15-150 Assignment 3

Nigel Nderi
ncn@andrew.cmu.edu
Section I
2/6/2018

3: Baa Baa Black Sheep

Task 3.2

```
fun addToEach (xs : int list, n : int) : int list =
  case xs of
    nil => nil
  | x :: xs' => x + n :: addToEach (xs', n)
                   .......
fun prefixSum (xs : int list) : int list =
  case xs of
       nil => nil
     | (x :: xs') =>
         (case addToEach(xs', x)  of
              nil => [x]
            | y :: ys => x :: prefixSum((y :: (addToEach(ys, ~x))))
         )
```

prefixSum has 1 argument: an int list that we will refer to as xs. Let n = the length of xs.
$W(n) = O(n^2)$

Task 3.4

```
fun prefixSumFast (nil : int list) : int list = nil
  | prefixSumFast( [x : int]) = [x]
  | prefixSumFast(x :: xs) =
    let
      val (y :: ys) = xs
    in
      x :: prefixSumFast((y + x) :: ys)
    end
```

prefixSumFast takes in 1 argument: an int list that we'll refer to as xs. Let n = the length of xs.
$W(n) = O(n)$

6: Work It Out

Task 6.1

listMax has 1 argument: a list of length n.

$$
\begin{aligned}
W(n) &= c_0 && \text{(n = 0)}\\
W(n) &= c_1 + W(n-1) && \text{(n > 0; assuming Int.max runs at constant time)}\\
&= c_1 + c_1 + W(n-2)\\
&= \sum_{i=1}^{n} c_1 + c_0 && \text{([1, n] contains n ints in; assuming listMax terminates)}\\
&= nc_1 + c_0\\
&= O(n)
\end{aligned}
$$

At each step, there is a constant time evaluation followed by calling listMax again with a list of size n - 1 until the list is of size 0.

Task 6.2

listMax has 1 argument: a list of length n.

$$
\begin{aligned}
S(n) &= c_0 && \text{(n = 0)}\\
S(n) &= c_1 + S(n-1) && \text{(n > 0; assuming Int.max runs at constant time)}\\
&= c_1 + c_1 + S(n-2)\\
&= \sum_{i=1}^{n} c_1 + c_0 && \text{([1, n] contains n ints in; assuming listMax terminates)}\\
&= nc_1 + c_0\\
&= O(n)
\end{aligned}
$$

At each step, you evaluate the first element of the list followed by calling listMax again with a list of size n - 1 until the list is of size 0.

Task 6.3

treeMax has 1 argument, a tree T with n nodes.

$$
\begin{aligned}
W(n) &= c_0 && \text{(n = 0)}\\
W(n) &= c_1 + 2W(\frac{n}{2}) && \text{(n > 0; assuming max runs at constant time)}\\
&= c_1 + 2(c_1 + 2W(\frac{n}{4}))\\
&= 3c_1 + 4W(\frac{n}{8})\\
&= \sum_{i=1}^{logn}((2^i - 1)c_1) + c_0\\
&= (2^{logn} - 1 - nlogn)c_1 + c_0 = 2^{logn}c_1 - nlognc_1 + c_0 - c_1\\
&= O(nlogn)
\end{aligned}
$$

At each iteration, there is a constant time evaluation from the 2 Int.max calls and then two calls of treeMax of a tree of size n/2 until you reach 0.

Task 6.4
treeMax has 1 argument, a tree T with n nodes.

$$\begin{aligned}
S(n) &= c_0 && \text{(n = 0)}\\
S(n) &= c_1 + W(\frac{n}{2}) && \text{(n > 0; assuming max runs at constant time)}\\
&= c_1 + (c_1 + W(\frac{n}{4}))\\
&= 2c_1 + W(\frac{n}{8})\\
&= \sum_{i=1}^{logn}(c_1) + c_0\\
&= logn * c_1 + c_0\\
&= O(logn)
\end{aligned}$$

At each iteration, there is a constant time evaluation from the 2 Int.max calls and then two calls of treeMax of a tree of size n/2 until you reach 0.

Task 6.5
treeMax takes in 1 argument: a tree with depth d

$$\begin{aligned}
W(d) &= k_0 && \text{(d = 0)}\\
W(d) &= k_1 + 2W(d-1) && \text{(d > 1)}\\
&= k_1 + 2k_1 + 4W(d-2)\\
&= 3k_1 + 4k_1 + 8W(d-3)\\
&= \sum_{i=1}^{d}((2^i - 1)k_1) + k_0\\
&= (2^{d+1} - 2 - \frac{d^2+d}{2})k_1\\
&= O(2^d)
\end{aligned}$$

Task 6.6
treeMax takes in 1 argument: a tree with depth d

$$\begin{aligned}
S(d) &= k_0 && \text{(d = 0)}\\
S(d) &= k_1 + W(d-1) && \text{(d > 1)}\\
&= k_1 + k_1 + W(d-2)\\
&= 2k_1 + k_1 + W(d-3)\\
&= \sum_{i=1}^{d}(k_1) + k_0\\
&= dk_1\\
&= O(d)
\end{aligned}$$

 Nigel Nderi
ncn@andrew.cmu.edu

7: Proofs

(Option 1)

```
14 fun heads (x : int, nil : int list) : int = 0
15   | heads(x, xs) = ( let
16                                    val (y :: ys) = xs
17                                  in
18                                      case  x = y of
19                                          true  => 1 + heads(y, ys)
20                                        | false => 0
21                                    end)
                    ........
33 fun tails (x : int, xs : int list) : int list =
34   case heads(x, xs) of
35        0 => xs
36     | _ => (let
37                 val (y :: ys) = xs
38                in
39                  tails(x, ys)
40                 end
41                 )
```

Th$^{\mathbf{m}}$: Let L be an int list and x be some random integer. heads(x, L) + length(tails(x, L)) ≅ length(L)

Base Case: $L = nil \implies length(L) = 0 \wedge heads(x, L) = 0 \wedge (length(tails(x, L)) = length(nil) = 0) \implies 0 + 0 = 0$ which is true so the base case holds

Inductive Hypothesis: Let L = y :: ys and x be some arbitrary integer. heads(x, ys) + length(tails(x, L)) = length(L) - 1

Induction Step:

Case 1: x isn't the first element of L

$$\begin{aligned}
heads(x, L) + length(tails(x, L)) &= 0 + length(tails(x, L)) && \text{(clause 1 of heads)}\\
&= 0 + length(L) && \text{(case 1 of tails)}\\
&= length(L) && \text{(math)}
\end{aligned}$$

Case 2: x is the first elemnt of L

$$\begin{aligned}
heads(x, y :: ys) &= 1 + heads(x, ys) && \text{(line 18)}\\
heads(x, L) + length(tails(x, L)) &= 1 + heads(x, ys) + length(tails(x, L) && \text{(substitution)}\\
&= 1 + length(L) - 1 && \text{(Induction Hypothesis)}\\
&= length(L) && \text{(math)}
\end{aligned}$$

(Option 2)

```
fun swap_up (Empty : tree, right : tree) : tree = right
  | swap_up (Node(L, x, R), right) = Node(L, x, swap_up(R, right))
                                          .............
fun remove_evens (Empty : tree) : tree = Empty
  | remove_evens(Node(L, x, R)) =
        case (x mod 2) of
             0 => remove_evens(swap_up(L, R))
           | _ => Node(remove_evens(L), x, remove_evens(R))
```

Th$^{\mathbf{m}}$: Let T be a tree. preorder (`remove_evens` T) $\cong$ `remove_evens_list` (preorder T)

Base Case: T = Empty $\implies$ (preorder (`remove_evens` T) = preorder Empty = nil) $\wedge$ (`remove_evens_list` (preorder T) = `remove_evens_list` nil = nil) $\implies$ nil = nil which is true; base case holds

Induction Hypothesis:

Induction Step:

Case 1: T contains no even nodes

$$
\begin{aligned}
preorder(remove\_evens(T)) &= preorder(T) && \text{(second case of remove\_evens)} \\
remove\_evens\_list(preorder(T)) &= preorder(T) && \text{(second case of remove\_evens\_list)}
\end{aligned}
$$