Technique: Queue with Max

Level: Hard

Implement a Queue with O(1) lookup of the Maximum element.

Questions to Clarify:
Q. Can we assume that it will have integers?
A. Yes

Solution:
We use a similar approach as stacks - keep two queues - a main queue and a max queue. The main queue contains the elements. The max queue contains the max elements. The max queue is a double ended queue (deque) because we want to be able to remove elements from both ends.

Let's say we have the following:

```
Main Queue: 1           <-- front of queue
Max Queue : 1           <-- front of queue
```

Now, let's say we insert a 4 into the queue. the main queue will look as follows:

```
Main Queue: 4 -> 1      <-- front of queue
```

In the max queue, we don't need 1 anymore, since 1 can never be the max of this queue now. So we remove 1 and insert 4.

```
Main Queue: 4 -> 1      <-- front of queue
Max Queue : 4           <-- front of queue
```

Now let's say we insert 2 into the queue. 2 is not the max, but it can be a max if we dequeue 1 and 4 from the queue. So, we insert it onto the max queue:

```
Main Queue: 2 -> 4 -> 1      <-- front of queue
Max Queue : 2 -> 4           <-- front of queue
```

If we insert a 3 into the queue, we can get rid of the 2 from the max queue, because 2 can no longer be the max of the queue, even if 4 and 1 are dequeued. Our queues become:

```
Main Queue: 3 -> 2 -> 4 -> 1  <-- front of queue
Max Queue : 3 -> 4            <-- front of queue
```

In the process of inserting 3, we removed elements from the back of the max queue until we found an element >= 3. This is because elements < 3 could never be max after 3 is inserted. This is the algorithm for inserting an element.

To lookup the max, we just check the front of the Max Queue.

While dequeuing elements, we check if they are equal to the front of the max queue, and if so, we dequeue from the max queue too. For example, after dequeuing 1, lets say we want to dequeue 4. We see that 4 is the front of the max queue, so we remove both the 4s. This makes sense as 4 can no longer be the max after it's removed.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a few
lines quickly to plan out your solution. Your actual solution should be in a real
language and use good syntax.)

init main queue and max queue

enqueue(n)
     main.enqueue(n)
     while (max.back < n)
          max.remove_back()
     max.enqueue(n)

dequeue()
     value = main.dequeue()
     if (max.front() == value)
         max.dequeue()

findMax()
     return max.front()
```

Test Cases:
Edge Cases: empty queue
Base Cases: single element in queue
Regular Cases: insert equal element as back, remove equal element,
elements in decreasing/increasing order

Time Complexity:
Insertion: O(n) worse case, O(1) amortized because we remove at most N elements for N insertions.

Deletion and Max lookup: O(1)

Space Complexity: O(n) on the Max queue

```
public class QueueWithMax {
    Queue<Integer> main;
    Deque<Integer> max;

    public QueueWithMax() {
        main = new LinkedList<>();
        max = new LinkedList<>();
    }

    public void enqueue(int item) {
        main.add(item);
        while (!max.isEmpty() && max.getLast() < item)
            max.removeLast();
        max.add(item);
    }

    public void dequeue() throws QueueEmptyException {
        if (main.isEmpty())
            throw new QueueEmptyException();
        int item = main.remove();
        if (max.getFirst() == item)
            max.remove();
    }

    public int findMax() throws QueueEmptyException {
        if (max.isEmpty())
            throw new QueueEmptyException();
        return max.getFirst();
    }
}

/*
 * Helper code, ask the interviewer if they want you to implement.
 */
public class QueueEmptyException extends Exception {
    public QueueEmptyException() {

    }
}
```