Discussion Solutions Week 6

CS 173: Discrete Structures

Tuesday

Problem 15.2. in Discussion Manual

(a) maxthree computes the largest sum of 3 numbers in the list.

(b) $T(3) = c_1$
$T(n) = n \cdot T(n-1) + c_2 n + c_3$
There loop on lines 4 - 7 runs n times and makes a recursive call each time on a list one shorter. It also does some constant work inside and outside the loop.

(c) $\frac{n!}{3!}$. We will reach the last level of the tree (base case) when the list is of length 3, and at each level, each node has as many children as the length of the list. So, the number of leaves is $n \cdot (n-1) ... \cdot 4$.

(d) No. The running time of maxthree is at least $O(n!)$ since there are $\Theta(n!)$ leaves (even with no other work). It is easy to see that the big-O running time of maxthree is more than 2^n.

Problem 15.3. in Discussion Manual

(a) crunch computes how many nonnegative numbers are in the array.

(b) $T(1) = d$
$T(n) = 2T(\frac{n}{2}) + c$

(c) Answer: $\Theta(n)$.

Justification using unrolling:

- $T(n) = 2T(\frac{n}{2}) + c$
- $T(n) = 2[2T(\frac{n}{2^2}) + c] + c = 2^2 T(\frac{n}{2^2}) + 2c + c$
- $T(n) = 2^2[2T(\frac{n}{2^3}) + c] + 2c + c = 2^3 T(\frac{n}{2^3}) + 2^2 c + 2c + c$

Based on the above, we predict the general form is that for any k,

$$T(n) = 2^k T(\frac{n}{2^k}) + \sum_{i=0}^{k-1} 2^i c = 2^k T(\frac{n}{2^k}) + c(2^k - 1)$$

.

When we choose k such that $2^k = n$, this becomes $nT(1) + c(n-1) = dn + cn - c$, which is $\Theta(n)$.

Problem 15.4. in Discussion Manual

(a)
```
FindPeak(-1,3,6,7,0):
- skip several false ifs
- set k=3
- skip line 8's if
- line 10: since 6<7, we return FindPeak(7,0)+3

FindPeak(7,0):
- line 3: since 7>0, we return 1

Thus the original call returns 1+3=4
```

And the peak is indeed at position 4 (starting from that 7, the array strictly decreases in both directions until its ends)

(b) **3**. If n were 1, we would have returned on line 1. If n were 2, we would return on either line 4 or line 6 (because the first item is either greater than or less than the second/last). However on an input array with 3 elements whose peak is in the center, like $[5, 6, 4]$, we can reach line 7. *(Note that to argue that* 3 *is the smallest, we had to argue* both *that* 3 *works* and *that no smaller number works.)*

(c) $T(1) = T(2) = c$
$T(n) = T(n/2) + d$

(d) $\Theta(\log(n))$. We find this by unrolling: $T(n) = T(n/2) + d = T(n/2^2) + 2d = T(n/2^3) + 3d = \cdots = T(n/2^k) + kd = T(n/2^{\log(n)}) + \log(n)d = c + \log(n)d$

Problem 15.5. in Discussion Manual

(a) Foo(n) computes the nth Fibonacci number.

1. $O(n)$. We have a for-loop which does a constant amount of work $O(n)$ times; everything else in the program just adds an additional constant amount of work.

(b)
```
RecursiveFoo(n: non-negative integer)
    if n=0 or n=1
        return n
    else
        return RecursiveFoo(n-1) + RecursiveFoo(n-2)
```

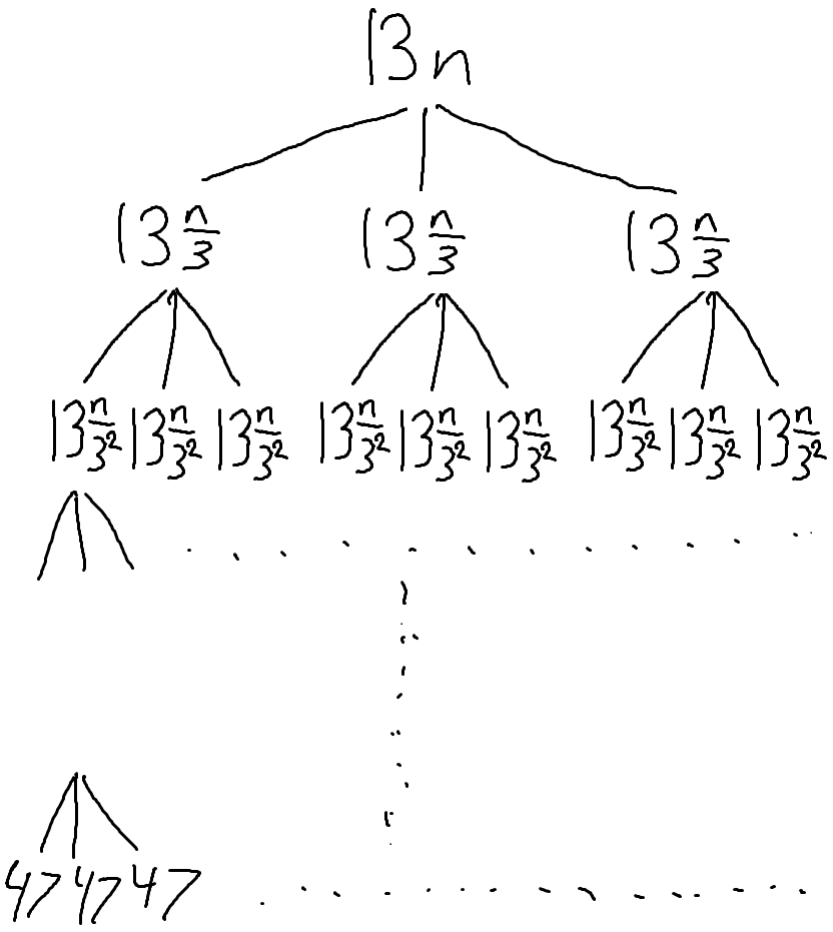
This algorithm just follows the (recursive) definition of Fibonacci exactly - to compute the nth Fibonacci number, it just computes and then adds together the $(n-1)$th and $(n-2)$th.

(c) We've established Foo runs in linear time; meanwhile RecursiveFoo is exponential time with respect to n. We can write a recurrence for RecursiveFoo's runtime: $T(0) = T(1) = c$, $T(n) = T(n-1) + T(n-2) + d$. Computing the closed form for that recurrence is outside the scope of this class, but it's definitely exponential - one way to see that is to first bound it below by a similar recurrence where $T(n) = 2T(n-2) + d$ instead.

Wednesday

Problem 13.1. in Discussion Manual

(a) Assume n is a power of 3 so that the input will always be an integer. Then we get the following recursion tree:



The tree is described by the following table:

level	"problem size"	# nodes	work per node	total for level
0	n	1	$13n$	$13n$
1	$\frac{n}{3}$	3	$13\frac{n}{3}$	$13n$
2	$\frac{n}{3}$	3^2	$13\frac{n}{3^2}$	$13n$
3	$\frac{n}{3^2}$	3^3	$13\frac{n}{3^3}$	$13n$
...				
k	$\frac{n}{3^k}$	3^k	$13\frac{n}{3^k}$	$13n$
...				
h	$\frac{n}{3^h} = 1$	3^h	$T(1) = 47$	$47 * 3^h$

(Notice that the final row (the leaf level) follows the same pattern for problem size and number of nodes as the rows above it, but that we also know the problem size must be 1 *since that's the function's base case - this is why I've written both* $\frac{n}{3^h}$ *and* 1 *in that cell, and this is how we are able to solve for* h*. Note that the work per node and hence total for level does* not

follow the pattern of the levels above it; this is why our later summation only sums through $h-1$ and then we have to add in the work in the leaves separately.)

We have $\frac{n}{3^h} = 1$, i.e. $h = \log_3 n$, so there are $3^{\log_3 n} = n$ leaves. Thus the total work at the leaves is $n \cdot T(1) = 47n$.

From the table, the total work for all non-leaf levels is

$\sum_{k=0}^{(\log_3 n)-1} 13n = 13n \log_3 n$.

Putting it all together, our final closed form is $47n + 13n \log_3 n$.

(b) We'll just describe the tree with a table instead of drawing it:

level	"problem size"	# nodes	work per node	total for level
0	n	1	3	3
1	$n-1$	2	3	6
2	$n-2$	4	3	12
3	$n-3$	8	3	24
...				
k	$n-k$	2^k	3	$3 \cdot 2^k$
...				
h	$n-h=1$	2^h	T(1) = 1	$1 \cdot 2^h$

From $n - h = 1$ we get $h = n - 1$, so there are $2^h = 2^{n-1}$ leaves, for a total work in the leaves of $2^{n-1} \cdot T(1) = 2^{n-1}$.

The total work for all non-leaf levels is

$\sum_{k=0}^{n-2}(3 \cdot 2^k) = 3\sum_{k=0}^{n-2} 2^k = 3(2^{n-1} - 1)$.

Thus our closed form is $2^{n-1} + 3(2^{n-1} - 1) = 4 \cdot 2^{n-1} - 3 = 2^{n+1} - 3$.

Friday

Problem 16. in Discussion Manual

(a) Suppose not. That is, suppose $\sqrt{2}+\sqrt{6} \geq \sqrt{15}$. Then we get the following:

$$\begin{aligned} \sqrt{2}+\sqrt{6} &\geq \sqrt{15} \\ 2+2\sqrt{12}+6 &\geq 15 && \text{(both sides were positive; square them)} \\ 2\sqrt{12} &\geq 7 \\ 48 &\geq 49 && \text{(both sides were positive; square them)} \end{aligned}$$

This is a contradiction, so our initial supposition must be false. Thus we have shown $\sqrt{2}+\sqrt{6}<\sqrt{15}$.

(b) Assume towards contradiction that there are a *finite* number of integers of the form $4k+3$. Then there must be some largest of them. *(For claims where it isn't obvious, one should first show that there is at least one before claiming there's a largest - in this case we could say* $43 = 4 \cdot 10 + 3$ *so there's at least one.)* Let that largest be $m = 4y+3$. Now consider $m+4$. It is larger than m, so it does not have the $4k+3$ form. Yet $m+4 = (4y+3)+4 = 4(y+1)+3$, so it *does* have the $4k+3$ form - contradiction! Thus our assumption is false and there are actually infinitely many integers of that form.