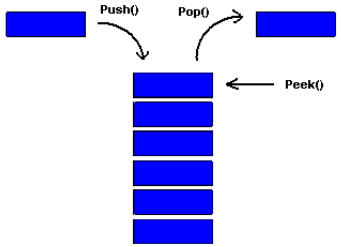
Al Hussein
Technical University
جامعة الحسين التقنية

Data Structures and Algorithms

Stack



Prepared by:

Eng. Malek Al-Louzi

School of Computing and Informatics – Al Hussein Technical University

Spring 2021/2022

Outlines

- Introduction to Stack
- Stack ADT
- Stack Operations
- Stack Applications
- Stack Implementation
- Stack Implementation using :Array & Linked List

Eng. Malek Lozi

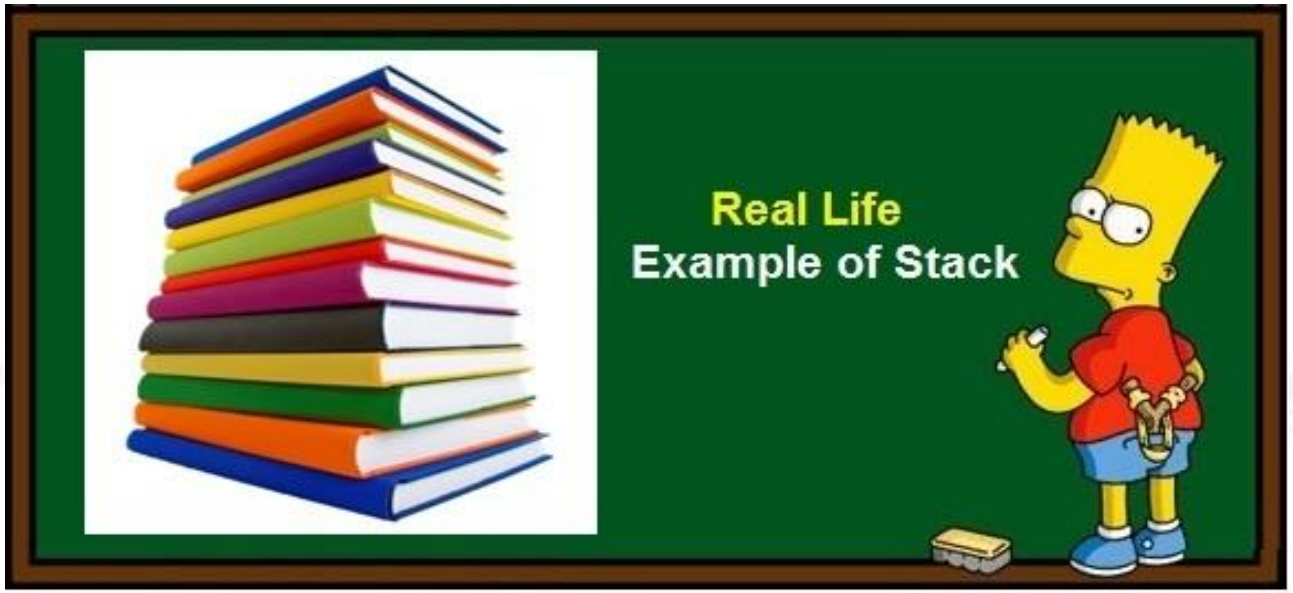
WHAT IS Stack ?

Eng. Malek Lozi

What is Stack?

- Stack is a data structure.
  - Data structures is away to store and organize data.
    - Arrays.
    - Linked Lists.
- We will define a stack as ADT.
  - Features and operations without implementation details.

Eng. Malek Lozi



A stack of plates

A stack of tennis balls

Eng. Malek Lozi

What is Stack?

- Stack is basically a collection with a property that an item in the stack must be inserted or removed from the same end.
- This end is called the Top of the stack.
- The Stack also called Last-In First-Out (LIFO) Data Structure.

Eng. Malek Lozi

Stack ADT

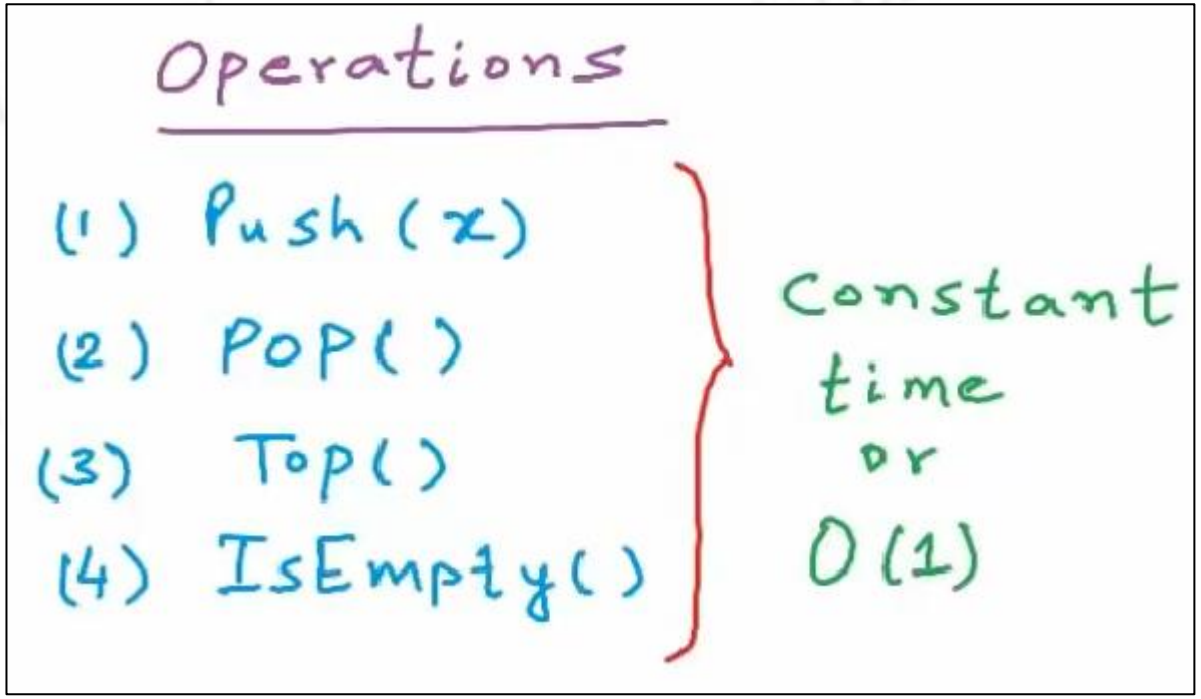
- Formal definition of a stack ADT:

A list with the restriction that insertion and deletion can be performed only from one end, called the top.

Eng. Malek Lozi

Stack ADT

- The main two operations are:
  - Insertion an element, called **Push.**
  - Removing an element, called **Pop.**
- **Top**, returns the element at the top of stack.
- **IsEmpty**, Check if stack is empty or not.



- Push and Pop One element at a time.
- **All operations should be performed in constant time -> O(1).**

Logical Representation of Stack

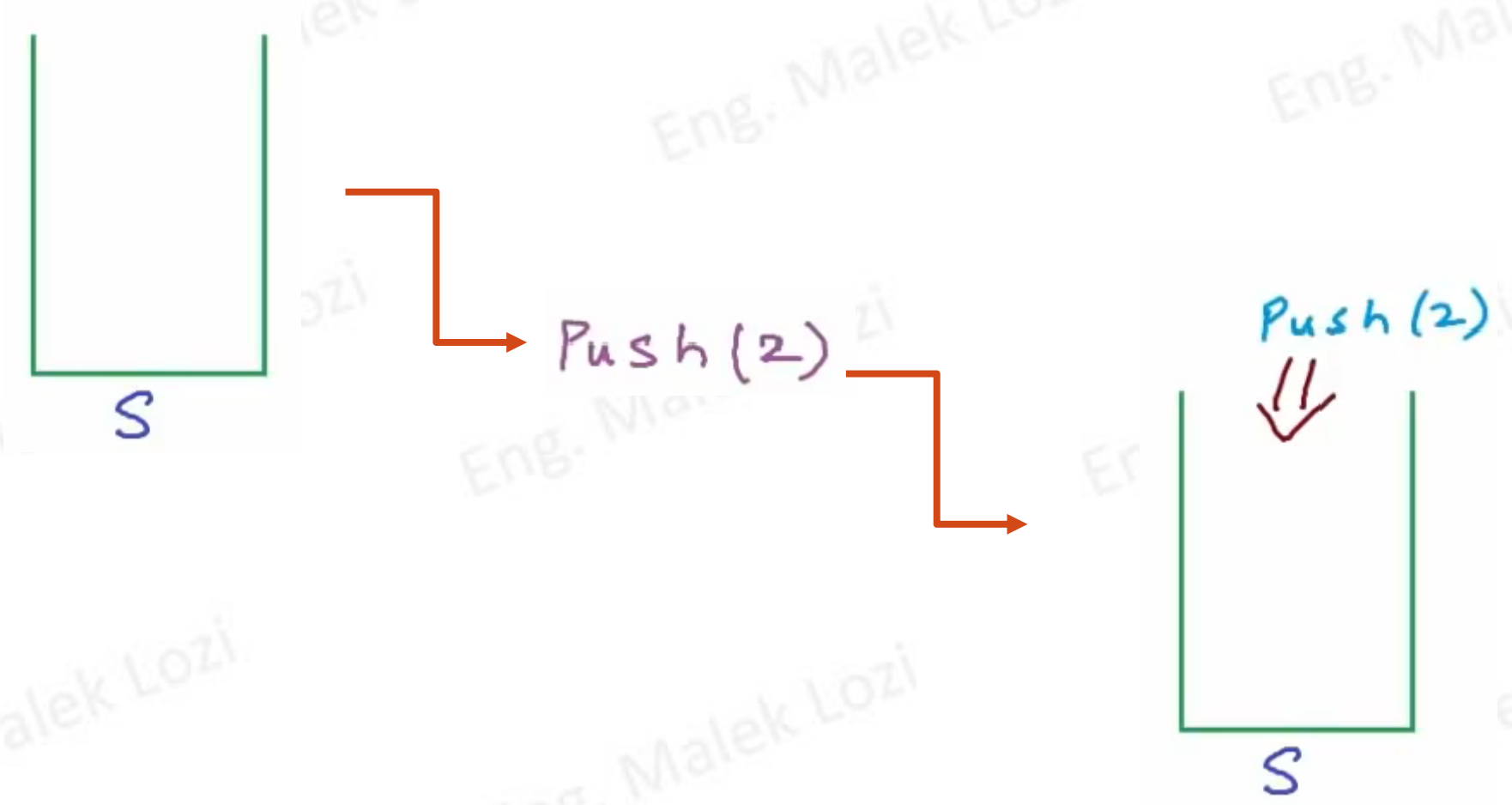
- Logical representation of a stack is a **Container opened from one side.**

Stack

Eng. Malek Lozi

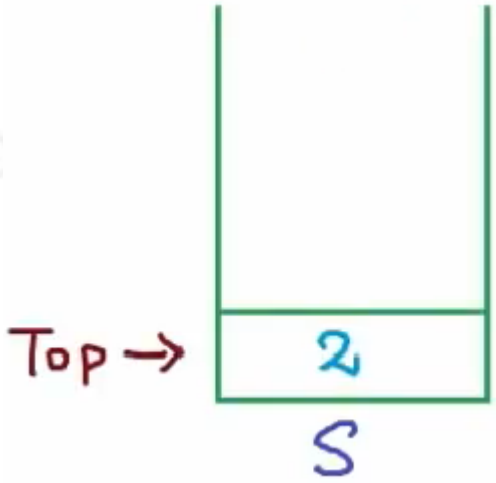
Stack Operations (1/4)

- This is an empty stack called S, and we will insert number “2” in it.



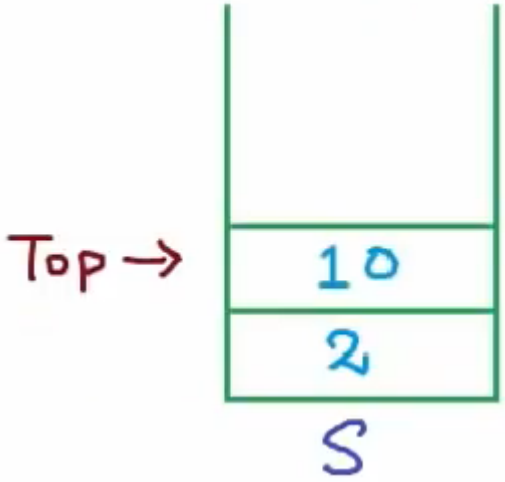
Stack Operations (2/4)

- After the Push, the stack will look like:
  - It contains one integer at the top.



- Inserting number 10.

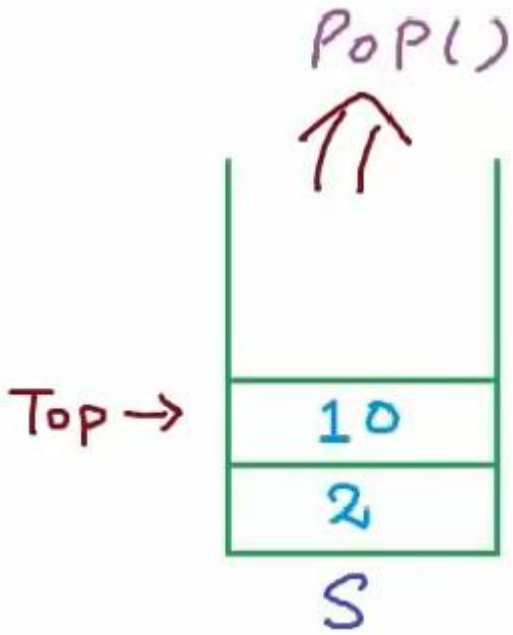
Push(10)



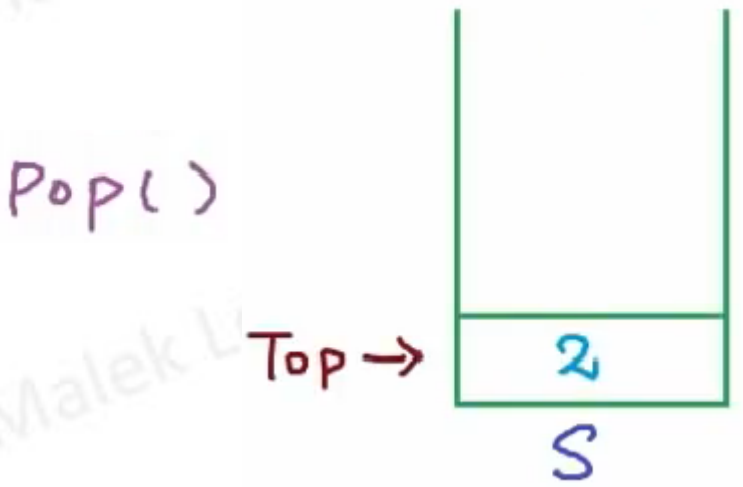
Eng. Malek Lozi

Stack Operations (3/4)

- Removing an element.



- After the Pop operation.

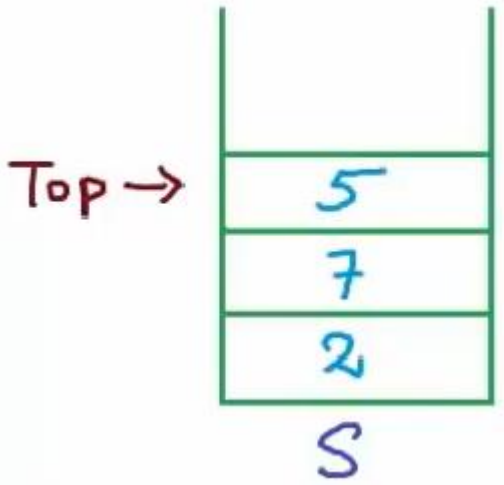


Eng. Malek Lozi

Stack Operations (4/4)

- After another Two Push operations.

Push(7)
Push(5)



- At this stage of the stack:
  - Top will return 5.
  - IsEmpty will return False.

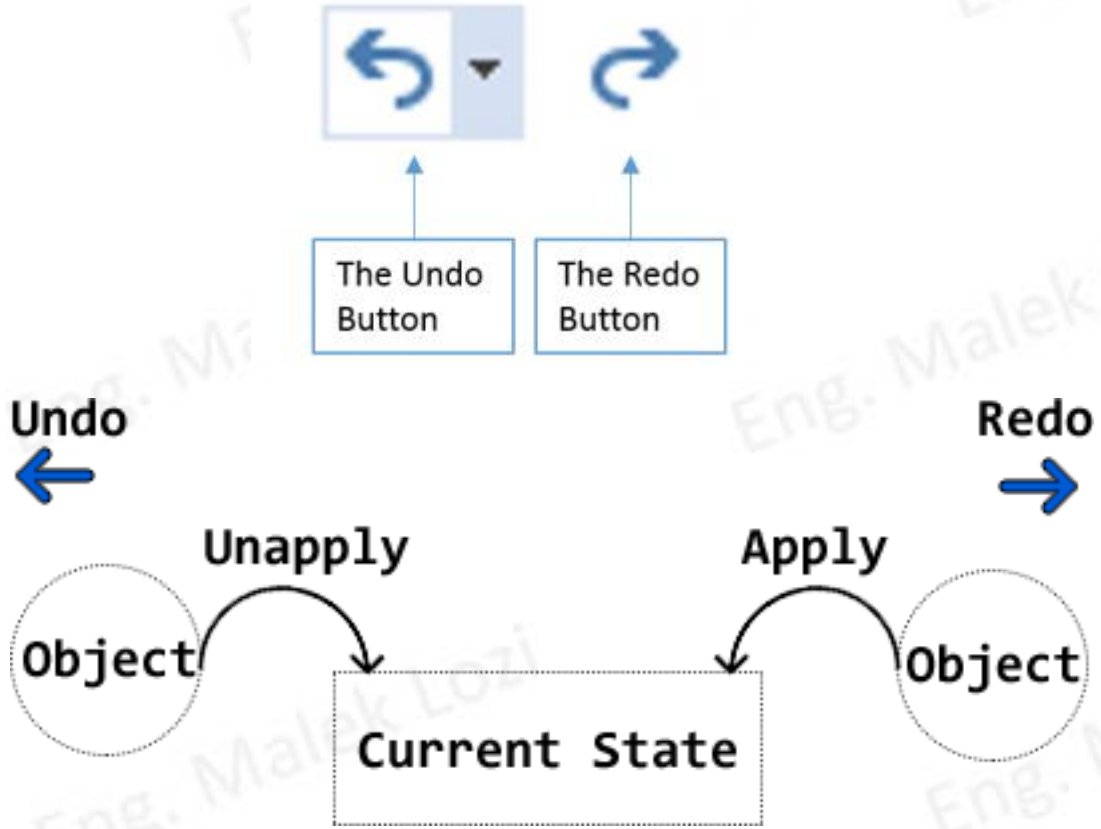
Top() ⇒ 5
IsEmpty() ⇒ false

Eng. Malek Lozi

Stack Applications (1/2)

- Real applications of Stack.

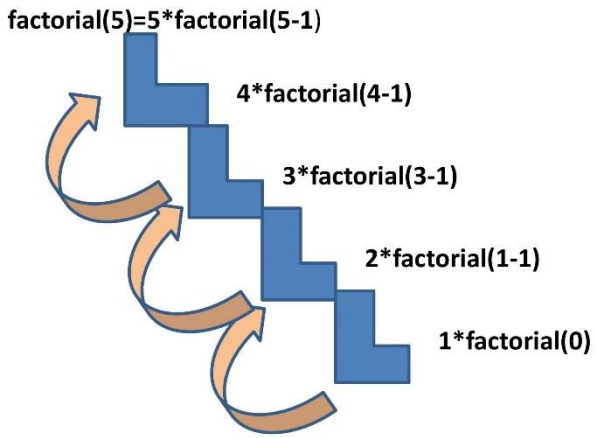
1- Undo operation in a text editor.



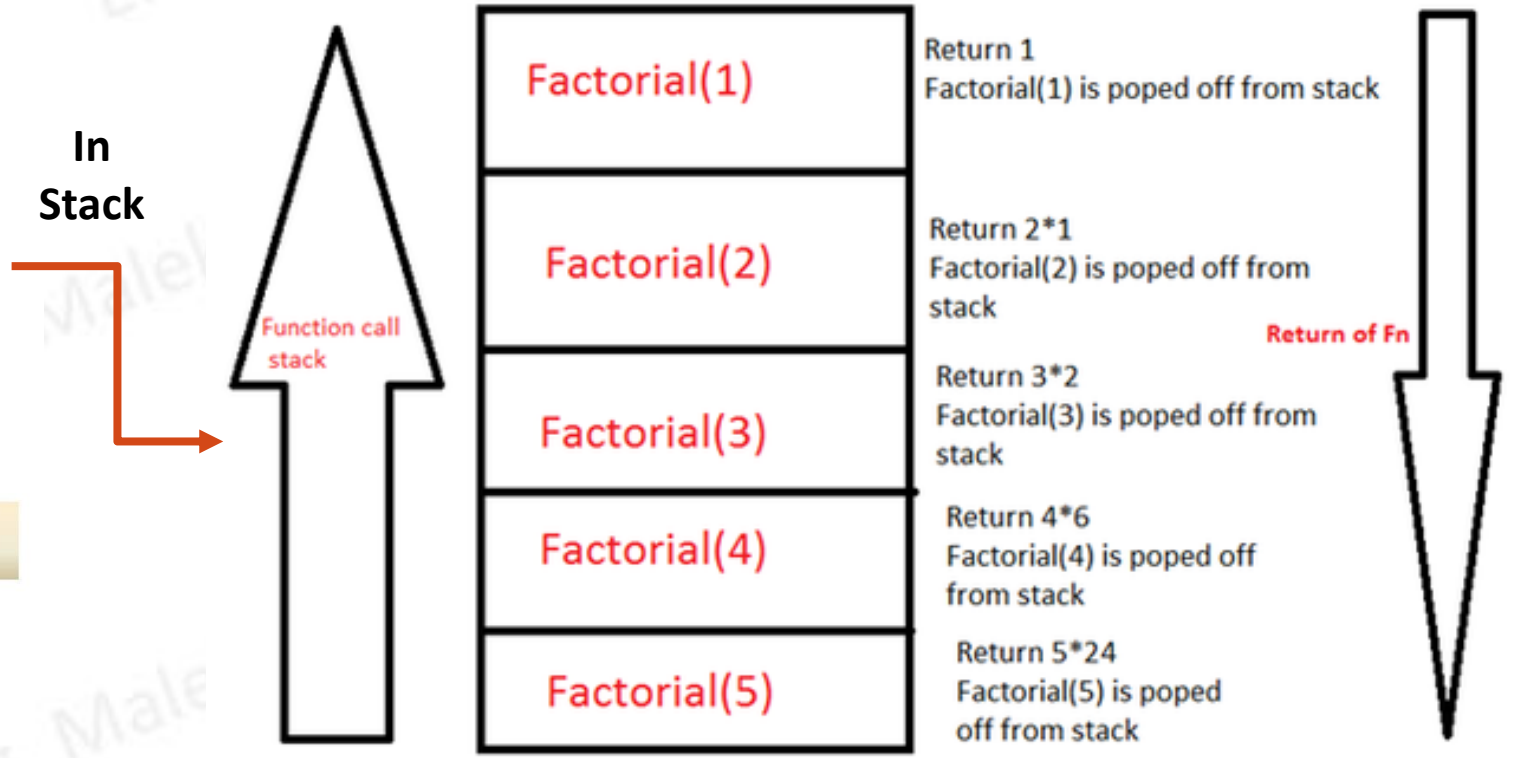
Stack Applications (2/2)

- Real applications of Stack.

  2- Function calls / recursion.



** Recursion expands when n>0*
** Its starts winding up on hitting the base*

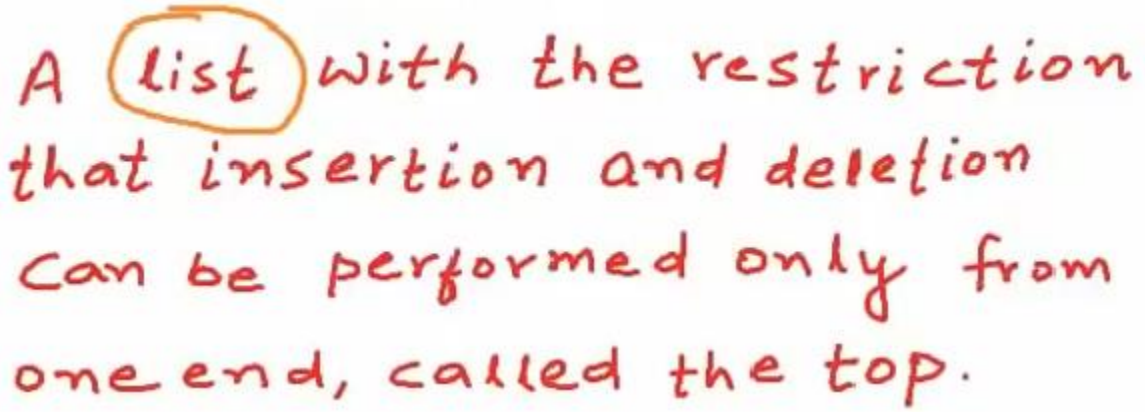


Think about other applications for a Stack?

Eng. Malek Lozi

Stack Implementation (1/2)

- From the definition of the Stack:



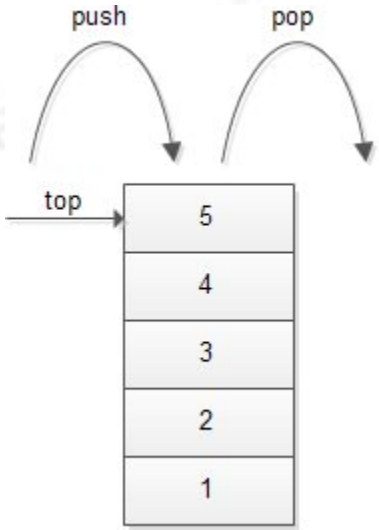
- We can add only this one extra constraint to an implementation of a List, then we can get a Stack.
- The constraint is that insertion and deletion must be performed only from one end.

Stack Implementation (2/2)

- There are two popular ways of creating lists.
  - **Arrays.**
  - **Linked Lists.**
- We can use any of them to create a Stack.

Eng. Malek Lozi

Stack Implementation Using Array

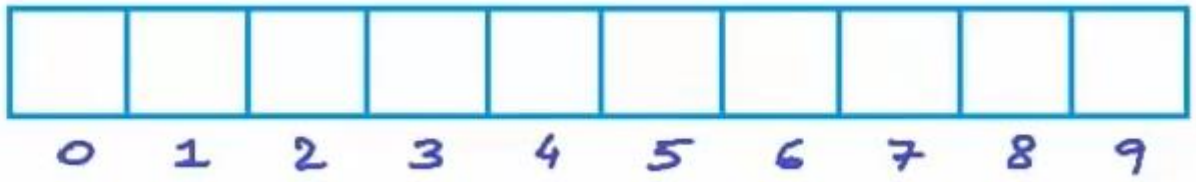


Eng. Malek Lozi

Stack Implementation Using Array (1/19)

- We want to create a Stack of integers.
- First, we can create an array of integers.
  - Array of Ten integers.

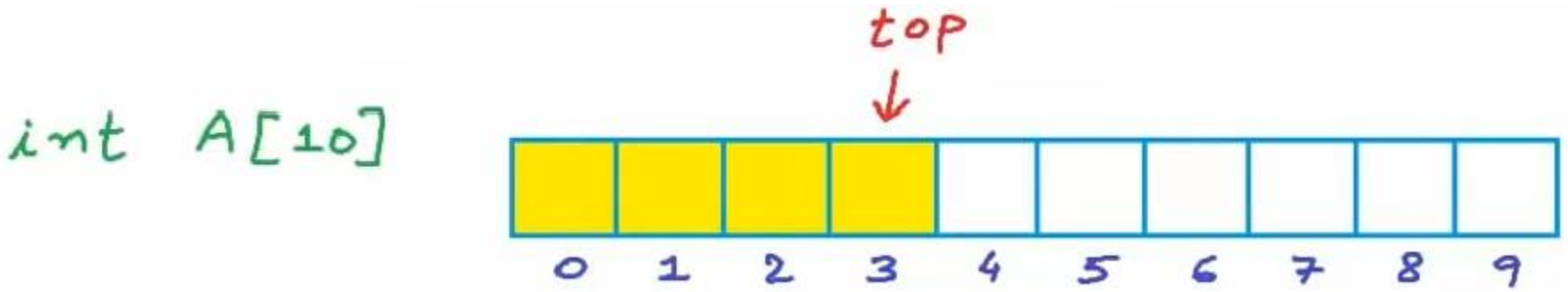
```
int A[10]
```



Eng. Malek Lozi

Stack Implementation Using Array (2/19)

- At any point, some part of this array starting index Zero to index marked as **top** will be our stack.



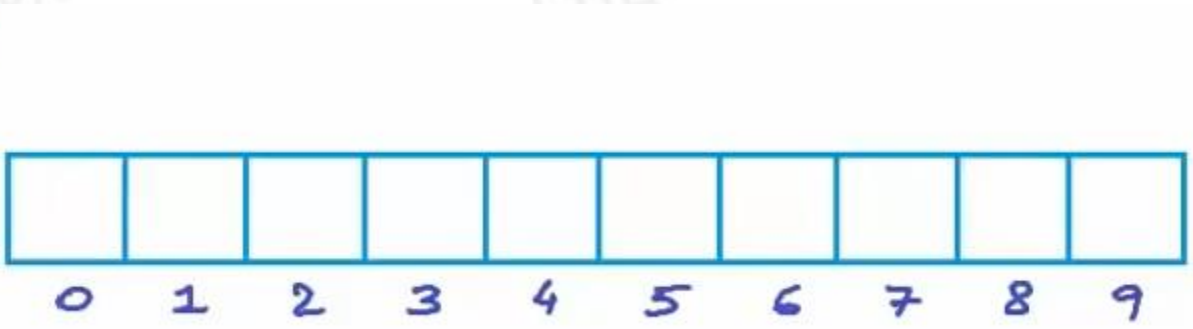
- We can create a variable named **top** to store the index of top of the stack.

Eng. Malek Lozi

Stack Implementation Using Array (3/19)

- For an empty stack top will set to -1.

```
int A[10]
top ← -1   //empty stack
```



Eng. Malek Lozi

Stack Implementation Using Array (4/19)

- For Push operation, we can write a function named **push** that will take an integer x as an argument.

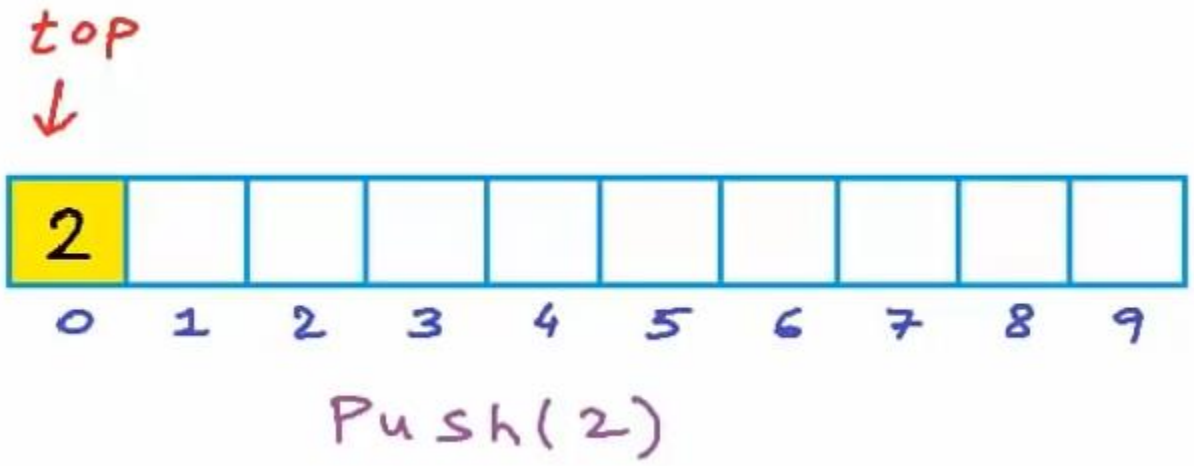
```
int  A[10]
top ← -1   //empty stack

Push(x)
{
   top ← top+1
   A[top] ← x
}
```

Eng. Malek Lozi

Stack Implementation Using Array (5/19)

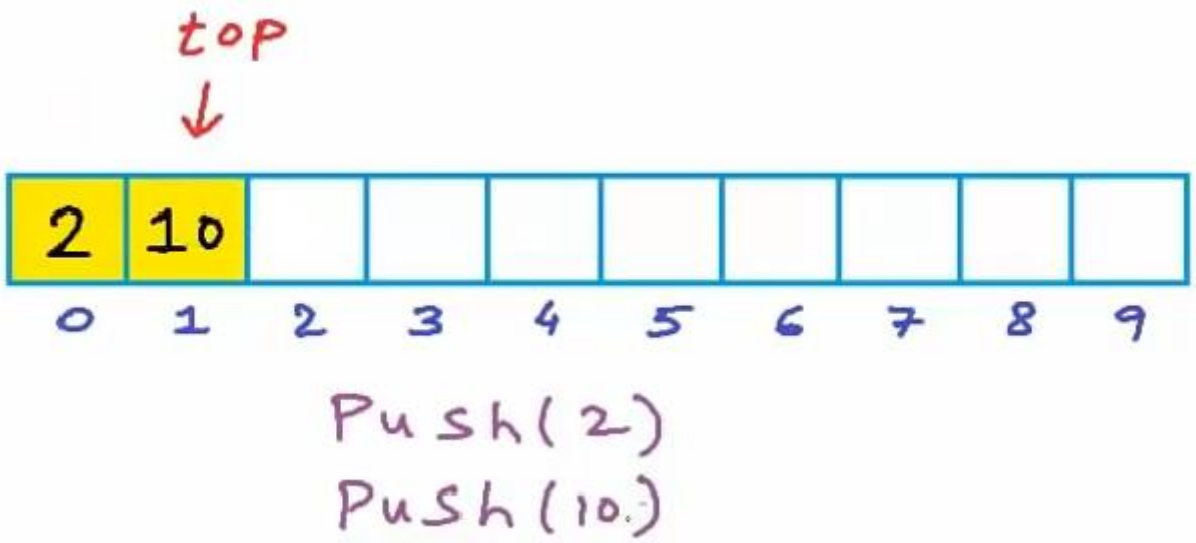
- The stack is still empty.
- Let us insert data into the stack.
- We will call the **push** function.



Eng. Malek Lozi

Stack Implementation Using Array (6/19)

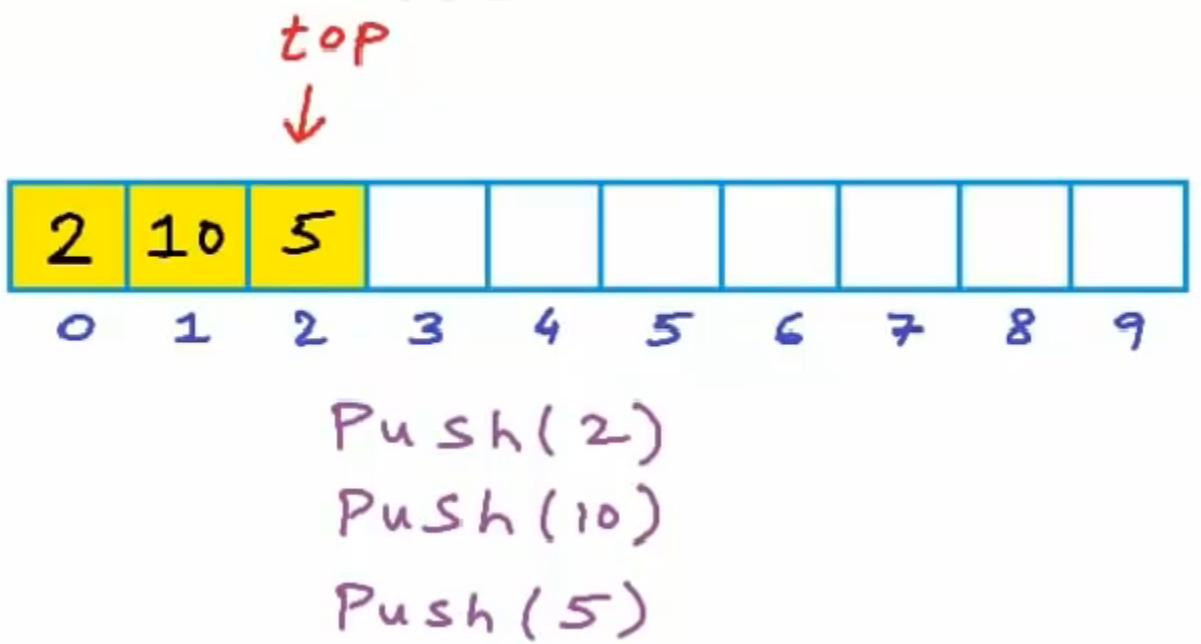
- We will push another item onto the stack.



Eng. Malek Lozi

Stack Implementation Using Array (7/19)

- With each push, the stack will expand towards higher indices in the array.



Eng. Malek Lozi

Stack Implementation Using Array (8/19)

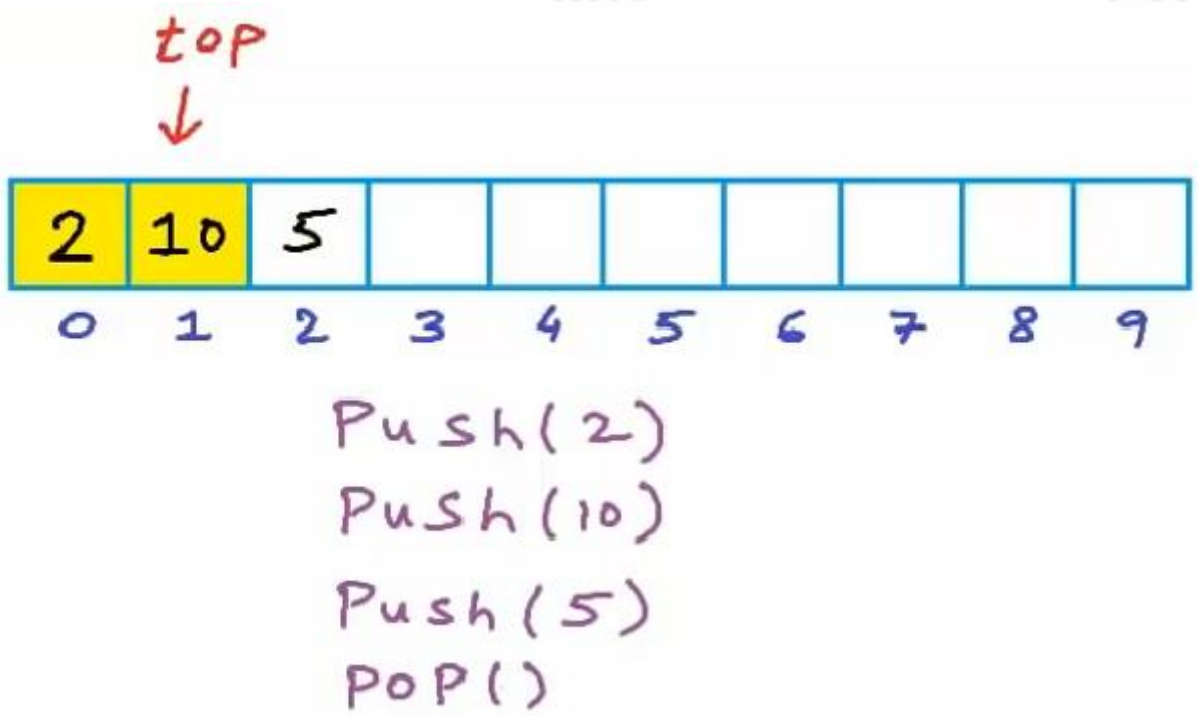
- Now, we will write a **pop** function, to pop an element from the stack.
- All we need to do is decrementing the top by One.

```
PoP()
{
   top ← top - 1
}
```

Eng. Malek Lozi

Stack Implementation Using Array (9/19)

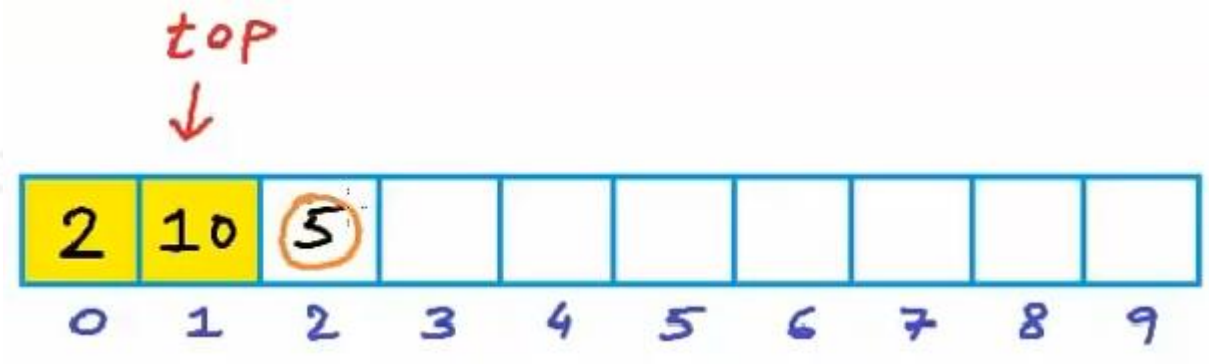
- Let us make a call to the **pop** function.



Eng. Malek Lozi

Stack Implementation Using Array (10/19)

- The cells in Yellow in the figures are part of the stack.
- We don't need to reset this cell before or after popping.

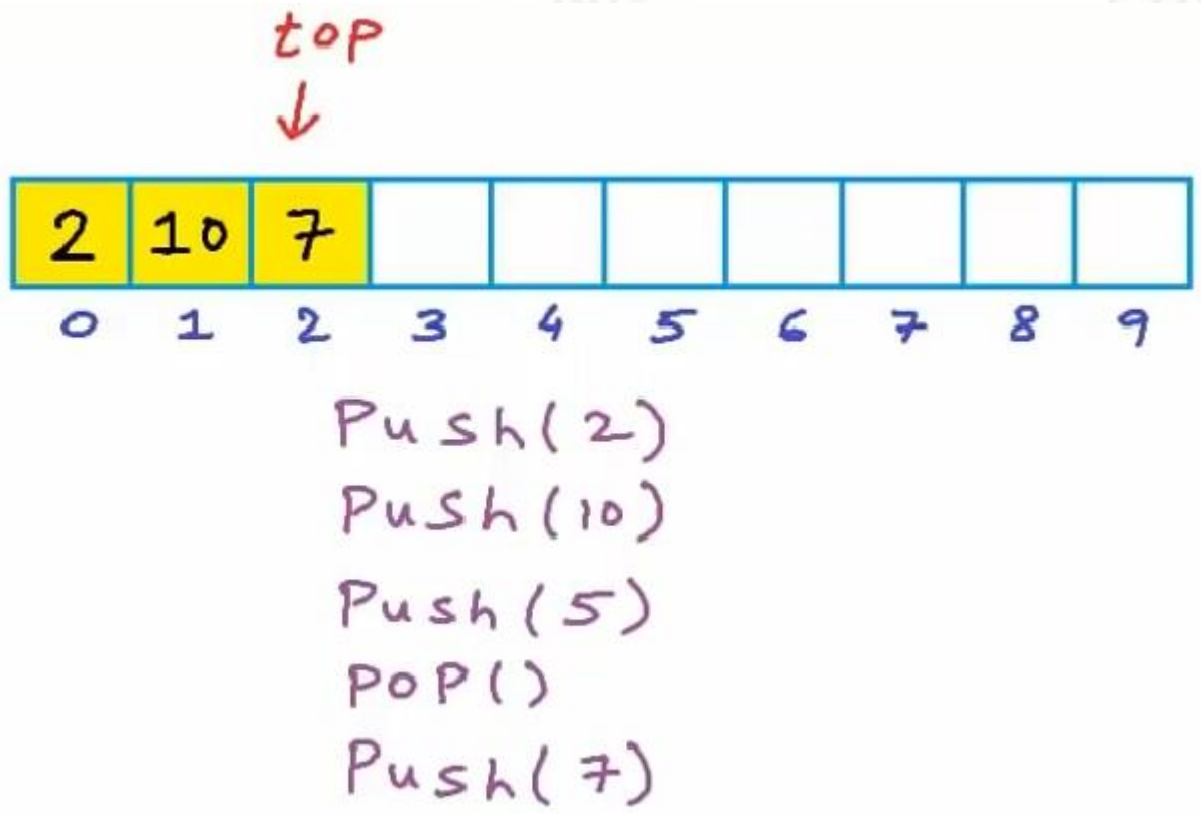


- If a cell is not part of the stack anymore, we don't care what garbage lies there.
  - Next time when we will push, we will modify it anyway.

Eng. Malek Lozi

Stack Implementation Using Array (11/19)

- After the pop operation, let us perform a push operation again.



Stack Implementation Using Array (12/19)

- The Two function that we have written (push and pop), will take a constant time.
  - We have simple operations in these two functions.
  - The execution time will not depend on the size of the stack.
- When we defined a Stack ADT, we had said that all the operations must take constant time.
  - The time complexity should be O(1).

Eng. Malek Lozi

Stack Implementation Using Array (13/19)

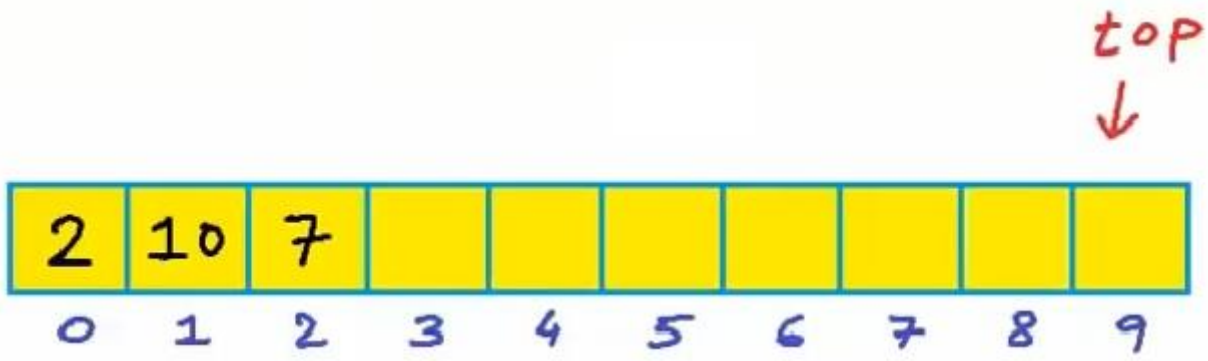
```
Push(x)
{
   top ← top+1
   A[top] ← x
}
PoP()
{
   top ← top - 1
}
```

Constant time
or
O(1)

Eng. Malek Lozi

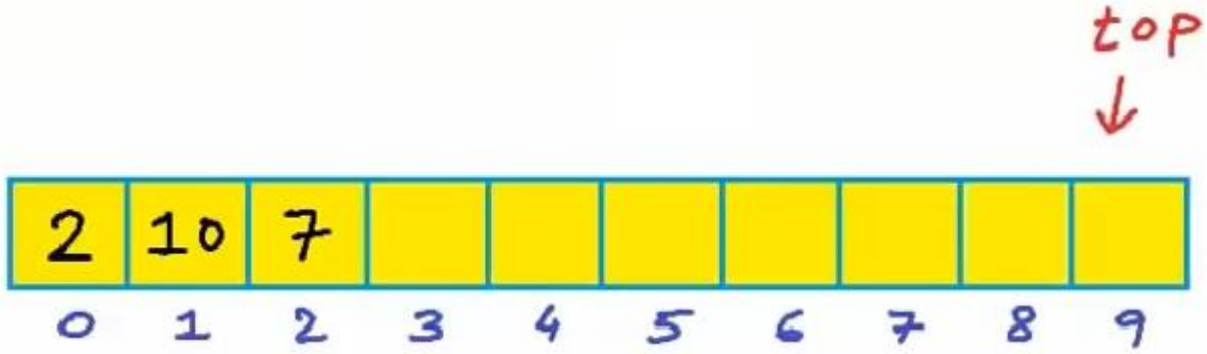
Stack Implementation Using Array (14/19)

- One important thing here, we can push onto the stack only till array is not exhausted (all elements are used).
- We can have a situation where stack would consume the whole array.
  - Top will equal the highest index of the array.



Eng. Malek Lozi

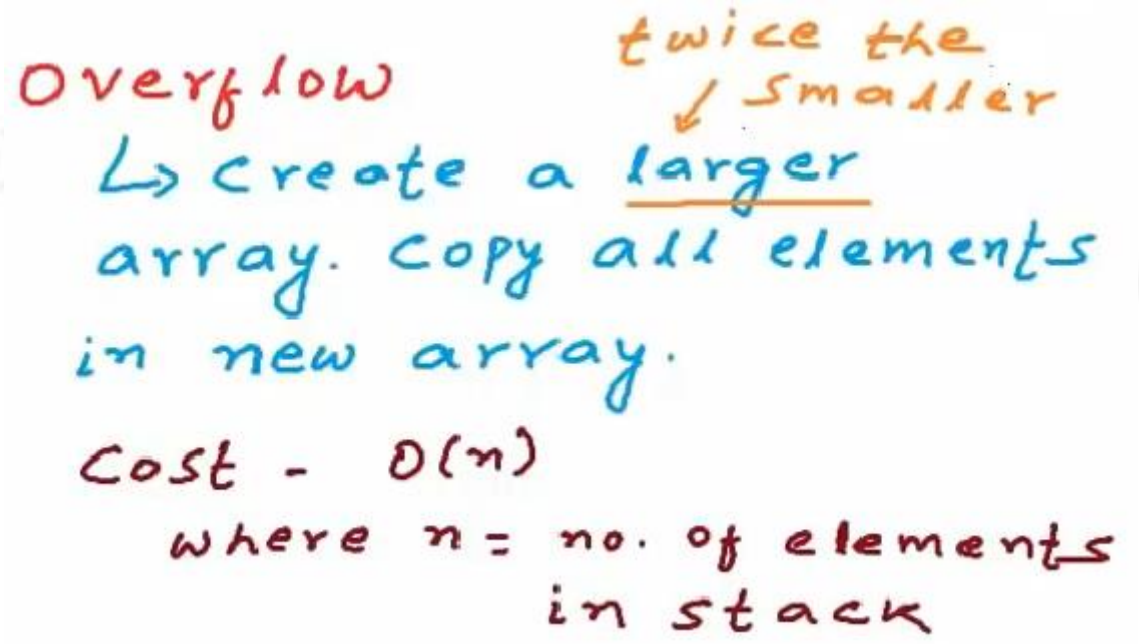
Stack Implementation Using Array (15/19)



- A further push will not be possible because it will result an overflow.
- This is one limitation with array-based implementation.
- We must handle overflow in our implementation.
  - In push function, check whether array is exhausted or not.
    - A) Throw an error in case of an overflow, push operation will not succeed.
    - B) We can use the concept of dynamic array.

Stack Implementation Using Array (16/19)

- If we used the dynamic array, the time complexity of push with this strategy:



Stack Implementation Using Array (17/19)

- The time complexity of push operation with this strategy:

Push - O(1) - best case
O(n) - Worst case

Eng. Malek Lozi

Stack Implementation Using Array (18/19)

- We still have two more operations in the definition of stack ADT:
  - Top.
  - IsEmpty.
- Top will simply return the element at the top:

```
Top()
{
return A[top]
}
```

Eng. Malek Lozi

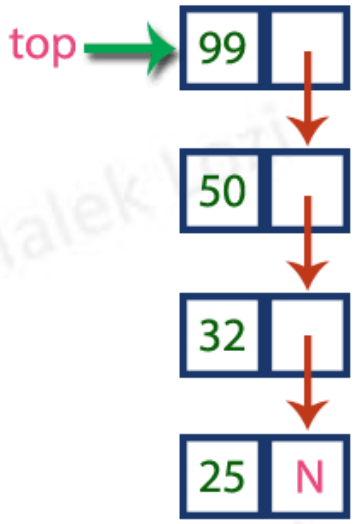
Stack Implementation Using Array (19/19)

- To verify the stack IsEmpty or not:

```
IsEmpty()
{
   if (top==-1)
      return true
   else
      return false
}
```

Eng. Malek Lozi

Stack Implementation Using Linked List



Eng. Malek Lozi

Stack Implementation Using Linked List (1/7)

- Remember:

Stack ADT (LIFO)

A list with the restriction that insertion and deletion can be performed only from one end, called the top.

Operations

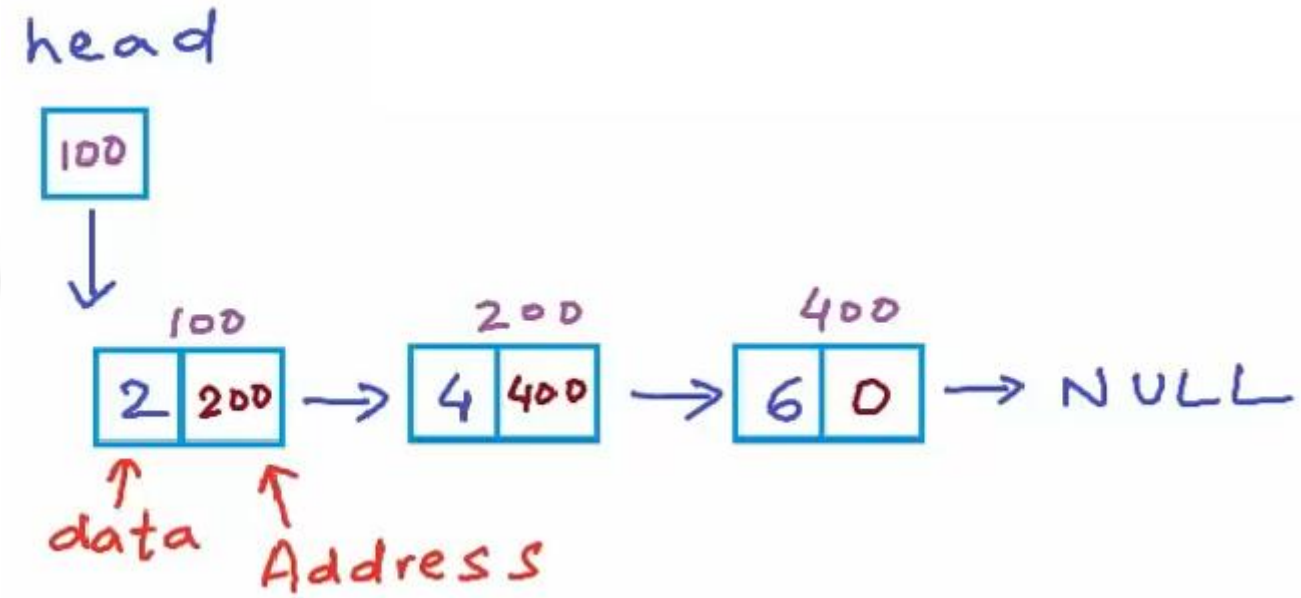
(1) Push(x)
(2) Pop()
(3) Top()
(4) IsEmpty()

Constant time or O(1)

Eng. Malek Lozi

Stack Implementation Using Linked List (2/7)

- We have a linked list of integers here:



Eng. Malek Lozi

Stack Implementation Using Linked List (3/7)

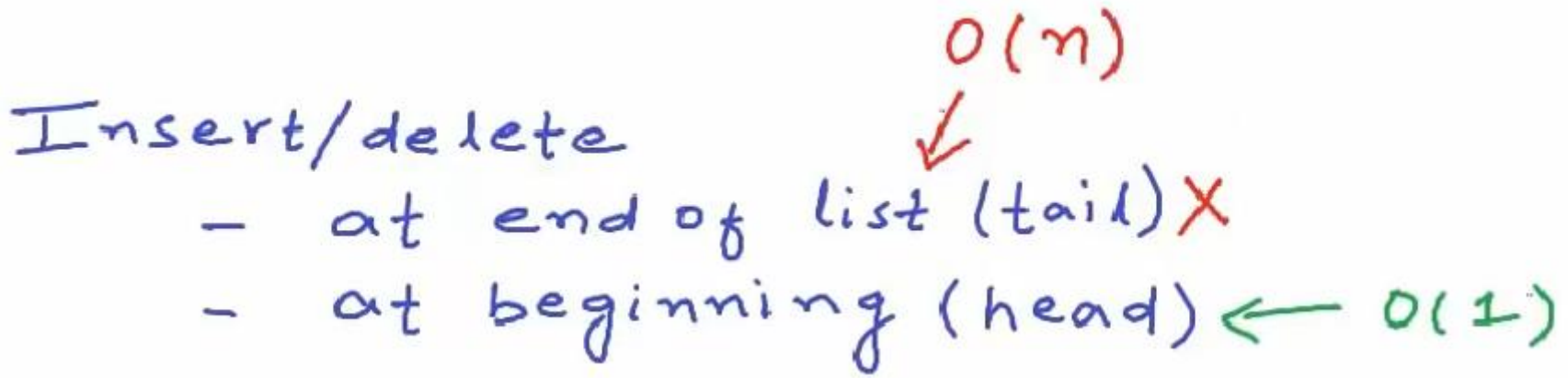
- Unlike arrays, linked lists are not fixed size and elements are not stored in a continues block of memory.
- To use linked list for a stack, we want the insertion and deletion must always happen from the same end.
- We have Two options:

Insert/delete
- at end (tail)
- at beginning (head)

Eng. Malek Lozi

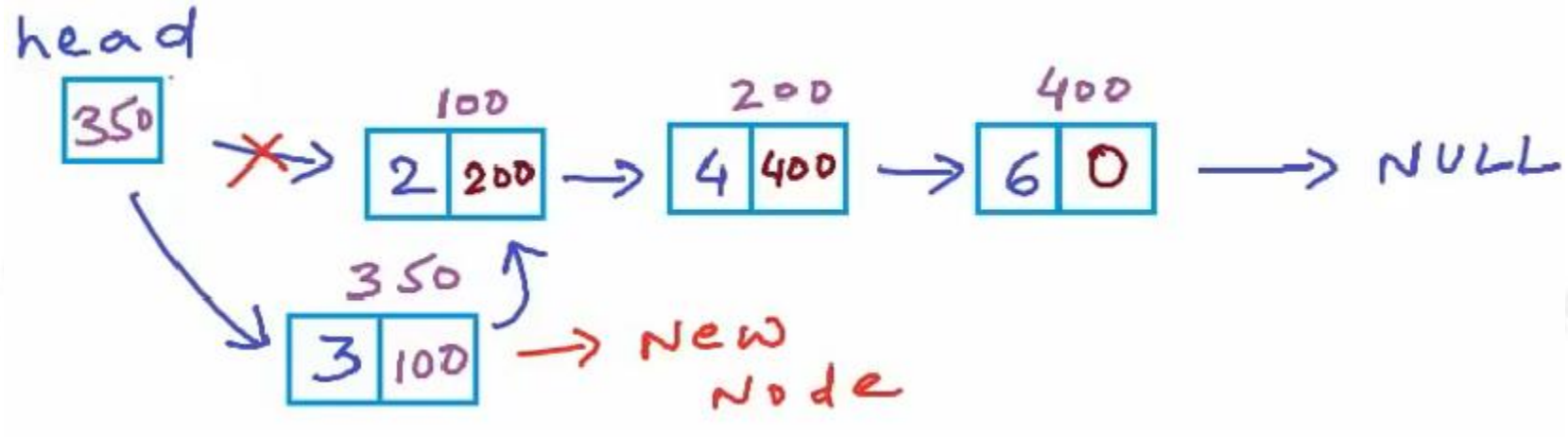
Stack Implementation Using Linked List (4/7)

- The cost of insertion at the end of a linked list is O(n).
- It is not an option for us because we will not be able to do push and pop in constant time.



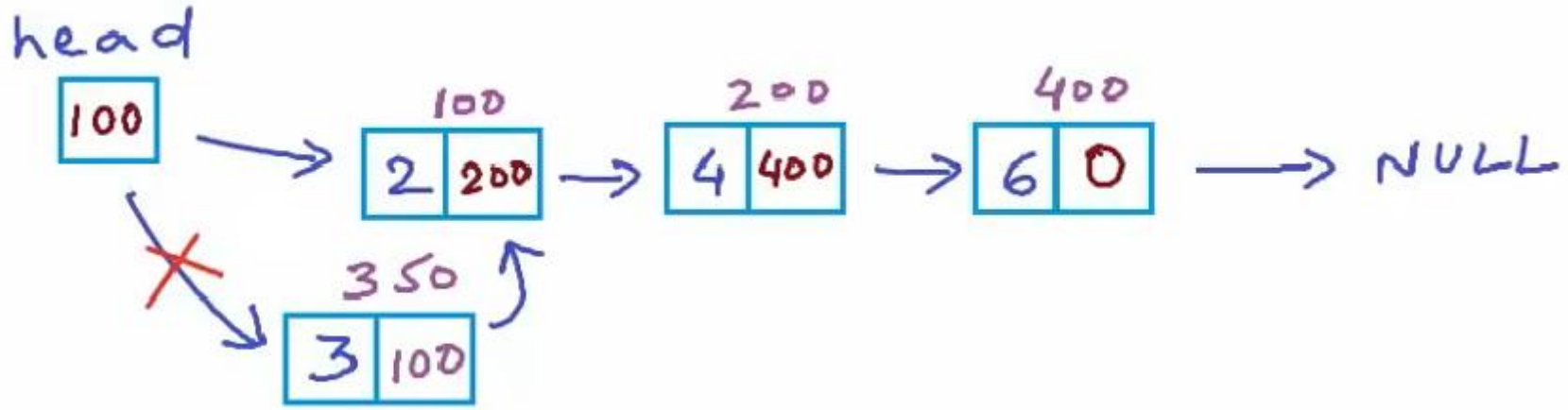
Stack Implementation Using Linked List (5/7)

- To insert a node at the beginning:
  - Create a new node.
  - Build two links.
    - From the new node to the first node.
    - From head to the new node.



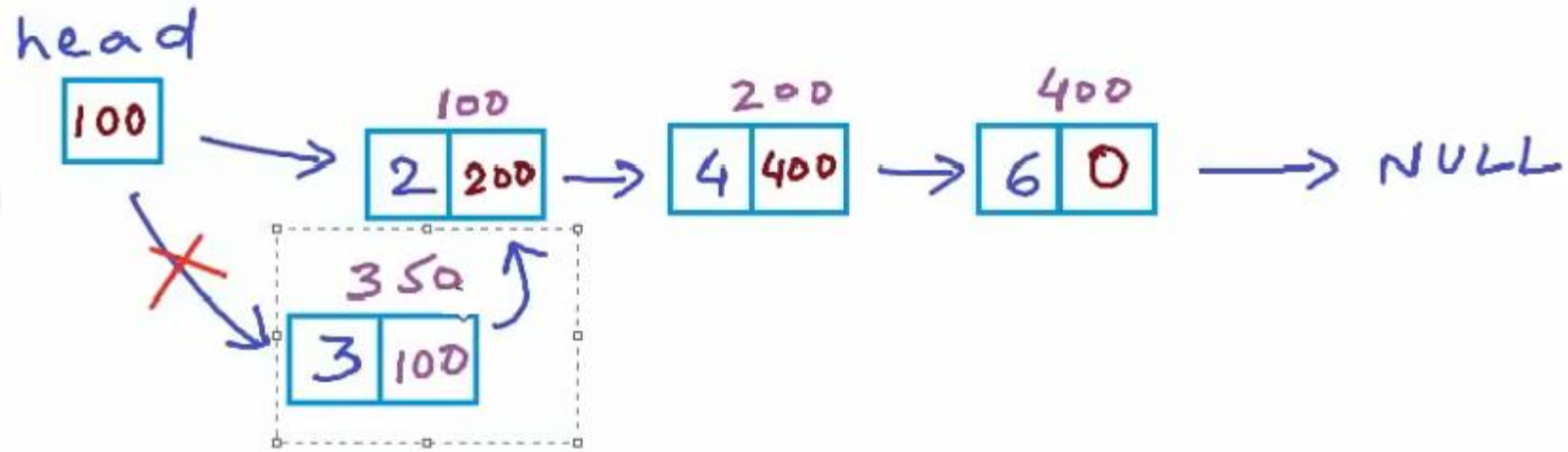
Stack Implementation Using Linked List (6/7)

- To delete a node from the beginning:
  - Cut the link from head to the first node.
  - Build a new link from head to the second Node.
- This means resetting the reference in the head.



Stack Implementation Using Linked List (7/7)

- Do we need to free the memory allocated to the deleted node?



Eng. Malek Lozi

Any Questions???…

Eng. Malek Lozi