15-150 Assignment 7

Karan Sikka
ksikka@andrew.cmu.edu
G
April 4, 2012

1: Task 4.1

Tabulate performs an operation n times, where n is the sum of the lengths of `s1:'a Seq.seq` and `s2:'a Seq.seq`. Each operation is constant time, since in the `true` case, `nth` is const time and in the `false` case, `nth` and `length` are const time. Since a constant time operation is performed n times, myAppend has work of $\mathbf{O(n)}$ where n is the sum of the lengths of the sequences.

2: Task 4.2

Tabulate performs an operation n times, where n is the sum of the lengths of `s1:'a Seq.seq` and `s2:'a Seq.seq`. Note that these operations may be done in parallel. Each operation is constant time, since in the `true` case, `nth` is const time and in the `false` case, `nth` and `length` are const time. Since tabulate has span of $\mathbf{O(1)}$ and each operation is const time, myAppend has span of $O(1)$.

3: Task 4.3

If we draw the mapreduce tree, we can see that for each of the $log(n)$ levels, myAppend does n total work. Singleton is const time, and happens n times. Seq.empty is also const time, and happens fewer than n times. Therefore, the work is tightly upper bounded by $\mathbf{O(nlog(n))}$. This differs from the other reverse function because in that function, a constant time operation occurred n times. Here a linear operation occurs $log(n)$ times.

4: Task 4.4

The span is the span of the longest branch of the mapreduce tree. The height of the mapreduce tree is proportional to $log(n)$, and the span of myAppend is constant, so the span of of `reverse'` is $O(log(n))$. Since `reverse'` uses mapreduce, there is a dependency on previous operations to finish whereas in `reduce`, each tabulate operation can be computed independently

5: Task 4.5

Suffixes has a tight upper bound of $\mathbf{O(n^2)}$ because it evaluates Seq.length once (const time) and Seq.drop, which has linear work, n times.

6: Task 4.6

`Seq.drop` has a span of $O(1)$, and tabulate has a span of $O(1)$ for a constant time operation. Therefore `suffixes` has a span of $\mathbf{O(1)}$.

7: Task 4.7

Seq.zip does linear work, and suffixes does linear work. Both functions are called once. Therefore the overall work of withSuffixes is $\mathbf{O(n)}$.

8: Task 4.8

The span of suffixes is $O(1)$. After this executes, Seq.zip is called, which has a span of $O(1)$. All in all, withSuffixes has a span of $\mathbf{O(1)}$.

9: Task 4.9

First, `maxS` is applied to each sequence in the outer sequence. Since mapS makes a call to reduce using the comparator Int.max, the total work is $O(nk_i)$ where k_i is $max(k_1...k_n)$. Then maxS is called on the resulting sequence of the previously operation, which has length n. Since $O(n)$ is a lower bound than the previous bound, the overall bound for the work is $\mathbf{O(nk_i)}$.

10: Task 4.10

First, `maxS` is applied to each sequence in the outer sequence. Since mapS makes a call to reduce using the comparator Int.max, the total span is $O(log(k_i))$ where k_i is $max(k_1...k_n)$. Then maxS is called on the resulting sequence of the previously operation, which has length n and span $O(log(n))$. Since this step depends on the first, the span is $O(log(n) + log(k_i))$ which can be rewritten as $\mathbf{O(log(nk_i))}$.

11: Task 4.11

First, `withSuffixes` is called, and as previously determined, this has work of $O(n)$, where n is the length of s. Then, the map within a map happens. n times, an n-times map occurs. Therefore the work of this is $O(n^2)$. The result is an n-length sequence containing sequences of length at most n. As previously found, the work of `maxAll` is $O(nk_i)$, which is $O(n^2)$ in this case. Since these steps happen sequentially, the work is bounded by the largest time bound, which is $\mathbf{O(n^2)}$.

12: Task 4.12

First, `withSuffixes` is called, and as previously determined, this has span of $O(1)$. Then, the map within a map happens. The inner map has span of O(1) since the operation is constant time, and therefore the outer map is also O(1). The overall span for this step is O(1). Then, for `maxAll`, we know the span of this is $O(log(nk_i))$, or in this case, $O(log(n^2)) = O(2log(n)) = \mathbf{O(log(n))}$. Since these steps happen sequentially, the span is bounded by the sum of these spans, which is $\mathbf{O(log(n))}$.