```
enqueue(element):
            // Add an element to the back of the queue
            if capacity >= 0 and size() >= capacity:
                        throw "Queue is full"
            inbox.push(element)
            // O(1)

dequeue():
            // Remove and return the front element
            if outbox.empty():
                        transfer_inbox_to_outbox()
            if outbox.empty():
                        throw "Queue is empty"
            value = outbox.top()
            outbox.pop()
            return value
            // best case O(1); worst-case O(n) when transfer happens

front():
            // Return (but do not remove) the front element
            if outbox.empty():
                        transfer_inbox_to_outbox()
            if outbox.empty():
                        throw "Queue is empty"
            return outbox.top()
            // best case O(1); worst-case O(n)

size():
            return inbox.size() + outbox.size()
            // O(1)

isEmpty():
            return inbox.empty() and outbox.empty()
            // O(1)

isFull():
            if capacity < 0:
                        return false
            return size() >= capacity
            // O(1)

maxSize():
            return capacity;  //based on initially set capacity
            // O(1)
```

Design choices:

- I used 2 stacks because stacks only allow you to push, pop, and top. By using 2 stacks we can actually implement a queue where the oldest element is removed rather than the newest like in a stack
- We use a capacity value set by the user initially.