Work and Span Lab (Solution)

`workspan_lab_sol`

Principles of Functional Programming — Summer 2023

Contents

1 Work/Span Lab **2**

1.1 Work and Span . . . 2

1.1.1 Tree Method for Finding Work/Span . . . 2

1.1.2 Writing and Solving Recurrences . . . 3

Task 1.1. . . . 3

1.1.3 Common Recurrences . . . 3

1.1.4 Common Summations . . . 5

1.2 Work and Span for Trees . . . 6

1.2.1 Balanced vs. Unbalanced . . . 6

1.2.2 Nodes vs. Depth . . . 6

Task 1.2. . . . 6

Task 1.3. . . . 7

2 Mult **9**

2.1 Analyzing `mult` . . . 9

Task 2.1. . . . 9

Task 2.2. . . . 9

Task 2.3. . . . 9

3 Trees **11**

Task 3.1. . . . 11

Task 3.2. . . . 11

Task 3.3. . . . 11

Task 3.4. . . . 12

4 Work It Out **13**

Task 4.1. . . . 13

Task 4.2. . . . 13

Task 4.3. (Recommended) . . . 14

Task 4.4. (Recommended) . . . 14

Task 4.5. . . . 15

Task 4.6. . . . 15

Task 4.7. (Recommended) . . . 17

Task 4.8. . . . 17

Task 4.9. (Recommended) . . . 18

Task 4.10. . . . 18

Task 4.11. . . . 18

Task 4.12. . . . 19

1 Work/Span Lab

1.1 Work and Span

- **Work:** The number of steps to sequentially evaluate code.
- **Span:** Assuming infinite processors, the number of steps to evaluate code in parallel.

We can express and solve for the work and span of recursive functions with *recurrences*. One way to gain intuition about the summation form of a recurrence is the *tree method* [1].

1.1.1 Tree Method for Finding Work/Span

Tree Method:

1. Write the recurrence - base case and recursive case, if applicable.
2. Identify the number of levels (L), nodes at level i (n_i), and the non-recursive work/span per node at level i (w_i).
3. Solve the summation $\sum_{i=0}^{L} n_i * w_i$.
4. Use this summation to find the big-O bound.

[1] While it may seem easy to demonstrate recurrence bounds by induction, these proofs are trickier than they seem! The nature of a bound makes such proofs hard to write correctly, so we strongly recommend you use the tree method on assignments and exams.

1.1.2 Writing and Solving Recurrences

Task 1.1.

```
fun one 0 = 1
  | one n = one (n div 2)
```

$W_{\texttt{one}}(0) =$

$W_{\texttt{one}}(n) =$

$W_{\texttt{one}}(n) \in O(\qquad\qquad)$

```
fun min [] = 0
  | min (x::xs) =
      Int.min (x, min xs)
```

$W_{\texttt{min}}(0) =$

$W_{\texttt{min}}(n) =$

$W_{\texttt{min}}(n) \in O(\qquad\qquad)$

Solution 1.

One

$$W_{\texttt{one}}(0) = k_0$$
$$W_{\texttt{one}}(n) = W_{\texttt{one}}\left(\frac{n}{2}\right) + k_1$$

Since every recursive call's input size is halved, there are $\log n$ levels. There is one node per level, each with cost k_1. Thus, the total cost is:

$$W_{\texttt{one}}(n) = \sum_{i=0}^{\log n} k_1 = k_1(\log n + 1) \in O(\log n)$$

Min

$$W_{\texttt{min}}(0) = k_0$$
$$W_{\texttt{min}}(n) = W_{\texttt{min}}(n-1) + k_1$$

Since each recursive call's input size is reduced by one, `min` has n levels. There is one node per level, each with cost k_1. Thus, the total cost is:

$$W_{\texttt{min}}(n) = \sum_{i=0}^{n} k_1 = k_1(n+1) \in O(n)$$

1.1.3 Common Recurrences

Below is a table of asymptotic bounds on common recurrences.

Recurrence	Asymptotic Upper Bound
$T(n) = T(n-1) + c$	$O(n)$
$T(n) = T(n/2) + c$	$O(\log n)$
$T(n) = 2T(n/2) + c$	$O(n)$
$T(n) = T(n/2) + c_1 n + c_0$	$O(n)$
$T(n) = 2T(n/2) + c_1 n + c_0$	$O(n \log n)$
$T(n) = T(n-1) + c_1 n + c_0$	$O(n^2)$
$T(n) = 2T(n-1) + c$	$O(2^n)$

Note: You may not simply cite the table of recurrences given in lab to justify the work and span bounds you derive on the homework or exams. While the table may be helpful for double checking that your work is indeed correct, you must ***always*** show the steps you take to get from your recurrence to your final bound.

1.1.4 Common Summations

Here are some formulas that you may find useful when trying to solve summations:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \tag{1}$$

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \tag{2}$$

$$\sum_{i=0}^{\log n} \frac{1}{2^i} \leq \sum_{i=0}^{\infty} \frac{1}{2^i} = 2 \in O(1) \tag{3}$$

$$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a} \qquad \text{(generalization of 1)} \tag{4}$$

1.2 Work and Span for Trees

1.2.1 Balanced vs. Unbalanced

You may only assume the tree is balanced if it is stated in the problem statement. Otherwise, you must analyze the cost bounds given the worst-case scenario.

1.2.2 Nodes vs. Depth

When the input to a function is a tree, we can analyze the work and span through either the number of nodes in the tree, n, or the depth of the tree, d.

Task 1.2.

For the following questions, consider both input sizes (number of nodes, n, and depth, d).

Given a "standard" tree function (i.e. one that makes two recursive calls on the left and right subtree), what would be the work recurrences for a balanced tree?

What about a maximally unbalanced tree (e.g. each node only has children in its left subtree)?

Solution 2.

When a tree is balanced, we know that half the remaining nodes are in the left subtree and the other half are in the right subtree.

Thus, for balanced trees, we will commonly get recurrences that look like

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

$$W(d) = 2W(d-1) + \cdots$$

where n is the number of nodes in our tree and d is the depth of our tree.

When a tree is maximally unbalanced, all the remaining nodes are in one of the subtrees (e.g. all remaining nodes are in the left subtree).

Thus, we will commonly get recurrences that look like

$$W(n) = W(n_l) + W(n_r) + \cdots = W(n-1) + \cdots$$

$$W(d) = W(d_l) + W(d_r) + \cdots = W(d-1) + \cdots$$

where n is the number of nodes in our tree, and n_l and n_r are the number of nodes in the left and right subtrees, and similarly d is the depth of our tree and d_l and d_r are the depths of the left and right subtrees.

Note: a maximally unbalanced tree is not always the worst case for an unbalanced tree. The worst case depends on the structure of the function. In this course, however, you will likely only need to find the work/span for a maximally unbalanced tree.

Task 1.3.

```
datatype tree = Empty | Node of tree * int * tree

fun insert (Empty : tree, n : int) : tree = Node (Empty, n, Empty)
  | insert (Node (L, x, R) : tree, n : int) : tree =
      if n <= x then insert (L, x)
      else insert (R, x)
```

What is the work and span of `insert` in terms of the number of nodes in the tree, n? Assume that the tree is balanced.

What are the work and span of `insert` in terms of the depth of our tree, d? Do not assume that the tree is balanced.

Solution 3.

Balanced Tree

$$W_{\texttt{insert}}(0) = k_0$$
$$W_{\texttt{insert}}(n) = W_{\texttt{insert}}\left(\frac{n}{2}\right) + k_1$$

$$S_{\texttt{insert}}(0) = k_2$$
$$S_{\texttt{insert}}(n) = S_{\texttt{insert}}\left(\frac{n}{2}\right) + k_3$$

Since every recursive call's input size is halved, there are $\log n$ levels. There is one node per level, each with work cost k_1 or span cost k_3. Thus, the total cost is:

$$W_{\texttt{insert}}(n) = \sum_{i=0}^{\log n} k_1 = k_1(\log n + 1) \in O(\log n)$$
$$S_{\texttt{insert}}(n) \in O(\log n) \qquad \text{(no parallelism)}$$

Not Necessarily Balanced Tree

We'll assume that the tree is maximally unbalanced on the left (i.e. all of the nodes are in the left subtree). In the worst case, the element we're inserting will be less than all the elements in the tree, so we'll have to traverse the entire tree to insert it.

$$\begin{aligned} W_{\texttt{insert}}(0) &= k_0 \\ W_{\texttt{insert}}(d) &= W_{\texttt{insert}}(d-1) + k_1 \end{aligned}$$

$$\begin{aligned} S_{\texttt{insert}}(0) &= k_2 \\ S_{\texttt{insert}}(d) &= S_{\texttt{insert}}(d-1) + k_3 \end{aligned}$$

Since the recursive call's input size decreases by 1, there are d levels. There is one node per level, each with work cost k_1 or span cost k_3. Thus, the total cost is:

$$W_{\texttt{insert}}(d) = \sum_{i=0}^{d} k_1 = k_1(d+1) \in O(d)$$

$$S_{\texttt{insert}}(d) \in O(d) \qquad \text{(no parallelism)}$$

2 Mult

2.1 Analyzing `mult`

Recall the tree datatype:

```
datatype tree = Empty | Node of tree * int * tree
```

Consider this function, which computes the product of integers in a tree:

```
fun mult Empty = 1
  | mult (Node(L,x,R)) = x * mult L * mult R
```

Note that function application takes precedence over infix operators such as `*`.

For this section, you may assume that the trees are *balanced*, meaning that the tree is either an empty tree or a tree whose subtrees are themselves balanced and whose subtrees differ in height by at most one.

Task 2.1.

Given that `T` is a balanced `tree` with depth d, write recurrences for the work and span of evaluating `mult T`, in terms of d.

Solution 1.

$$\begin{aligned} W_{\texttt{mult}}(0) &= k_0 \\ W_{\texttt{mult}}(d) &= k_1 + W_{\texttt{mult}}(d_l) + W_{\texttt{mult}}(d_r) = k_1 + 2W_{\texttt{mult}}(d-1) \\ S_{\texttt{mult}}(0) &= k_2 \\ S_{\texttt{mult}}(d) &= k_3 + \max\{S_{\texttt{mult}}(d_l), S_{\texttt{mult}}(d_r)\} = k_3 + S_{\texttt{mult}}(d-1) \end{aligned}$$

Task 2.2.

Given that `T` is a balanced `tree` with n nodes, write recurrences for the work and span of evaluating `mult T`, in terms of n.

Solution 2.

$$\begin{aligned} W_{\texttt{mult}}(0) &= k_0 \\ W_{\texttt{mult}}(n) &= k_1 + W_{\texttt{mult}}(n_l) + W_{\texttt{mult}}(n_r) = k_1 + 2W_{\texttt{mult}}(n/2) \\ S_{\texttt{mult}}(0) &= k_2 \\ S_{\texttt{mult}}(n) &= k_3 + \max\{S_{\texttt{mult}}(n_l), S_{\texttt{mult}}(n_r)\} = k_3 + S_{\texttt{mult}}(n/2) \end{aligned}$$

Task 2.3.

Using the recurrences you found in the previous tasks, what are the big-O bounds for the work and the span respectively? Show your work. You should have 2 big-O bounds in terms of d (one for work and one for span) and 2 big-O bounds in terms of n (one for work and one for span).

Solution 3.

For the work (in terms of d), we get that the work tree has d many levels (since we are decreasing d by 1 each level) and that level i has cost $2^i k_1$. So therefore we get that

$$W_{\texttt{mult}}(d) = \sum_{i=0}^{d} 2^i k_1 = (2^{d+1} - 1)k_1 = O(2^d)$$

For span (in terms of d), we get that the span tree has d levels, and that level i has cost k_3. So therefore we get that:

$$S_{\texttt{mult}}(d) = \sum_{i=0}^{d} k_3 = k_3 * (d+1) = O(d)$$

For the work (in terms of n), we get that the work tree has $\log n$ many levels (since we are dividing n by 2 each level) and that level i has cost $2^i k_1$ (because the ith level has 2^i many nodes). So therefore we get that

$$W_{\texttt{mult}}(n) = \sum_{i=0}^{\log n} 2^i k_1 = (2^{\log n+1} - 1)k_1 = O(n)$$

For the span (in terms of n), we get that the span tree has $\log n$ many levels (since we are dividing n by 2 each level) and that level i has cost k_3 (because each level only has 1 node). So therefore we get that

$$S_{\texttt{mult}}(n) = \sum_{i=0}^{\log n} k_3 = k_3 * (\log n + 1) = O(\log n)$$

3 Trees

In the Datatypes Lab, you wrote the function `invert`, which inverts a tree so that the in-order traversal of the inversion comes out backwards:

```
fun invert (Empty : tree) : tree = Empty
  | invert (Node (L, x, R)) = Node (invert R, x, invert L)
```

In this problem, you will analyze the work and span of this function.

Task 3.1.

Determine the recurrence for the work of `invert`, in terms of the size (number of elements) of the tree. You may assume the tree is balanced.

Solution 1.

We will determine the work, $W_{\texttt{invert}}(n)$, based on the number n of elements in the tree t.

We assume that the tree is balanced so the work of the evaluation of each recursive call in `Node (invert R, x , invert L)` takes at most $W_{\texttt{invert}}\left(\frac{n}{2}\right)$ steps.

Thus the recurrence for the work of `invert` is:

$$W_{\texttt{invert}}(0) = k_0$$
$$W_{\texttt{invert}}(n) = W_{\texttt{invert}}(n_l) + W_{\texttt{invert}}(n_r) + k_1 = 2W_{\texttt{invert}}\left(\frac{n}{2}\right) + k_1$$

Task 3.2.

Use the closed form of the recurrence from the previous step to determine the big-O of $W_{\texttt{invert}}(n)$.

Solution 2.

Since every recursive call's input size is halved, there are $\log_2 n$ levels. There are 2^i nodes at level i, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} W_{\texttt{invert}}(n) &= \sum_{i=0}^{\log_2 n} 2^i k_1 \\ &= k_1 \sum_{i=0}^{\log_2 n} 2^i \\ &= k_1(2n-1) \in O(n) \end{aligned}$$

Task 3.3.

Determine the recurrence for the span of your `invert` function, in terms of the size of the tree. You may assume the tree is balanced.

Solution 3.

We will determine the span, $S_{\texttt{invert}}(n)$, based on the number n of elements in the tree t. We assume that the tree is balanced so the span of the evaluation of each recursive call in `Node (invert R, x , invert L)` is at most $S_{\texttt{invert}}\left(\frac{n}{2}\right)$. This gives the following recurrence:

$$S_{\texttt{invert}}(0) = k_0$$
$$S_{\texttt{invert}}(n) = \max\{S_{\texttt{invert}}(n_l), S_{\texttt{invert}}(n_r)\} + k_1 = S_{\texttt{invert}}\left(\frac{n}{2}\right) + k_1$$

Task 3.4.

Use the closed form of the recurrence from the previous step to give a big-O for $S_{\texttt{invert}}$.

Solution 4.

Since every recursive call's input size is halved, there are $\log_2 n$ levels. There 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} S_{\texttt{invert}}(n) &= \sum_{i=0}^{\log n} k_1 \\ &= k_1(\log n + 1) \in O(\log n) \end{aligned}$$

4 Work It Out

Consider the following two functions:

```
fun listMax ([] : int list) : int = 0
  | listMax (x::xs) = Int.max (x, listMax xs)

fun treeMax (Empty : tree) : int = 0
  | treeMax (Node (l,x,r)) =
      Int.max (treeMax l, Int.max (x, treeMax r))
```

For each of the functions below:

- Write the function's recurrence relations.
- Solve the recurrence with tree method.
- Derive a tight big-O bound.
- Briefly explain why your answer is correct.

You may not use the table of common recurrences for these problems.

Task 4.1.

Find the *work* of `listMax`, in terms of the length, n.

Solution 1.

$$W_{\texttt{listMax}}(0) = k_0$$
$$W_{\texttt{listMax}}(n) = W_{\texttt{listMax}}(n-1) + k_1$$

Since every recursive call's input size decreases by 1, there are n levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} W_{\texttt{listMax}}(n) &= \sum_{i=0}^{n} k_1 \\ &= k_1(n+1) \in O(n) \end{aligned}$$

Task 4.2.

Find the *span* of `listMax`, in terms of the length, n.

Solution 2.

$$
\begin{aligned}
S_{\texttt{listMax}}(0) &= k_0 \\
S_{\texttt{listMax}}(n) &= S_{\texttt{listMax}}(n-1) + k_1
\end{aligned}
$$

Since every recursive call's input size decreases by 1, there are n levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$
\begin{aligned}
S_{\texttt{listMax}}(n) &= \sum_{i=0}^{n} k_1 \\
&= k_1(n+1) \in O(n)
\end{aligned}
$$

For the following tasks (4.3 - 4.6), assume the tree is balanced.

Task 4.3. (Recommended)

Find the *work* of `treeMax`, in terms of the number of nodes in the tree, n.

Solution 3.

$$
\begin{aligned}
W_{\texttt{treeMax}}(0) &= k_0 \\
W_{\texttt{treeMax}}(n) &= W_{\texttt{treeMax}}(n_l) + W_{\texttt{treeMax}}(n_r) + k_1 = 2W_{\texttt{treeMax}}\left(\frac{n}{2}\right) + k_1
\end{aligned}
$$

Since every recursive call's input size halves, there are $\log n$ levels. There are 2^i nodes at level i, each with cost k_1. Thus, the total cost is:

$$
\begin{aligned}
W_{\texttt{treeMax}}(n) &= \sum_{i=0}^{\log_2 n} 2^i k_1 \\
&= k_1(2n-1) \in O(n)
\end{aligned}
$$

Task 4.4. (Recommended)

Find the *span* of `treeMax`, in terms of the number of nodes in the tree, n.

Solution 4.

$$S_{\texttt{treeMax}}(0) = k_0$$
$$S_{\texttt{treeMax}}(n) = \max\{S_{\texttt{treeMax}}(n_l), S_{\texttt{treeMax}}(n_r)\} + k_1 = S_{\texttt{treeMax}}\left(\frac{n}{2}\right) + k_1$$

Since every recursive call's input size halves, there are $\log n$ levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} S_{\texttt{treeMax}}(n) &= \sum_{i=0}^{\log_2 n} k_1 \\ &= k_1(\log_2 n + 1) \in O(\log n) \end{aligned}$$

Task 4.5.

Find the *work* of `treeMax`, in terms of the tree's depth, d.

Solution 5.

$$W_{\texttt{treeMax}}(0) = k_0$$
$$W_{\texttt{treeMax}}(d) = W_{\texttt{treeMax}}(d_l) + W_{\texttt{treeMax}}(d_r) + k_1 = 2W_{\texttt{treeMax}}(d-1) + k_1$$

Since every recursive call's input size decreases by 1, there are d levels. There are 2^i nodes at level i, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} W_{\texttt{treeMax}}(d) &= \sum_{i=0}^{d} 2^i k_1 \\ &= k_1(2^{d+1} - 1) \in O(2^d) \end{aligned}$$

Task 4.6.

Find the *span* of `treeMax`, in terms of the tree's depth, d.

Solution 6.

$$S_{\texttt{treeMax}}(0) = k_0$$
$$S_{\texttt{treeMax}}(d) = \max\{S_{\texttt{treeMax}}(d_l) + S_{\texttt{treeMax}}(d_r)\} + k_1 = S_{\texttt{treeMax}}(d-1) + k_1$$

Since every recursive call's input size decreases by 1, there are d levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned} S_{\texttt{treeMax}}(d) &= \sum_{i=0}^{d} k_1 \\ &= k_1(d+1) \in O(d) \end{aligned}$$

For the following tasks (4.7 - 4.11), assume the tree is maximally unbalanced.

Task 4.7. (Recommended)

Find the *work* of `treeMax`, in terms of the number of nodes in the tree, n.

Solution 7.

$$\begin{aligned} W_{\texttt{treeMax}}(0) &= k_0 \\ W_{\texttt{treeMax}}(n) &= W_{\texttt{treeMax}}(n_l) + W_{\texttt{treeMax}}(n_r) + k_1 = W_{\texttt{treeMax}}(n-1) + k_0 + k_1 \end{aligned}$$

Since every recursive call's input size decreases by 1, there are n levels. There is 1 node per level, each with cost $k_0 + k_1$. Thus, the total cost is:

$$\begin{aligned} W_{\texttt{treeMax}}(n) &= \sum_{i=0}^{n} k_0 + k_1 \\ &= (k_0 + k_1)(n+1) \in O(n) \end{aligned}$$

Task 4.8.

You should have gotten the same asymptotic bound for the work of `treeMax` in both the balanced case and the maximally unbalanced case. It turns out that the work of `treeMax` only depends on n, the number of nodes in the tree, and not the structure of the tree. Why is this the case? (A brief justification suffices.)

Hint: Suppose there are n_l nodes in the left subtree and n_r nodes in the right subtree. Write the recurrence for this setup.

Solution 8.

Assuming there are n_l nodes in the left subtree and n_r nodes in the subtree, we get the following recurrence:

$$\begin{aligned} W_{\texttt{treeMax}}(0) &= k_0 \\ W_{\texttt{treeMax}}(n) &= W_{\texttt{treeMax}}(n_l) + W_{\texttt{treeMax}}(n_r) + k_1 \end{aligned}$$

Note that $n_l + n_r = n - 1$, so the combined input size decreases by 1. Since the non-recursive work per call is constant, we can think of it as performing a constant amount of work per node for every node in the tree.

Note: Solving the recurrence for a tight asymptotic bound for this task has been omitted, but unless otherwise stated, you should be showing all steps (of the tree method) in your solutions!

Task 4.9. (Recommended)

Find the *span* of `treeMax`, in terms of the number of nodes in the tree, n.

Solution 9.

$$\begin{aligned}
&S_{\texttt{treeMax}}(0) = k_0 \\
&S_{\texttt{treeMax}}(n) = \max\{S_{\texttt{treeMax}}(n_l), S_{\texttt{treeMax}}(n_r)\} + k_1 = S_{\texttt{treeMax}}(n-1) + k_1
\end{aligned}$$

Since every recursive call's input size decreases by 1, there are n levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned}
S_{\texttt{treeMax}}(n) &= \sum_{i=0}^{n} k_1 \\
&= k_1(n+1) \in O(n)
\end{aligned}$$

Task 4.10.

Find the *work* of `treeMax`, in terms of the tree's depth, d.

Solution 10.

$$\begin{aligned}
&W_{\texttt{treeMax}}(0) = k_0 \\
&W_{\texttt{treeMax}}(d) = W_{\texttt{treeMax}}(d_l) + W_{\texttt{treeMax}}(d_r) + k_1 = W_{\texttt{treeMax}}(d-1) + k_0 + k_1
\end{aligned}$$

Since every recursive call's input size decreases by 1, there are d levels. There is 1 per level, each with cost $k_0 + k_1$. Thus, the total cost is:

$$\begin{aligned}
W_{\texttt{treeMax}}(d) &= \sum_{i=0}^{d} k_0 + k_1 \\
&= (k_0 + k_1)(d+1) \in O(d)
\end{aligned}$$

Task 4.11.

Find the *span* of `treeMax`, in terms of the tree's depth, d.

Solution 11.

$$\begin{aligned}
S_{\texttt{treeMax}}(0) &= k_0 \\
S_{\texttt{treeMax}}(d) &= \max(S_{\texttt{treeMax}}(d_l), S_{\texttt{treeMax}}(d_r)) + k_1 = S_{\texttt{treeMax}}(d-1) + k_1
\end{aligned}$$

Since every recursive call's input size decreases by 1, there are d levels. There is 1 node per level, each with cost k_1. Thus, the total cost is:

$$\begin{aligned}
S_{\texttt{treeMax}}(d) &= \sum_{i=0}^{d} k_1 \\
&= k_1(d+1) \in O(d)
\end{aligned}$$

Task 4.12.

Compare your answers to 4.5 and 4.10. Which kind of tree (balanced or maximally unbalanced) is the worst case for `treeMax` when analyzed in terms of depth?

How can we determine the worst case for `treeMax` without doing both analyses?

Solution 12.

The worst case for `treeMax` is a balanced tree.

We can determine the worst case for `treeMax` by looking at the structure of the code. There are two recursive calls in `treeMax`, which means, for a balanced tree with depth d, we would be making two recursive calls on an input of size $d-1$. However, for a maximally unbalanced tree, we would effectively be only doing one recursive call on an input of size $d-1$, since all of the nodes are on one side of the tree. This tells us that the balanced tree should result in a higher bound, and is thus the worst case.