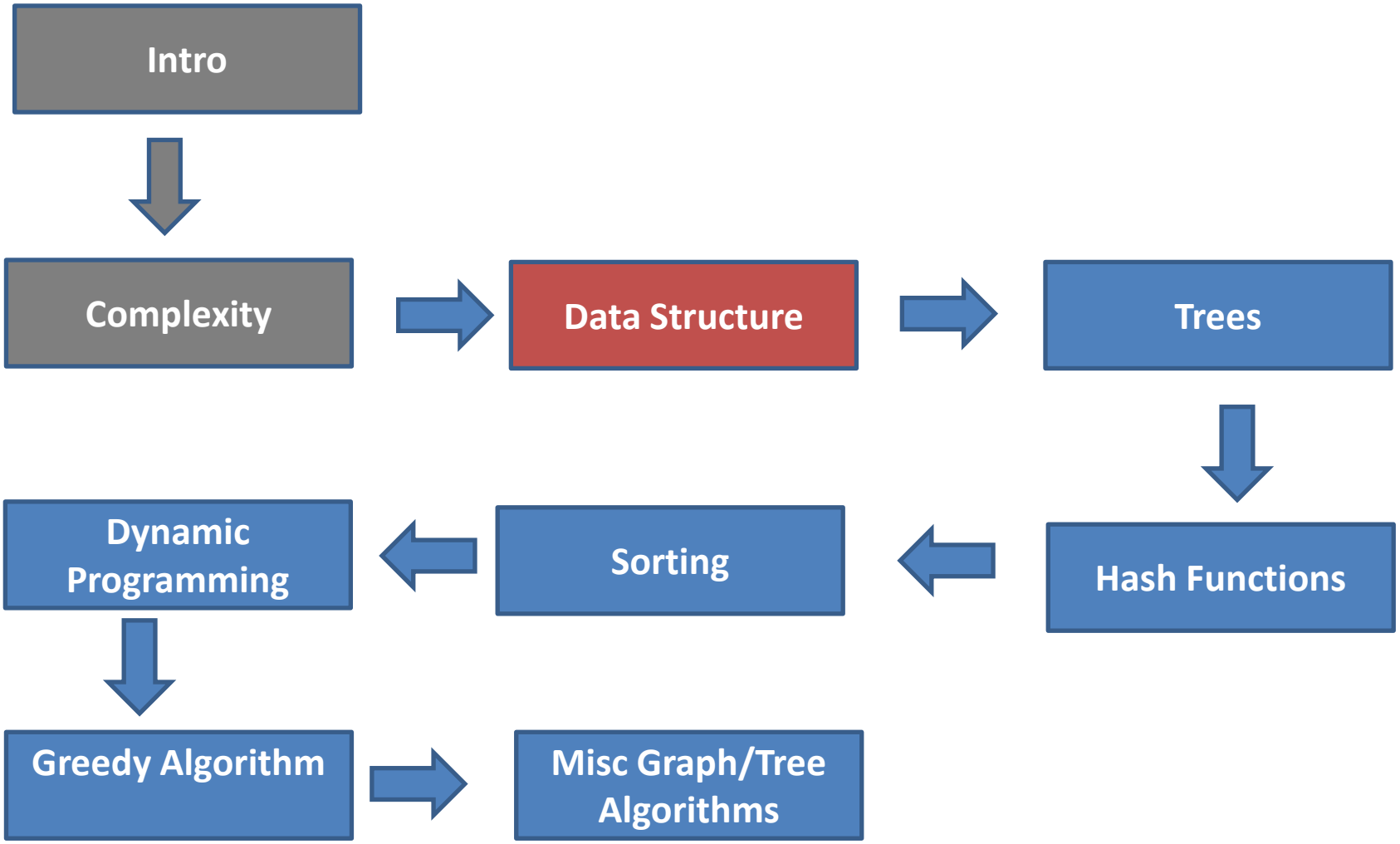
An Introduction to Algorithms

By

Hossein Rahmani

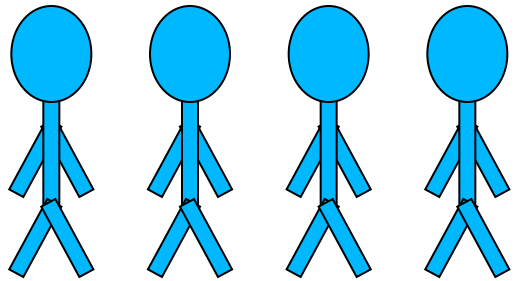
h_rahmani@iust.ac.ir

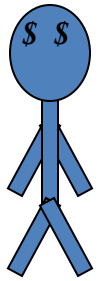
http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

The Queue Operations

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



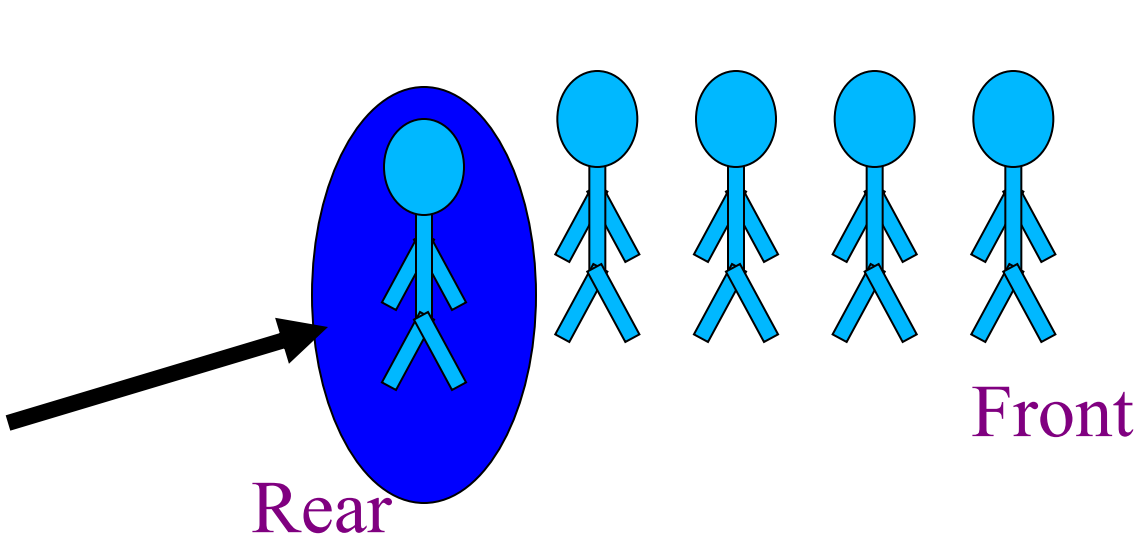


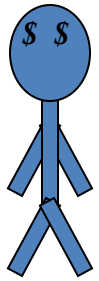
Front

Rear

The Queue Operations

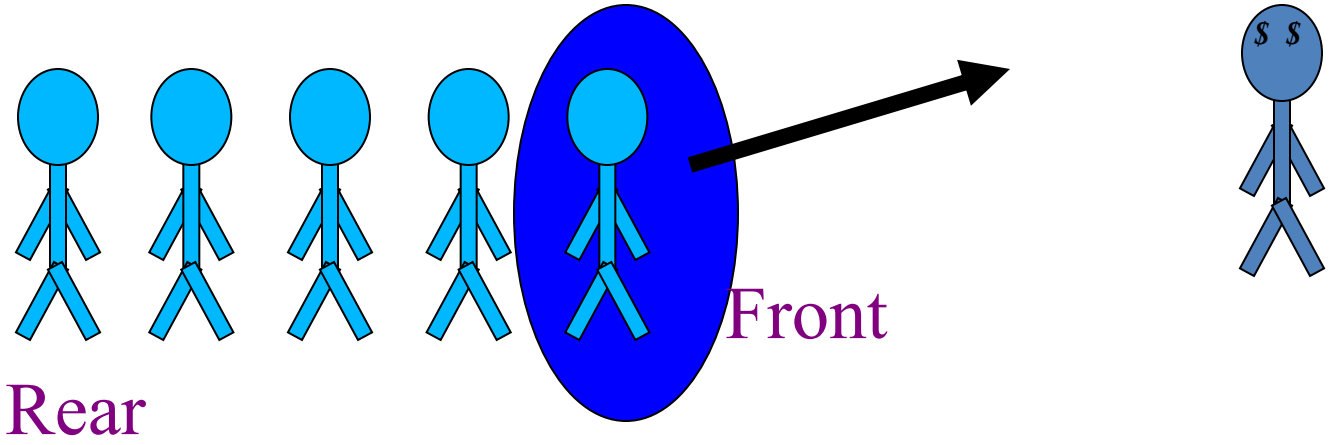
- New people must enter the queue at the rear. The C++ queue class calls this a **push**, although it is usually called an **enqueue** operation.





The Queue Operations

- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a **pop**, although it is usually called a **dequeue** operation.



The Queue ADT

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(object): inserts an element at the end of the queue
  - object dequeue(): removes and returns the element at the front of the queue
- Auxiliary queue operations:
  - object front(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyQueueException

Queue Example

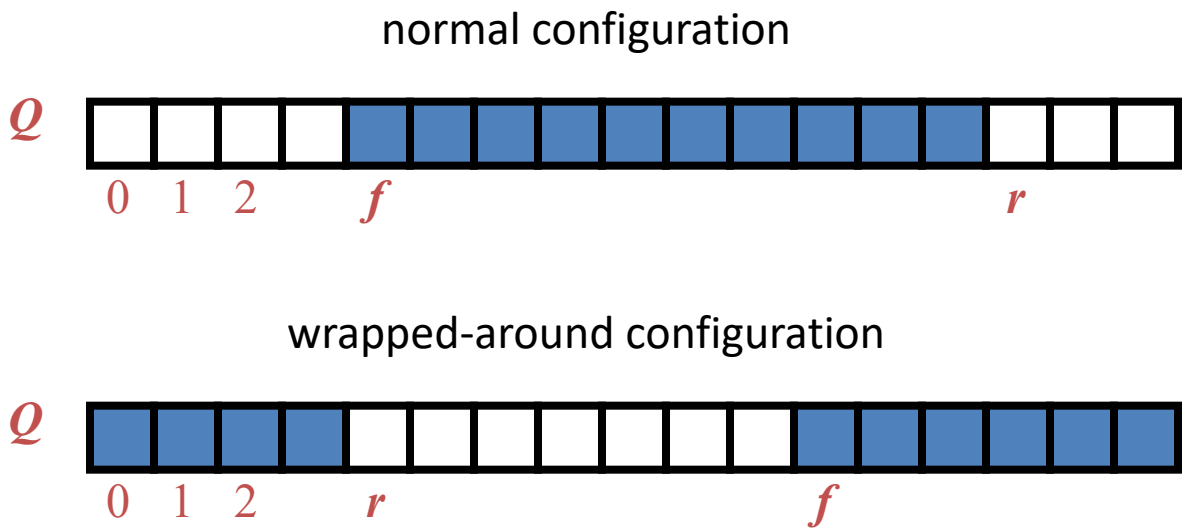
Operation	***Output***	*Q*
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
front()	*7*	(7)
dequeue()	*7*	()
dequeue()	*"error"*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

Applications of Queues

- Direct applications
  - Waiting lists
  - Access to shared resources (e.g., printer)
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and rear
  - f index of the front element
  - r index immediately past the rear element
- Array location r is kept empty

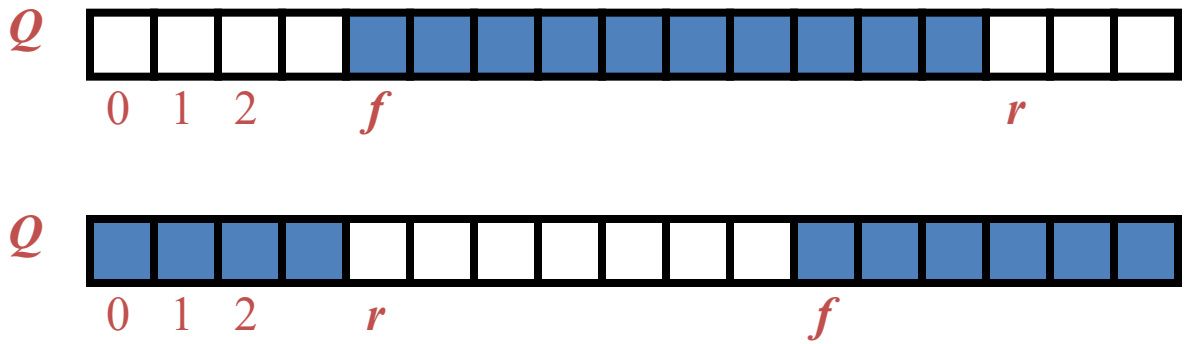


Queue Operations

- We use the modulo operator (remainder of division)

Algorithm *size*()
  return $(N - f + r) \bmod N$

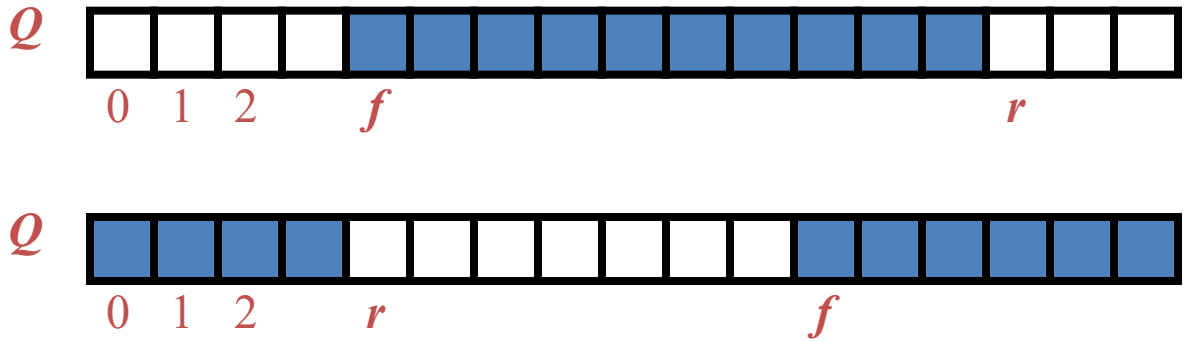
Algorithm *isEmpty*()
  return $(f = r)$



Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

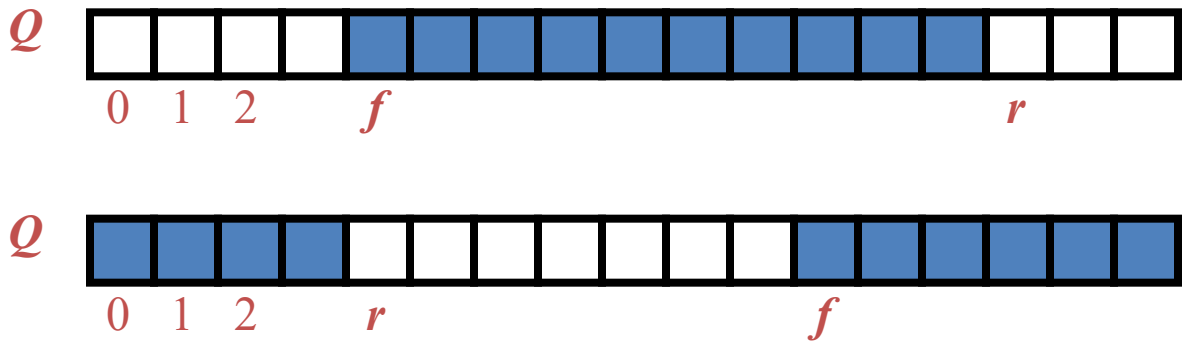
Algorithm *enqueue*(*o*)
  if *size*() = *N* – 1 **then**
    throw *FullQueueException*
  else
    Q[*r*] ← *o*
    r ← (*r* + 1) mod *N*



Queue Operations (cont.)

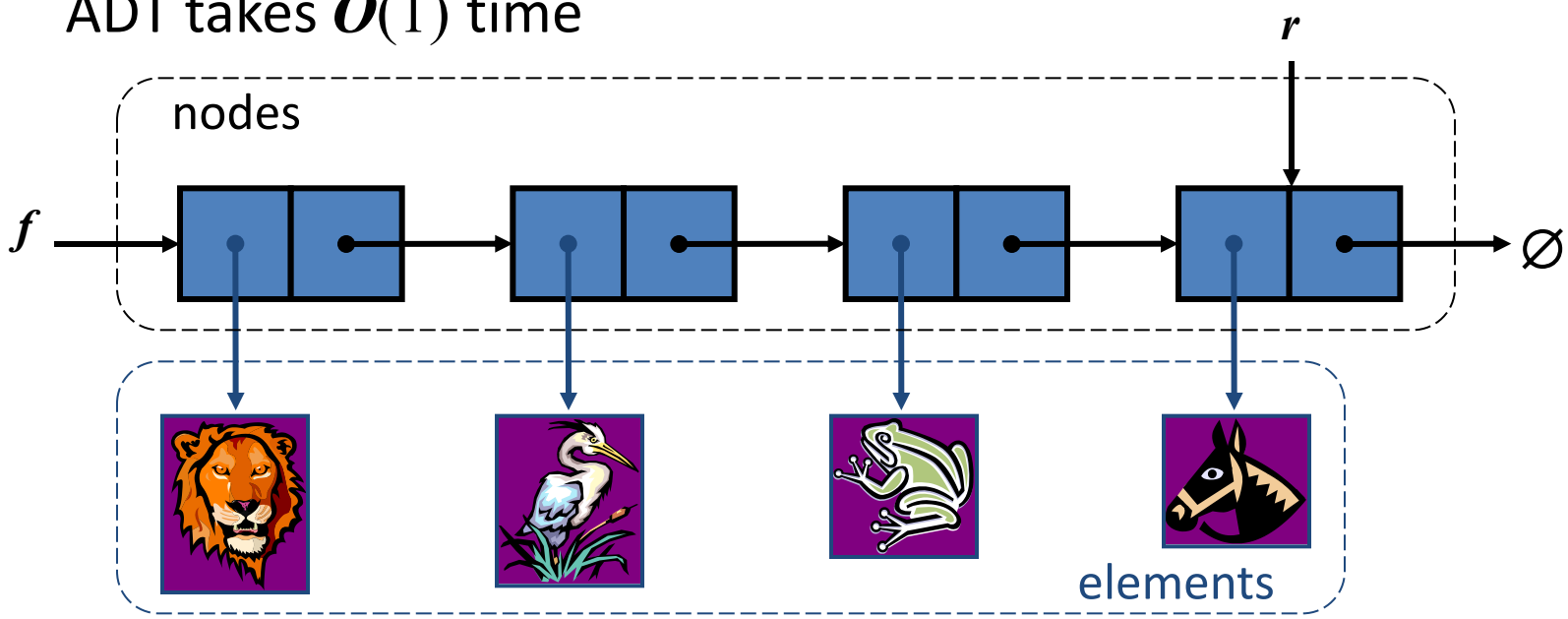
- Operation dequeue throws an exception if the queue is empty
- This exception is specified in the queue ADT

Algorithm *dequeue*()
  if *isEmpty*() **then**
    throw *EmptyQueueException*
  else
    $o \leftarrow Q[f]$
    $f \leftarrow (f + 1) \bmod N$
    return o



Queue using a Doubly-Linked List

- We can implement a queue with a doubly linked list
  - The front element is stored at the first node
  - The rear element is stored at the last node
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time



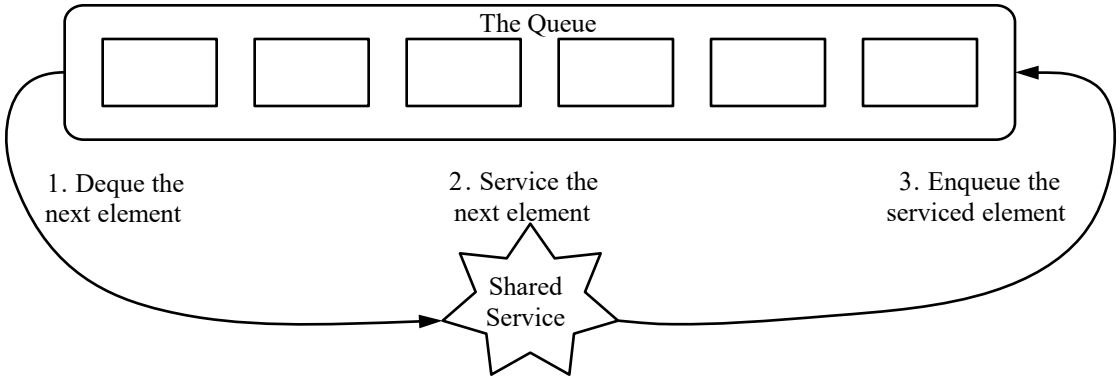
Queue Interface in Java

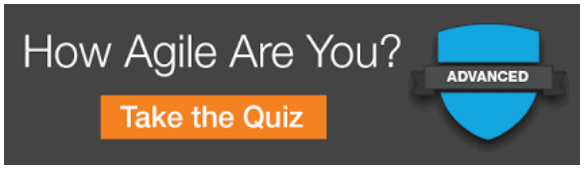
- Java interface corresponding to our Queue ADT
- Requires the definition of class EmptyQueueException
- No corresponding built-in Java class

```
public interface Queue {

   public int size();

   public boolean isEmpty();

   public Object front()
         throws EmptyQueueException;

   public void enqueue(Object o);

   public Object dequeue()
         throws EmptyQueueException;
}
```

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue, *Q*, by repeatedly performing the following steps:
  1. *e* = *Q*.dequeue()
  2. Service element *e*
  3. *Q*.enqueue(*e*)





Quiz 1

Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

```
void fun(int n)
{
    IntQueue q = new IntQueue();
    q.enqueue(0);
    q.enqueue(1);
    for (int i = 0; i < n; i++)
    {
        int a = q.dequeue();
        int b = q.dequeue();
        q.enqueue(b);
        q.enqueue(a + b);
        ptint(a);
    }
}
```