Array

→ arr[i] ⇒ TC: O(1)

→ O(1) random access.

int arr[5];

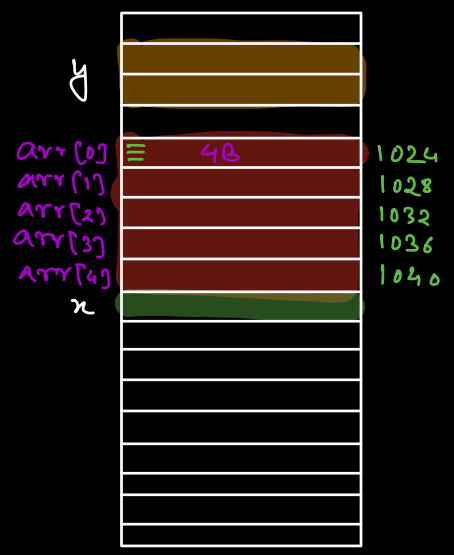
↓

4B

reference|name

(1024)

→ Static Array



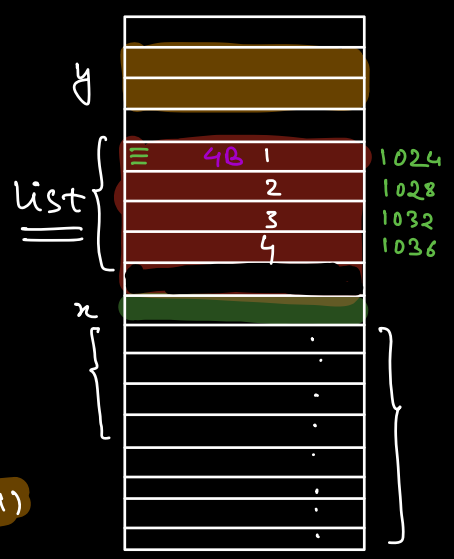
Dynamic Array (ArrayList | Vector)

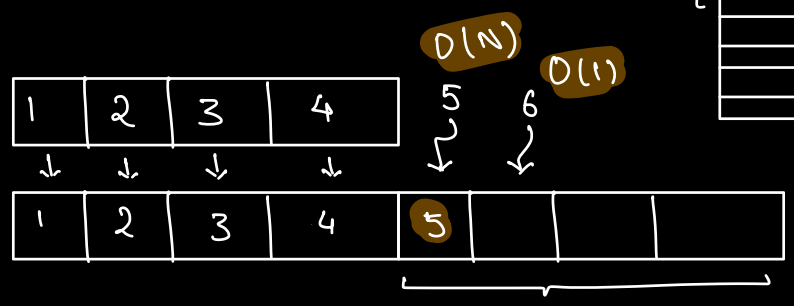
ArrayList ↓ Java

Vector ↓ C++

list<int> l;

l.add(1)
l.add(2)
l.add(3)
l.add(4)
l.add(5) ←



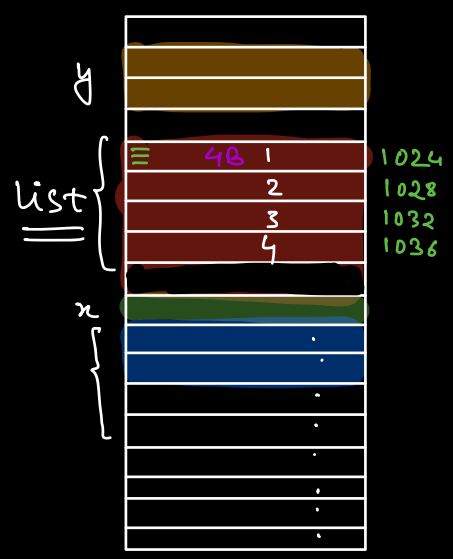


Worst Case TC of insertion in Dynamic Array

$\Rightarrow$ $O(N)$

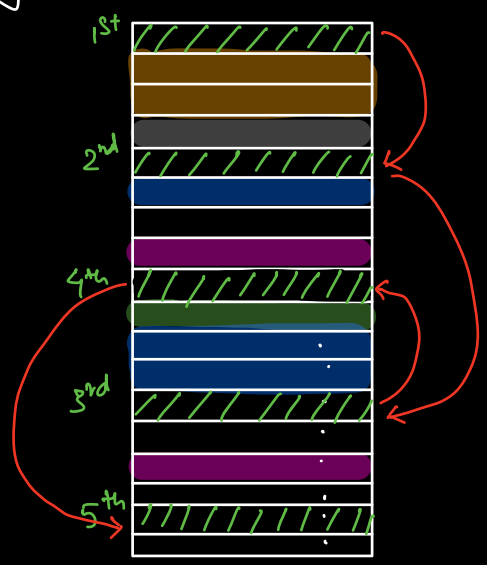
Insertion in Dynamic Array $\Rightarrow$ $O(1)$ Amortized TC

$\Rightarrow$ $O(1)$ random access.

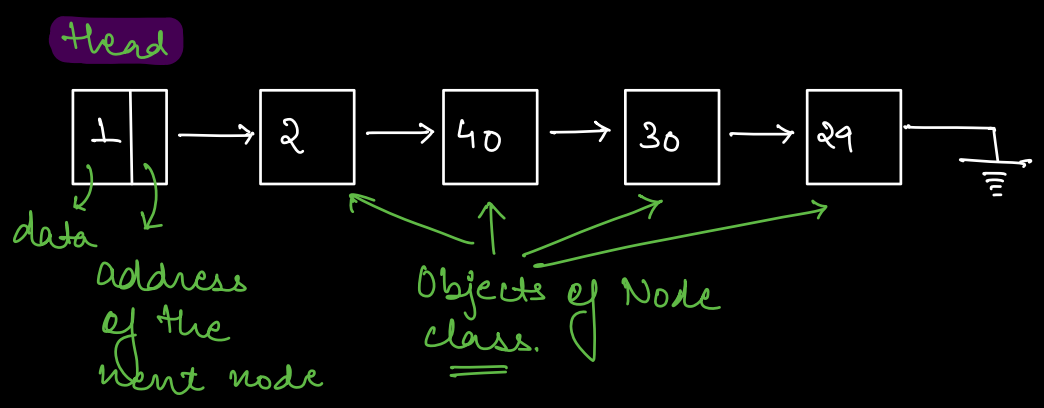


$\Rightarrow$ unoptimised way of using memory.

$\Rightarrow$ If we don't want $O(1)$ random access?



Linked list

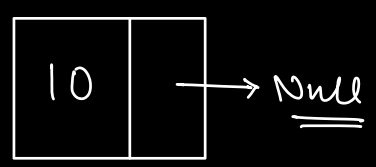


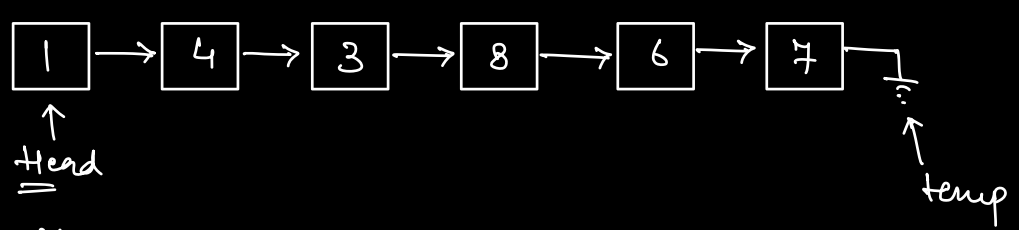
Q. Given a L.L, find its length.

Head node is given.

```
* Class Node {
      int data;
      Node next;
      Node (int x) {
          this.data = x
          this.next = null;
      }
}
```

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





d = 0x23486

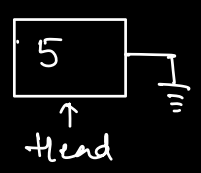
```
int length (Node head) {
    int len = 0
    Node temp = head;
    while ( temp != Null) {
        len++
        temp = temp.next;
    }
    return len;
}
```

TC: O(N)

→ NEVER UPDATE HEAD OF THE L.L

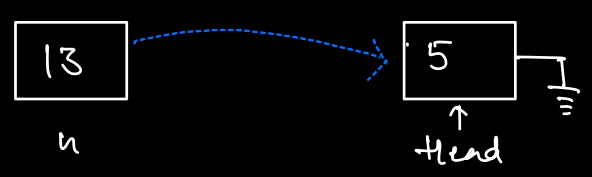
Insert a node in the L.L

```
Node head = new Node(5);
```

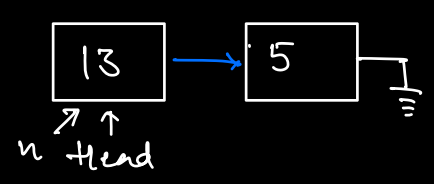


1) Front

- `Node n = new Node (13);`

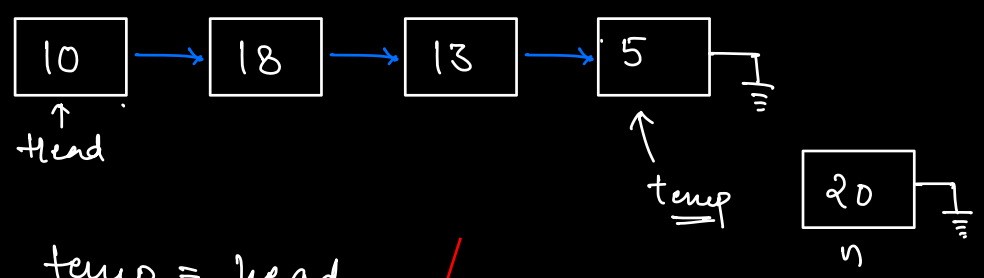


- `n.next = head;`
- `head = n;`



TC: O(1) vs O(N) in Array.

2) End



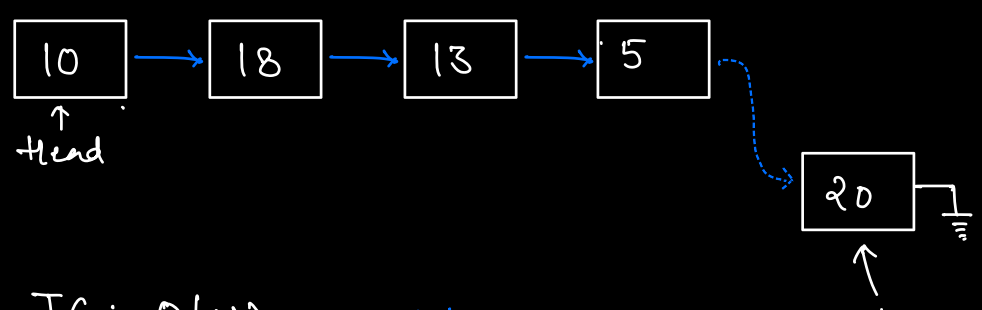

```
temp = head
while( temp.next != Null){
        temp = temp.next;
```

Will not work for head=null

3

→ Node n = new Node(20);

temp.next = n;

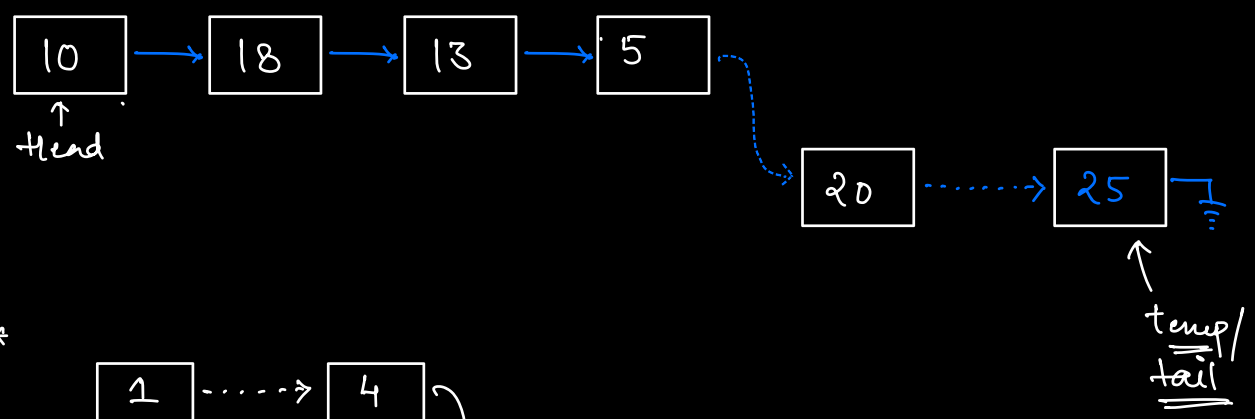


TC: O(N) vs O(N) Worst case Array

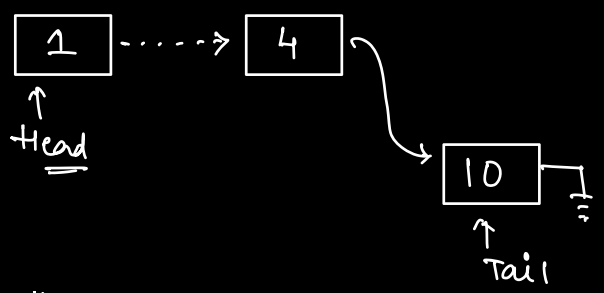
⇒ Optimisation of insertion at end :-

Start maintaining the tail reference.

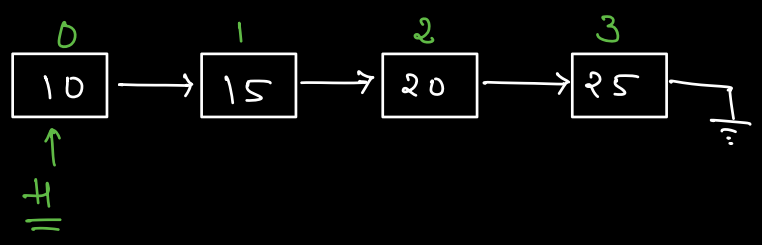
TC: O(1)



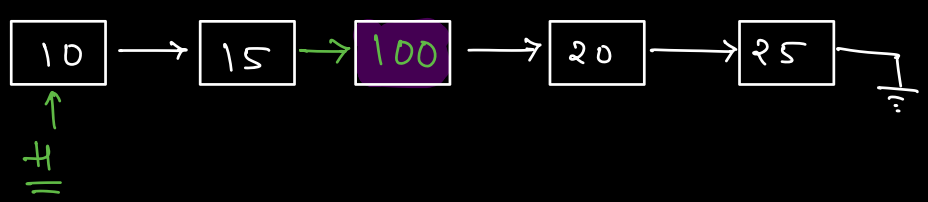
*



3) K^{th} position

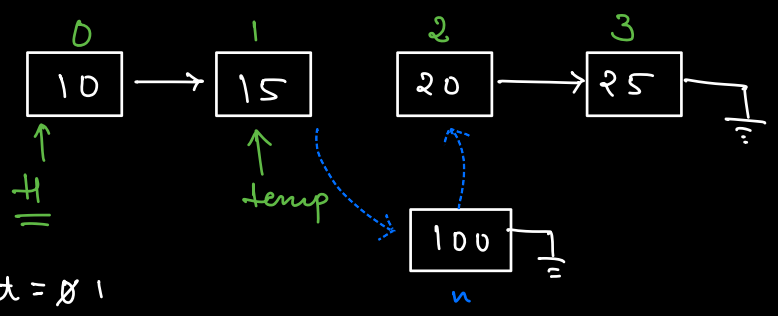


K = 2
100



K = 0

K=2



Count = ~~0~~ 1
< 1

```
* Node n = new Node(100);
  Node temp = head;
  count = 0
  while (count < K-1) {
        temp = temp.next;
        count++;
  }
  n.next = temp.next;
  temp.next = n;
```

Will not work for k = 0

TC: O(N) vs O(N) worst case array

Important pointers

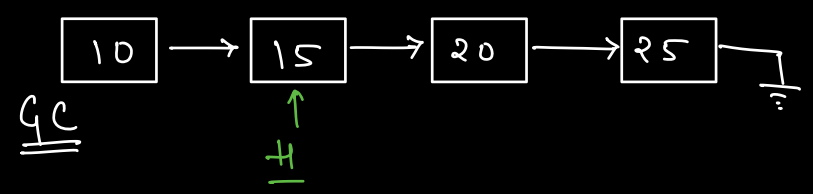
1) Write code on pen-paper / doc
2) Dry run on various TC's.
3) Edge cases
   - → Null L.L / head = null.
   - → size = 1/2/3
   - → Problem specific TC's.

Edge case :- Test input for below values

Todo K = 0 | 1 | > N | . . . -1, minus values also

Delete

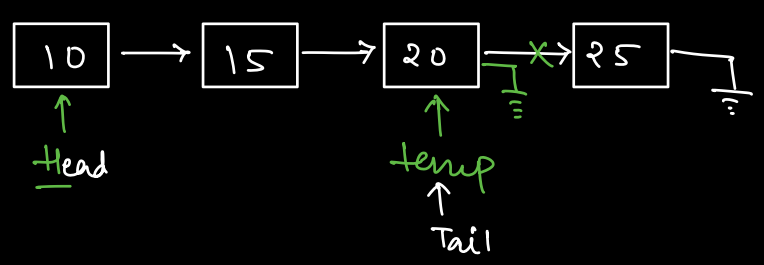
1) Front



head = head.next;

TC: O(1) vs O(N) Array.

2) End

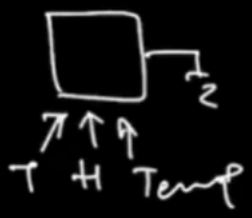


⇒ reference of the second last node.

```
temp.next = null;
tail = temp;
```

TC: O(N)

—— * ——



temp. next next ⇒ NPE

Null.

. . .

a.b.c.d

x

x

x