CSCC24 Summer 2019 – Assignment 1
Due: Wednesday, June 12, midnight
This assignment is worth 10% of the course grade.

In this assignment, you will work in Haskell with a data structure given by a recursive data type and an abstract data type from the standard library, and you will write interesting recursive functions over the recursive data type.

As usual, you should also aim for reasonably efficient algorithms and reasonably lucid code.

Quadtrees

A quadtree is a tree data structure in which each node has 0 or 4 children. We will use this to represent a square grayscale image with possible subdivision into 4 square quadrants, recursively. To this end, our Haskell quadtree is defined as:

```
data Quadtree = QNode Int Int Int Word8 QKids
data QKids = Q0 | Q4 Quadtree Quadtree Quadtree Quadtree
```

The 3 'Int' fields stand for the (x, y) coordinates of the top-left corner of the square and the width. The 'Word8' field (a byte) is a grayscale value between 0 and 255, and is the rounded average over the square region.

Note 1: Following image convention, the y-axis points down, not up.

Note 2: When there are 4 children, the order is: top-left, bottom-left, top-right, bottom-right.

A quadtree could be a compressed (possibly lossy) representation of an image by applying cutoff conditions: a cap on tree depth, or do not subdivide if the maximum grayscale difference in grayscales is sufficiently small.

Example: Here is a 4x4 image (in matrix notation to show the actual grayscale values):

$$\begin{bmatrix} 0 & 0 & 1 & 3 \\ 0 & 0 & 3 & 1 \\ 2 & 4 & 10 & 11 \\ 6 & 2 & 11 & 11 \end{bmatrix}$$

```
array ((0 ,0) , (3 ,3)) [
((0,0),0), ((0,1),0), ((0,2),2), ((0,3),6),
((1,0),0), ((1,1),0), ((1,2),4), ((1,3),2),
((2,0),1), ((2,1),3), ((2,2),10), ((2,3),11),
((3,0),3), ((3,1),1), ((3,2),11), ((3,3),11) ]
```

Here is a quadtree represention with depth capped to 1, so we only subdivided into 2x2 blocks, each 2x2 block bearing only its average grayscale:

```
QNode 0 0 4 4 (Q4
  (QNode 0 0 2 0 Q0)
  (QNode 0 2 2 4 Q0)
  (QNode 2 0 2 2 Q0)
  (QNode 2 2 2 11 Q0))
```

If there is no cap on depth (or the depth cap is ≥ 2 so it doesn't matter for 4x4), but instead we stop subdividing when the maximum grayscale difference is ≤ 2, then only the bottom-left 2x2 block is further split into 1x1 blocks:

```
  QNode 0 0 4 4  (Q4
    (QNode 0 0 2 0 Q0)
    (QNode 0 2 2 4 (Q4
      (QNode 0 2 1 2 Q0)
      (QNode 0 3 1 6 Q0)
      (QNode 1 2 1 4 Q0)
      (QNode 1 3 1 2 Q0)))
    (QNode 2 0 2 2 Q0)
    (QNode 2 2 2 11 Q0))
```

If we use the latter quadtree to reconstruct the image, it becomes

$$\begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 2 & 4 & 11 & 11 \\ 6 & 2 & 11 & 11 \end{bmatrix}$$

**array ((0 ,0) , (3 ,3)) [
((0,0),0), ((0,1),0), ((0,2),2), ((0,3),6),
((1,0),0), ((1,1),0), ((1,2),4), ((1,3),2),
((2,0),2), ((2,1),2), ((2,2),11), ((2,3),11),
((3,0),2), ((3,1),2), ((3,2),11), ((3,3),11)]**

There is more information about quadtrees (not necessarily relevant to this assignment) in the Wikipedia quadtree entry.

Image in Array

Grayscale images in this assignment are represented by the 'Array' type from the standard library (imported from 'Data.Array').

The 'Array' type takes two type parameters: one for index type, one for element type. In this assignment, the whole type is 'Array (Int, Int) Word8', with '(Int, Int)' for 2-dimension indexes (x, y), and 'Word8' for grayscale elements.

By way of examples, the first 4x4 image example could be coded as:

```
pic1 :: Array (Int,Int) Word8
pic1 = array ((0,0), (3,3)) [
    ((0,0),0), ((0,1),0), ((0,2),2), ((0,3),6),
    ((1,0),0), ((1,1),0), ((1,2),4), ((1,3),2),
    ((2,0),1), ((2,1),3), ((2,2),10), ((2,3),11),
    ((3,0),3), ((3,1),1), ((3,2),11), ((3,3),11)   ]
-- List order does not matter because indexes are explicit.
```

or another way:

```
pic1 :: Array (Int,Int) Word8
pic1 = listArray ((0,0), (3,3)) [0,0,2,6, 0,0,4,2, 1,3,10,11, 3,1,11,11]
-- List order matters because indexes are implicitly (0,0), (0,1), ...
```

In this assignment, you may assume that all images are squares, and furthermore widths (and heights) are always powers of 2 (2^k). As for the starting index: For simplicity, do *not* assume that the top-left coordinates are $(0, 0)$—that's right, your job is simpler with arbitrary top-left coordinates! Note that each array knows its index range, and you can query with the 'bounds' function.

The Problems

1. [8 marks] Implement reconstruction of image from quadtree:

   ```
   quadtreeToPic :: Quadtree -> Array (Int, Int) Word8
   ```

2. [8 marks] Implement building of quadtree from image:

   ```
   picToQuadtree :: Word8                       -- threshold
                 -> Int                         -- depth cap
                 -> Array (Int, Int) Word8      -- image
                 -> Quadtree
   ```

   Do not subdivide if: maximum grayscale difference $\leq$ threshold, or depth cap hits 0.

End of questions.