Recitation 2 – *Practice with Sequences*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2015)

January 20th, 2015

1 Announcements

- Lab 0 – *SignalLab* is due on Monday, January 26.
- Lab 1 – *SkylineLab* will be released on Monday, January 26.
- Library Documentation for 15210 can be found at `http://www.cs.cmu.edu/~15210/`

2 Warmup

The following functions should take close to one line to implement. Fill in the implementations, then answer the questions about the work and span of your implementations.

1. `divisors`
   Input: (n : int)
   Output: $\langle x \in [n] \mid n \bmod x = 0 \rangle$

```
fun divisors(n : int) : (int seq) =
```

2. `alternatingSum`
   Input: (S : int seq)
   Output: $\sum_{i=0}^{n-1} (-1)^i S_i$
   We will be doing this in two ways, once using `iter` and once using `reduce`

```
fun alternatingSumIter(S : int seq) : int =
```

```
fun alternatingSumReduce(S : int seq) : int =
```

3. Write `filter` in terms of `flatten`

```
fun filterViaFlatten (p:'a -> bool) (S:'a seq) : ('a seq)=
                                                  _____
```

What are the assymptotic complexities for the work and span of the functions?

`divisors` : W(n) = _____ S(n) = _____

`alternatingSumIter` : W(n) = _____ S(n) = _____

`alternatingSumReduce` : W(n) = _____ S(n) = _____

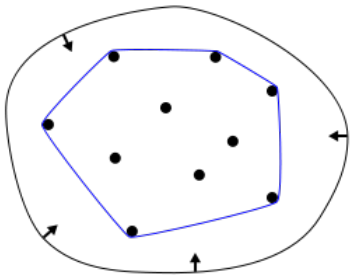
`filterViaFlatten` : W(n) = _____ S(n) = _____

3 Exercise: Sorting via Convex Hull

A *convex set*, C, is a set of points such that $\forall x, y \in C$, every point p in the line segment connecting x and y is also in C. The *convex hull* of a set of points P is the the smallest convex set that contains every element of P. You can think of it as if you had pegs at every point in P and wanted to stretch a rubber band around around them.

To compute the convex hull of a set of points, it is enough to output a non-ambiguos representation of the convex hull. In this case, we will output a sequence of the verticies that define the envelope of the convex hull.

NOTE: We will ignore inputs where any three point are collinear. Different implentations deal with them in different ways (include, ignore, alert user, undefined behavior). In particular, as with many computational geometry problems, floating point errors can make it difficult to decide what the output should be.



http://en.wikipedia.org/wiki/Convex_hull

Task 1: You are given a black-box function

```
fun convexHull(P : (int * int) seq) : (int*int) seq
```

which evaluates to the sequence of verticies on the envelope of the convex hull of P. You may assume that the sequence starts with the leftmost vertex (if there are more than one, then it starts with the bottom one) and traverses counter-clockwise.

For example, if $P = \langle(3,0),(0,3),(1,1),(0,0)\rangle$, then `convexHull(P)` $= \langle(0,0),(3,0),(0,3)\rangle$.

Design an algorithm that uses `convexHull` to sort a sequence of integers. Then determine that function's work and span in terms of the work and span of `convexHull`, $W_{CH}(n)$ and $S_{CH}(n)$.

```
fun sortViaConvexHull (S : int seq) : (int seq) =
  let

    _____________________________________________

    _____________________________________________

    _____________________________________________
  in

    _____________________________________________
  end
```

W(n) = _____ S(n) = _____

Task 2: There are multiple ways to define the convex hull of a set of points. How can we accomodate the following new conditions? Write the pseudocode for your ideas.

- The points in the output sequence are given in clockwise order rather than counter-clockwise order.
- There are no guarantees about the starting point in the output. In our previous example, this would mean that $\langle(0,0),(3,0),(0,3)\rangle$ is just as likely to be the ouptut as $\langle(3,0),(0,3),(0,0)\rangle$ and $\langle(0,3),(0,0),(3,0)\rangle$.

4 Bonus: Convex Hulls

Implement `convexHull`. It is a little more involved than the rest of the recitation, but very doable. If you are interested in learning more about it, there is an entire wikipedia page on convex hull algorithms: `http://en.wikipedia.org/wiki/Convex_hull_algorithms`.