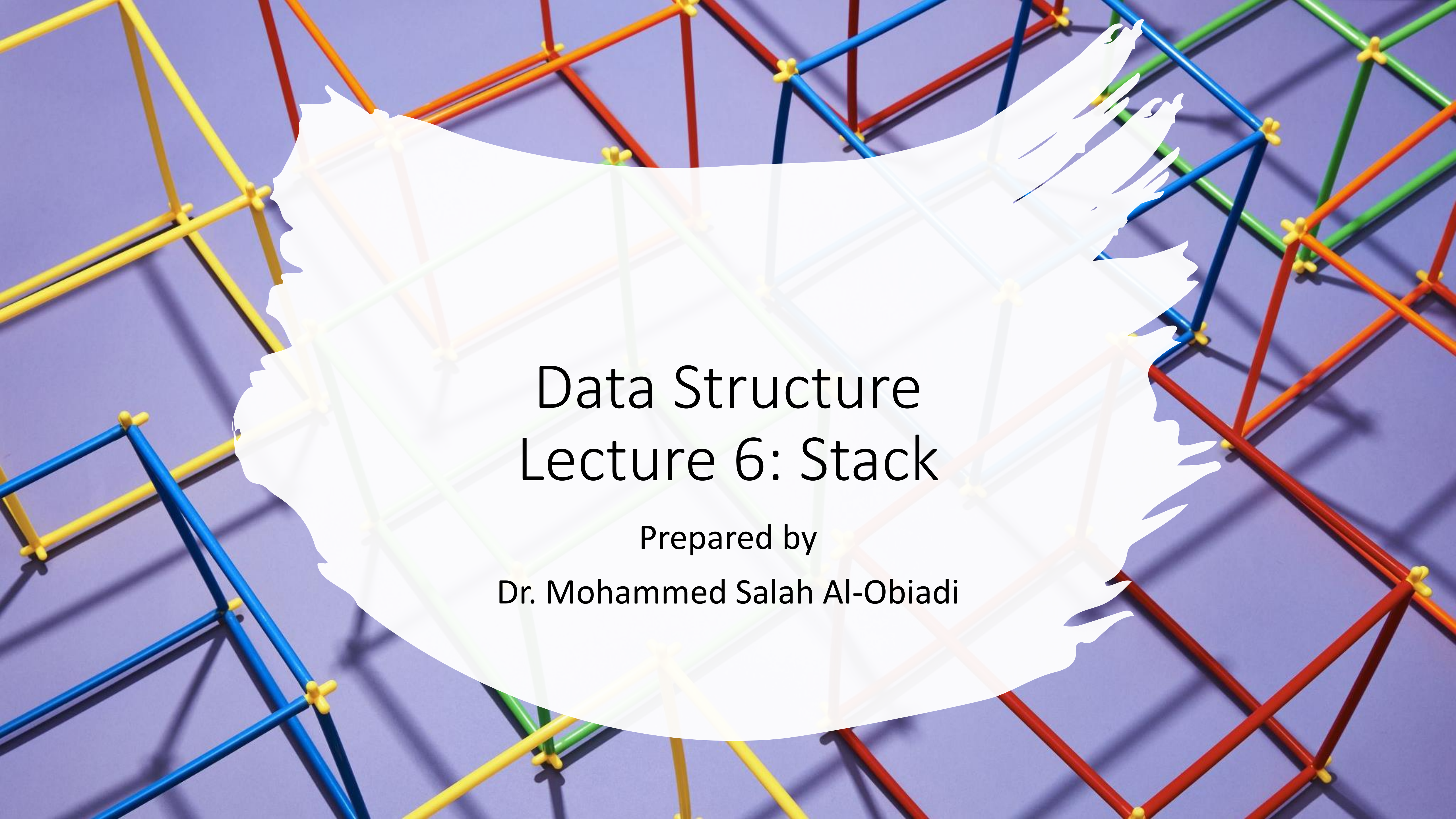


Data Structure
Lecture 6: Stack

Prepared by

Dr. Mohammed Salah Al-Obiadi

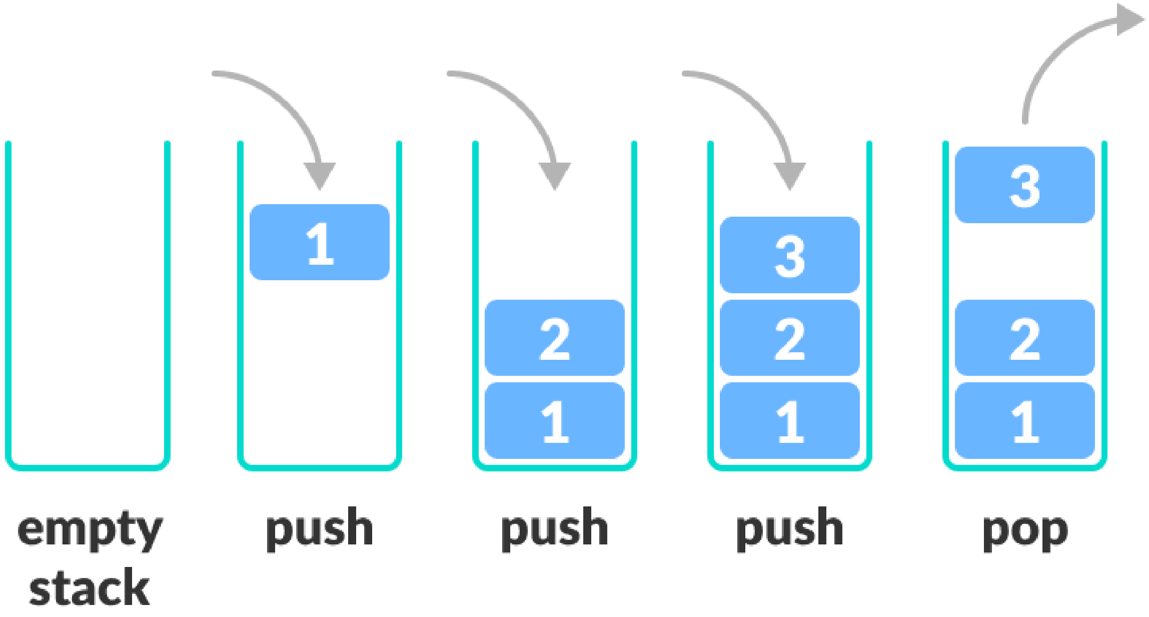
STACK

- Stack is a linear data structure.
- Follows the principle of LIFO **(Last in First Out).**
- Any data structure use the LIFO principle, it can be called as STACK.

Operations Performed With STACK

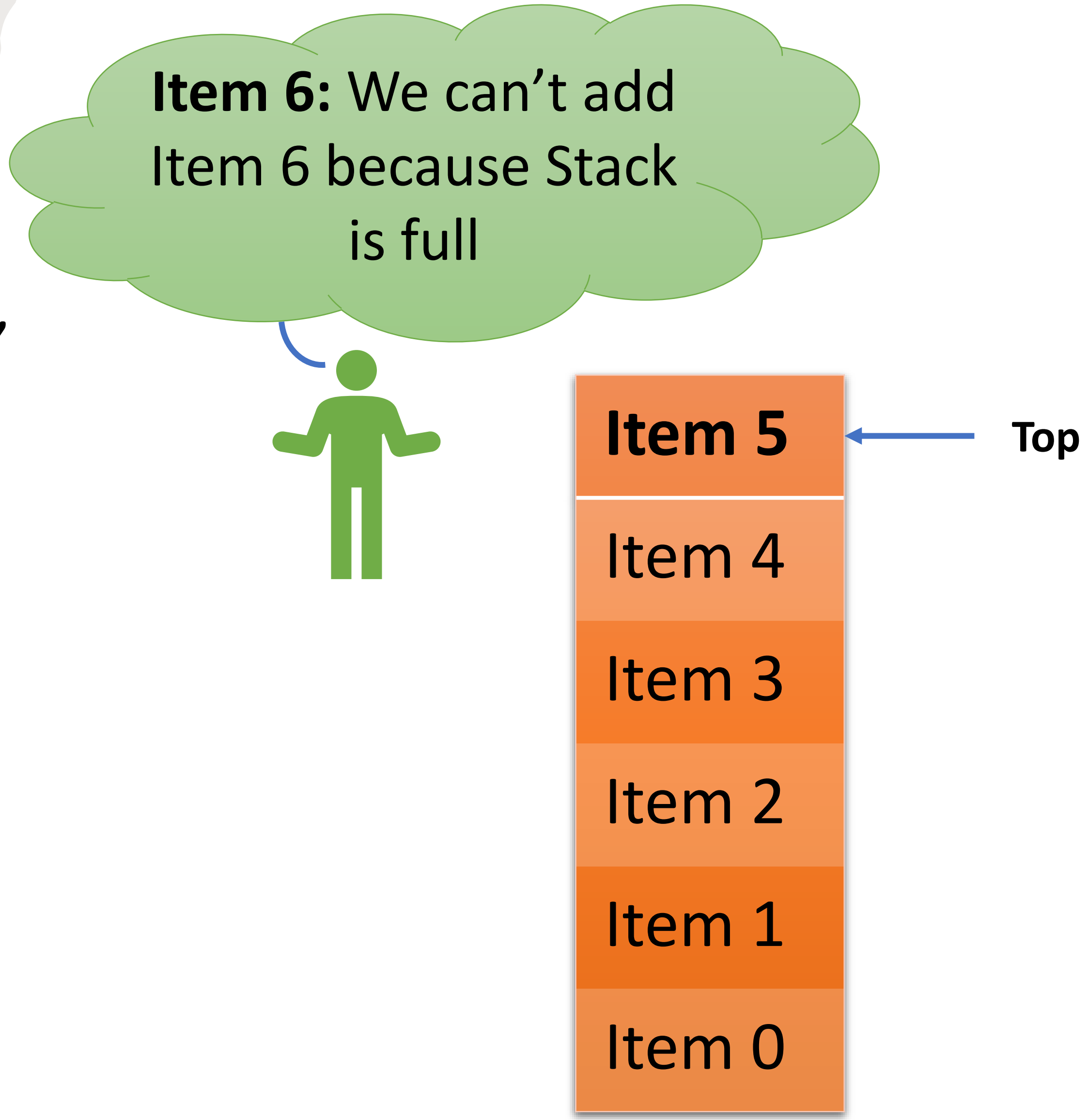
1- PUSH: which adds an element to the collection.

2- POP: which removes the most recently added element.



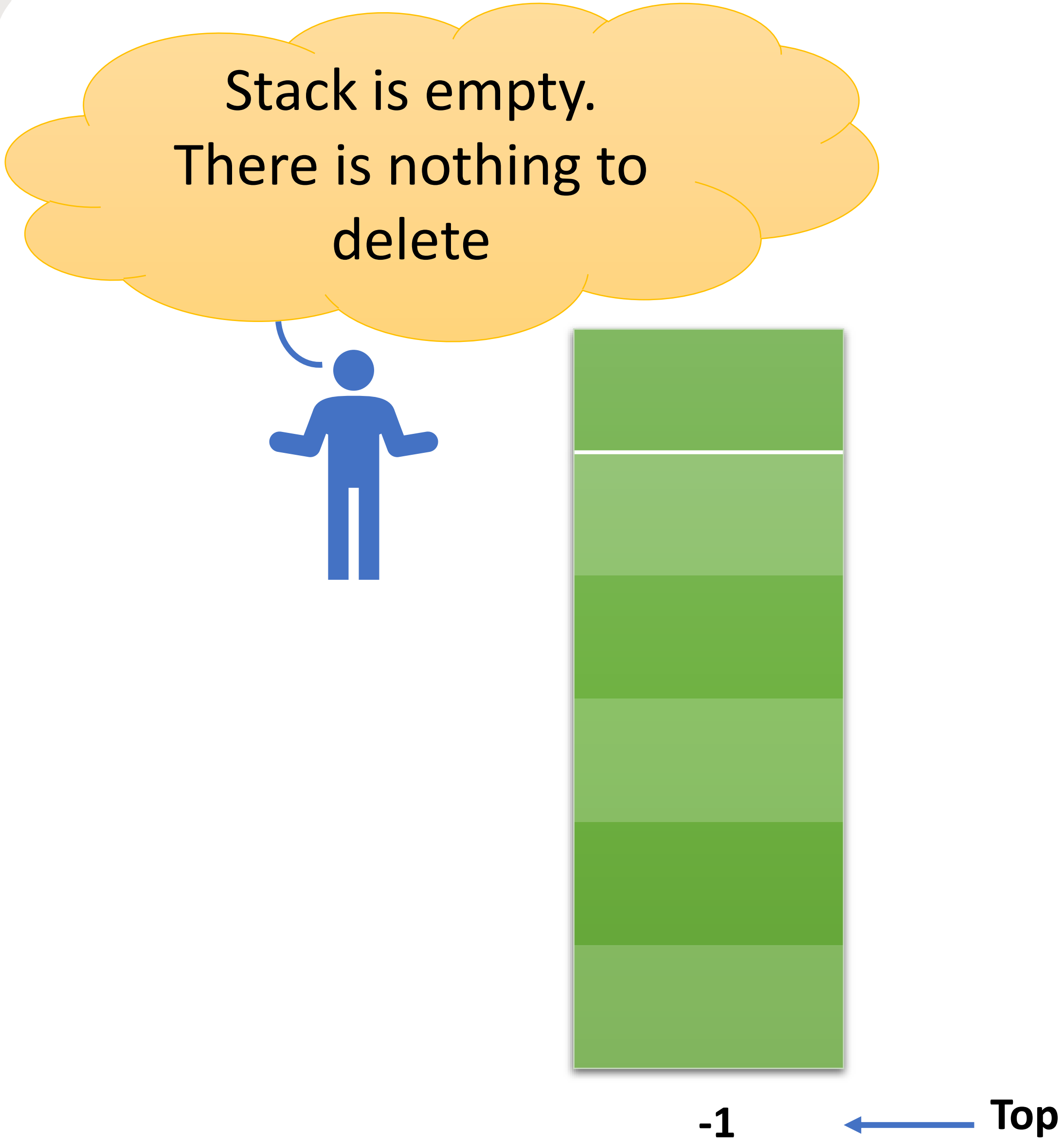
Overflow conditions

- During the **PUSH** (add) operation, we have to check the condition for overflow
- Condition for OVERFLOW
    - Top = size −1 (for the STACK starts with 0)
- Example of stack of size 6.
    - Now the stack has 6 items so we can't add any item.



Underflow conditions

- During the **POP** (delete) operation, we have to check the condition for underflow.
- Condition for
  - Top =-1 (for the STACK starts with 0)
- Example of stack of size 6.
  - Now the stack is empty and Top=-1, so we can't remove any item.



EXAMPLES

STACK[5]

0	1	2	3	4

top = −1 (CONDITION FOR EMPTY STACK)

PUSH(5)

5				
0	1	2	3	4

top = 0

PUSH(25)

5	25			
0	1	2	3	4

top = 1

PUSH(53)

5	25	53		
0	1	2	3	4

top = 2

PUSH(78)

5	25	53	78	
0	1	2	3	4

top = 3

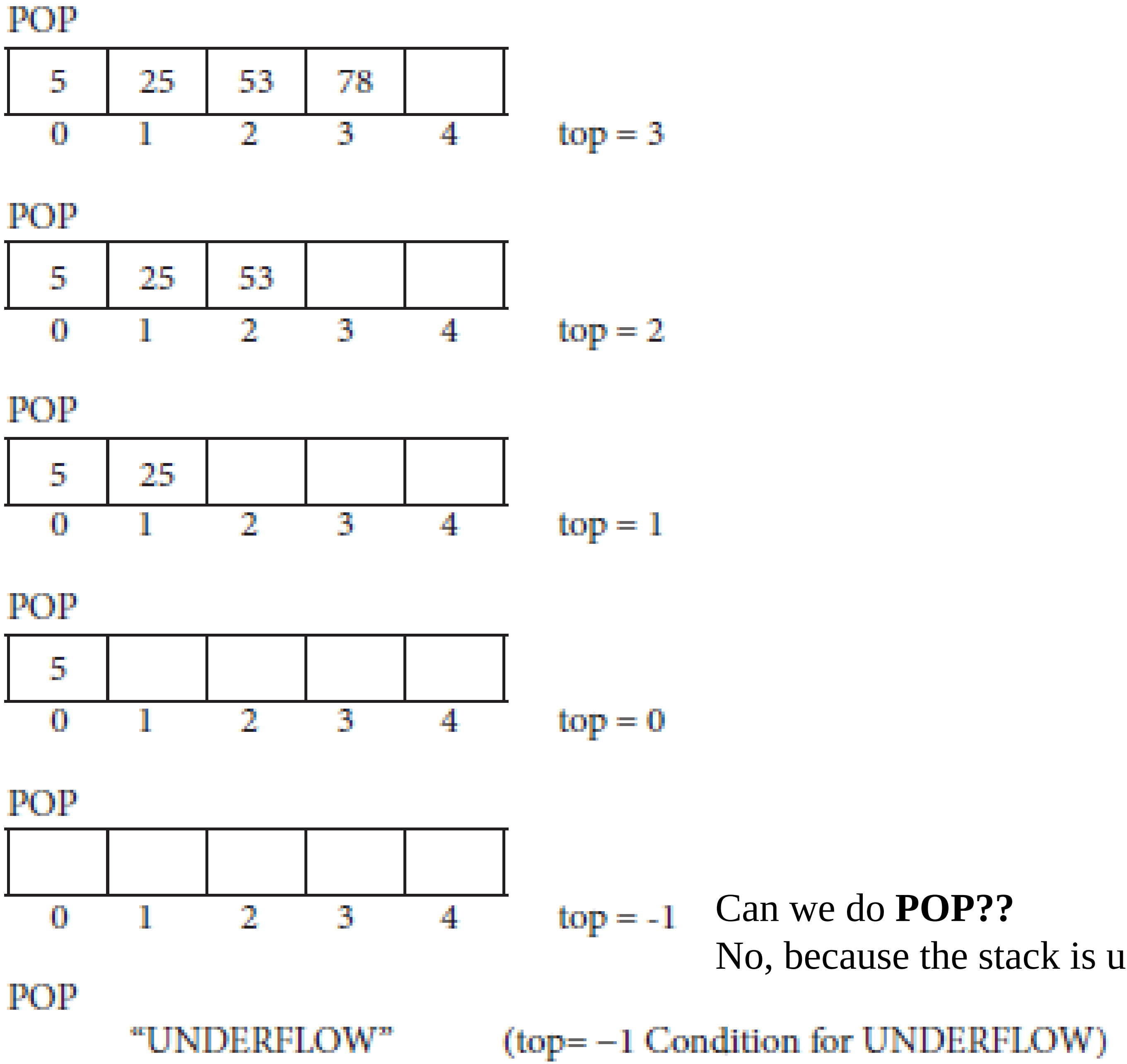
PUSH(99)

5	25	53	78	99
0	1	2	3	4

top = 4

Can we do **PUSH(76)??**

No, because OVERFLOW (top = size −1 Condition for OVERFLOW)



POP

5	25	53	78		
0	1	2	3	4	top = 3

POP

5	25	53			
0	1	2	3	4	top = 2

POP

5	25				
0	1	2	3	4	top = 1

POP

5					
0	1	2	3	4	top = 0

POP

0	1	2	3	4	top = -1

Can we do **POP??**

No, because the stack is underflow (top = −1 Condition for underflow)

POP

"UNDERFLOW" (top= −1 Condition for UNDERFLOW)

Algorithm For Push Operation

```
PUSH(stack[size], no, top) [no is the number to insert] [top is the position of the stack]
    step-1 : if (top = size - 1) then :
                    write : “overflow”
                    return
    step-2 : top : = top +1
                stack[top] := no
    step-3 : return
```

Algorithm For POP Operation

pop(stack[size], top) [stack[size] is the stack] [top is the position of the stack]

```
Step-1 : if (top = - 1) then :
                   write : “underflow”
                   return
Step-2 : write : stack[top]
         top := top -1
Step-3 : return
```

Algorithm For Traverse Operation

```
Traverse(stack[size], top)
Step-1 : if (top = - 1) then :
             write : “stack is empty”
             return
Step-2 : set i : = 0
Step-3 : repeat for i = top to 0 by -1
               write : stack[i]
Step-4 : return
```

Algorithm For Update Operation

Can you do it?

Applications of STACK

1- Checking of the parenthesis of an expression

2- Reversing of a string

3- In Recursion

4- Evaluation of Expression