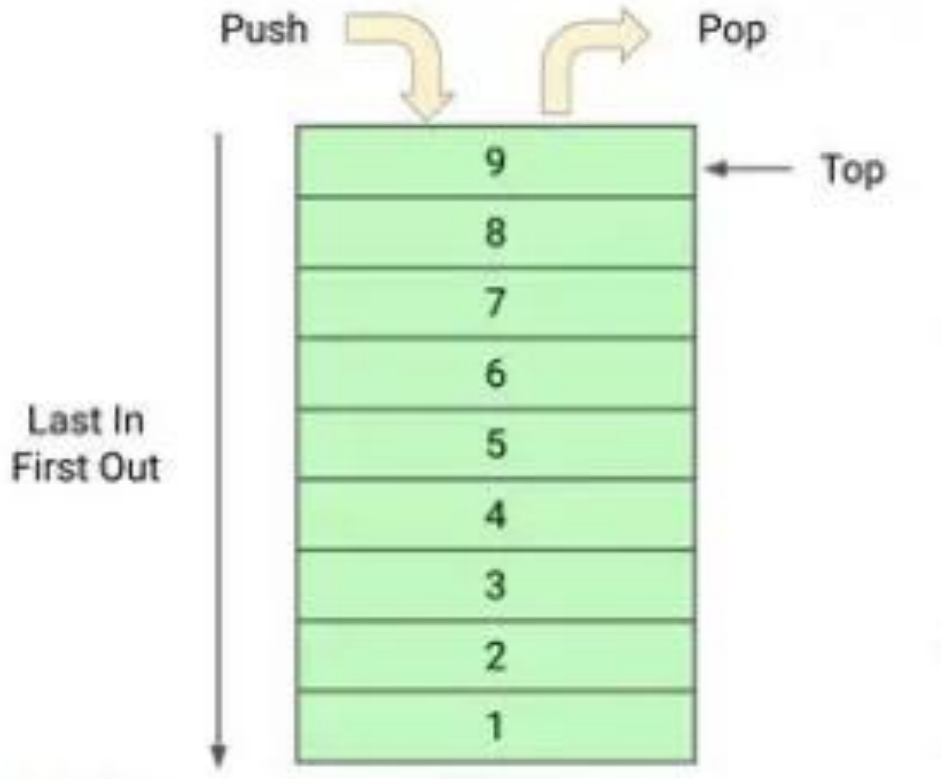
Kavita K. Bharti
Asst. Prof.
Department of Computer Science

Stack

Introduction

A **Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO(Last In First Out)** or **FILO(First In Last Out)**. **LIFO** implies that the element that is inserted last, comes out first.
It behaves like a stack of plates, where the last plate added is the first one to be removed. **Think of it this way:**

- Pushing an element onto the stack is like adding a new plate on top.
- Popping an element removes the top plate from the stack.



Stack

Implementation

Stack is implemented in memory in two ways

1. Implement Stack using Array
2. Implement a stack using singly linked list

Implement Stack using Array

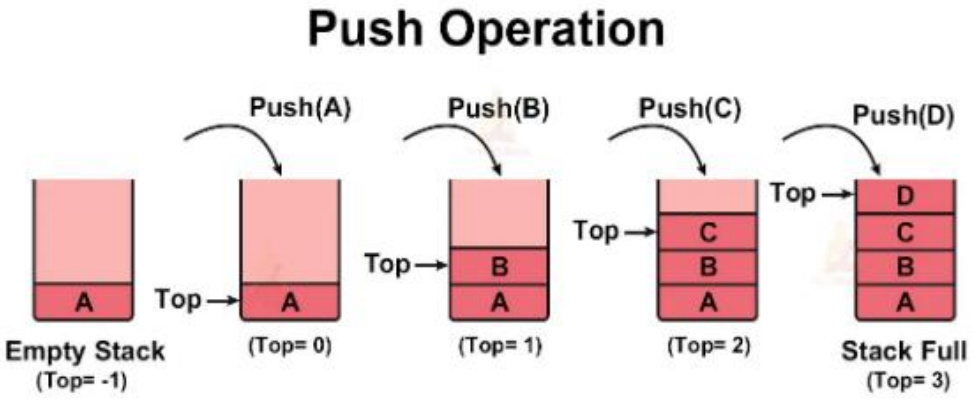
An stack can be implemented in memory using array. Use an array to store stack elements. Maintain a variable top to indicate the current top of the stack.

Algorithm for push operation

PUSH(stack, item,top,max) : here stack is the name of the stack in which we want to insert element. The element that we want to store is stored in item, top indicates the current top element in the stack and max indicates the maximum number of elements that can be stored in stack.

1. If top = max then "overflow and exit"
2. top = top +1
3. stack[top] = item
4. exit

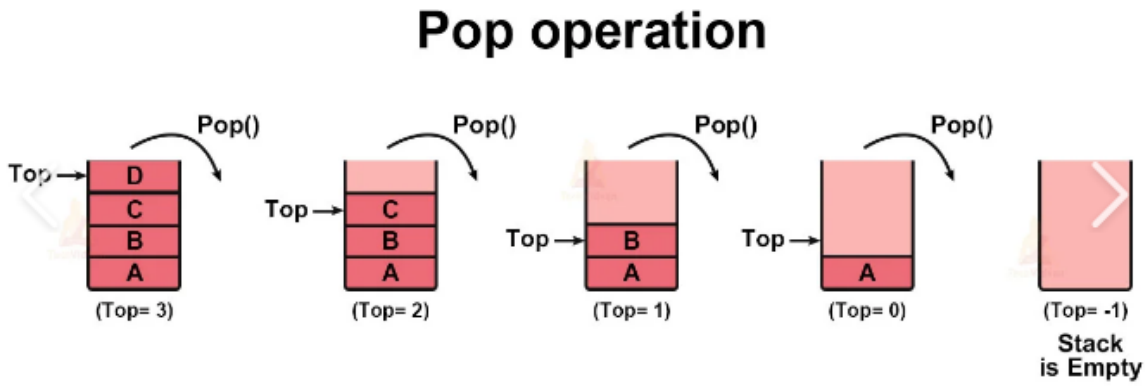
The following figure shows push operation



Algorithm for push operation

POP(stack, item,top) : here stack is the name of the stack in which we want to insert element. The element that we delete is stored item, top indicates the current top element in the stack

1. If top = -1 then “underflow and exit”
2. item = stack[top]
3. top = top - 1
4. exit



Advantages of Array Implementation

- Simple and fast.
- Direct access using index.
- Efficient for small, fixed-size stacks.

Limitations

- **Fixed size**: Can't grow beyond declared size (unless using dynamic arrays).
- Risk of **overflow** if size isn't managed.
- Not memory-efficient if size is overestimated.