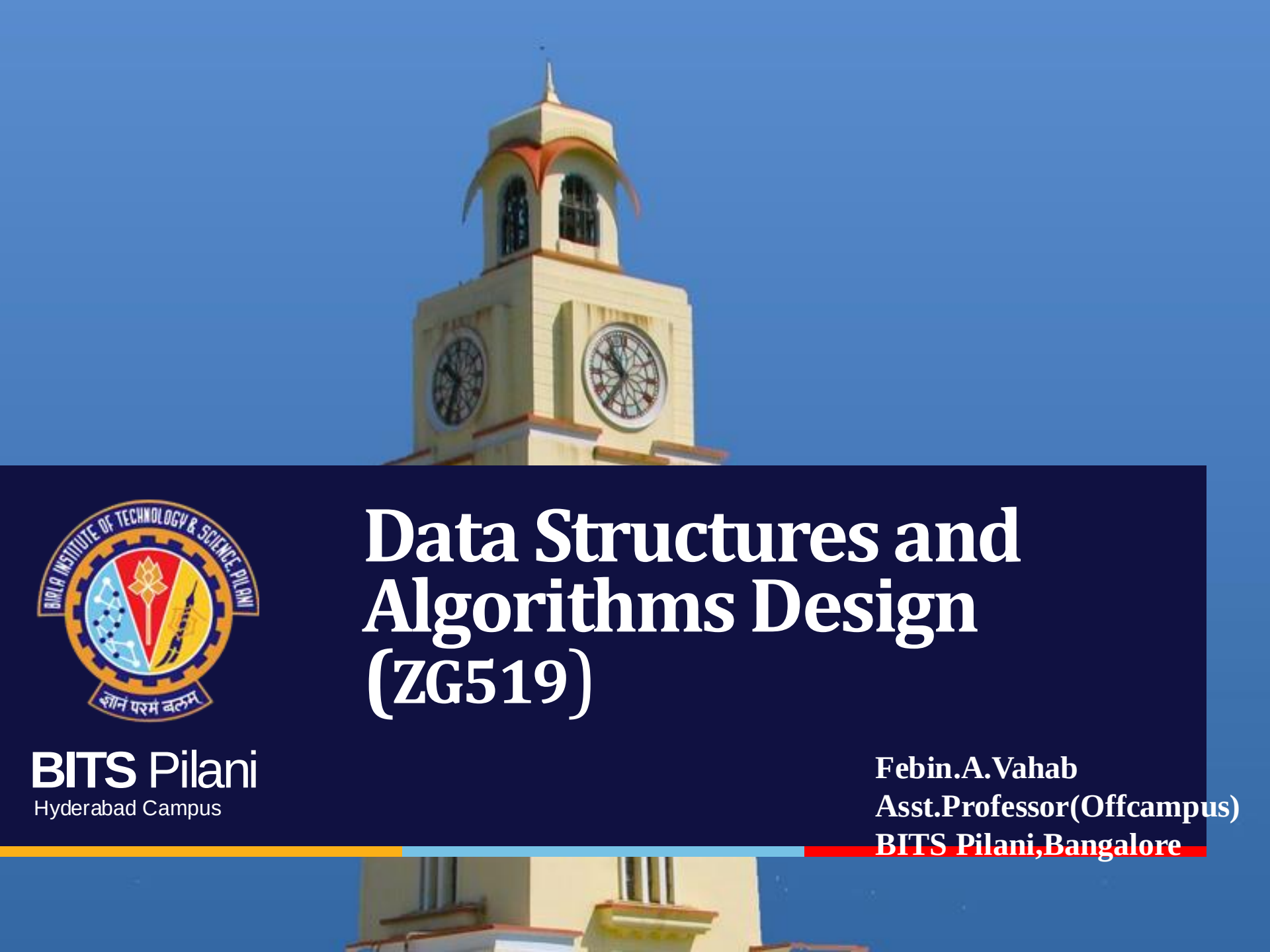


Data Structures and Algorithms Design (ZG519)

BITS Pilani
Hyderabad Campus

Febin.A.Vahab
Asst.Professor(Offcampus)
BITS Pilani,Bangalore

SESSION 3 -PLAN

innovate achieve lead

Online Sessions(#)	List of Topic Title	Text/Ref Book/external resource
3	Stack ADT and Implementation. Queue ADT and Implementation, Applications	T1: 2.1

Abstract Data Type

- In computer science, an **abstract data type** (**ADT**) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

Abstract Data Type

- A method for achieving abstraction for data structures and algorithms
- ADT = model + operations
- Describes **what** each operation does, but not **how** it does it
- An ADT is independent of its implementation

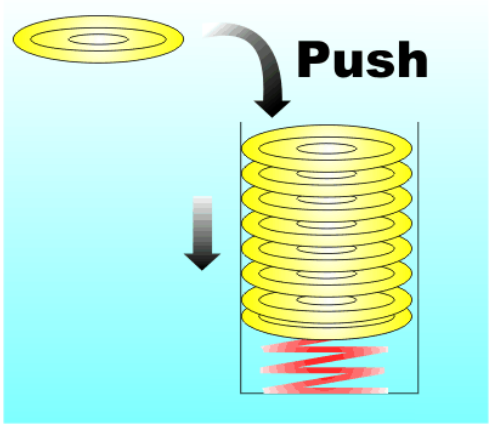
Abstract Data Type

- Simple ADTs
  - ***Stack***
  - ***Queue***
  - Vector
  - ***Lists***
  - Sequences
  - Iterators

  All these are called Linear Data Structures

Stacks

- A **stack** is a container of objects that are inserted and removed according to the last-in-first-out (**LIFO**) principle.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- Inserting an item is known as "**pushing** " onto the stack.
- "**Popping**" off the stack is synonymous with removing an item.



Stacks

- Stack "S" is a abstract data type supporting following methods:
  - ***Push(o)***- insert object o at the top of the stack
  - ***Pop()***- remove from the stack and return the top object on the stack. Error occurs for empty stack.
  - Supporting methods
    - ***Size()***- return the number of objects in the stack
    - ***isEmpty()***- return Boolean indicating if stack is empty
    - ***Top()***- return value of top object on the stack. Error occurs for empty stack.

Stacks: An Array Implementation

- Create a stack using an array by specifying a maximum size N for our stack.
- The stack consists of an N-element array S and an integer variable t, the index of the top element in array S.

- Array indices start at 0, so we initialize t to -1

$t = -1$

Stacks: An Array Implementation

- **Pseudo code**

```
Algorithm push(o)
if size()==N then
    return Error
t=t+1
S[t]=o

Algorithm pop()
 if isEmpty() then
    return Error
 t=t-1

return S[t+1]
```

```
Algorithm size()
return t+1

Algorithm isEmpty()
if t<0 return

Algorithm top()
if isEmpty() then
    return Error
return S[t]
```

Stacks: An Array Implementation

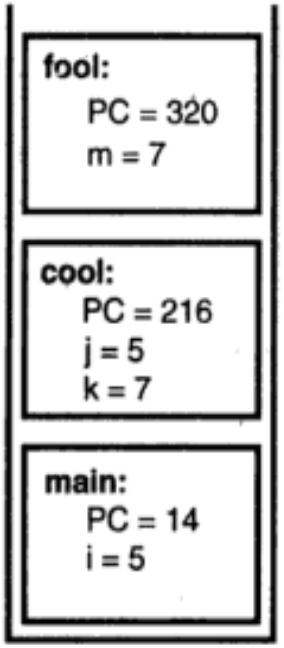
- The array implementation is simple and efficient (**methods performed in O(1))/constant time.**

Disadvantage

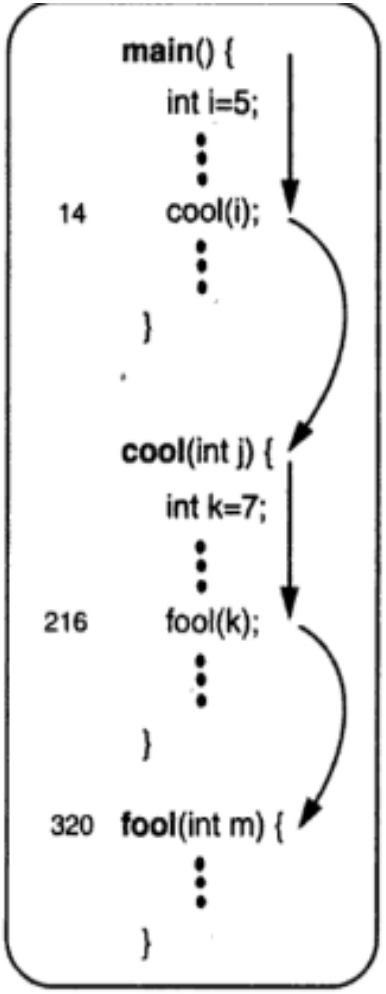
- There is an upper bound, *N*, on the size of the stack.
- The arbitrary value *N* may be too small for a given application OR a waste of memory.

Stacks: Applications

- Procedure calls
- Recursion



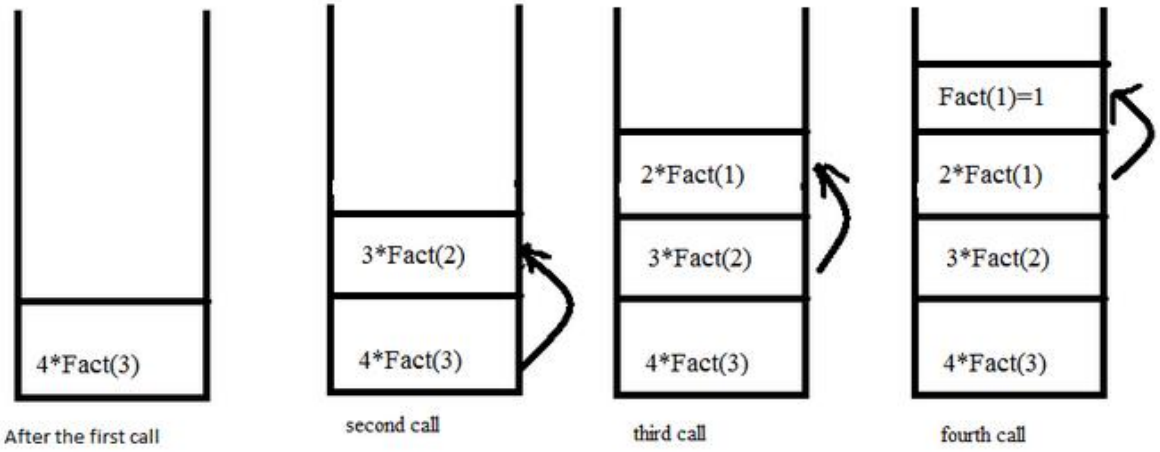
Stack



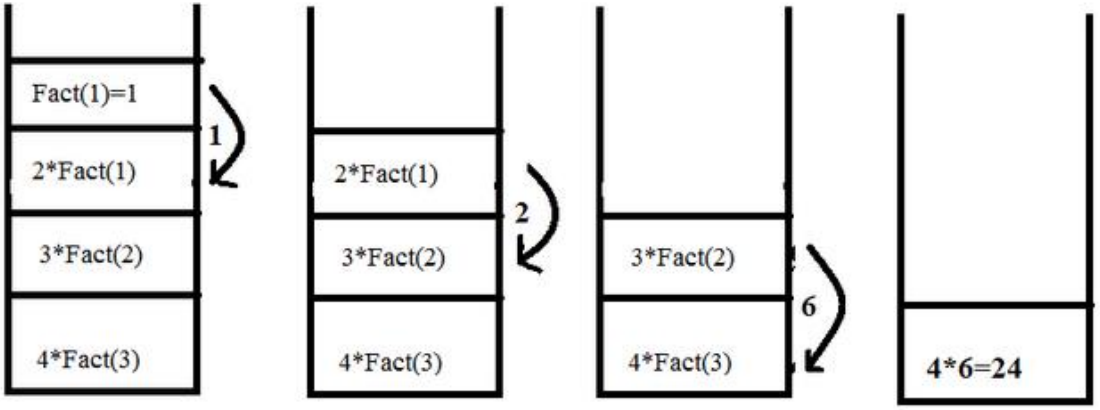
Program

Stacks: Applications

When function call happens previous variables gets stored in stack



Returning values from base case to caller function



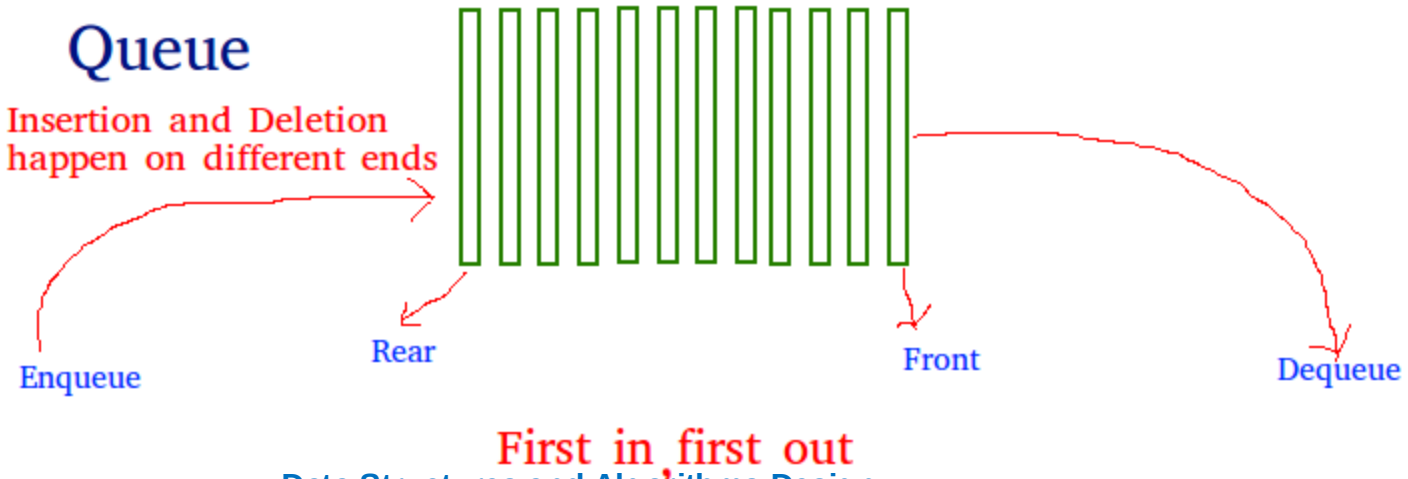
Stacks: HW-Stock Span problem

- Given a list of prices of a single stock for N number of days,find the stock span for each day.
- ***Stock span is the number of consecutive days prior to the current day when the price of the stock was less than or equal to the prices at the current day***

Day	Price
1	200
2	120
3	140
4	160
5	130

Queue ADT

- A queue differs from a stack in that its insertion and removal routines follows the **first-in-first-out** (FIFO) principle.
- Elements may be inserted at any time, but only the element which has been in the queue the longest may be removed.
- Elements are inserted at the **rear** (enqueued) and removed from the **front** (dequeued)



Queue ADT

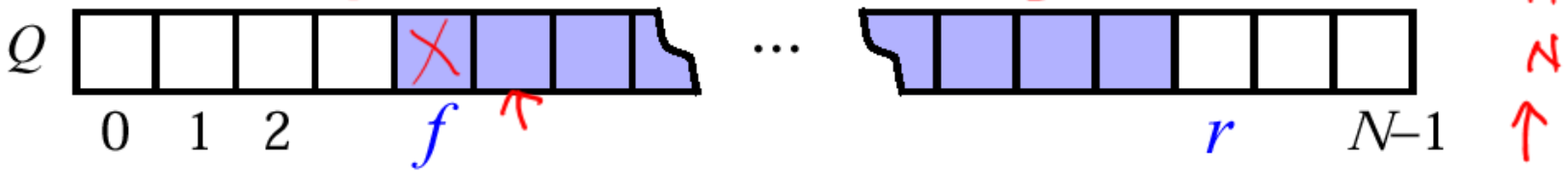
- The queue ADT supports the following two fundamental methods:
  - ***enqueue(o):*** Insert object o at the rear of the queue.
  - ***dequeue():*** Remove and return from the queue the object at the front;an error occurs if the queue is empty.

Additionally, the queue ADT includes the following supporting methods:

- ***size():*** Return the number of objects in the queue.
- ***isEmpty():*** Return a Boolean value indicating whether queue is empty.
- ***front():*** Return, but do not remove, the front object in the queue;an error occurs if the queue is empty

Queues: An Array Implementation

- A maximum size N is specified.
- The queue consists of an N-element array Q and two integer variables:
  - f, is an index to the cell of Q storing the first element of the queue which is the next candidate to be removed by a dequeue operation, unless the queue is empty (in which case f = r)
  - r , is an index to the next available array cell in Q.
  - Initially, f=r=0 and the queue is empty if f=r



```
Algorithm enqueue(o)
    if size() = N - 1 then
        Qfull(Exception
    Q[r] ← o
    r ← (r + 1) mod N
```

```
Algorithm dequeue(o)
    if isEmpty() then
        throw Exception
    temp ← Q[f]
    Q[f] ← null
    f ← (f + 1) mod N
```

Queues: An Array Implementation

- The "normal" configuration with f<=r

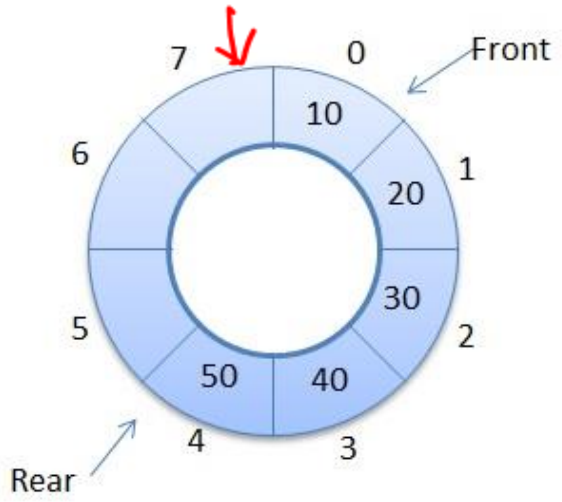
- The "wrapped around" configuration with r < f.

  The cells storing queue elements are highlighted.

Queues: An Array Implementation

- Circular Array
  - r=(r+1)mod N
  - f=(f+1)mod N

- When array is full f=r (Same as array empty condition)



r=0 f=0
r=1 f=0
r=2 f=0
r=2 f=1
r=2 f=2
r=3 f=2

Queues: An Array Implementation

- Queue is empty: $f = r$
- When r reaches and overlaps with f, the queue is full: $r = f$

- To distinguish between empty and full states, we impose a constraint: ***Q* can hold at most *N* – 1 objects** (one cell is wasted). So r never overlaps with f, except when the queue is empty.

Queues: An Array Implementation

- Queue Operations

```
Algorithm dequeue()
if isEmpty() then
    return Error
Q[f]=null
f=(f+1) mod N
```

```
Algorithm enqueue(o)
if size = N - 1 then
    return Error
Q[r]=o
r=(r +1) mod N
```

```
Algorithm size()
return (N-f+r) mod N
```

```
Algorithm isEmpty()
if(f=r) return
```

```
Algorithm front()
if isEmpty() then
    return Error
return Q[f]
```

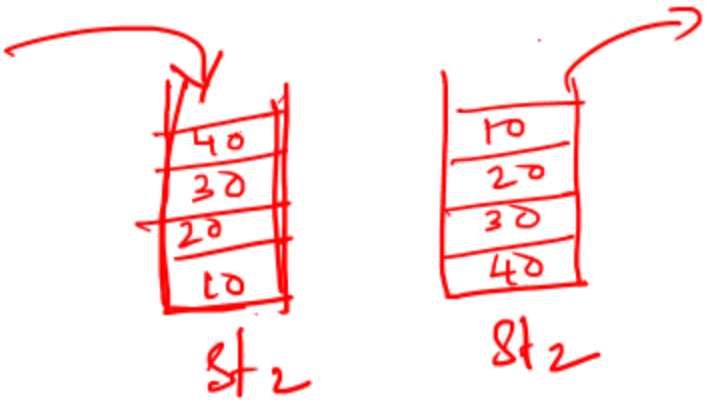
- Each method runs in O(1) time.

Queues: Application

- Multiprogramming

Queues: HW

- Write an Algorithm to implement Queue using Stack(s).Discuss different approaches.



THANK YOU!

BITS Pilani
Hyderabad Campus