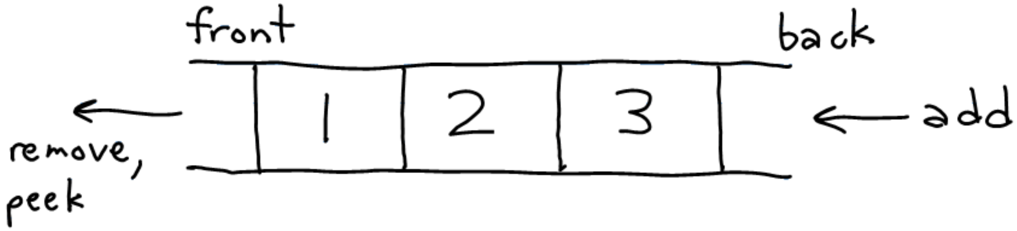
CSE 373: Data Structures and Algorithms

Lecture 2: Queues

Queue ADT

- **queue**: A list with the restriction that insertions are done at one end and deletions are done at the other
  - First-In, First-Out ("FIFO")
  - Elements are stored in order of insertion but don't have indexes.
  - Client can only add to the end of the queue, and can only examine/remove the front of the queue.



- basic queue operations:
  - **add** (enqueue): Add an element to the back.
  - **remove** (dequeue): Remove the front element.
  - **peek**: Examine the top element.

Queues in computer science

- Operating systems:
  - queue of print jobs to send to the printer
  - queue of programs / processes to be run
  - queue of network data packets to send

- Programming:
  - modeling a line of customers or clients
  - storing a queue of computations to be performed in order

- Real world examples:
  - people on an escalator or waiting in a line
  - cars at a gas station (or on an assembly line)

Using `Queue`s

`add(`**value**`)`	places given value at back of queue
`remove()`	removes value from front of queue and returns it; throws a `NoSuchElementException` if queue is empty
`peek()`	returns front value from queue without removing it; returns `null` if queue is empty
`size()`	returns number of elements in queue
`isEmpty()`	returns `true` if queue has no elements

```
Queue<Integer> q = new LinkedList<Integer>();
q.add(42);
q.add(-3);
q.add(17);                               // front [42, -3, 17] back
System.out.println(q.remove());          // 42
```

- **IMPORTANT**: When constructing a queue you must use a new `LinkedList` object instead of a new `Queue` object.

Queue idioms

- As with stacks, must pull contents out of queue to view them.

```
while (!q.isEmpty()) {
    do something with q.remove();
}
```

  - another idiom: Examining each element exactly once.

```
int size = q.size();
for (int i = 0; i < size; i++) {
    do something with q.remove();
    (including possibly re-adding it to the queue)
}
```

    - Why do we need the `size` variable?

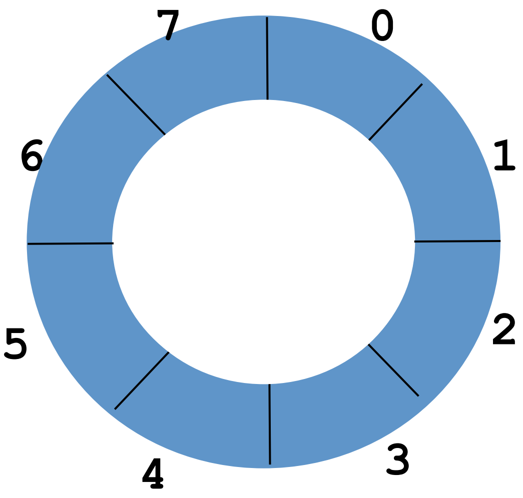
Implementing Queue ADT: Simple Array Queue

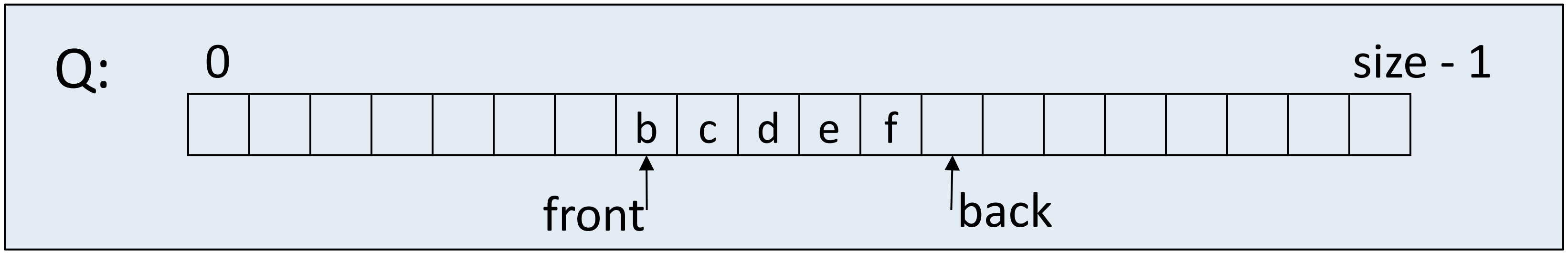
- Keep track of the number of elements in the queue, `size`.
- Enqueue at the back of the array (`size`).
- Dequeue at the front of the array (index 0).

  - what is bad about this implementation?
  - what if we enqueue at 0 and dequeue at `size`?

Implementing Queue ADT: Circular Array Queue

- **Neat trick**: use a ***circular array*** to insert and remove items from a queue in constant time.
- The idea of a circular array is that the end of the array "wraps around" to the start of the array.

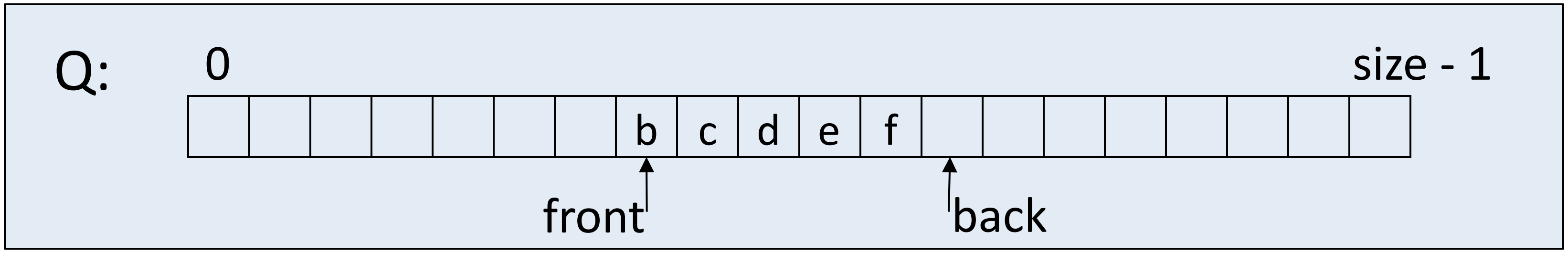



Q: 0 size - 1

b c d e f

front back

Circular Array Queue

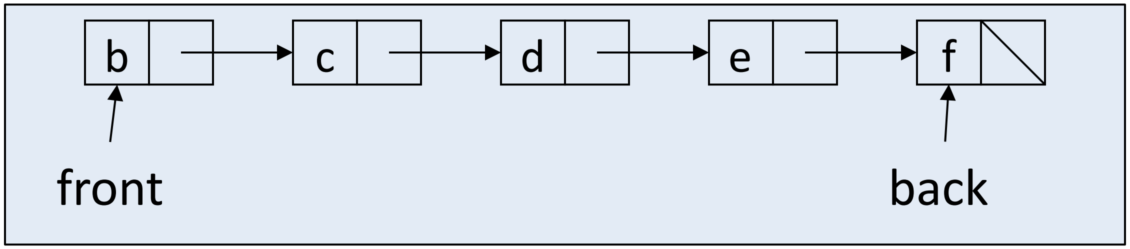


```
// Basic idea only!
enqueue(x) {
  Q[back] = x;
  back = (back + 1) % size
}
```

```
// Basic idea only!
dequeue() {
  x = Q[front];
  front = (front + 1) % size;
  return x;
}
```

Exercise: Linked List Queue Implementation

Implement a queue class that stores String values using a singly linked list with both nodes to indicate the front and the back of the queue as below. The queue should implement the interface on the next slide.



Exercise: Linked List Queue Implementation (cont.)

```
/**
 * Interface for a queue of Strings.
 */
public interface StrQueue {
    /**
     * Tests if the queue is empty.
     */
    public boolean isEmpty();

    /**
     * Inserts an element at the end of the queue.
     */
    public void enqueue(String str);

    /**
     * Deletes and returns the element at the front of the queue.
     * @return the deleted value; throws NoSuchElementException if empty
     */
    public String dequeue();
}
```

Circular Array vs. Linked List

Array:

List:

Circular Array vs. Linked List

Array:

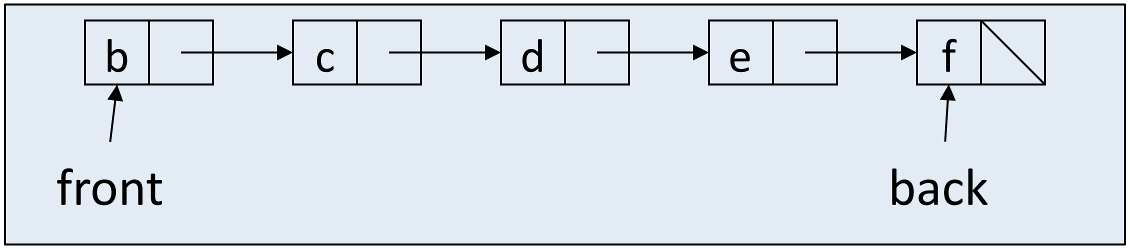
- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast

List:

- Always just enough space
- But more space per element
- Operations very simple / fast

- If we wanted add the ability to access the kth element to our queue, could both implementations support this?

Linked List Queue



```
// Basic idea only!
enqueue(x) {
  back.next = new Node(x);
  back = back.next;
}
```

```
// Basic idea only!
dequeue() {
  x = front.item;
  front = front.next;
  return x;
}
```