☆ Stack (LIFO)

7	3
23	2
3	1
11	0

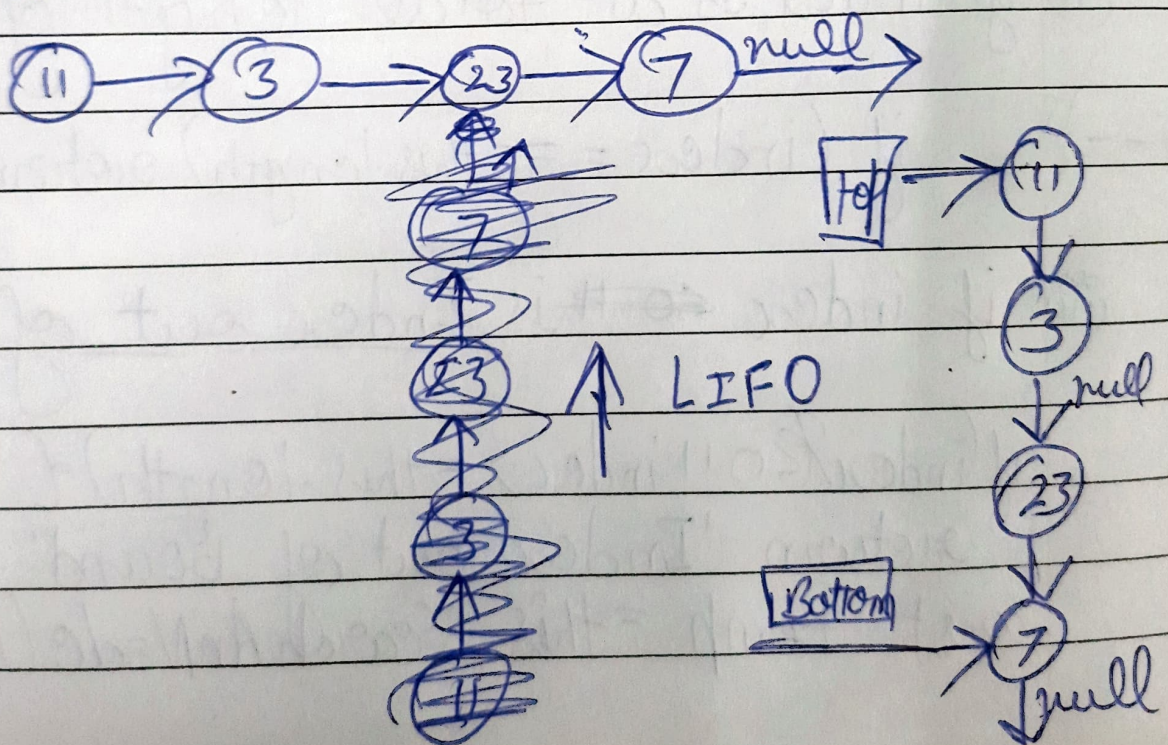
-> There are two ways to creat stack through Linked List or by Array

=> By Array

11	3	23	7	9
0	1	2	3	4

-> LIFO
(Last into First out)

=> By Linked List



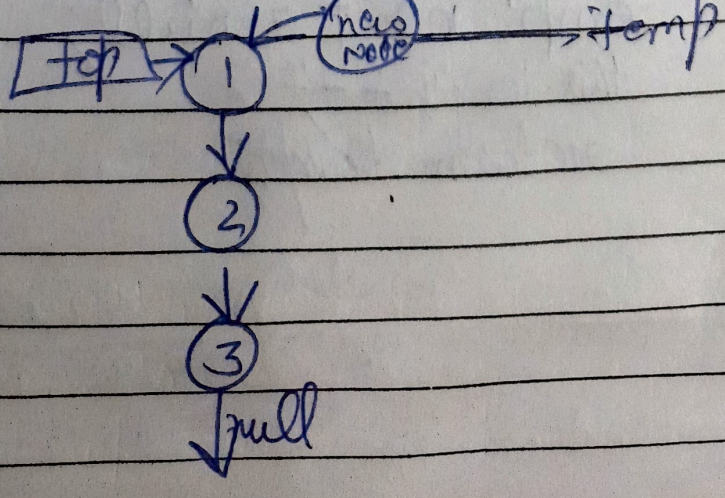
(i) Pop from (LL) is O(n)
(ii) Push in (LL) is O(1)
(iii) Insertion at Beggining in (LL) is O(1)
(iv) Deletion at Beggining in (LL) is O(1)

✱ Constructor for Stacks

```
class Node {
    constructor(value){
        this.value = value;
        this.next = null;
    }
}
```

```
class ~~My~~Stack {
    constructor(value){
        const newNod = newNode(val);
        this.top = newNode;
        this.length = 1;
    }
}
```

✱ push method Node in Stack



→ There is two case
(i) if stack is empty
(ii) if Stack has element

(i)
```
if(!this.top){
    this.top = newNode
}else{
    newNode.next = this.top;
    this.top = newNode;
}
this.length++;
return this;
```

★ Pop method in Stack

→ There is two case :-
(i) if stack is empty.
(ii) if stack has element.

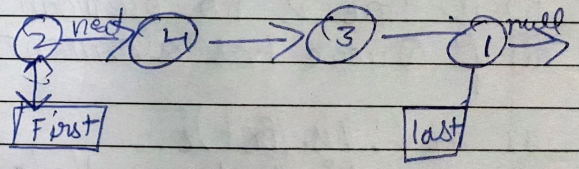
(i) if(!this.top) return 'Stack is Empty';
(ii)
```
let temp = this.top
this.top = this.top.next
temp.next = null;
this.length--;
return temp;
```

★ Queue (FIFO)

→ In array queue pop & push is O(1)
→ In array of queue insertion at Beg & deletion at Begining is O(n)

★ on the other hand in linked list
(i) push is O(1)
(ii) pop is O(n)
(iii) And Both insertion at Beg & deletion at Beg O(1)

★ Queue as Linked List



★ Queue constructor method

```
class Node {
    constructor(value){
        this.value = value;
        this.next = null;
    }
}
```

```
class Queue {
    constructor(value){
    const newNode = new Node(value);
     this.first = newNode;
     this.last = newNode;
     this.length = 1;
    }
```

★ enqueue method in Queue :-

(i) this method is same as like push method in linkedList

→ there is two case :-

(i) if there is no element in Queue
(ii) if Queue has element

```
a) if (!this.first){
       this.first = newNode;
       this.last = newNode;
   }else{
       this.last.next = newNode
       this.last = newNode;
   }
   this.length++;
   return this;
```

★ dequeue method in Queue :-

→ There is three case in dequeing method.

(i) if there is no element in Queue
(ii) if there is only one element in Queue
(iii) it has more than one element

```
let temp = this.first;
if (!this.first) return 'Queue is Empty';
if (this.length === 1){
    this.first = null;
    this.last = null;
}else{
    this.first = this.first.next;
    temp.next = null;
}
this.length--;
return temp;
```