Arrays :-

↳ faster access.

↳ O(1) random access.

arr[i] → O(1)

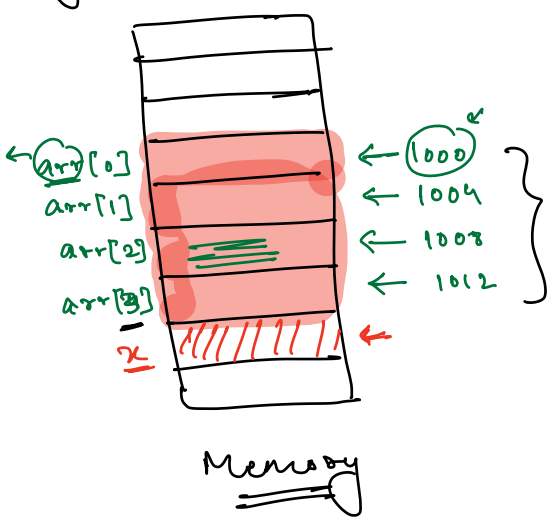
⇒ How array provides O(1) access ?

↳ Contigous memory allocation.

int arr [4] ;

↓

4B.



⇒ Can we append array ?

↳ NO

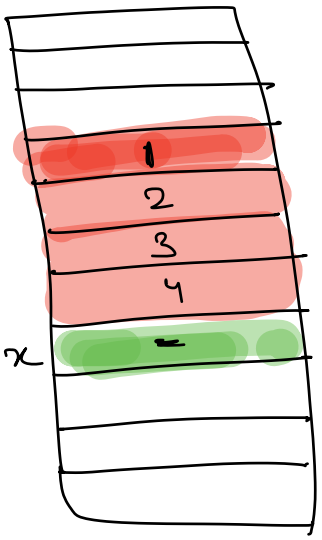
⇒ ArrayList / Vectors / List :-

↳ Dynamic Array.

→ Access time :- O(1)

↓

Continous memory.

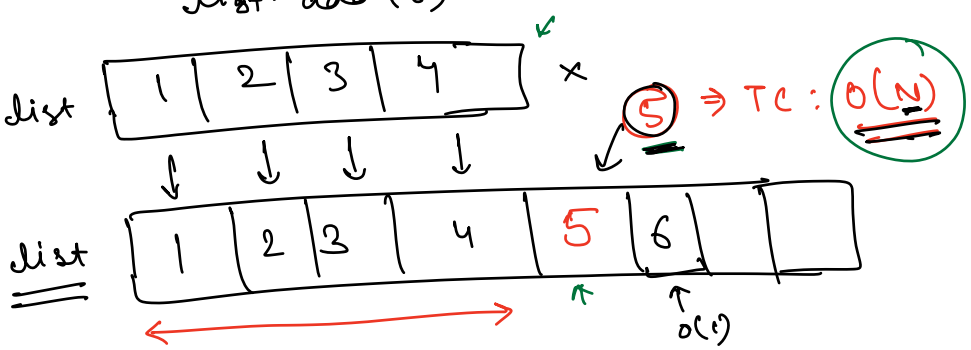


List<Int> list;

list.add(1)
(2)
(3)
(4)
→ O(1)

list.add(5)

list.add(6)



1) Worst case TC of insertion in Dynamic Array

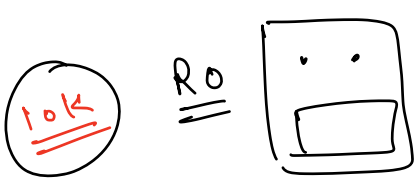
⇒ O(N)

2) Insertion in Dynamic Array : O(1) [Amortized TC]

↑

Avg. TC when we are performing many operations

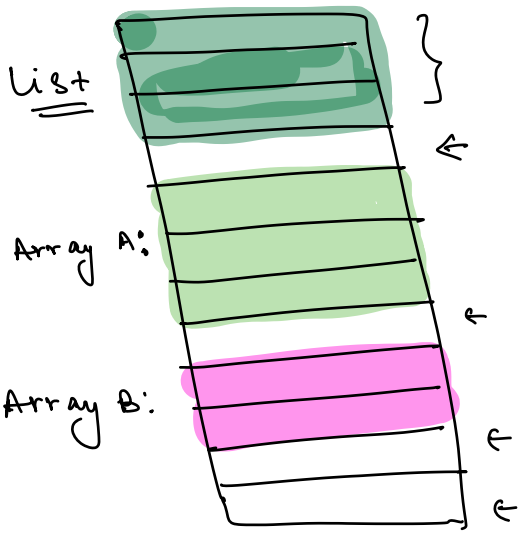
⇒ 



⇒ Dynamic Array

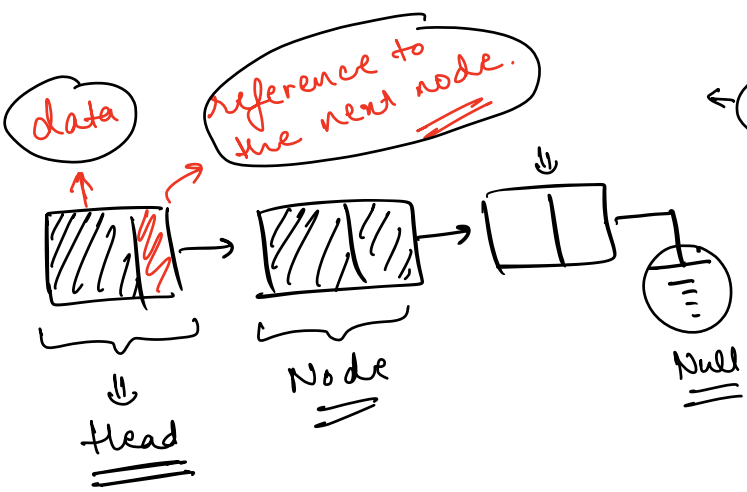
⇓

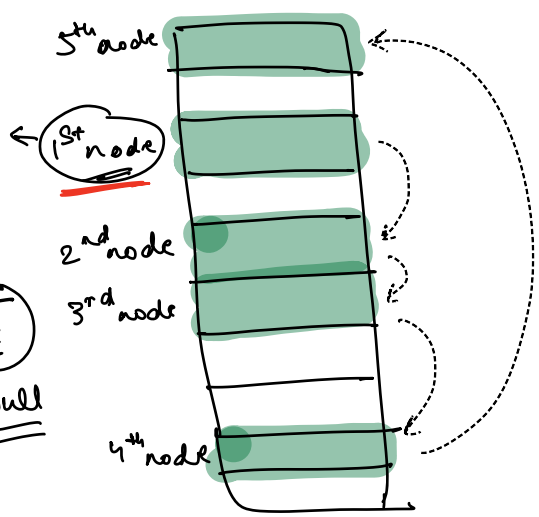
[Contigous Memory]



⇒ If we don't want O(1) access time?

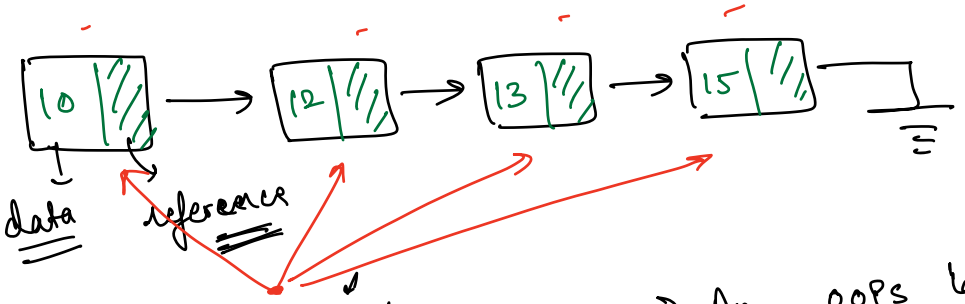
↳ Linked List





↳ Stores the reference of 1st node of L.L.

Implementation of L.L :-




```
Class Node {
    int data;
    Node next;
}
```

⇒ Any OOPs lang.

⇒

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

Constructor

```
Node newNode = new Node();
```

newNode.data ⇒ 0/garbage.

newNode.next → null/garbage

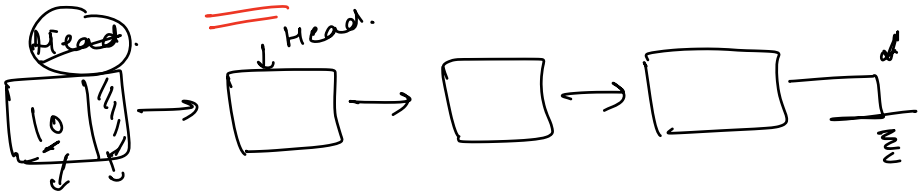
```
Node n = new Node(10);
```

n.data ⇒ 10

n.next ⇒ null.

Q Given a L.L, find its length ?

↳ Head

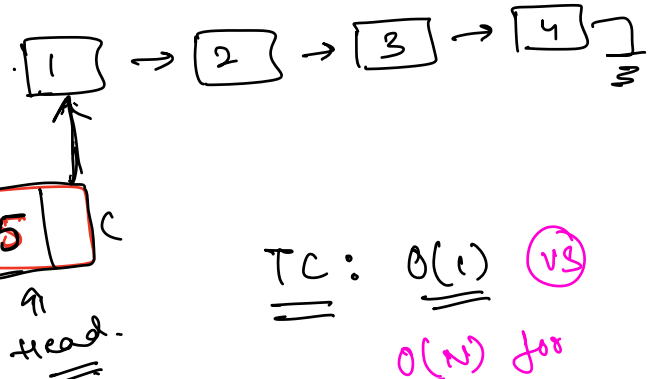



```
int getLength(Node head){
    int len = 0;
    Node temp = head;
    while(temp != null){
        len++;
        temp = temp.next;
    }
    return len;
}
```

⇒ Never Update Head of the L.L

Insert :-

1) At front :-



- Create a new node.

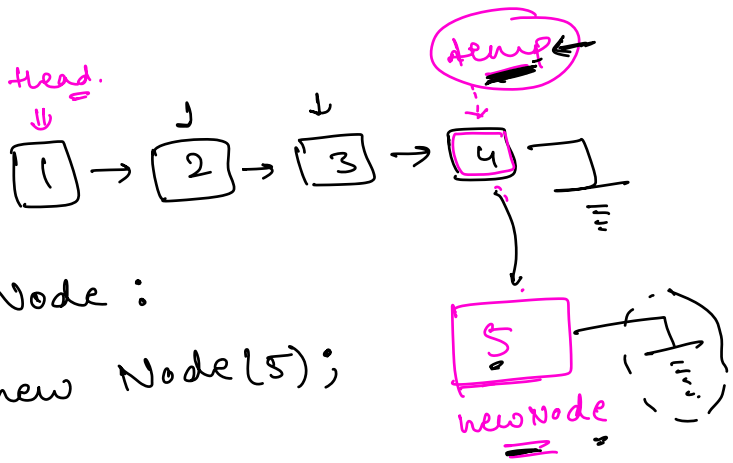
Node newNode = new Node(5);

- newNode.next = Head;
- Head = newNode;

TC : O(1) VS

O(N) for Array

ii) At Back :-



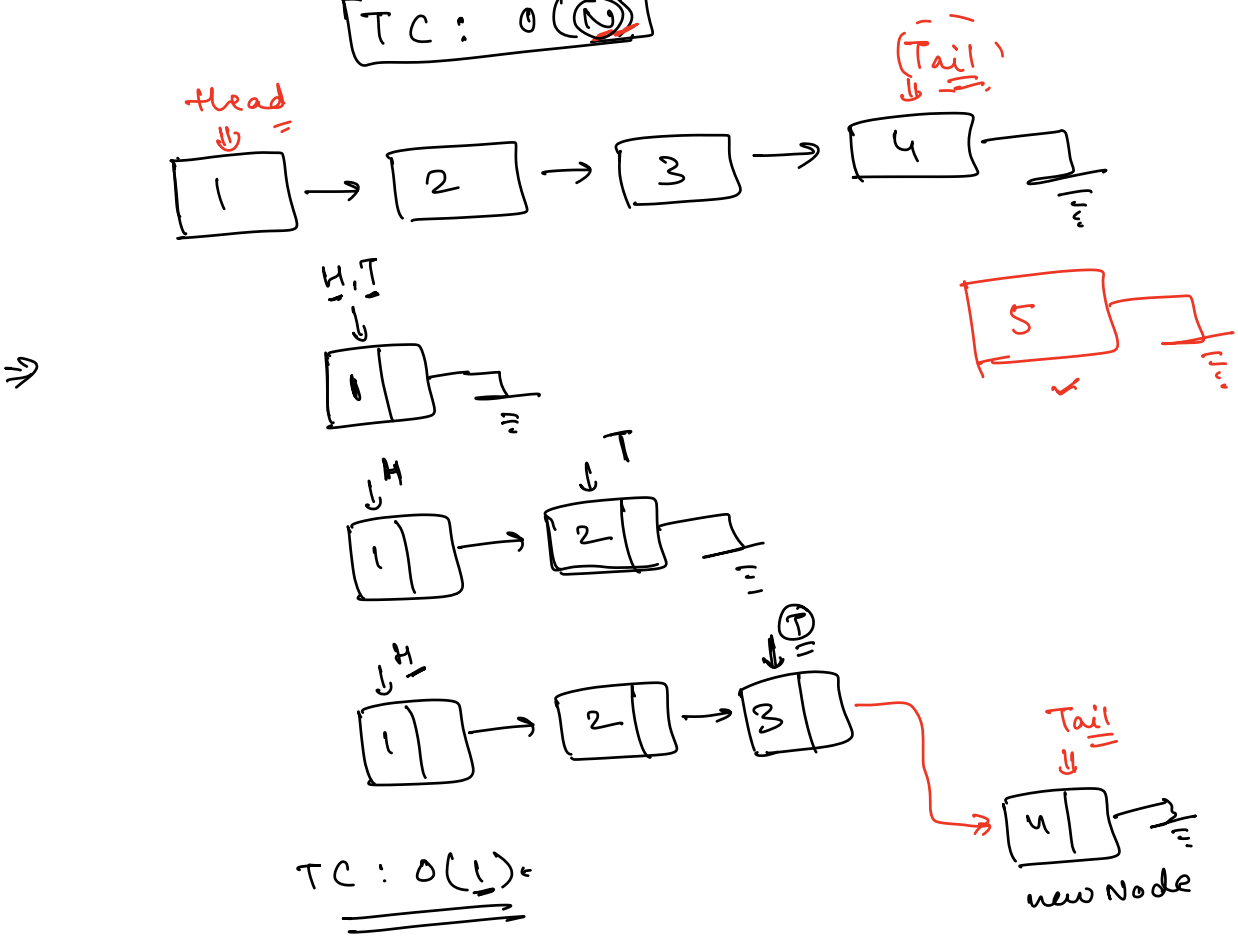
- Create the new Node :

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

- ```
  temp = head;
  while (temp.next != null)
      temp = temp.next;
  ```

- ```
  temp.next = newNode;
  ```

TC : O(N)



TC : O(1)

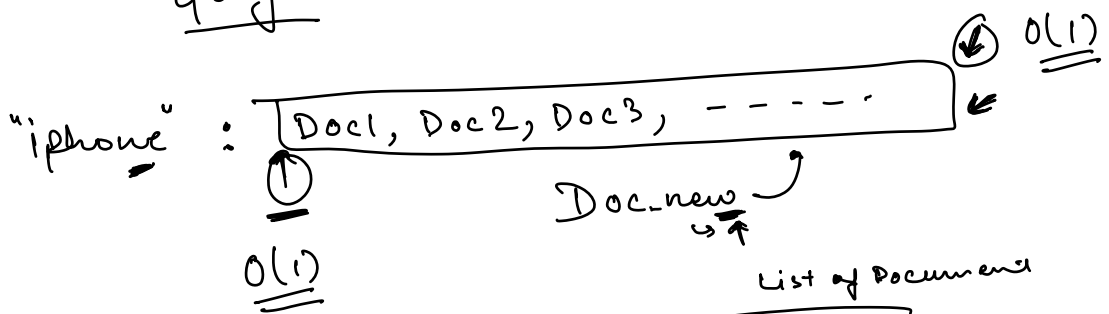
- Node newNode = new Node(4);
- Tail.next = newNode;
- Tail = newNode;

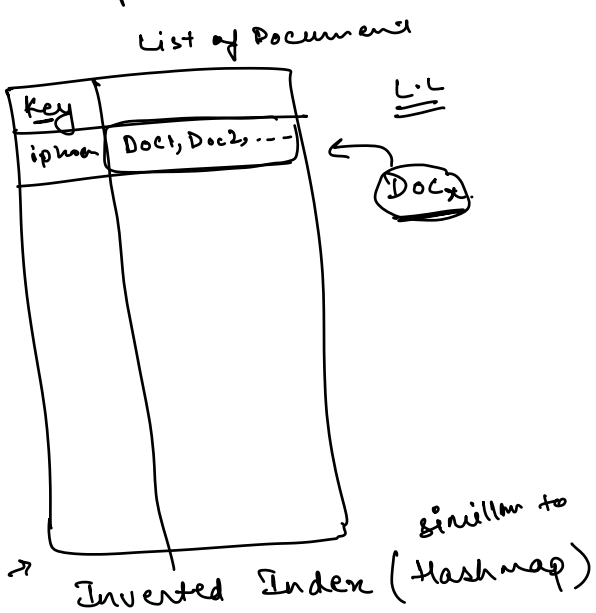
⇒ TC : O(1)

Applications of L.L :-

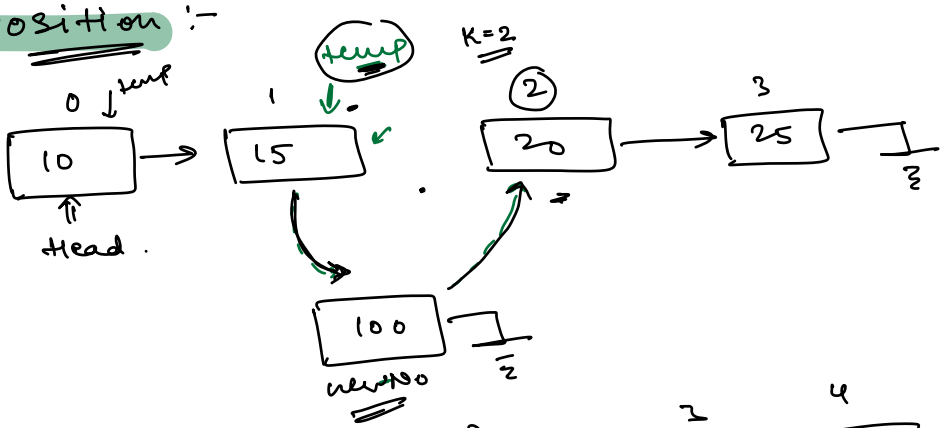
Elastic Search
Solr
Lucence
Google.

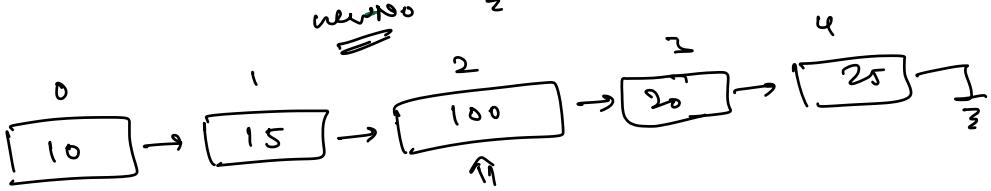
→ free text search



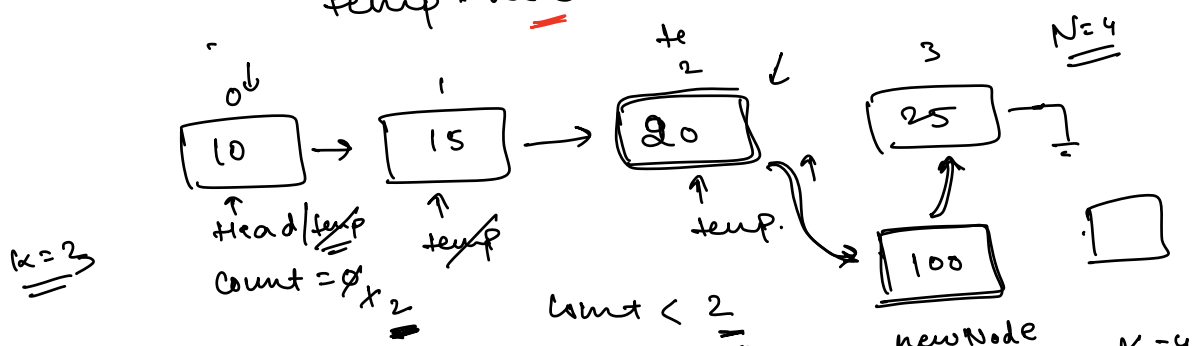


iii) K^{th} position :-






```
Count = 0
temp = head;          // K=2.
while (count < K-1) {
        temp = temp.next;
        count++;
}
Node newNode = new Node(100);
newNode.next = temp.next;
temp.next = newNode;
```



* $K \leq N$

⇒ → Reference manipulation is very important.

⇒ Steps :-

1) Write code on pen paper / google doc. ←

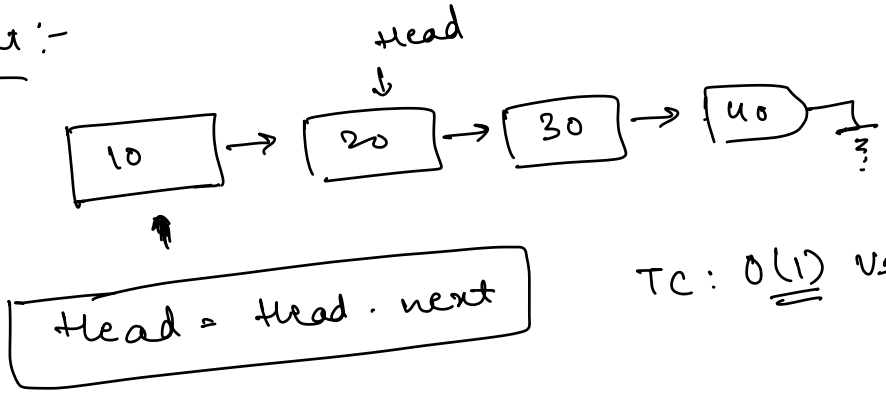
2) Dry run.

3) Edge cases :-

- → Null L.L / Head = null.
- → L.L of size = 1 | 2 | 3. →
- → Problem specific T.C's.

Deletion

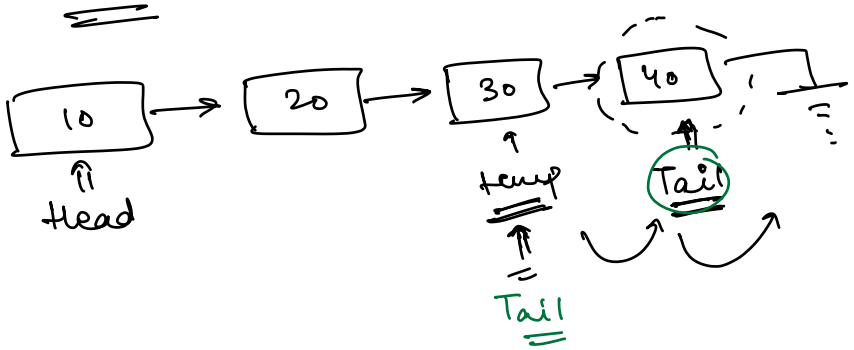
1) At front :-



TC : O(1) Vs O(N) in Array

C++ : Explicitly delete the reference.

2) from back :-



```
temp.next.next = null

while (temp.next.next != null) {
    temp = temp.next;
}
temp.next = null;
tail = temp;
```

TC: O(N)