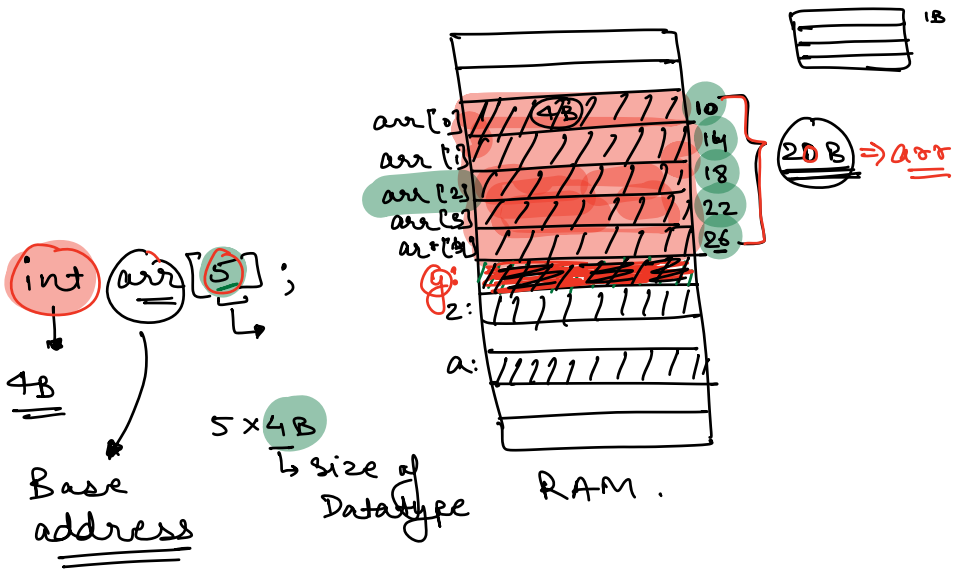
Array

↳ Easy access.

O(1) → random access.

arr[i] ⇒ O(1)

⇒ Continous memory.



int arr[5].

⇒ We can't extend the size of Array.

Dynamic Arrays / Vector / ArrayList :-

List<Int> list;

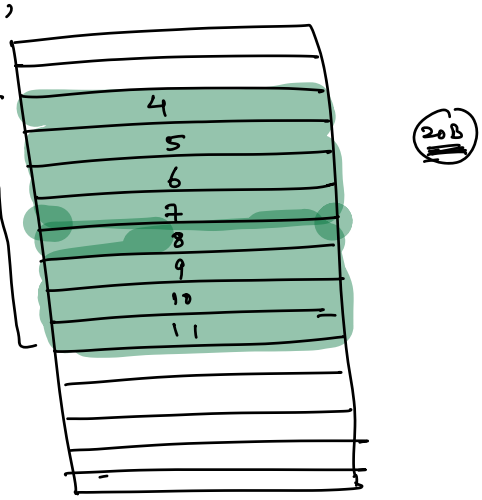
list.add(4)

list.add(5)

——(6)

——(7)

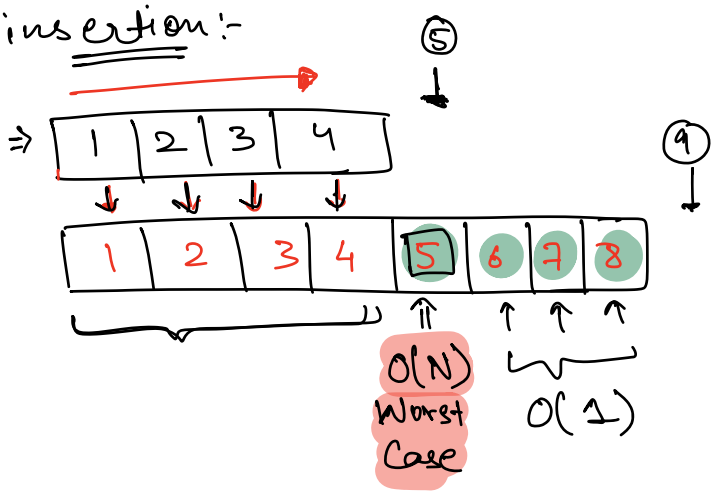
list.add(8);



ArrayList / Vector :-

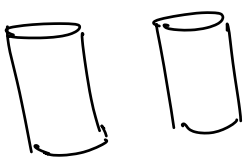
↳ TC of access : O(1)

Quiz TC of insertion :-

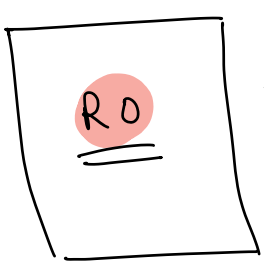


Insertion in Dynamic Array

⇒ O(1) [Amortized]

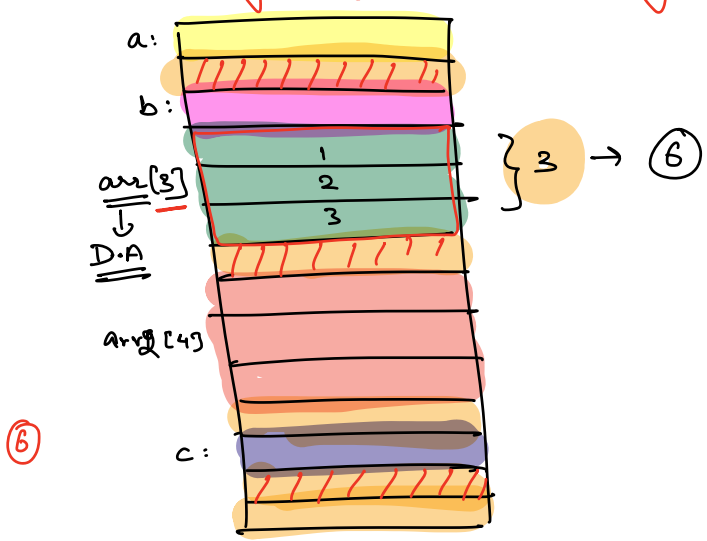


⇒ 150 Rs | week.



⇒ 10000 Rs

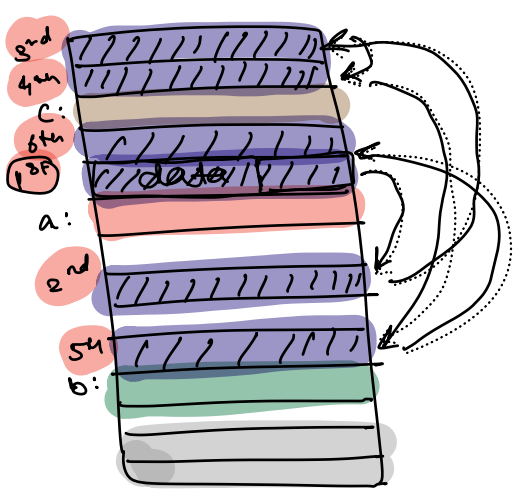
Drawback of Dynamic Array :-



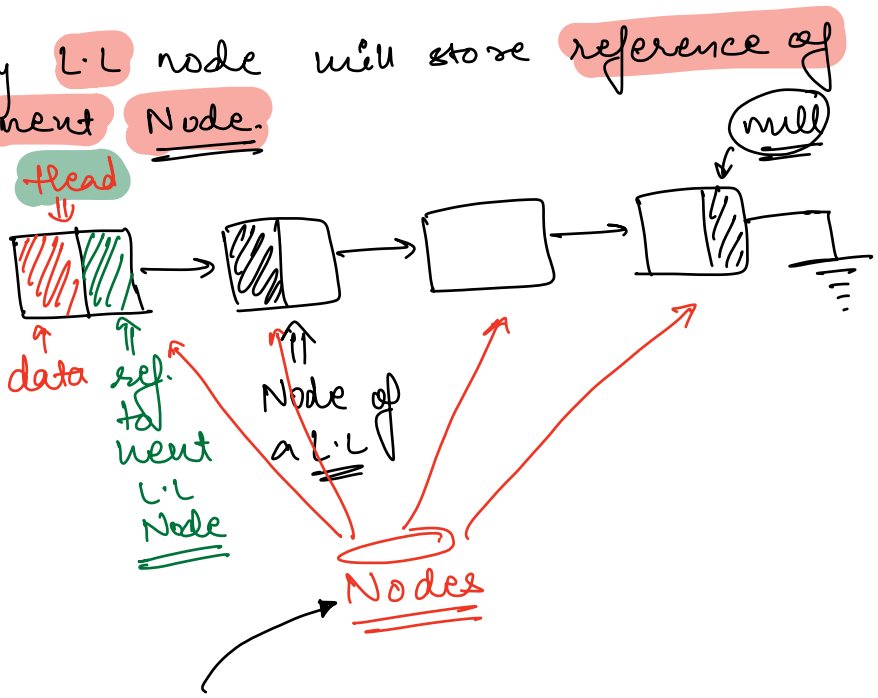
Arr.insert(4) ✗

⇒ Arrays | A.L | Vectors ⇒ O(1)
↓
Random access

⇒ If we don't want O(1) random access time :-
↳ Linked List

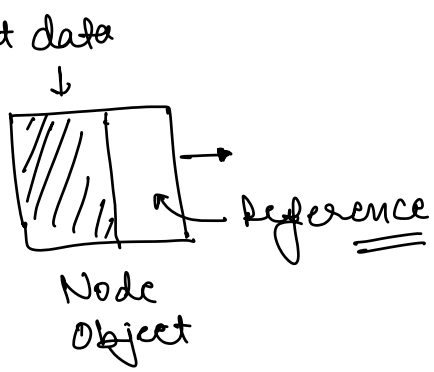


Every L.L node will store reference of its next Node.



Any OOPs lang.

```
Class Node {
    int data;
    Node next; ⇐ Reference of the next L.L Node
}
```

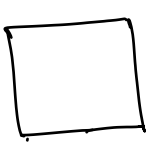


Constructor.

```
Node node = new Node();
```

node.data ⇒ 0 / garbage

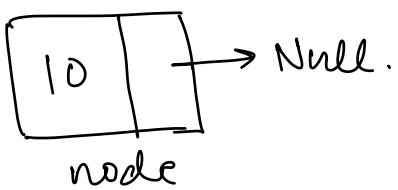
node.next ⇒ Null / garbage

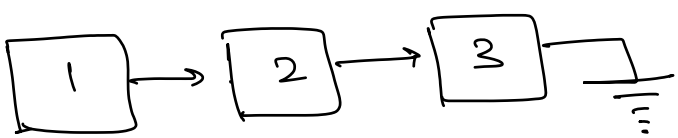


```
class Node {
    int data;
    Node next;
    Node(int a) {
        this.data = a;
        this.next = Null;
    }
}
```

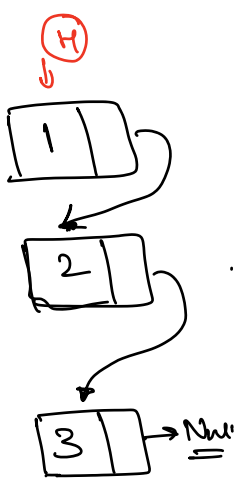
Constructor

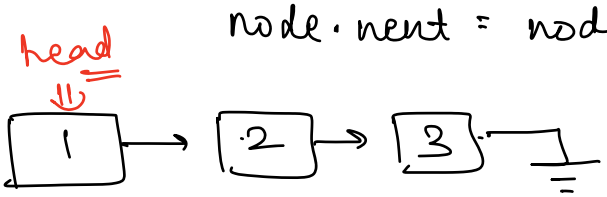
```
Node node = new Node(10);
```





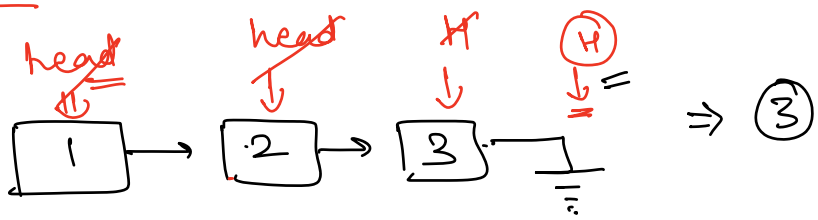

```
Node head = new Node(1);
Node node = new Node(2);
head.next = node;
Node node2 = new Node(3);
node.next = node2;
```





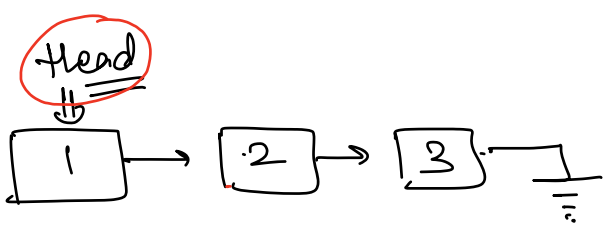
Q Given a L.L, find its length.

Head



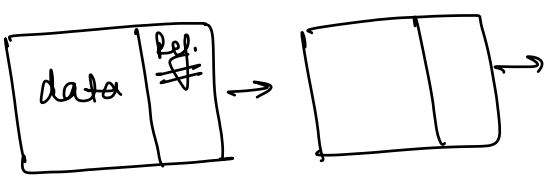
l = ✗✗3

```
int getLength(Node head){
    int len = 0;
    Node temp = head;
    while( temp != Null){
        len++
        temp = temp.next;
    }
    return len;
}
```



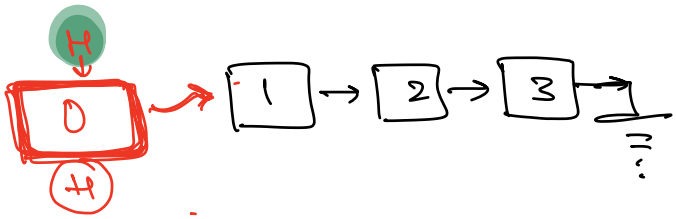
l = ✗✗3

We should not update head of the L.L.

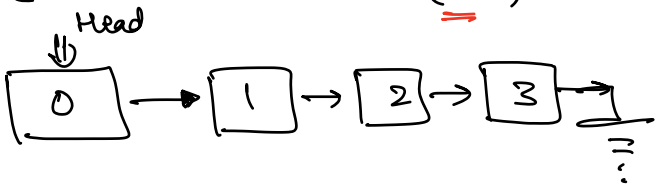


Insertion in LL

1) front

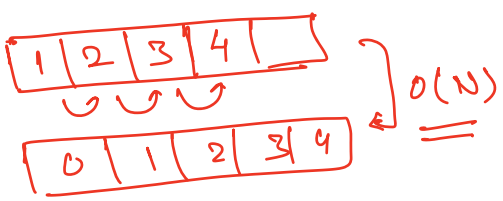


- Node node = new Node(0); ← O(1)

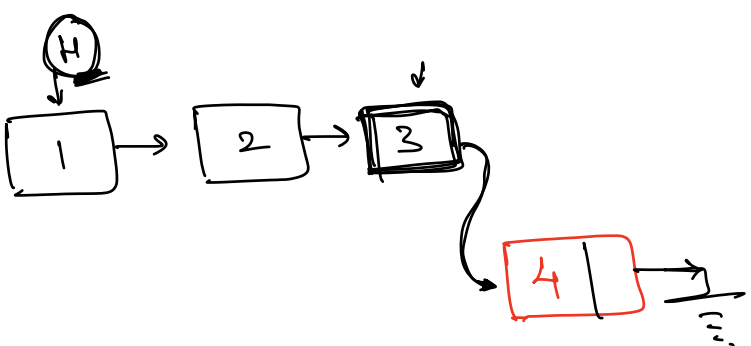


- node.next = head;] → O(1)
- head = node;

TC: O(1) VS [Array O(N)]



2) End

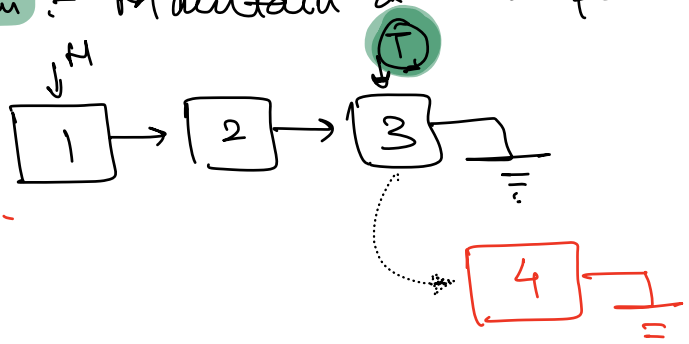


- Node node = new Node(4);
- temp = head;
  while (temp.next != null) {
  temp = temp.next;
  }
  } O(N)
- temp.next = node;

TC: O(N) vs [O(N) in Array]

NullPointer Exception

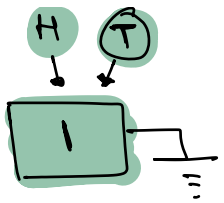
Optimization :- Maintain a tail pointer of L.L

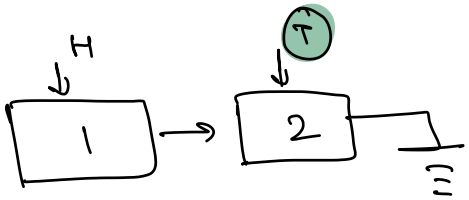


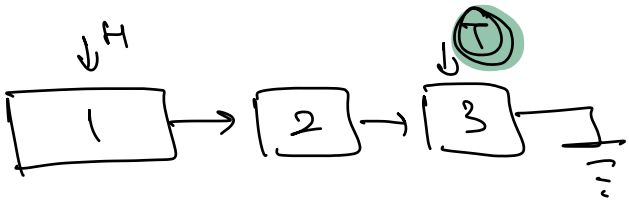
- Node node = new Node(4);
- Tail.next = node;
- Tail = node;

TC: O(1)

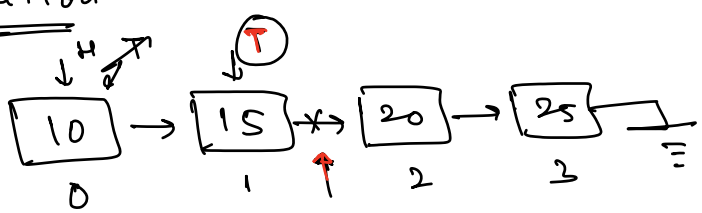
#

1) 

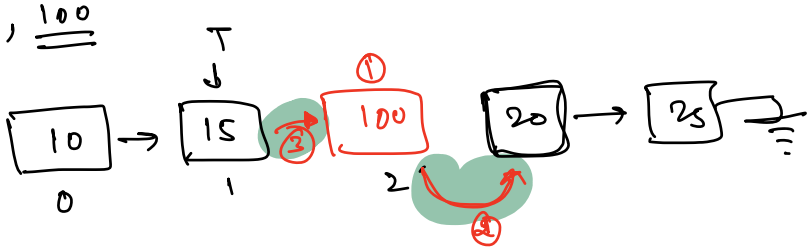
2) 

3) 

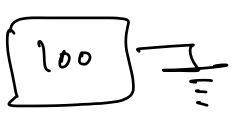
3) K^{th} position



K = 2, 100



- Node node = new Node(100);



- temp = head;
  count = 0;
  while (count < K-1) {
    count++;
    temp = temp.next;
  }

K=0
K=N
K=N+1

How

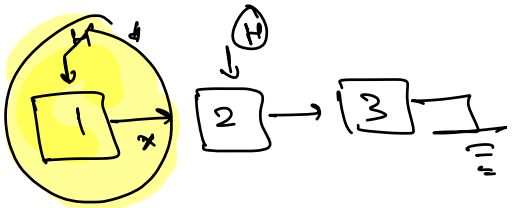
- node.next = temp.next;
- temp.next = node;

#

1. Write code on paper / google doc. (Draw L.L)
2. Dry-run.
3. Edge cases.
   - → size of L.L = 0 / Head = null.
   - → size of L.L = 1 | 2 | 3.
   - → Problem specific Edge cases.

Delete

1. front

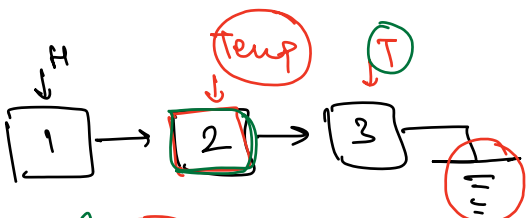


Garbage collector

- Head = Head.next;

TC: O(1) vs [O(N) Array]

2. End:



while (temp.next.next == null) {--}

temp.next = Null;

tail = temp;

TC : O(N)