Persistent Queues Revisited

This is *Queue 2*. Recall, this queue implementation is inefficient. Why?

We can instead implement queue like *Queue 3*:

```
queue * == ([*],[*])                    || (front,rear)
emptyqueue = ([],[])
isempty ([],[]) = True
isempty q = False
head ([x:front],rear) = x
enqueue (front,rear) x = make_proper_queue (front,x:rear)
dequeue (x:front,rear) = make_proper_queue (front,rear)

make_proper_queue ([],x:rear) = (reverse (x:rear),[])
make_proper_queue q = q
```


Persistent Usage

```
|| create queue with 1..100 in it
q = foldl enqueue emptyqueue [1..100]

|| access second element.
head (dequeue q)
head (dequeue q)
head (dequeue q)
head (dequeue q)
```

What is cost?

Recall: Get \$2 for `make_proper_queue` and pay \$$n$ for `reverse`.

Memoization

Consider:

```
<some-code>
   where
      x = <some-expression>
```

Because Miranda is lazy, `x` will only be evaluated if needed.
When `x` is first evaluated, its value will be *memoized*.

Subsequent references to `x` will not re-evaluate `<some-expression>`.
Instead, they will return `x`'s memoized value.

Lazyness & Memoization

What is the runtimes of the following?

1. `reverse [1..n] ! 0`
2. `([1,2,3] ++ reverse [1..n]) ! 0`
3. `([1,2,3] ++ reverse [1..n]) ! 3`

Here `reverse` is expensive.

We can use this idea to improve our queue implementation.

Queue 4

Instead of copying `reverse rear` to `front` when `front` is empty, copy it when `length rear > length front`.

Since length is expensive to compute, store lengths in queue tuple.

```
queue * == (num,[*],num,[*])          || (fl,front,rl,rear)
emptyqueue = (0,[],0,[])
isempty (0,[],0,[]) = True
isempty q = False
head (fl,(x:front),rl,rear) = x

enqueue (fl,front,rl,rear) x
       = make_proper_queue (fl,front,rl+1,x:rear)

dequeue (fl,x:front,rl,rear)
       = make_proper_queue (fl-1,front,rl,rear)

make_proper_queue (fl,front,rl,rear)
       = (fl+rl, front ++ reverse rear,0,[]), if rl > fl
make_proper_queue q = q
```


Amortized Analysis

What part of `make_proper_queue` is expensive?

It is `reverse`.

For queue length n, must call `dequeue` $n/2$ times before reverse happens because it is lazy!

Charge \$2 for `make_proper_queue` call.
Pay \$1 for no-op call.
Pay \$$k$ for call to reverse of k item list.

We must have $n/2$ no-op calls before we call `reverse`.

A persistent queue with amortized $O(1)$ cost per operation!!