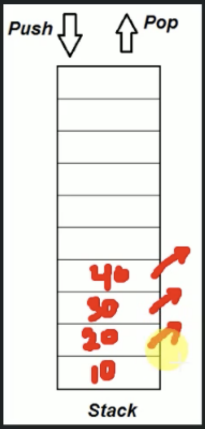
What is Stack ?



✓ Observation from above Picture:

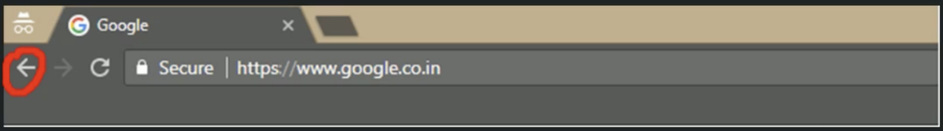
- ✓ Insertion/Removal of Bangles follows LIFO (Last in First Out) method.

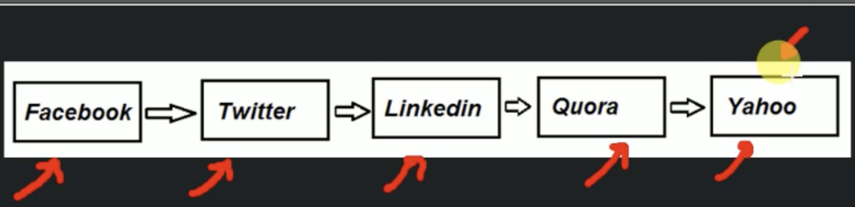
✓ Property of Stack:

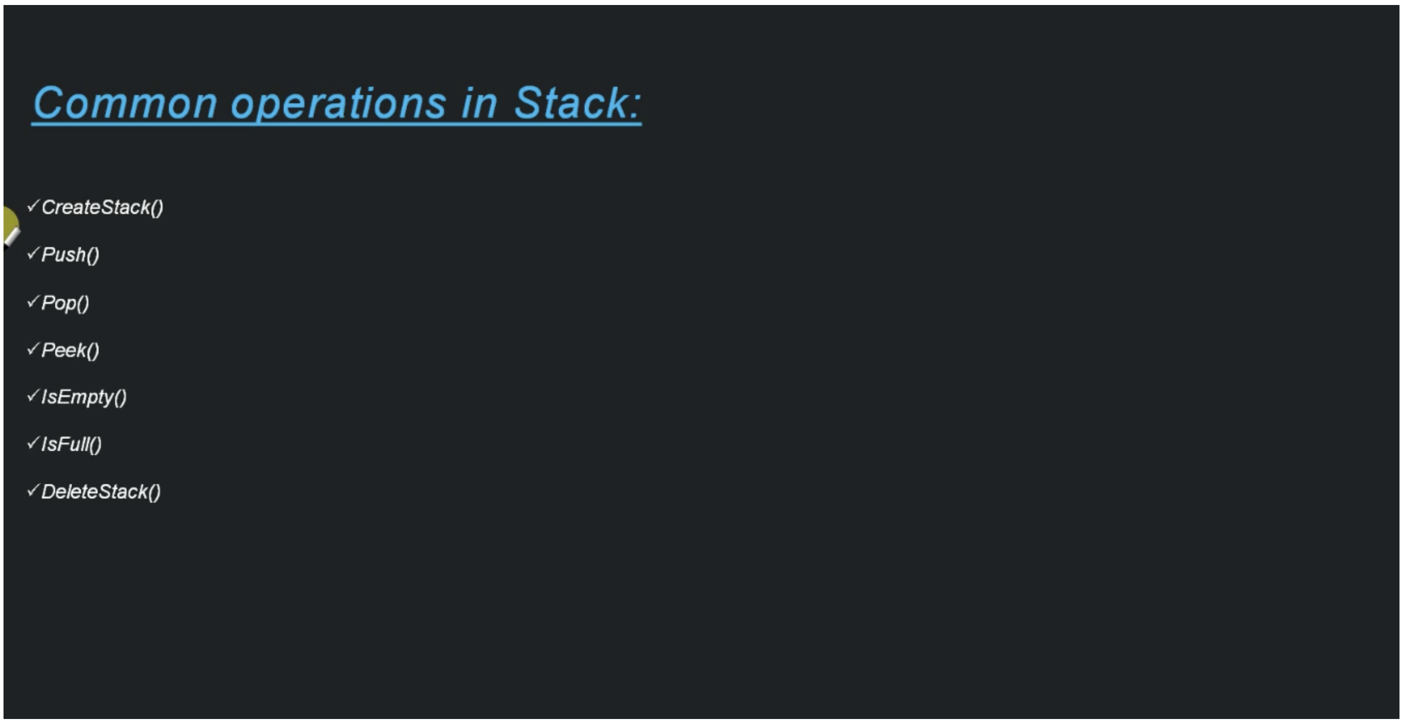
- ✓ follows LIFO (Last in First Out) method

Why should we learn Stack ?

- ✓Why ?
  - ✓ When we need to create an application which utilizes 'last incoming data first'.
  - ✓ Example: implementation of 'back' button in browser.







Common operations in Stack:

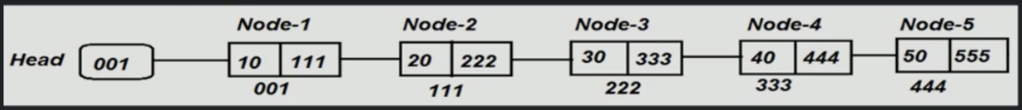
- ✓CreateStack()
- ✓Push()
- ✓Pop()
- ✓Peek()
- ✓IsEmpty()
- ✓IsFull()
- ✓DeleteStack()

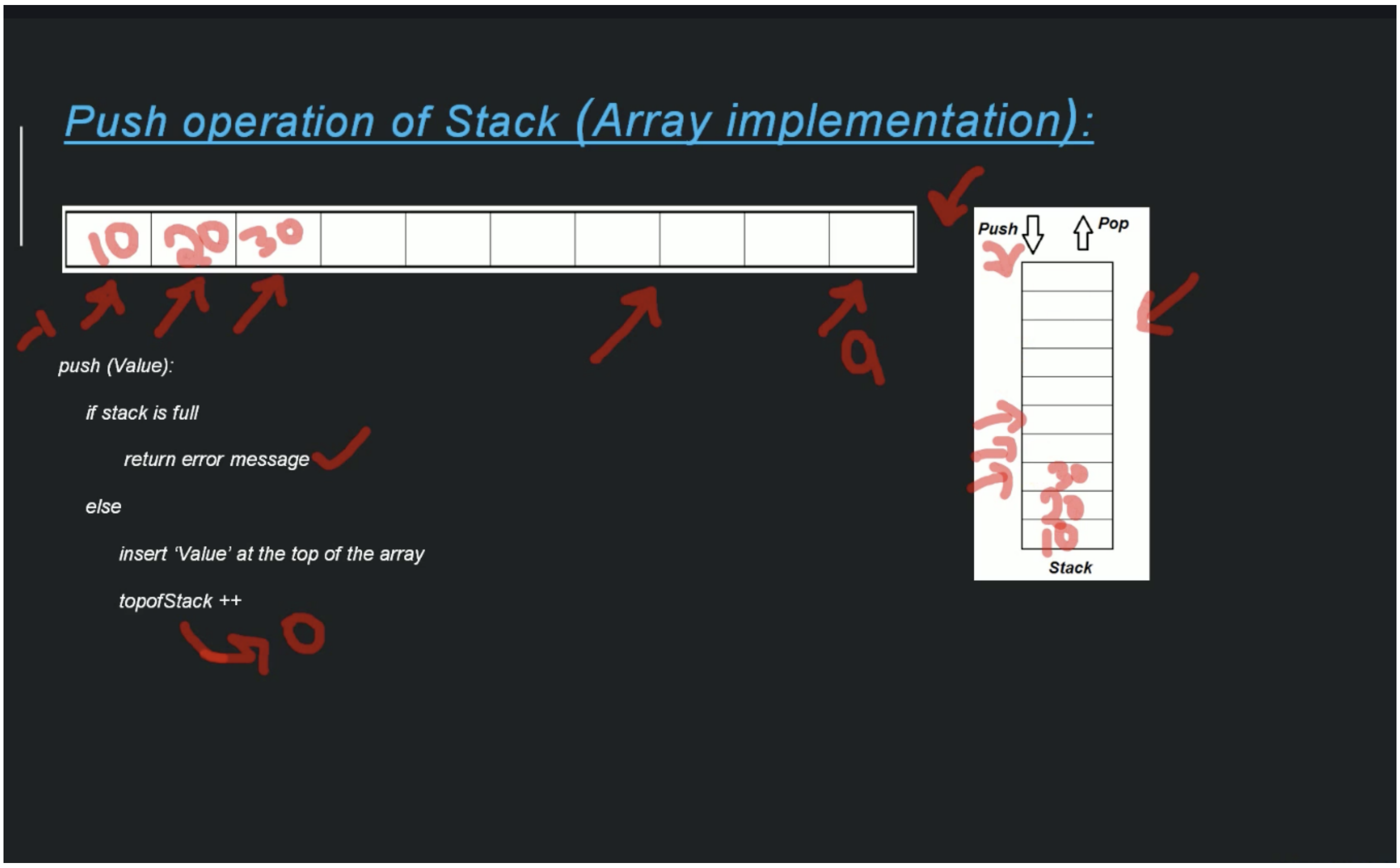
Implementation options of Stack:

- ✓ Array:
  - ✓ Pros:
    - ✓ Easy to implement
  - ✓ Cons:
    - ✓ Fixed Size

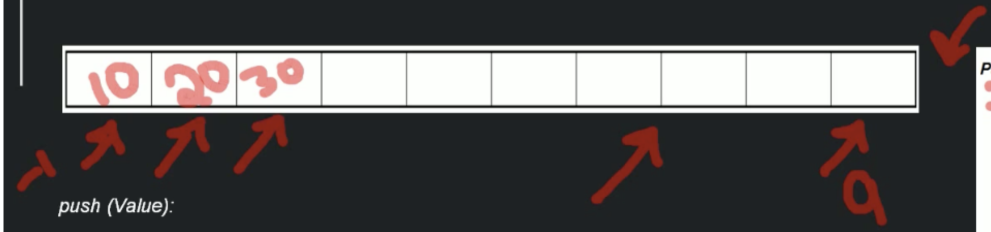
10	20	30	40			

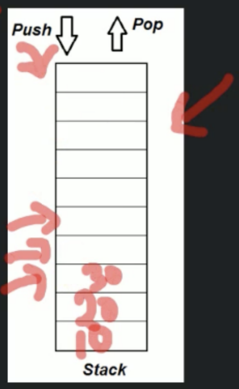
- ✓ Linked List:
  - ✓ Pros:
    - ✓ Variable Size
  - ✓ Cons:
    - ✓ Moderate in implementation





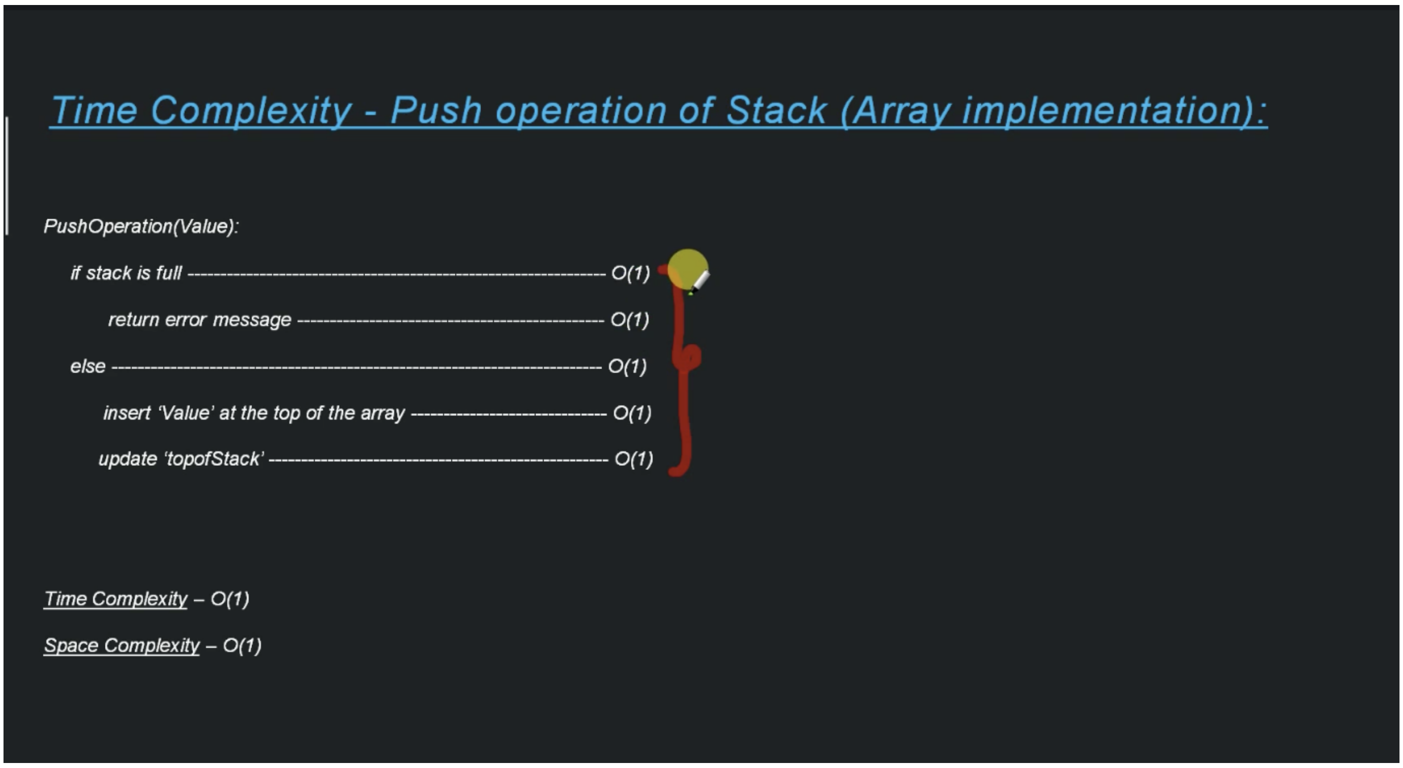
Push operation of Stack (Array implementation):





```
push (Value):
  if stack is full
      return error message
  else
      insert 'Value' at the top of the array
      topofStack ++
```

0

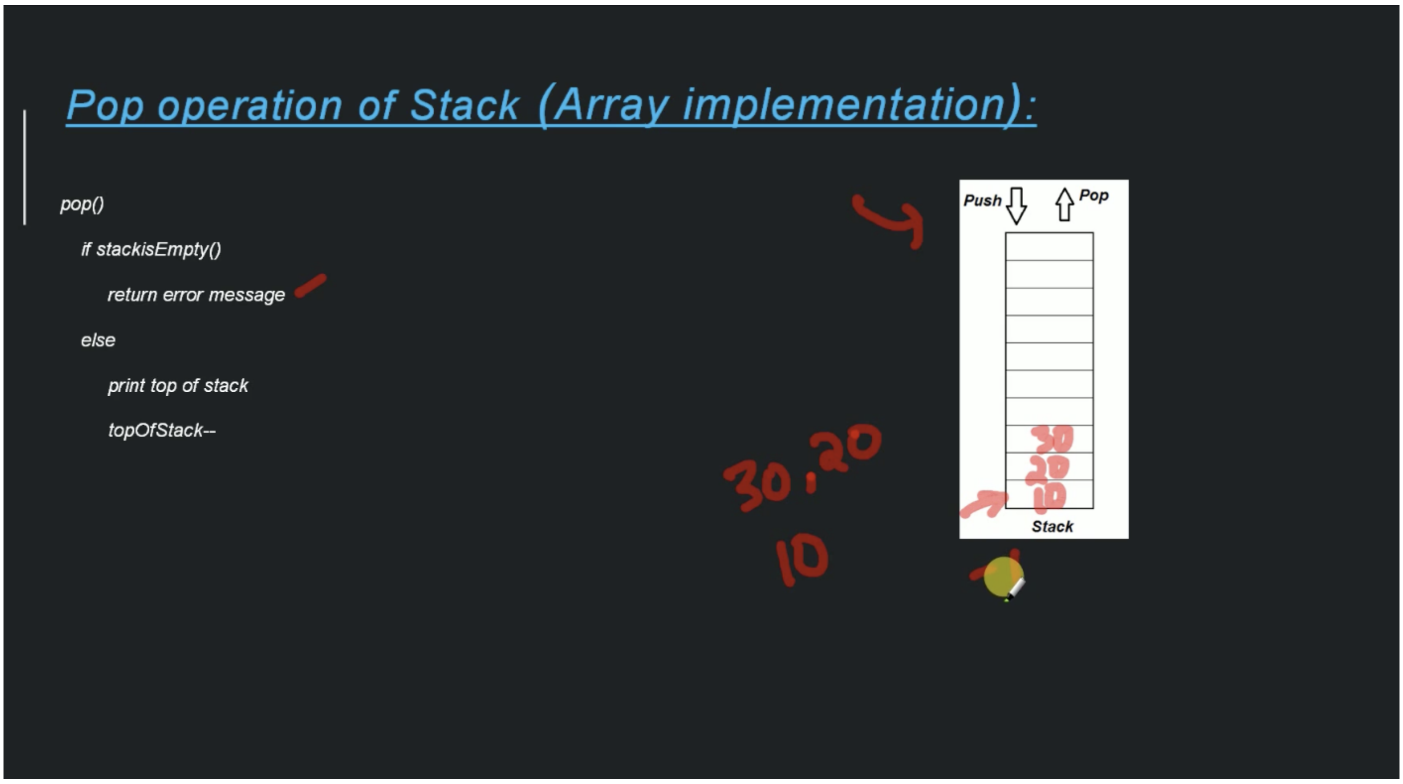


Time Complexity - Push operation of Stack (Array implementation):

```
PushOperation(Value):
    if stack is full ---------------------------------------------- O(1)
        return error message ---------------------------------- O(1)
    else ---------------------------------------------------------- O(1)
        insert 'Value' at the top of the array ----------------- O(1)
        update 'topofStack' ----------------------------------- O(1)
```

Time Complexity – O(1)

Space Complexity – O(1)

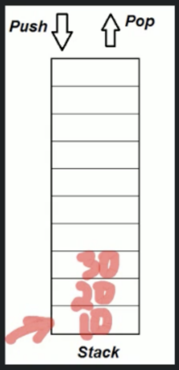


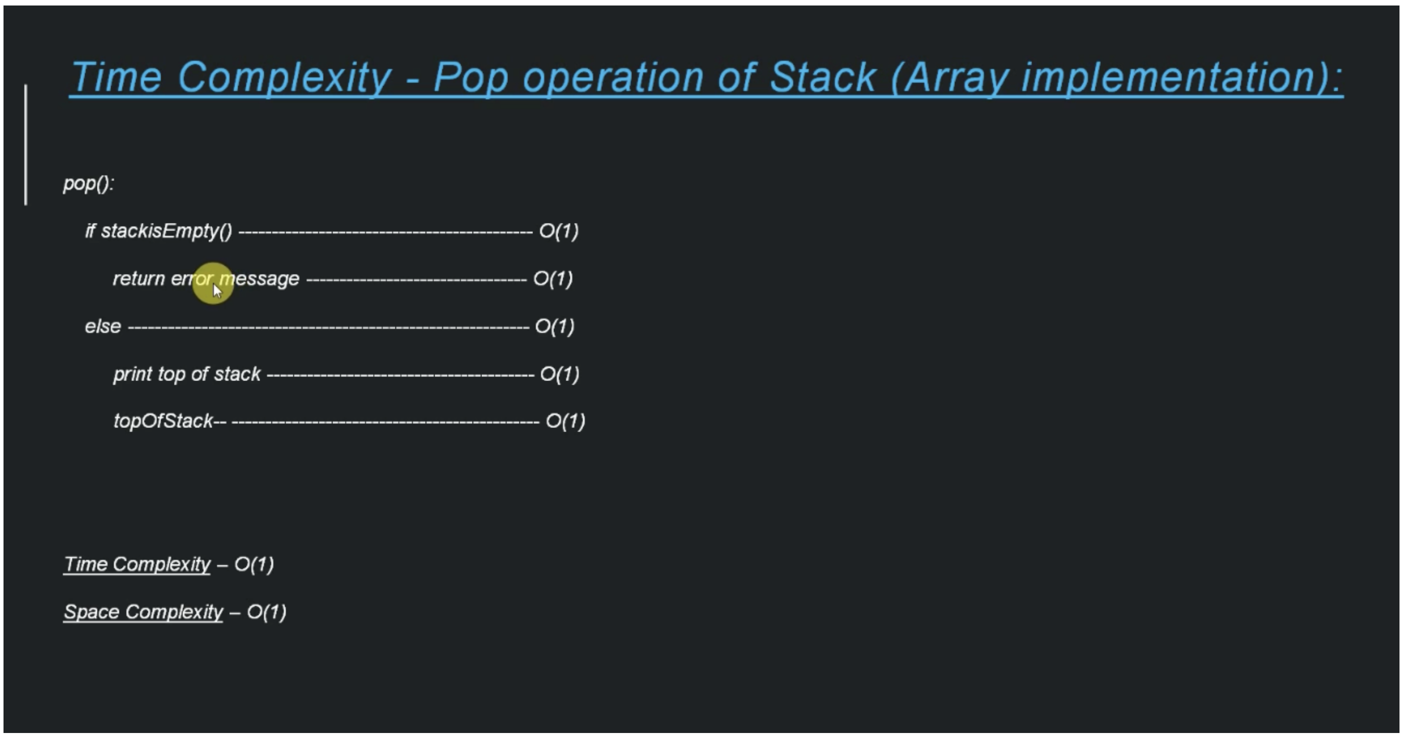
Pop operation of Stack (Array implementation):

```
pop()
  if stackisEmpty()
     return error message
  else
     print top of stack
     topOfStack--
```

30, 20

10





Time Complexity - Pop operation of Stack (Array implementation):

```
pop():
    if stackisEmpty() -------------------------------------------- O(1)
        return error message ---------------------------------- O(1)
    else ---------------------------------------------------------- O(1)
        print top of stack ------------------------------------- O(1)
        topOfStack-- --------------------------------------------- O(1)
```

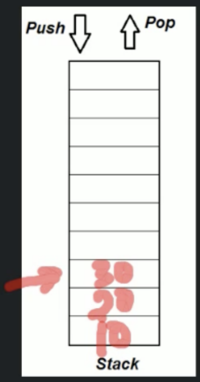
Time Complexity – O(1)

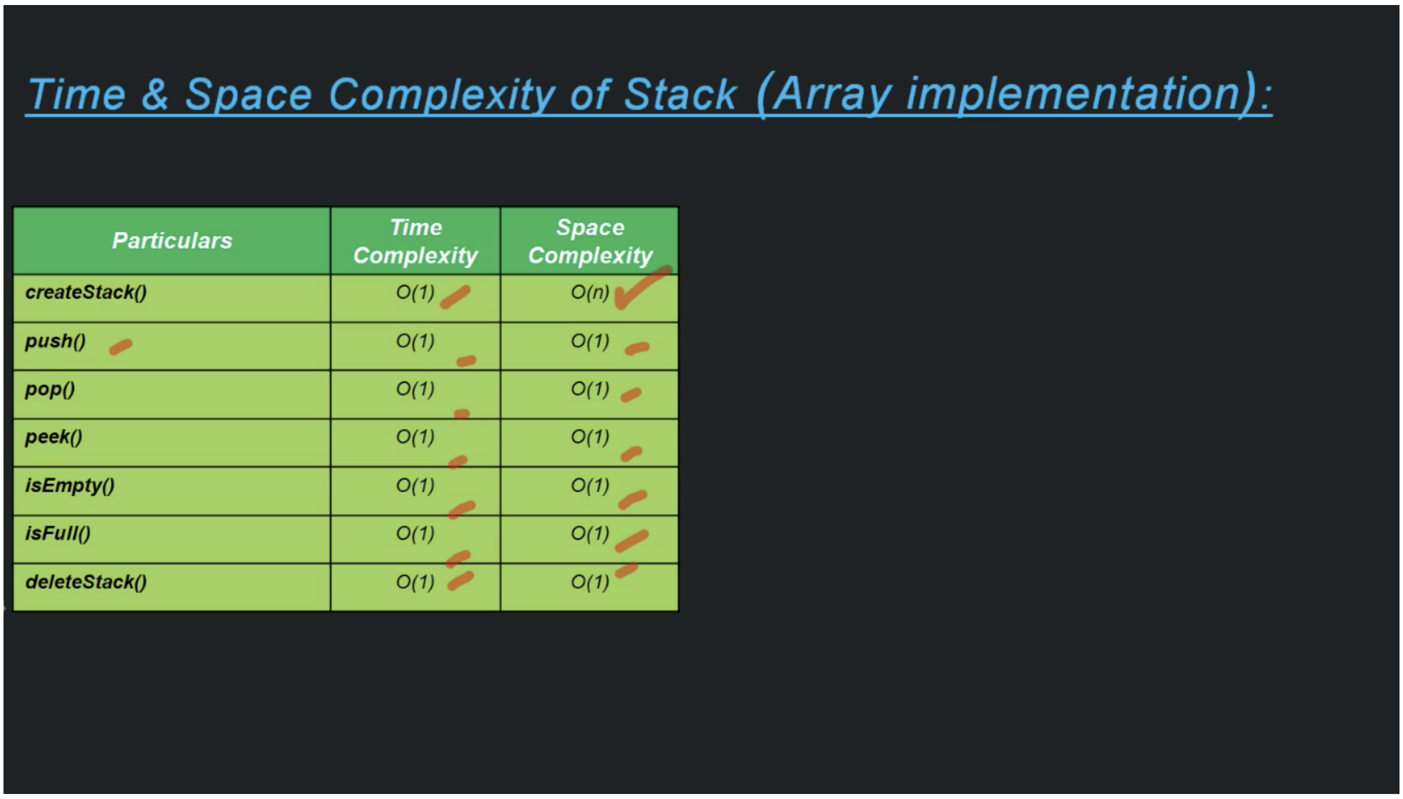
Space Complexity – O(1)

Peek operation of Stack (Array implementation):

```
peek()
   if stackisEmpty()
       return error message
   else
       print topOfStack
```

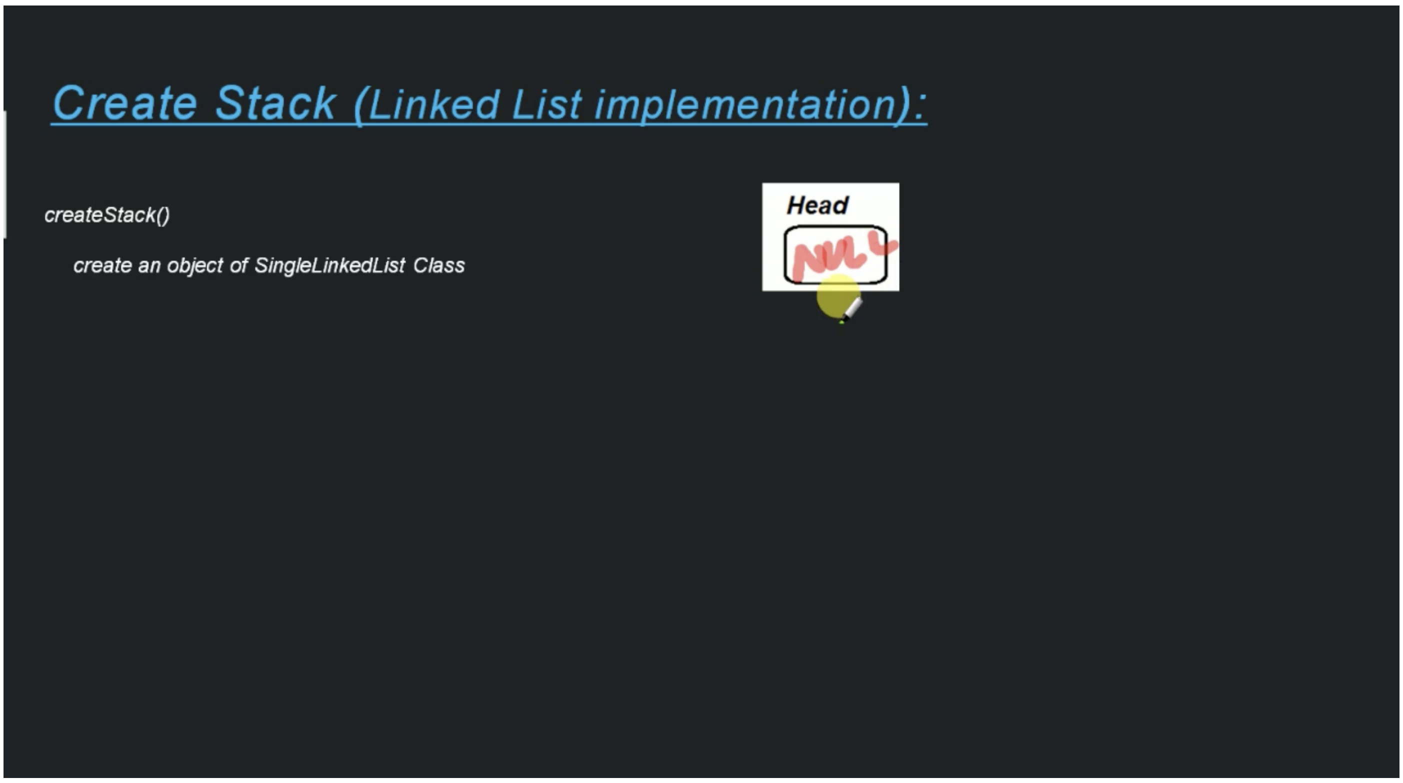
30,
30
30





Time & Space Complexity of Stack (Array implementation):

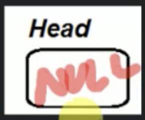
Particulars	Time Complexity	Space Complexity
createStack()	*O(1)* ✓	*O(n)* ✓
push() ✓	*O(1)* ✓	*O(1)* ✓
pop()	*O(1)* ✓	*O(1)* ✓
peek()	*O(1)* ✓	*O(1)* ✓
isEmpty()	*O(1)* ✓	*O(1)* ✓
isFull()	*O(1)* ✓	*O(1)* ✓
deleteStack()	*O(1)* ✓	*O(1)* ✓

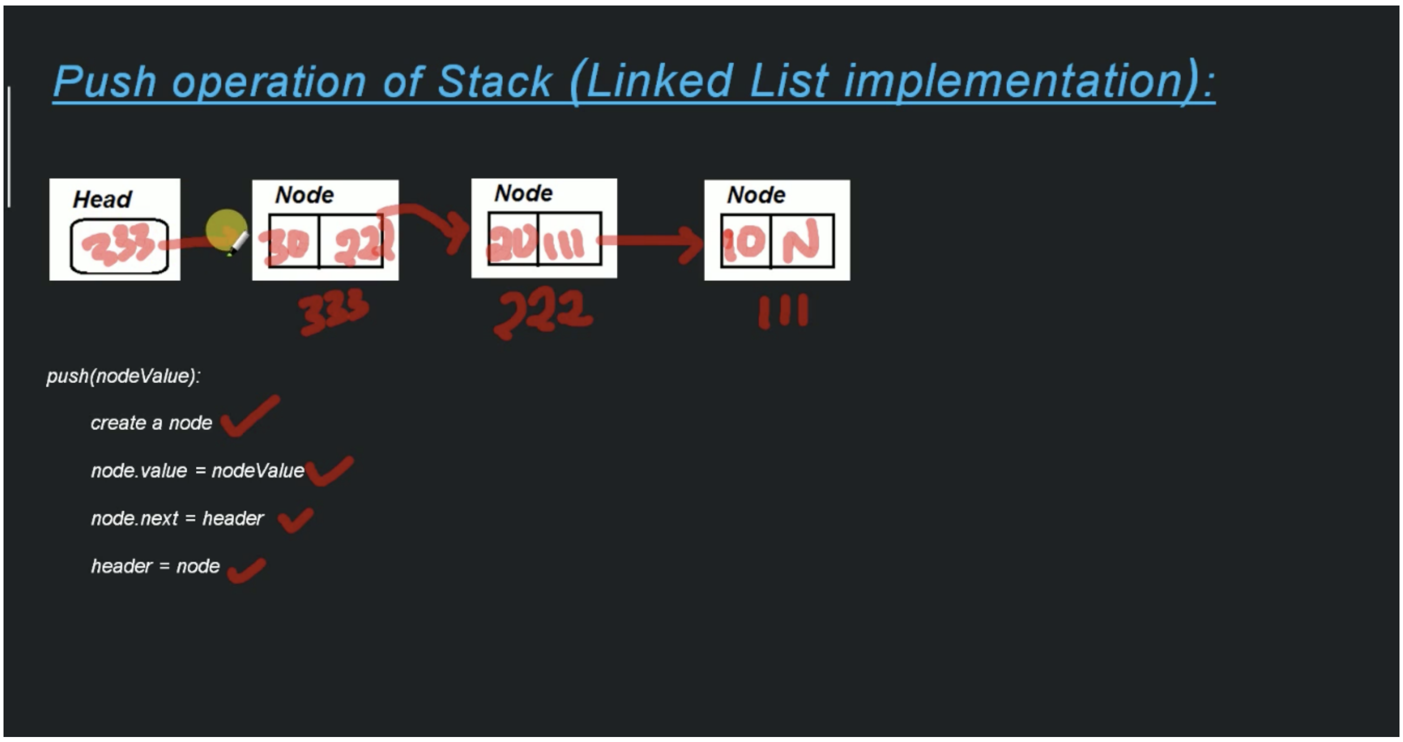


Create Stack (Linked List implementation):

createStack()

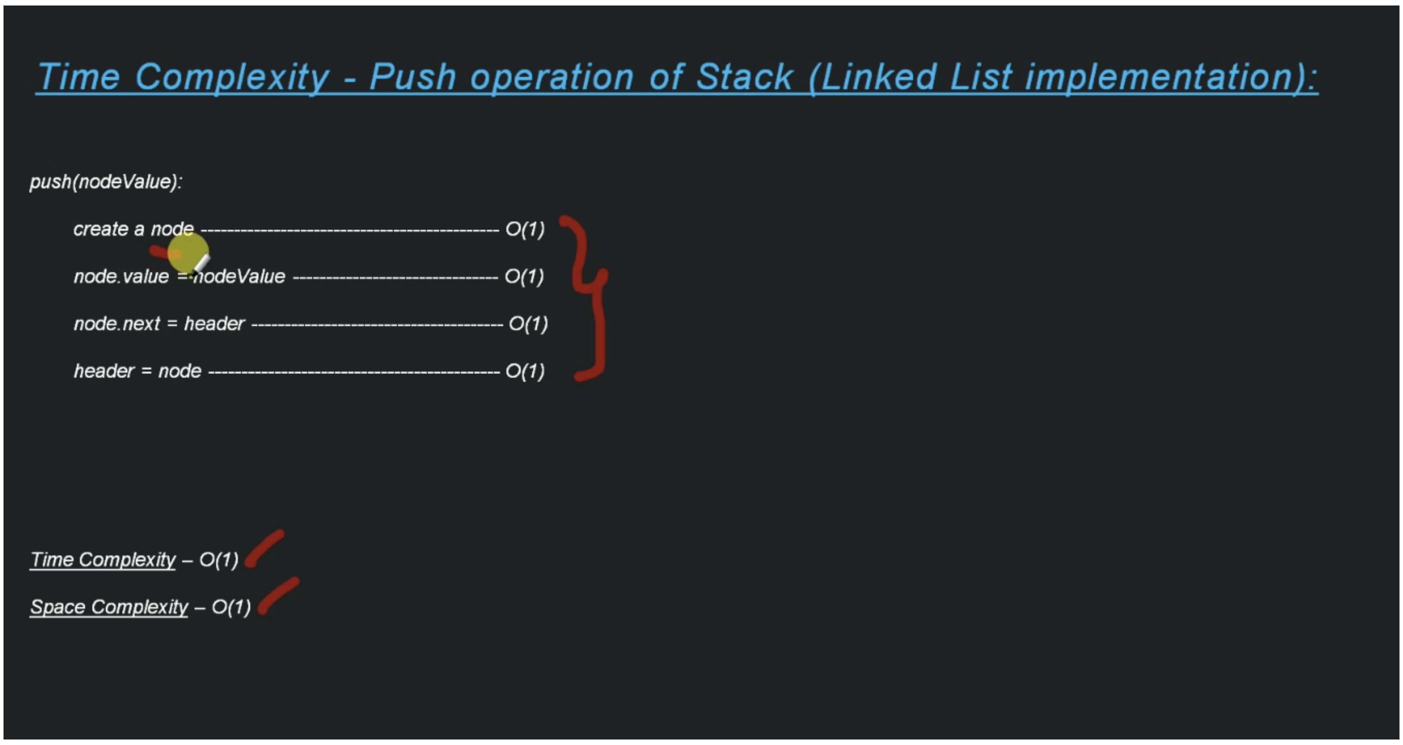
    create an object of SingleLinkedList Class





Push operation of Stack (Linked List implementation):

```
push(nodeValue):
    create a node ✓
    node.value = nodeValue ✓
    node.next = header ✓
    header = node ✓
```



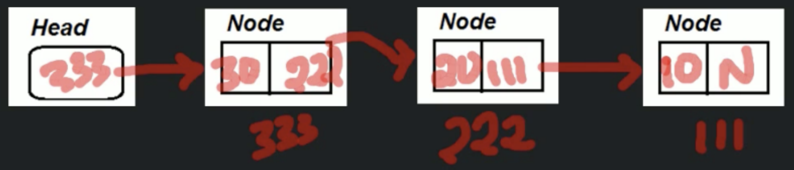
Time Complexity - Push operation of Stack (Linked List implementation):

```
push(nodeValue):
    create a node ------------------------------------------- O(1)
    node.value = nodeValue ------------------------------ O(1)
    node.next = header ------------------------------------ O(1)
    header = node ------------------------------------------ O(1)
```

Time Complexity – O(1) ✓

Space Complexity – O(1) ✓

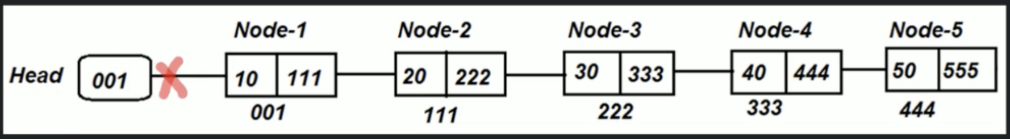
Pop operation of Stack (Linked List implementation):



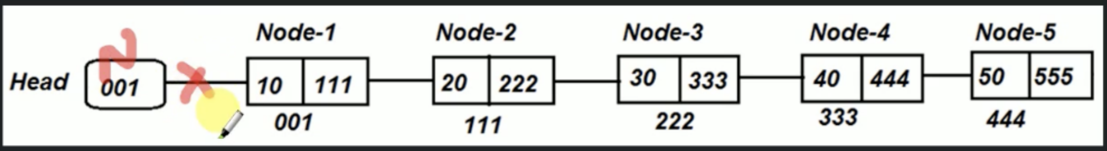
```
pop():
   if isEmpty() ✓
      return error message ✓
  else
      tmpNode = head
     header = header.next
     return tmpNode.value
```

IsEmpty operation of Stack (Linked List implementation):

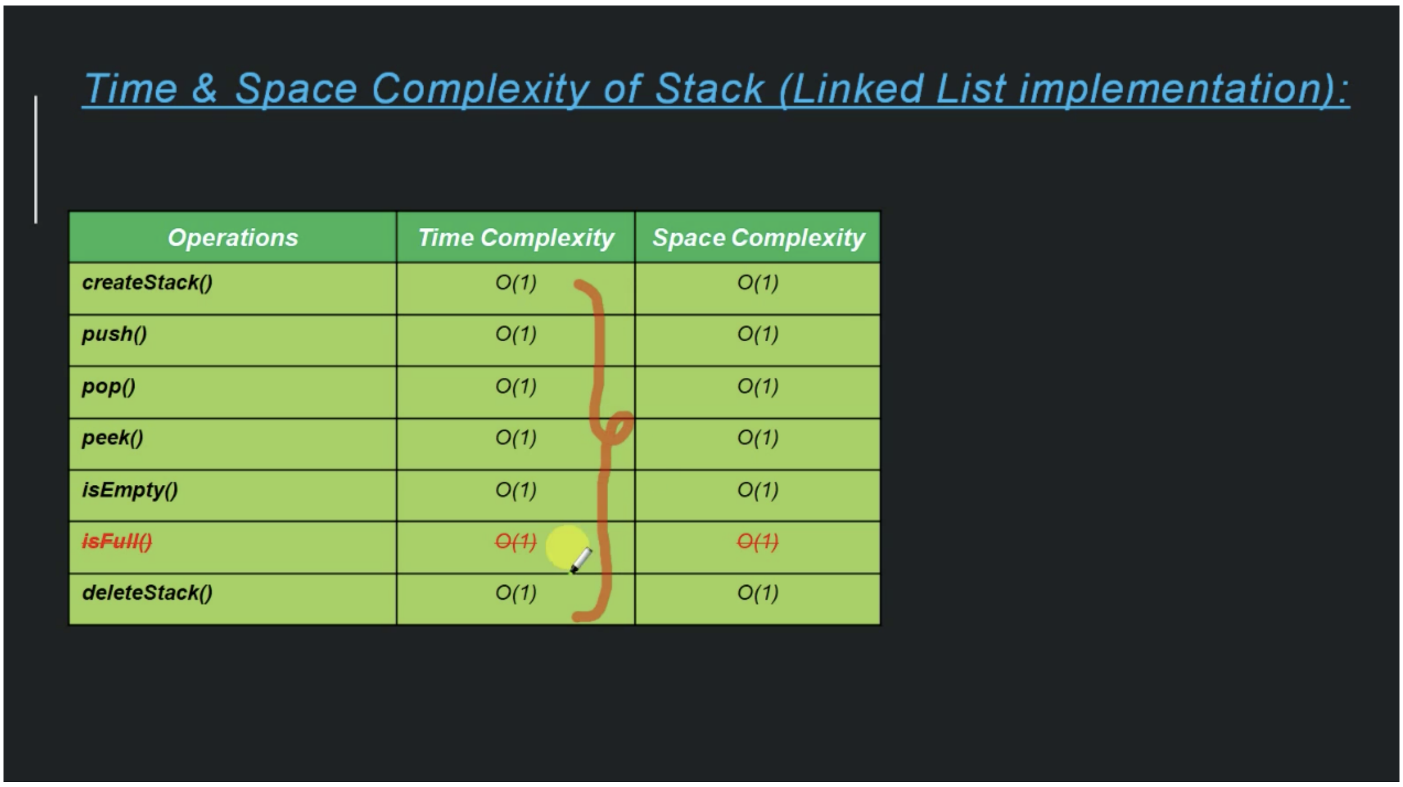
```
IsEmpty():
   if (header equals null)
        return true
   else
        return false
```



Deletion of entire Stack (Linked List implementation):

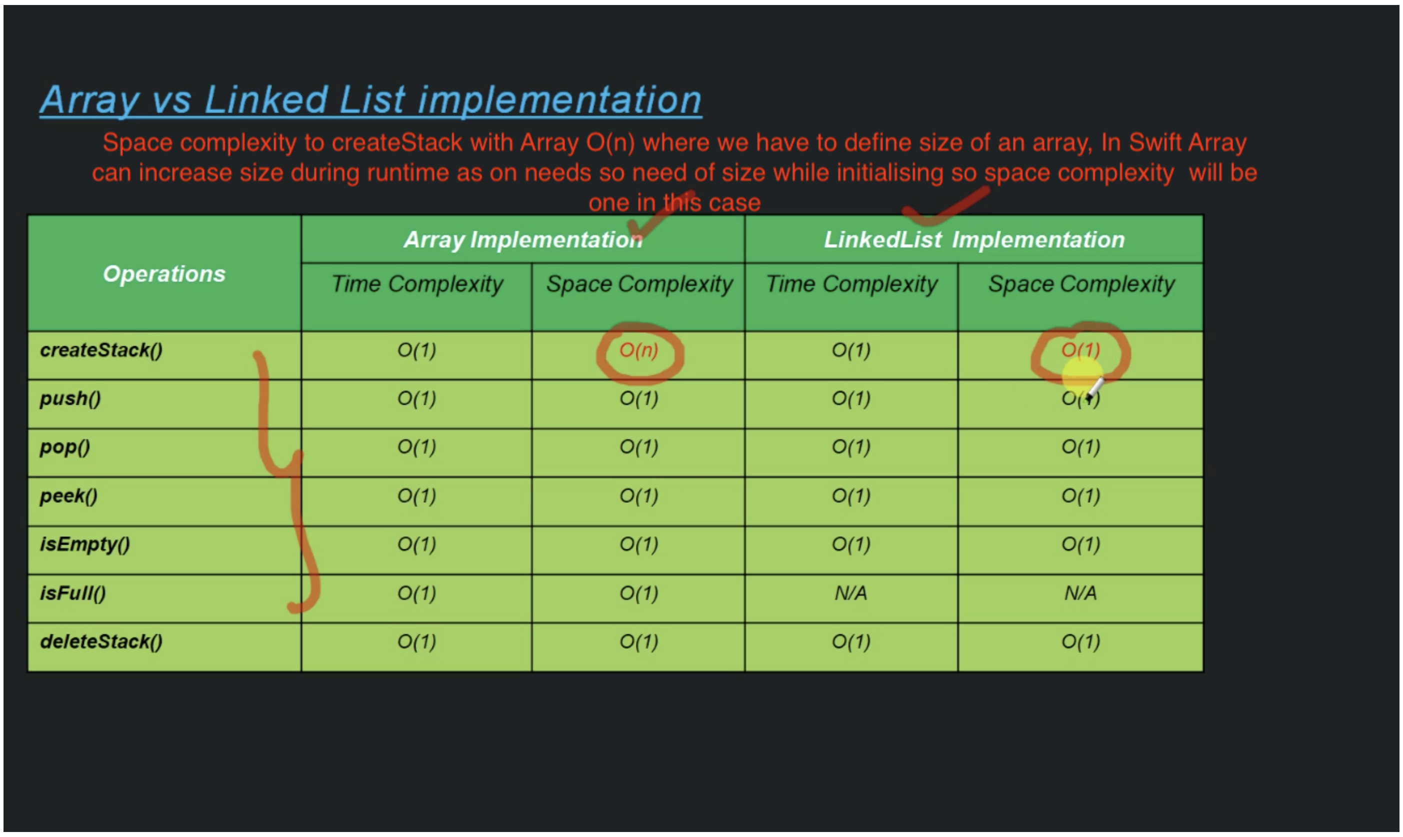


```
deleteStack():
  header = null ✓
```



Time & Space Complexity of Stack (Linked List implementation):

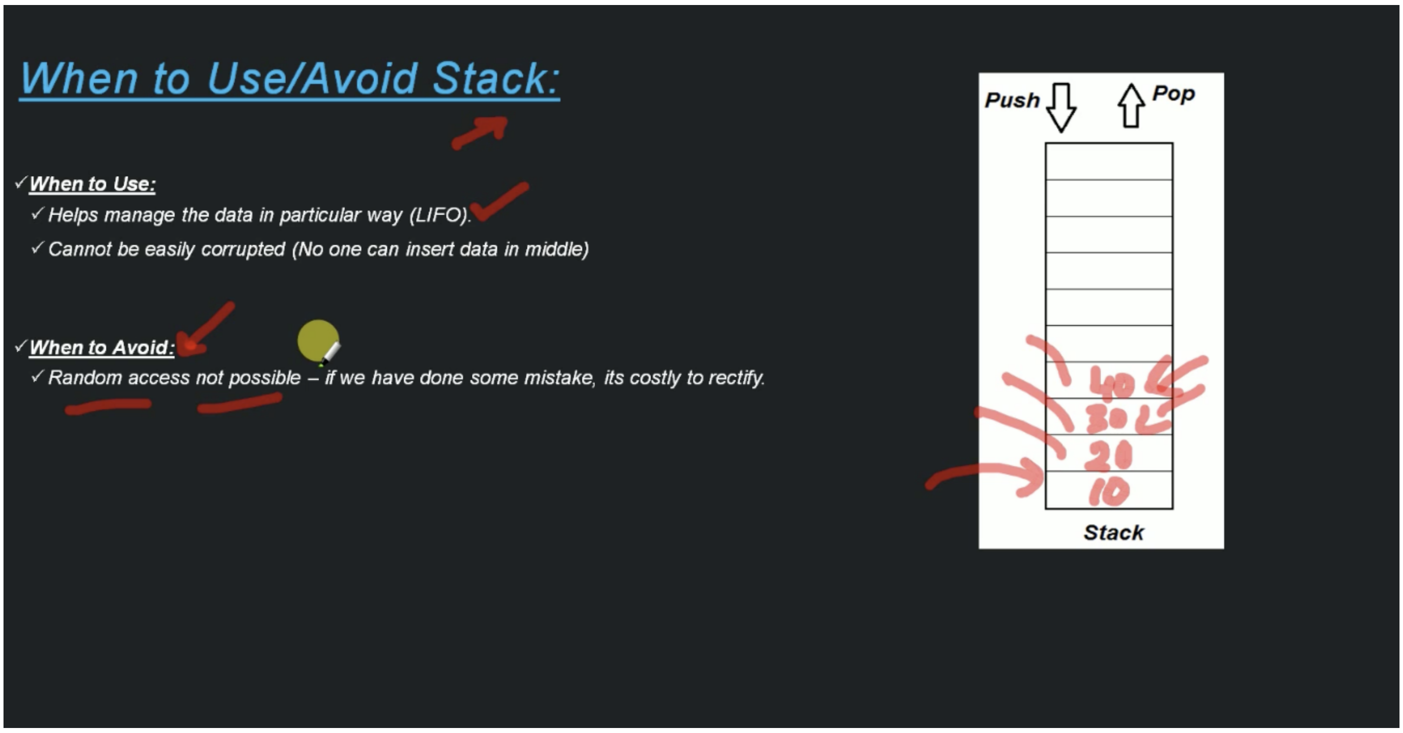
Operations	Time Complexity	Space Complexity
createStack()	*O(1)*	*O(1)*
push()	*O(1)*	*O(1)*
pop()	*O(1)*	*O(1)*
peek()	*O(1)*	*O(1)*
isEmpty()	*O(1)*	*O(1)*
~~***isFull()***~~	~~*O(1)*~~	~~*O(1)*~~
deleteStack()	*O(1)*	*O(1)*



Array vs Linked List implementation

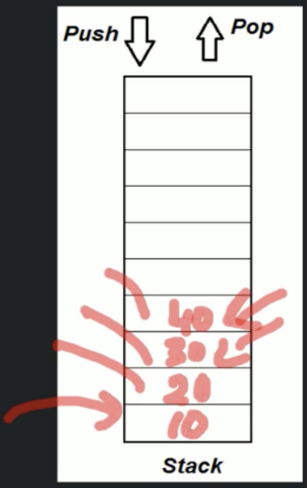
Space complexity to createStack with Array O(n) where we have to define size of an array, In Swift Array can increase size during runtime as on needs so need of size while initialising so space complexity will be one in this case

Operations	Array Implementation		LinkedList Implementation	
	Time Complexity	Space Complexity	Time Complexity	Space Complexity
createStack()	*O(1)*	*O(n)*	*O(1)*	*O(1)*
push()	*O(1)*	*O(1)*	*O(1)*	*O(1)*
pop()	*O(1)*	*O(1)*	*O(1)*	*O(1)*
peek()	*O(1)*	*O(1)*	*O(1)*	*O(1)*
isEmpty()	*O(1)*	*O(1)*	*O(1)*	*O(1)*
isFull()	*O(1)*	*O(1)*	*N/A*	*N/A*
deleteStack()	*O(1)*	*O(1)*	*O(1)*	*O(1)*



When to Use/Avoid Stack:

- ✓ **When to Use:**
  - ✓ *Helps manage the data in particular way (LIFO).*
  - ✓ *Cannot be easily corrupted (No one can insert data in middle)*
- ✓ **When to Avoid:**
  - ✓ *Random access not possible – if we have done some mistake, its costly to rectify.*


Push
Pop
40
30
20
10
Stack