# Recitation 4

# Scan Reloaded

## 4.1 Announcements

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

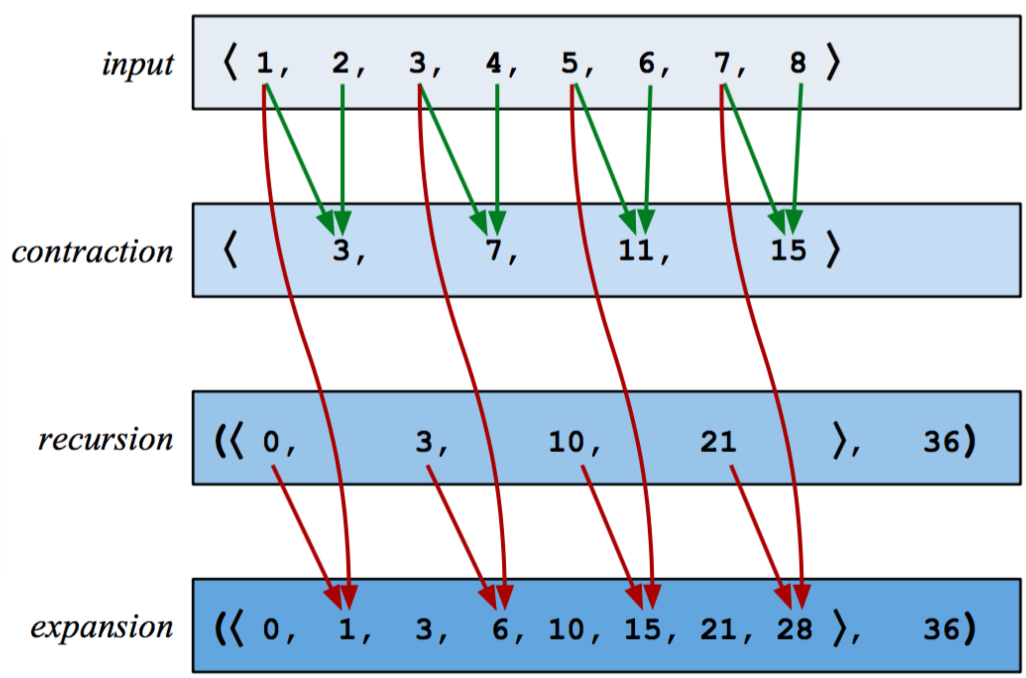
## 4.2 Implementation

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.*

```
1   fun scan f b S =
2     case |S| of
3       0 ⇒ (⟨ ⟩, b)
4     | 1 ⇒ (⟨b⟩, S[0])
5     | n ⇒
6         let
7           val S' = ⟨f(S[2i], S[2i+1]) : 0 ≤ i < n/2⟩
8           val (R, t) = scan f b S'
9           fun P(i) = if even(i) then R[i/2] else f(R[⌊i/2⌋], S[i−1])
10        in
11          (⟨P(i) : 0 ≤ i < n⟩, t)
12        end
```

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



## 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*

$$\texttt{scan (merge cmp)}\ \langle\,\rangle\ S$$

*assuming that $|S| = n$, $|x| \le m$ 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 $|A| + |B|$.

Our goal is to establish two recurrences $W(n, m)$ and $S(n, m)$. Let's walk through the steps:

- *Contraction.* We perform $n/2$ applications of `merge`, each of which requires $O(m)$ work and $O(\log m)$ span. Therefore this step requires $O(nm)$ work and $O(\log m)$ span.
- *Recursion.* We recurse on a sequence of half the length. Each element in this sequence is now twice as large. Therefore this step requires $W(n/2, 2m)$ work and $S(n/2, 2m)$ span.
- *Expansion.* Consider the even and odd positions of the output separately.
  - The even positions remain unchanged from the recursive result; copying them over to the output incurs a cost of $O(n)$ work and $O(1)$ span.
  - The odd positions are determined by $n/2$ applications of `merge`. The inputs to these calls, however, are of varying size. Specifically, the `merge` which generates the $(2i+1)^{\text{th}}$ position has inputs of size $2im$ and $m$, and therefore requires $O((i+1)m)$ work and $O\big(\log((i+1)m)\big)$ span. We add these up for $0 \le i < n/2$:
    * Work:
    $$\sum_{i=0}^{n/2-1} O((i+1)m) = O\left(m \sum_{j=1}^{n/2} j\right) = O(n^2 m)$$
    * Span:
    $$\operatorname*{MAX}_{i=0}^{n/2-1}\ O\big(\log((i+1)m)\big) = O(\log(nm))$$

  Therefore this step requires a total of $O(n^2m)$ work and $O(\log(nm))$ span.

We now have two recurrences to solve.

- *Work*: $W(n, m) = W(n/2, 2m) + O(n^2 m)$.

  Counting from $i = 0$ at the top, the $i^{\text{th}}$ level of this recurrence has a cost of

  $$O\left(\left(\frac{n}{2^i}\right)^2 2^i m\right) = O\left(\frac{n^2 m}{2^i}\right)$$

  and therefore this recurrence is root dominated, giving us that

  $$W(n, m) \in O(n^2 m).$$

- *Span*: $S(n, m) = S(n/2, 2m) + O(\log(nm))$.

  The $i^{\text{th}}$ level of this recurrence has a cost of

  $$O\left(\log\left(\frac{n}{2^i} 2^i m\right)\right) = O(\log(nm))$$

  and therefore this recurrence is balanced. There are $\log_2 n$ levels, giving us that

  $$S(n, m) \in O(\log n \cdot \log(nm)) = O(\log^2 n + \log n \cdot \log m).$$

## 4.4 Bonus Exercise: Factorials with Bignums

In this section, we write $**$ for bignum multiplication and $\overline{x}$ 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 ** ` $\overline{1}$ $\langle \overline{i} : 1 \leq i \leq n \rangle$

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

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

*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* $\overline{x}$ *is* $1 + \lfloor \log_2 x \rfloor$, *assuming* $x \geq 1$.

*Warning: this is pretty hard.*

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

.