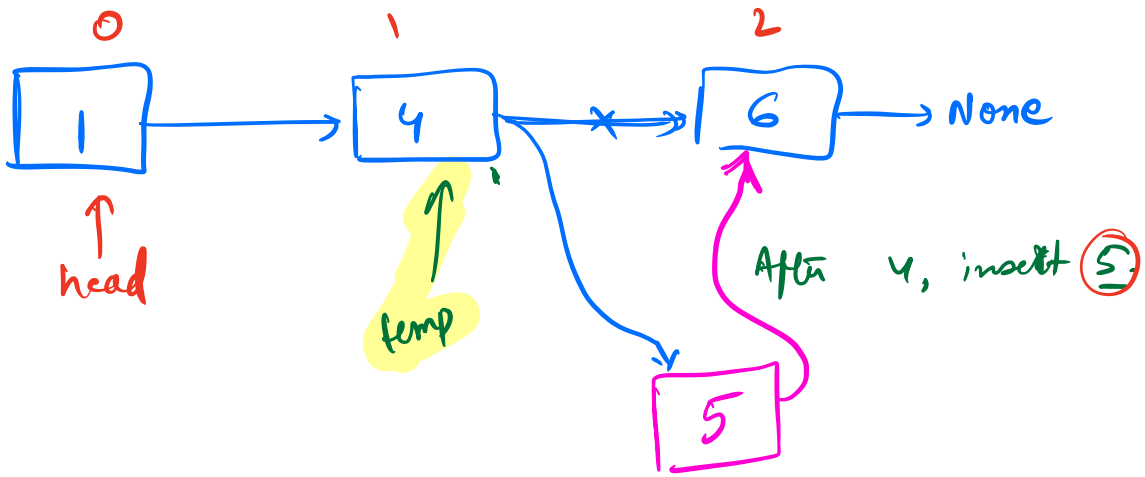
What if the list is empty?

head = None.

TC: O(N)

tail. node → O(1) time

** Insertion at the middle.




```
x = Node(5)
x.next = temp.next
temp.next = x
```

```
def insert_at_middle(val, k, head):
```

$0 \le K < N$

// Insert after 'K'. (0-indexed)

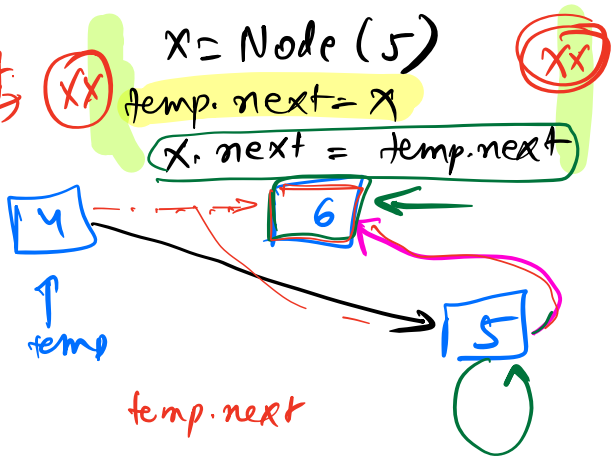
```
→ temp = findKthNode(head, K)
  if temp == None:
      insert_at_start(head, val)
```

```
x = Node(5)
x.next = temp.next
temp.next = x
```

or

Wrong

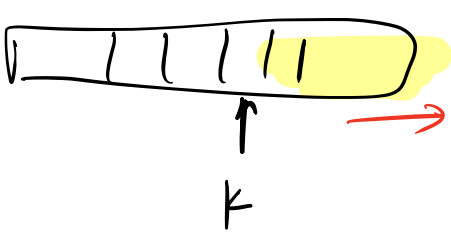
```
x = Node(5)
temp.next = x
x.next = temp.next
```

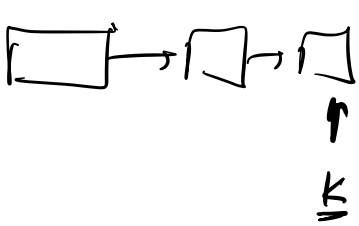


TC : O(N)

SC : O(1).

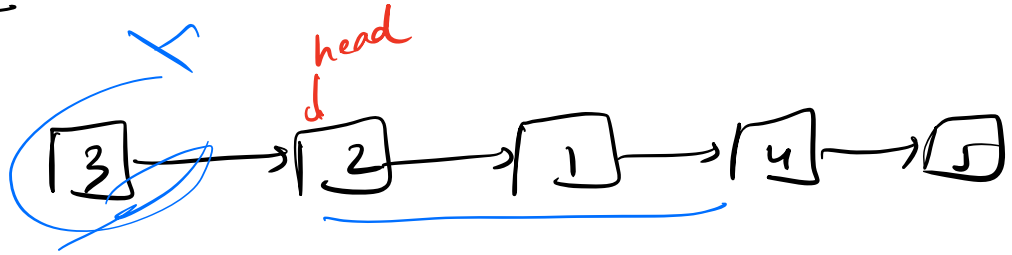
O(1)





$=$ O(1)

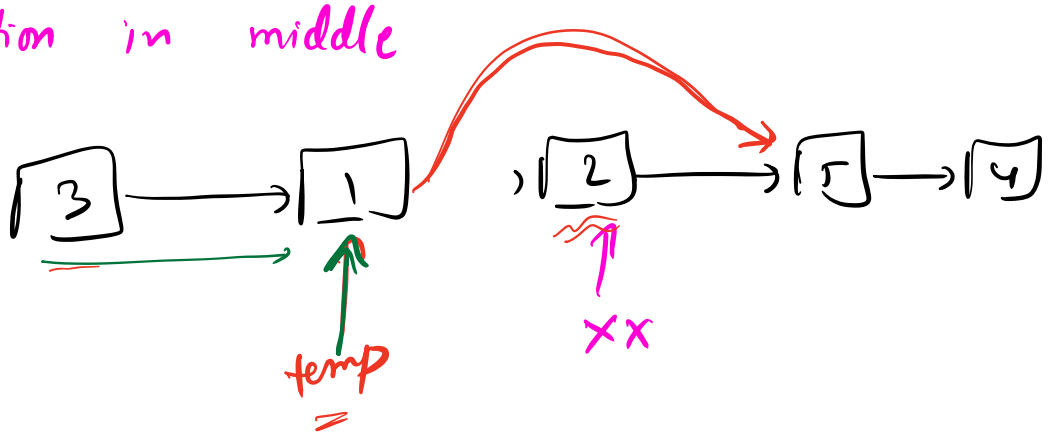
Deletion



① Delete the start node.

head = head.next

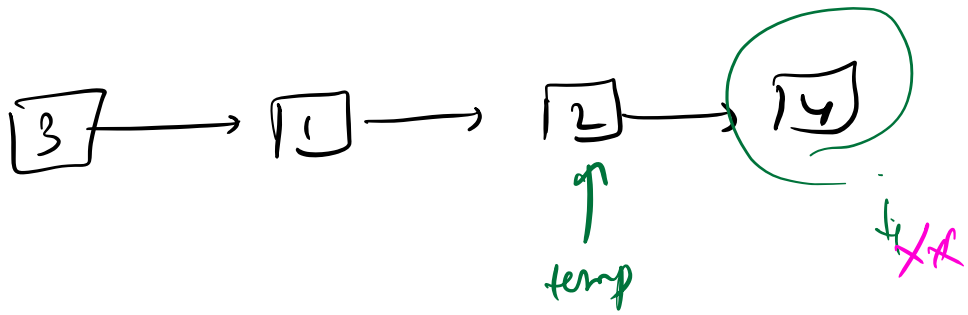
② Deletion in middle



temp.next = temp.next.next

Handle edge case.

③ Delete the last element.



temp.next = None.

```
while ( temp.next.next is not None)
```

Reach to the second last node.

HW Try implementing all of these on your own.

Handle all the edge cases.