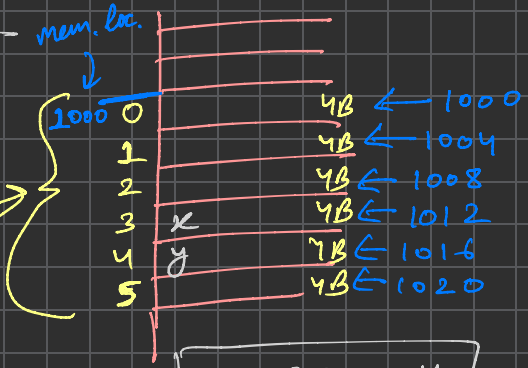
LINKED LISTS

Arrays



int arr[6]

arr[3] → x → O(1)

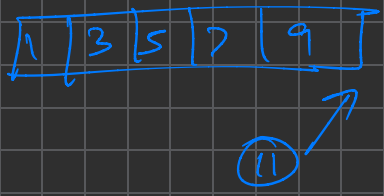
arr[4] → y → O(1)

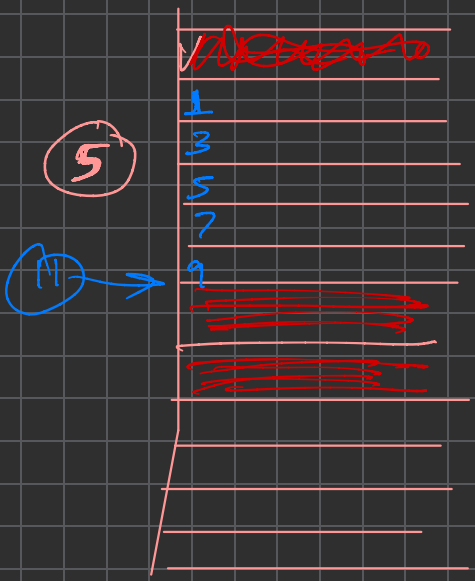
arr[4] = Addr of 0th element + 4 * size of arr. elements

= 1000 + 4*4 = 1016

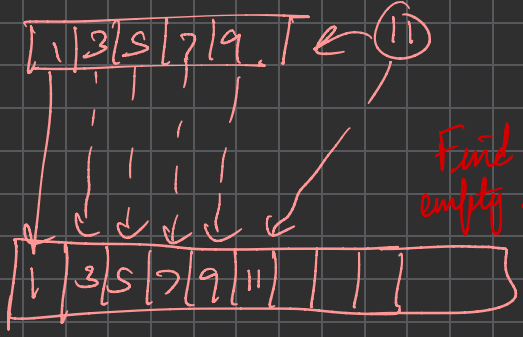
Accessing any element → O(1) time

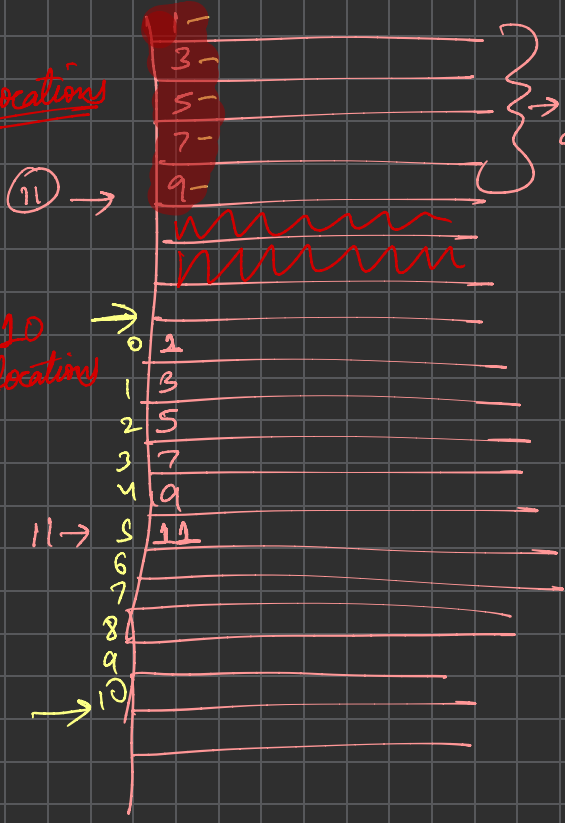
Array = Contiguous memory





ARRAYLISTS



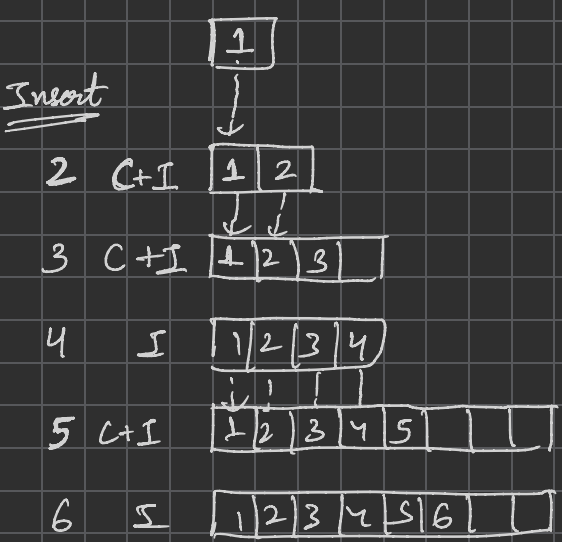


15 16

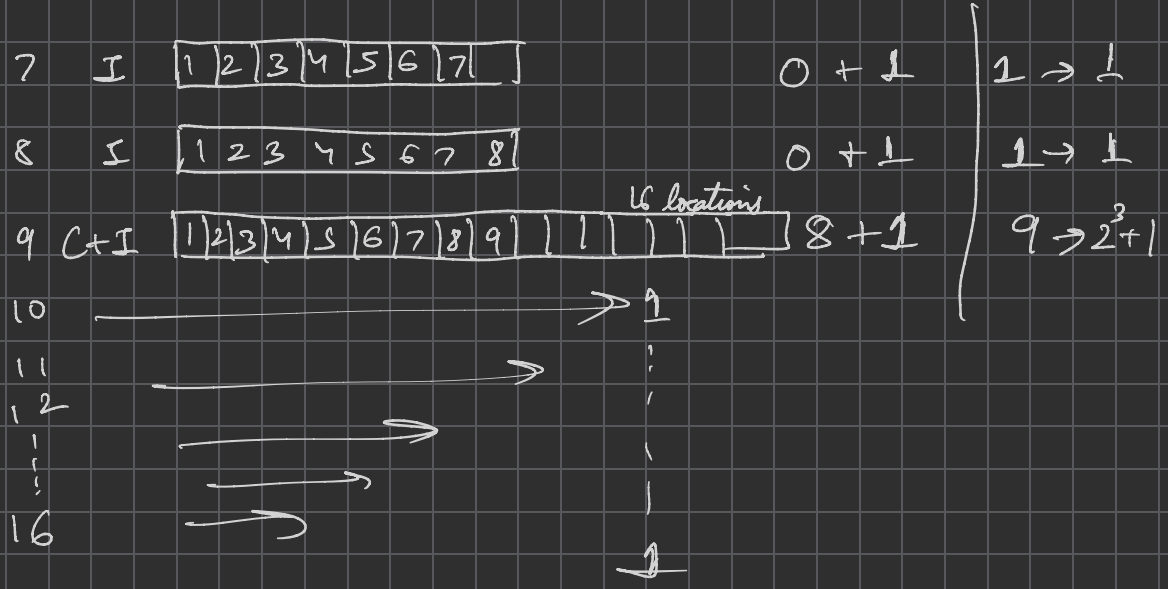
Accessing elements → O(1)

Insertion complexity ?

AMORTIZED T.C:



Copy	Insert	Total
1	+ 1	$2 \to 2^0+1$
2	+ 1	$3 \to 2^1+1$
0	+ 1	$1 \to 1$
4	+ 1	$5 \to 2^2+1$
0	+ 1	$1 \to 1$



Inserting 8 elements $= 2^0 + 2^1 + 2^2 + 2^3 + 8$

Inserting N elements $= \underbrace{2^0 + 2^1 + 2^2 + \cdots + 2^{\log_2 N}}_{G.P.} + N$

Sum of G.P. $= \frac{a(r^N - 1)}{r - 1}$

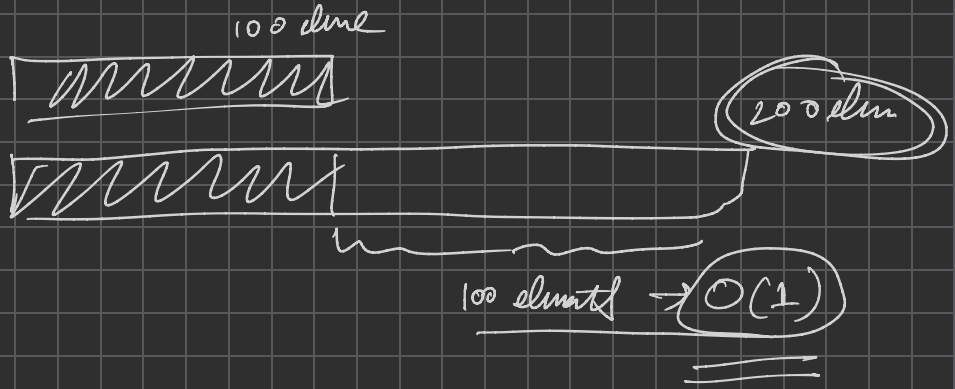
Inserting N elements $= \frac{2^0(2^{\log_2 N + 1} - 1)}{2 - 1} + N$

$= 2^{\log_2 N} * 2 - 1 + N$

$= 2N + N - 1 = 3N - 1 \approx O(N)$

Inserting N elements $= O(N)$

Inserting 1 element $= O(1)$ → Amortized T.C.



C++ → Vector

Java → Array list

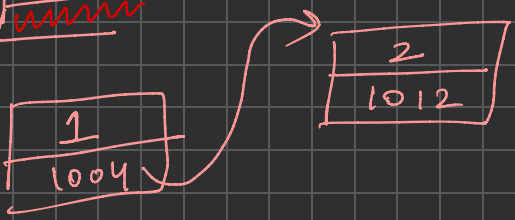
Python → Lists

C# → arraylist / list

JS → array

Total available spaces → 3

Array list, I can't insert elements

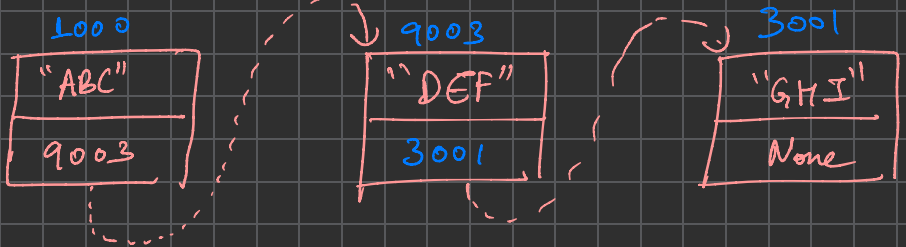


LINKED LISTS

Node

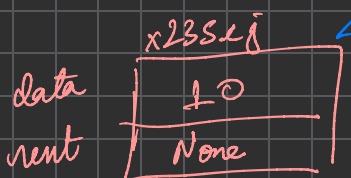
data : int, str, obj, data structure

addr : Memory address / reference to the next node

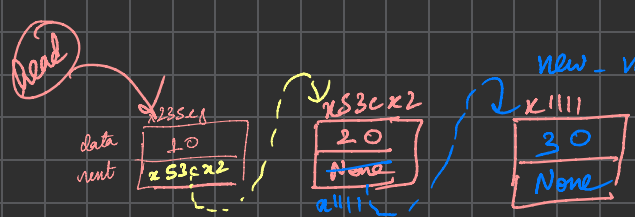



```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```
head = Node(10)
print(head.data) # 10
print(head.next) # None
```




```
new_node = Node(20)
head.next = new_node
new_node = Node(30)
print(head) # x235cj
print(new_node) # x1111
```



I don't want (x53cx2).next = x1111
new_node

(head.next).next = new_node

head.next.next = new_node

or

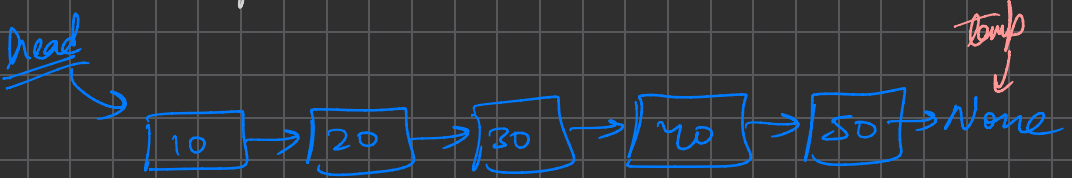
temp = head.next

print(temp) // 0x53c2

temp.next = new_node

Q: Given a linked list, find its length.

head of the L.L.



O(n) = 5

```
def get_len(head):
    length = 0
    temp = head
    while temp != None:
        length += 1
        temp = temp.next
    return length
```

T.C. → O(N)

S.C. → O(1)

length

0

1

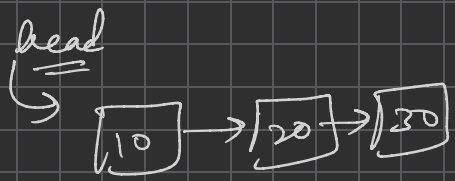
2

3

4

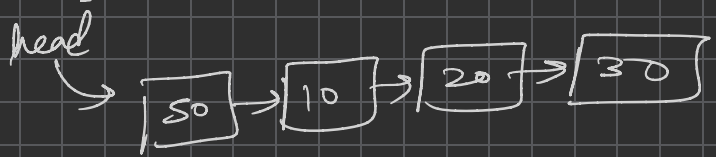
5

Q2 Given a L.L., add data in front of L.L.



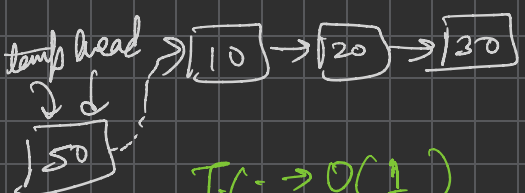
Add 50

O/P



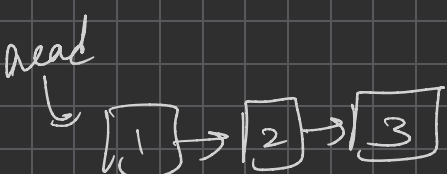
---

temp = Node (50)
temp.next = head
head = temp
return head.



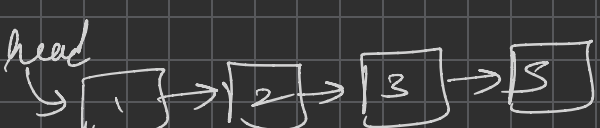
T.C. → O(1)
S.C. → O(1)

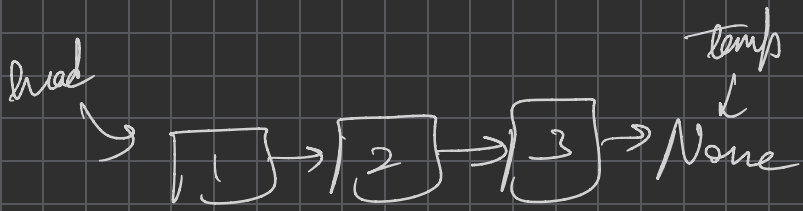
Q3 Given a L.L., add data at the end of the L.L.



Add 5

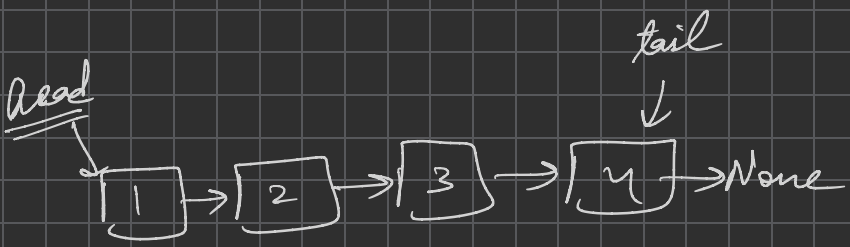
O/P

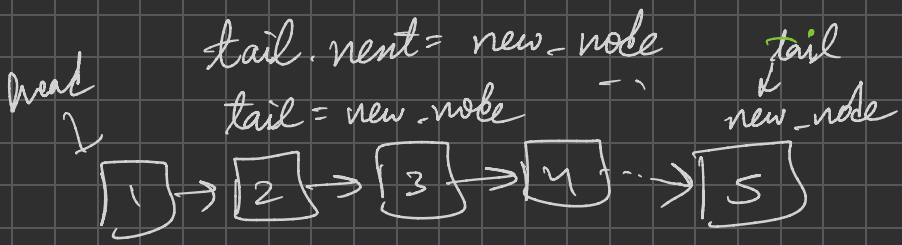





```
def ins_( ) :
    new_node = Node(5)
    temp = head
    while temp.next != None :
        temp = temp.next
    temp.next = new_node
```

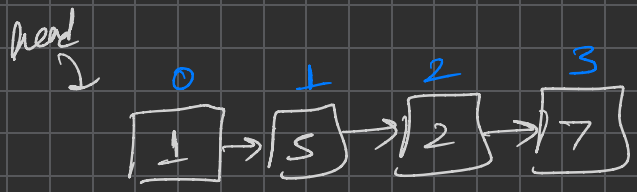
T.C → O(N)
S.C → O(1)



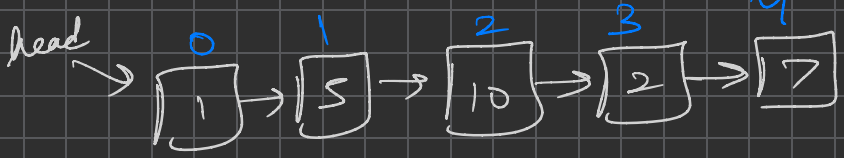


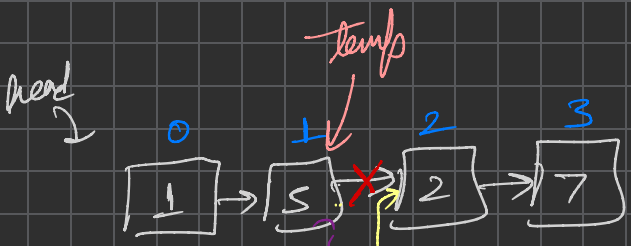
T.C → O(1)
S.C → O(1)

Q4 Given a L.L., add a node at K^{th} position.



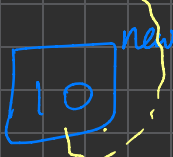
Add 10, at k=2





new_node.next = temp.next

temp.next = new_node




```
i = 0
temp = head
while (i < k-1) {
    i += 1
    temp = temp.next
new_node = Node(10)
new_node.next = temp.next
```

T.C. → O(N)

S.C. → O(1)

temp.next = new_node

H.W:
Q1 Delete a node from beg.
Q2 Delete a node from end.
Q3 Delete the k^{th} node.