MaxQueue
Arieh Chaikin

A naïve implementation of ArrayQueue would be sufficient to implement the enqueue, dequeue, and size methods, as the Java Docs indicate these methods run in amortized constant time. However, it would not suffice to implement the max method. When calling the dequeue method would result in the largest integer being removed, the queue would need to know which is the second largest integer. In order to do this, every integer would need to be checked to determine which is the new "max" integer, causing one of the methods to run in O(N) time. An implementation which can maintain record of the max integer, even when the current max integer is removed, is needed to maintain amortized constant time (or constant time when applicable) for all methods.

My implementation involves two ArrayQueue data structures which are used as Stacks, an input stack, and an output stack. These stacks where of type "Element", an inner class created. The reason this was important is that each Element not only has a get method for the value of the integers it represents, but it also has a "max" field and get method, used to store the largest integer from that point and below on the stack. When enqeue is called, one of two things happen: If the input stack is empty, a new Element is created with the inputted integer as its value, and the inputted integer as its max, and it is pushed onto the stack. If the input stack is not empty, a new Element is created with the inputted integer as its value, and either its value or the max value from the current top element of the stack as its max value, and it is pushed onto the stack. The advantage of this design is that when an element is removed from a stack, the new maximum integer is already stored in the new top of the stack.

The reason an output stack is needed is in order to remove integers in FIFO order, which means the bottom of the input stack must be removed first. For that reason, when dequeue is called with an empty output stack, every element is popped off the input stack and pushed onto the output stack, reversing the order. Instead of pushing the element itself onto the output stack, a new element is created maintain max order from that element and below as described earlier. Every subsequent call to the dequeue method simple pops the top element off the output stack. There are two size variables for each stack maintaining the size of each stack, adjusted at every push and pop. [For convenience's sake, I maintained a max variable of both the input and output variables which stores the top Element's "max" field to avoid having to check for exceptions if I were to call the peek method every time]

Size: O(1)
For both the input and output stack, there are corresponding size variables. Every time an element is pushed or popped, the size variables are adjusted according with simple addition or subtraction. These calculations run in constant time and thus do not take too much time during other operations. Because these variables are maintained throughout the other methods, the size method runs in constant time, simple returning the sum of the two variables storing the sizes of the stacks.

Enqueue: O(1)
When the enqueue method is called, a check is done to see if the input variable is a 0, a constant operation. Another constant time comparison is done to check if the inputted variable is greater than the current maximum element in the stack. An element is created with a value and max at constant time. The element is then pushed to the stack, which according to Java docs is a constant time method. Finally, the size variable tracking the input stack is increased by one, another constant time operation.

Dequeue: O(1) amortized
In most instances, Dequeue runs at constant time. If there are any elements in the output stack, the top element in the output slot is popped off, which is a constant time operation according to the Java docs. The size variable for the stack is then decreased, and the max variable is changed to the max variable stored in the new top element. However, if the there are no elements in the output slot (and are elements in the input slot), all the elements in the input slot are popped off and pushed into the output slot, an O(N) operation. Therefore, if there are, for example, 10 elements to be dequeued, and no elements in the output slot, there would be one O(N) operation and nine O(1) operations. This is represented by (N+9/N), which is considered to be amortized constant time.

Max: O(1)
Because the max integers of each stack are maintained throughout the queues and enqueues, all the max method does is check which stack's max is larger and returns it, a constant time operation. s