DATA STRUCTURES (ITPC-203)

Stacks and Queues

Mrs. Sanga G. Chaki

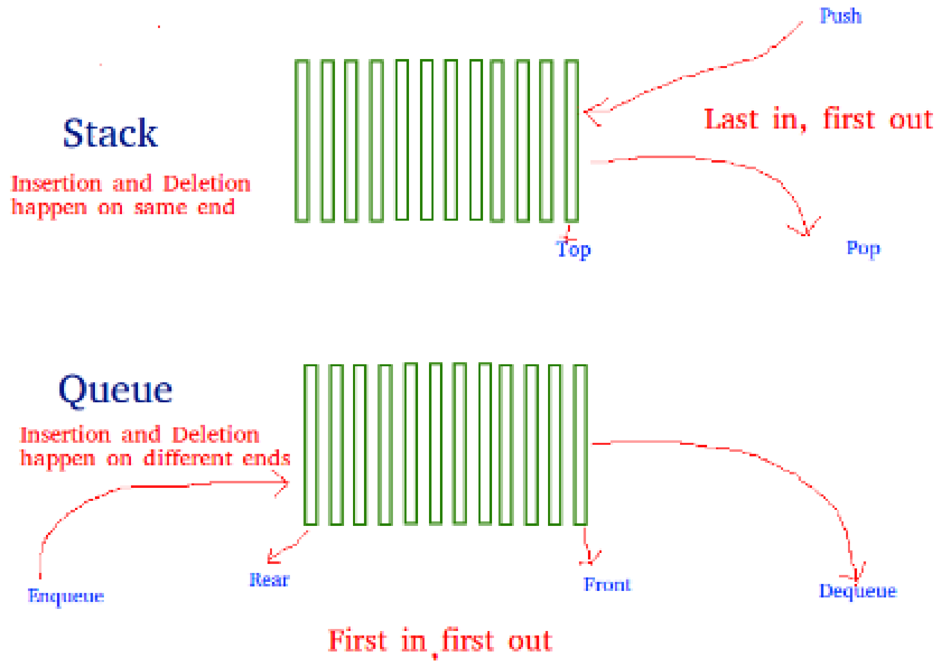
Department of Information Technology

Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Queue using Stacks

Queue (FIFO) using Stacks (LIFO)

1. We are given a stack data structure with push and pop operations
2. The task is to implement a queue using instances of stack data structure and operations on them.
3. A queue can be implemented using two stacks.
4. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2.
5. q can be implemented in two ways

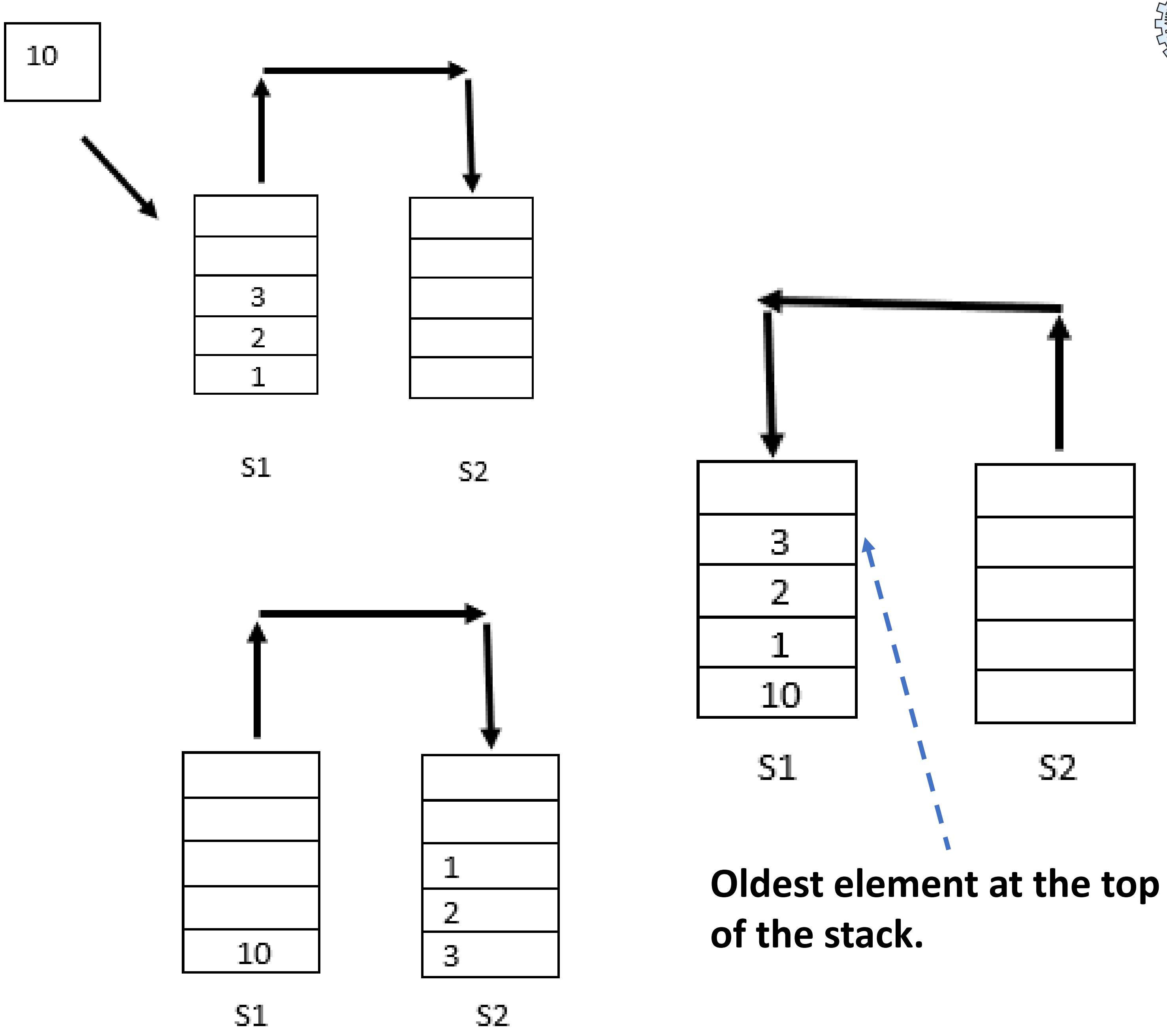

Stack
Insertion and Deletion happen on same end
Push
Last in, first out
Top
Pop
Queue
Insertion and Deletion happen on different ends
Enqueue
Rear
Front
Dequeue
First in first out

Queue (FIFO) using Stacks (LIFO)

1. Method 1 (By making enQueue operation costly)
   a) This method makes sure that oldest entered element is always at the top of stack 1,
   b) so that deQueue operation just pops from stack1.
   c) To put the element at top of stack1, stack2 is used.

Queue using Stacks

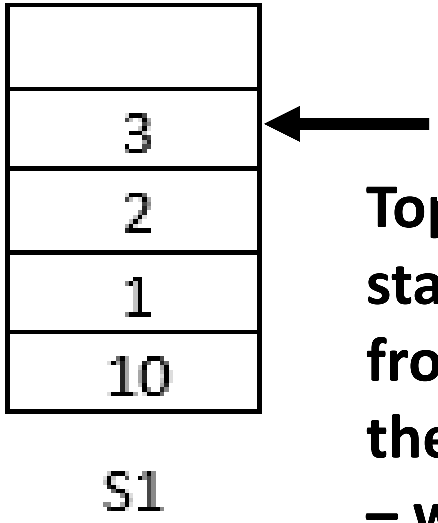
1. enQueue(q, x):
   I. While stack1 is not empty, push everything from stack1 to stack2.
   II. Push x to stack1.
   III. Push everything back to stack1.
2. Here time complexity will be O(n)



Oldest element at the top of the stack.

Queue using Stacks

1. deQueue(q):
   I. If stack1 is empty then error
   II. Pop an item from stack1 and return it
2. The topmost element in the stack is the first input into the queue implementation.
3. Here time complexity will be O(1)
4. **So, enqueue is costly.**



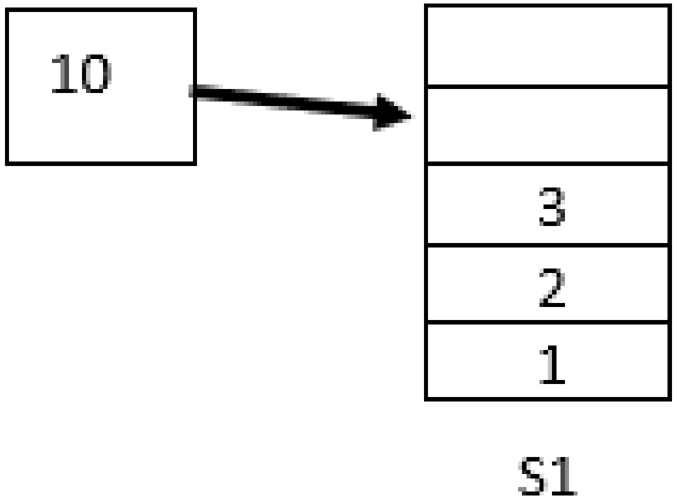
Top of the stack = front of the queue – where dequeue should take place.

Queue using Stacks

1. Method 2 (By making deQueue operation costly):
2. In this method, in enqueue operation, the new element is entered at the top of stack1.
3. In dequeue operation, if stack2 is empty then all the elements are moved to stack2 and finally top of stack2 is returned.

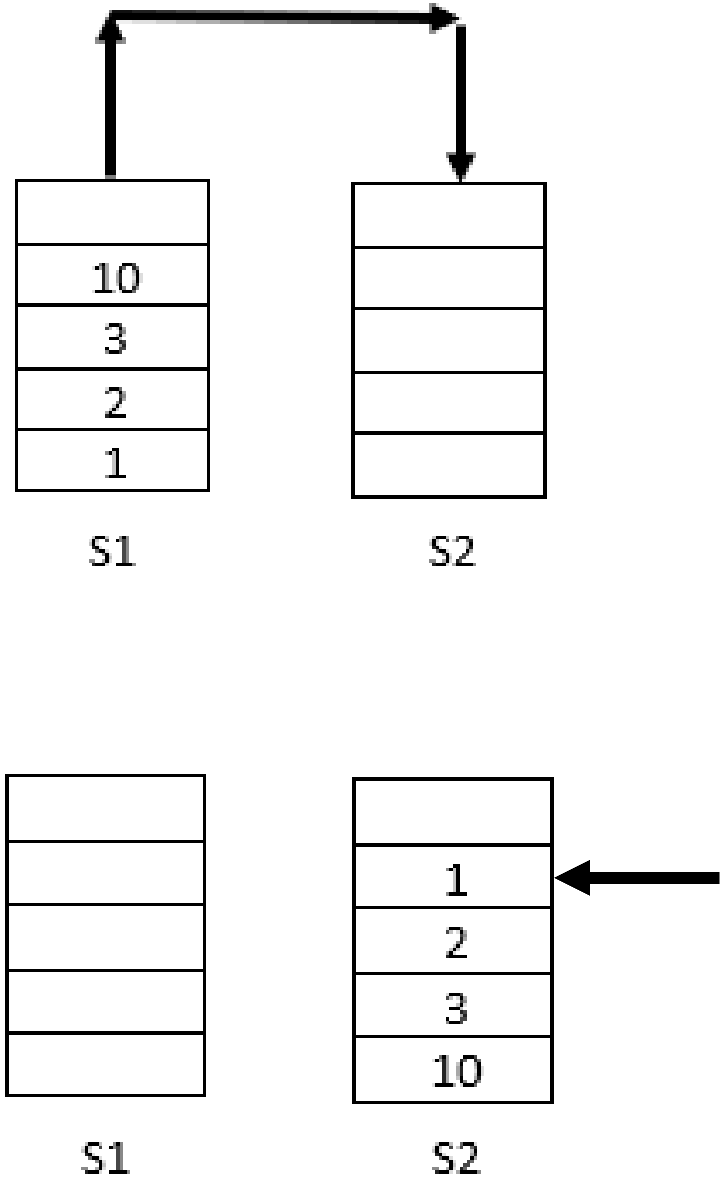
Queue using Stacks

1. enQueue(q, x):
    I. Push x to stack1
2. Here time complexity will be O(1)



Queue using Stacks

1. deQueue(q):
   I. If both stacks are empty then error.
   II. If stack2 is empty
      - While stack1 is not empty, push everything from stack1 to stack2.
   III. Pop the top element from stack2 and return it.
2. Here time complexity will be O(n)
3. So dequeue operation is costly.
4. What is the space complexity?

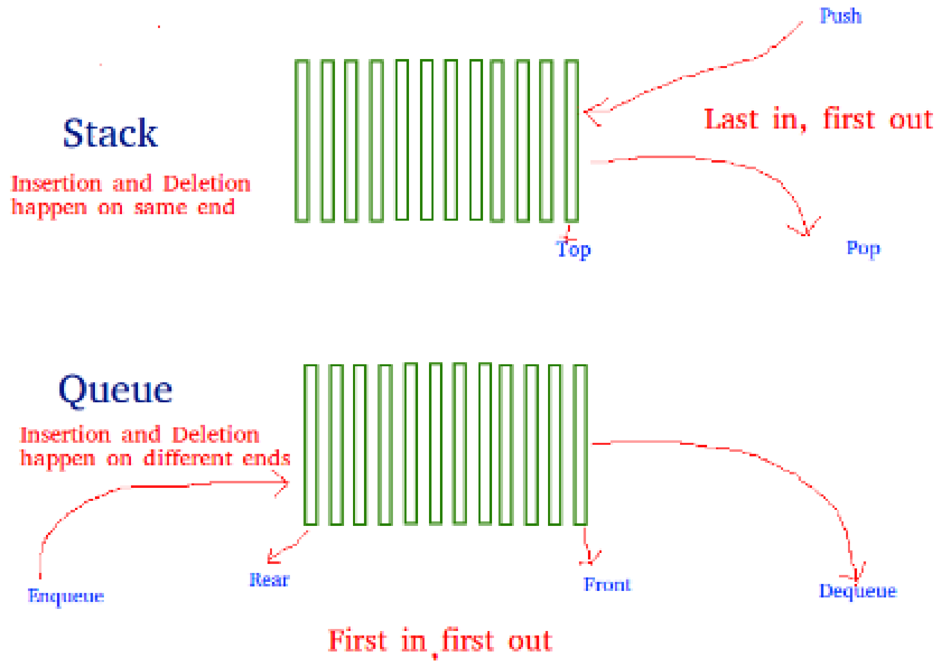


Top of the stack = front of the queue – where dequeue should take place.

Stack using Queue

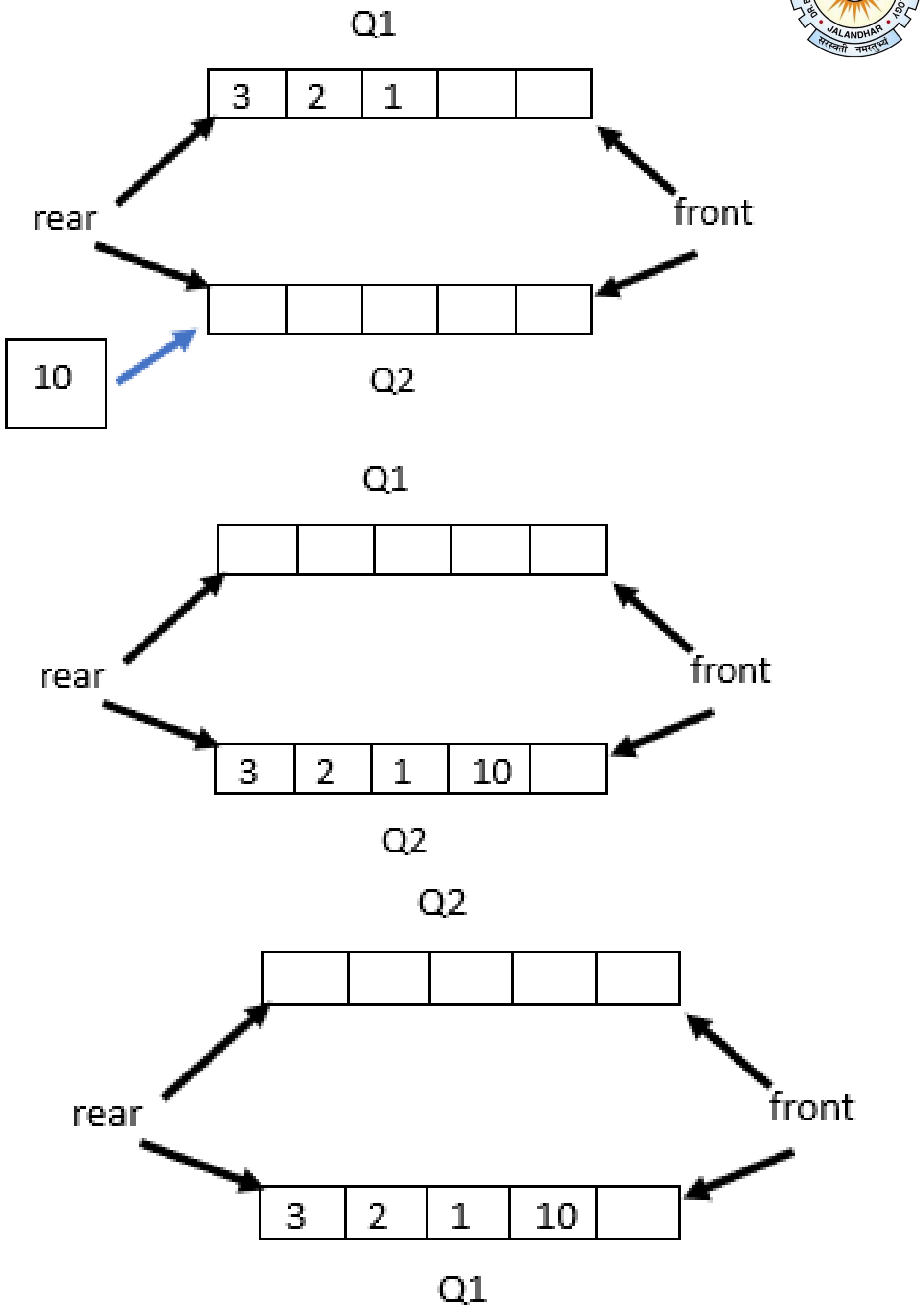
Stack (LIFO) using Queues (FIFO)

1. We are given a Queue data structure that supports standard operations like enqueue() and dequeue().
2. The task is to implement a Stack data structure using only instances of Queue and Queue operations allowed on the instances.
3. A Stack can be implemented using two queues. Let Stack to be implemented be ‘s’ and queues used to implement are ‘Q1’ and ‘Q2’.
4. Stack ‘s’ can be implemented in two ways


Stack
Insertion and Deletion happen on same end
Push
Last in, first out
Top
Pop
Queue
Insertion and Deletion happen on different ends
Enqueue
Rear
Front
Dequeue
First in first out

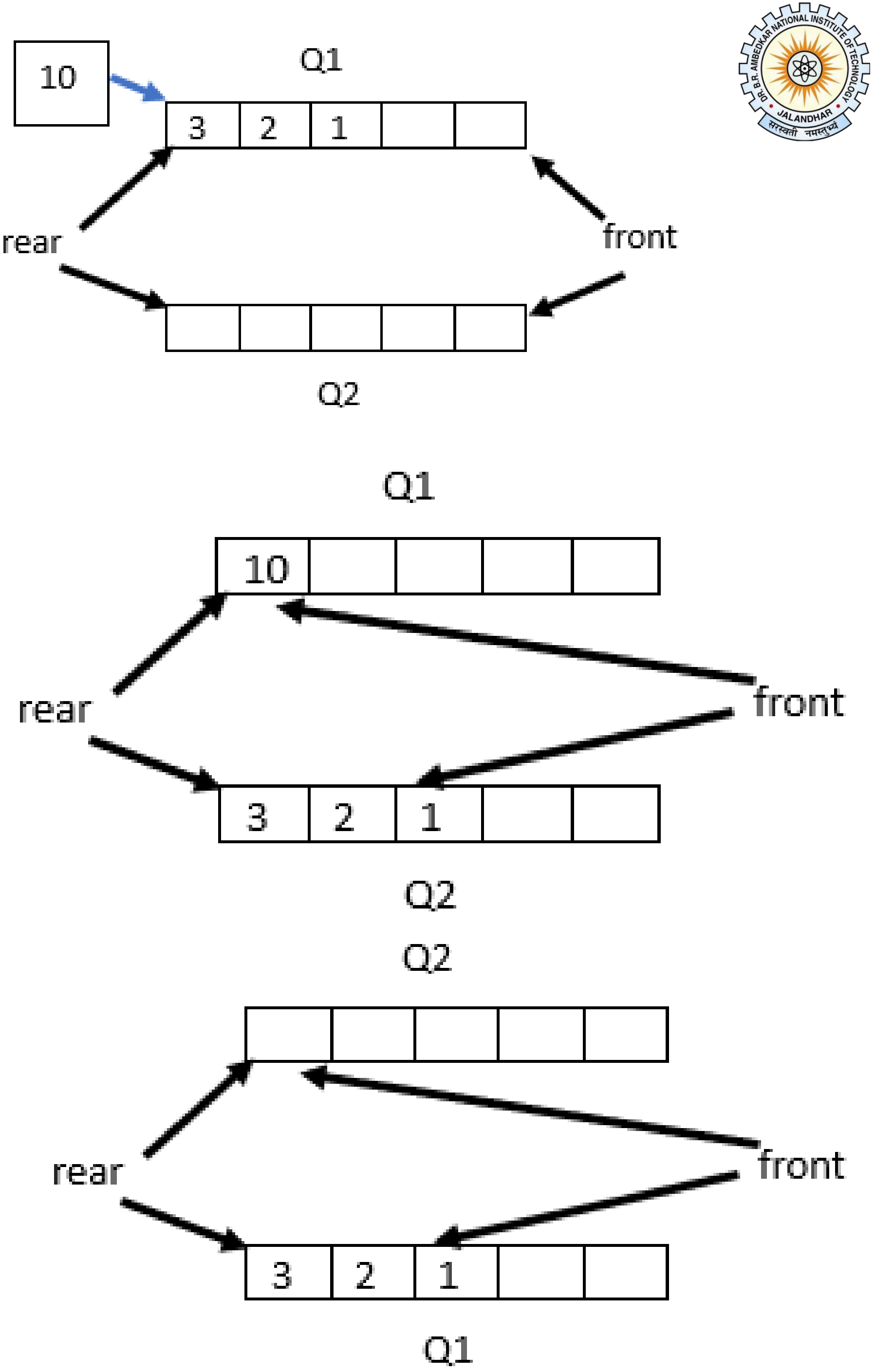
Stack using Queues

1. Implement Stack using Queues By making push() operation costly:
   a) keep newly entered element at the front of Q1, so that pop operation dequeues from Q1
   b) Q2 is used to put every new element in front of Q1.
2. push(s, x):
   I. Enqueue x to Q2.
   II. One by one dequeue everything from Q1 and enqueue to Q2.
   III. Swap the queues of Q1 and Q2.
3. pop(s):
   I. Dequeue an item from Q1 and return it.



Stack using Queues

1. Stack using Queues by making pop() operation costly:
   a) The new element is always enqueued to Q1.
   b) In pop() operation, if Q2 is empty then all the elements except the last, are moved to Q2. Finally, the last element is dequeued from q1 and returned.
2. push(s, x): Enqueue x to Q1.
3. pop(s):
   I. Dequeue everything except the last element from Q1 and enqueue to Q2.
   II. Dequeue the last item of Q1, the dequeued item is the result, store it.
   III. Swap Q1 and Q2
   IV. Return the item stored in step 2.



10
Q1
3
2
1
rear
front
Q2
Q1
10
rear
front
3
2
1
Q2
Q2
rear
front
3
2
1
Q1