Recitation 4 – *Scan Reloaded and Reductions*

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

February 3rd, 2015

1 Announcements

- How did Skyline go?
- Bignum is out—go go go!
- Questions about homework or lecture?

2 Scan Implementation Review

Scan is a complex operation, so we're going to work through one level of recursion.

Let $S = \langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$. We'll look at `scan op+ 0 S`.

First, `scan` contracts the sequence to a new S' by contracting every other pair of elements, giving us $S' = \langle$___, ___, ___, ___$\rangle$.

Then, a recursive call to `scan op+ 0 S'` will return:

($\langle$___, ___, ___, ___$\rangle$,___)

We then interleave values from S into S', giving us the final scan of

($\langle$___, ___, ___, ___, ___, ___, ___, ___$\rangle$,___)

Note that this particular sequence is much easier to scan than certain other sequences—why?

3 Parentheses Matching

3.1 Parentheses Pairing

We have a sequences of fully-matched open and close parentheses (every parens has exactly 1 proper match). However, each parens is lonely and wants to know where its matching parens is. Help the parens find their matches!

For example, the input:`<(,(,),),(,)>` produces the output: `<(0,3),(4,5),(1,2)>`.

Hint: How do you know if two parens are matched up? For each parens, you'll need to mark how many matched parens are enclosing it.

```
datatype paren = OP | CP

fun matchingPairs (S : paren seq) : (int * int) seq =
```

What is the work and span of your algorithm?

3.2 Parentheses Distance

Now that all of these parens are happily aware of where their partner is, we want to look at how happy the entire sequence of parens are. Because these parens are quite adventurous, the happiness of a sequence of parens depends on the total distance between matching parens. For example, `<(,(,),)>` would have a happiness of 2 while `<(,),(,)>` would have a happiness of 0. Using the output from matchingPairs, write a 1-line function that solves for the happiness of a parens sequence. Assume that `sort` is a stable sort and that the output order of the tuples of pairings doesn't matter.

```
fun parensDist (S : (int * int) seq) : int =
```

4 Minimum Gap Problem

You have been hired to choreograph stunts for the Pittsburgh Steelerettes - the cheerleading team for the Pittsburgh Steelers! In order to win the next Steelers' game, you need to pull off the best stunt ever and throw your flyer (person being thrown) as high as she can go. The clever choreographer that you are, you know that in order to optimize the height of the throw, you need to have two base (throwers) who are as close in height as possible.

Given a sequence of cheerleader heights in increasing order, find the minimum gap (smallest difference) between two cheerleaders. Implement a divide-and-conquer algorithm to solve this problem and win the game!

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

5 BignumLab Starter!

We have two input sequences X and Y, both of which are bit sequences where `datatype bit = ZERO | ONE`. We say that X and Y are non-negative, and they are represented from the least significant bit to the most, i.e. `nth X 0` is the least significant bit of X. Let's sequentially compute the sum of X and Y.

```
fun sum (X, Y) =
```