## Recitation 4

## Scan Reloaded

### 4.1 Announcements

- BignumLab has been released, and is due Thursday night. It's worth 175 points.
- RandomLab will be released on Thursday.


### 4.2 Implementation

Recall the implementation of scan for sequences of power-of-2 length. Note that we typically refer to line 7 as the contraction step, line 8 as the recursive step, and line 11 as the expansion step.

```
Algorithm 4.1. scan, assuming \(|S|\) is a power of 2.
fun scan \(f b S=\)
        case \(|S|\) of
            \(0 \Rightarrow(\rangle, b)\)
        \(\mid 1 \Rightarrow(\langle b\rangle, S[0])\)
        \(n \Rightarrow\)
            let
                val \(S^{\prime}=\langle f(S[2 i], S[2 i+1]): 0 \leq i<n / 2\rangle\)
                val \((R, t)=\operatorname{scan} f b S^{\prime}\)
                fun \(P(i)=\) if even \((i)\) then \(R[i / 2]\) else \(f(R[\lfloor i / 2\rfloor], S[i-1])\)
            in
                \((\langle P(i): 0 \leq i<n\rangle, t)\)
            end
```

A diagram should help clear up any confusion. Consider (scan $+0\langle 1,2,3,4,5,6,7,8\rangle$ ).


### 4.3 Cost Analysis

Since we so commonly use scan with a constant-time function argument, it is helpful to memorize that it has $O(n)$ work and $O(\log n)$ span in this case. But what about more complex functions? Let's try merge as an example.

Task 4.2. Analyze the work and span of

$$
\operatorname{scan}(\text { merge cmp) }\rangle S
$$

assuming that $|S|=n,|x| \leq m$ for every $x \in S$, and cmp is constant-time. Give your answers as tight Big-O bounds in terms of $n$ and $m$.

Recall that (merge cmp $(A, B)$ ) requires $O(|A|+|B|)$ work and $O(\log |A|+\log |B|)$ span, and it produces a sequence of length $|A|+|B|$.

### 4.4 Bonus Exercise: Factorials with Bignums

In this section, we write $* *$ for bignum multiplication and $\bar{x}$ for the bignum representation of $x$. We'll be using the same conventions here as in BignumLab.

Factorials quickly become too large to represent in a single 32-bit or 64-bit unsigned integer. ${ }^{1}$ This makes them the perfect candidate for bignums, which can be arbitrarily large. Consider the following code, which computes the first $n$ factorials (excluding 0 !):

## Algorithm 4.3. Bignum Factorials.

```
fun factorials n = Seq.scanIncl ** \overline{1}\langle\overline{i}:1\leqi\leqn\rangle
```

Exercise 4.4. Analyze the work of (factorials n). Note that you'll first need to determine

1. The work of $\bar{x} * * \bar{y}$, and
2. The bit width of $\bar{x} * * \bar{y}$.

The former is given by solving the recurrence given in BignumLab for multiplication, namely

$$
W(n)=3 W\left(\frac{n}{2}\right)+O(n)
$$

The latter can be determined via a little bit of algebra. Note that the bit width of a number $\bar{x}$ is $1+\left\lfloor\log _{2} x\right\rfloor$, assuming $x \geq 1$.

[^0]Built: September 21, 2015


[^0]:    ${ }^{1}$ With 32-bit unsigned integers, the largest factorial we can compute before encountering overflow is 11 !. For 64-bits, it's 19!.

