Programming Practice with QuickCheck

COMP 150FP

Fall 2013

1. Using the algebraic laws and Arbitrary instance developed in class, test the following implementation of queues (see also http://www.cs.tufts.edu/comp/150FP/Q.hs):

```haskell
module Q
    (Q, empty, isEmpty, put, get, rest)
where

data Q a = Q { front :: [a], back :: [a] }
    -- corresponds to front ++ reverse back

empty :: Q a
isEmpty :: Q a -> Bool
put :: Q a -> a -> Q a
get :: Q a -> a -- defined on nonempty queue only
rest :: Q a -> Q a -- defined on nonempty queue only

empty = Q [] []
isEmpty q = null (front q) && null (back q)
put q a = Q [] (a : back q)
get (Q [] []) = error "get from empty queue"
get (Q [] as) = get (Q (reverse as) [])
get (Q (a:as) _) = a

rest (Q [] []) = error "rest from empty queue"
rest (Q [] as) = rest (Q (reverse as) [])
rest (Q (a:as) _) = Q as []
```

Add other tests or extend your Arbitrary instance as needed.

Don’t forget shrink!

2. Using either a simple functional heap or a leftist heap, test an implementation of queues that looks like this:

```haskell
newtype Clocked a = Clocked { ticks :: Integer, value :: a }
data Q a = Q { elems :: Heap (Clocked a), clock :: Integer }
```

The clock field of the queue should measure the total number of put operations ever performed, and get should remove the element with the smallest ticks field.

Be sure that your heap is polymorphic and works with any type that is in class Ord.