Foundations of Algorithms
Advanced Hint on the Master Theorem

Some of the Master Theorem homework problems may be challenging. One problem in particular involves a substitution. **If you have attempted any such problem 3 times and are stuck, you may use this document, but please cite that you did so.** I want to demonstrate a pattern that will help you solve the problem. I will develop it over several pages, with hints given on each successive page. I don't want to rob you of the opportunity to solve this yourself, because some might find it rewarding.[**] Therefore, *look at one page at a time,* apply the info, and if you're still stuck then keep reading.

With that, let us solve a challenging problem; find the asymptotic complexity of:

$$T(n) = 3T(\sqrt[3]{n}) + 1$$

We want to be able to use the *Master Theorem* (your first hint), which has a form of

$$T(n) = aT(n/b) + f(n)$$

To do this, we need to make sure the recurrence term (function $T()$) has a format with a fraction in it. There are two substitutions that we need—find a replacement for n, then find a replacement for T. The key to this is to think, "how do I get rid of powers?" "Is there something that will help me convert powers into fractions?" Before you turn the page, take a moment and

THINK!

[**]Of course, some might consider an office filled with bees, put there by a good apiarist, to be a "reward," but I'm not keeping tabs.

The answer is *logarithms*. Logarithms help us get rid of powers. First, we need to review a few identities of logarithms and square roots.

1. $\sqrt[n]{x} = x^{1/n}$
2. $\log_b x^y = y \log_b x$
3. $\lg 2^x = x$, and generally, $\log_b b^x = x$
4. $2^{\lg x} = x$, and generally, $b^{\log_b x} = x$

We need *two* substitutions. The first substitution is to the argument to $T()$, and the second is to replace $T()$ with something else.

I need to make some points. First, the argument to $T()$ is just a number. If I say $T(\Box)$, the stuff in $\Box$ is just a number. $T(n)$ can be replaced by $T(2m)$ for all values m such that $n = 2m$.[†] When I do this, *there is no change* to the underlying relationship and importantly, *no change* to the other elements in the recurrence—that is, replacing n in

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

with $2m$ gives us

$$T(2m) = T(m) + 1$$

with no change to the "1". Again, the stuff inside $T()$ is just a number.

Next, I can replace function $T()$ with function $S()$ that transforms the inner arguments, again without changing the overall relationship. I could say "let $S(\Box) = T(2^{\Box})$" and then replace $T()$ with $S()$ everywhere. Now, on this little $T(n)$ example above, it does not make sense to do this, so before I go on, take in this page, try your problem again, and

THINK!

[†]This restricts inputs to $T()$ to just the even numbers, but that's fine for now.

Ok, you're still with me. I get it, it took me a while to understand it, too. I'm going to work the example from page 1 to show the substitutions of the variable. Here's our expression:

$$T(n) = 3T(\sqrt[3]{n}) + 1$$

I want to replace n with something that gets rid of the root. Identity 1 from the first page says we can convert the nasty root to a thing that looks like a fraction, changing $\sqrt[3]{n} \rightarrow n^{1/3}$. Next, Identity 2 says that $\lg n^{1/3} = \frac{1}{3} \lg n$. That looks more like a fraction, so we're getting closer to something that "fits" into the form of the Master Theorem.

Now I will show my replacement, replacing the number n with something involving a different number m. If I let $m = \lg n$, then I also have $n = 2^m$ by Identity 3, and $n^{1/3} = 2^{m/3}$ by the rule that $(x^y)^z = x^{yz}$.

Substituting the above in $T()$ changes my expression into:

$$T(2^m) = 3T(2^{m/3}) + 1$$

With me so far? Note that the $+1$ again does not change! Why is that? Because stuff inside $T(\square)$ is just a number; we are just re-expressing what goes into $T()$.

What's left to do? Oh yeah, get rid of the $2^{\square}$ stuff. How can we do that? Hint - we need a new function $S()$. With that, take a moment before you go on, and

THINK!

Let's show this substitution. Define a new relation $S(\square) = T(2^{\square})$. With $S(\square) = T(2^{\square})$, we can set $\square$ to any number that we want, and it leads to a new form of the relation. We can put terms involving m into the left and right hand $\square$'s as follows:

$$\begin{aligned} T(2^m) &= 3T(2^{m/3}) + 1 \\ T(2^{\square}) &= 3T(2^{\square}) + 1 \end{aligned} \tag{1}$$

Using the pattern of $S(\square) = T(2^{\square})$, this gives us[†]

$$S(\square) = 3S(\square) + 1 \tag{2}$$

which becomes

$$S(m) = 3S(m/3) + 1$$

Recurrence relations are just meant to capture the *pattern* of the recurrence. We have substituted values and even recurrence functions to get down to this point. Now, we have something that looks like the Master Theorem.

For a talk-through on the Master Theorem, continue going, but first,

THINK!

[†] Don't think of $\square$ as a variable, think of it as a placeholder. Really, it's an argument to a function in a different domain, but that's confusing mathspeak.

Solving $S(m) = 3S(m/3) + 1$ by the Master Theorem means identifying a and b in the equation

$$T(m) = aT(m/b) + f(m)$$

I'm using m instead of n to remember that we have a substitution to do later on. In the above expression, we get $a = 3$, $b = 3$, and $f(m) = 1$. Using the values of a and b, we see that our m term in the Master Theorem has the form: $m^{\log_3 3}$ which reduces to m by Identity 4 given earlier.

Since $f(1)$ is 1, we must find a case where we can change m to a 1: Case 1 applies if we set $\epsilon = 1$, resulting in $m^{\log_3 3-1} = m^0 = 1$. Therefore, this is Case 1 of the Master Theorem, giving $\theta(m^{\log_3 3})$ which is $\theta(m)$.

We're not done yet—we need to express this in terms of n. To change from m to n, we have to apply the substitution $m = \lg n$ from a few pages ago. Making this substitution into $\theta(m)$ gives us our final answer— the asymptotic complexity of our relation is $\theta(\lg n)$.

Intuitively, does this answer make sense? The recurrence relation says that the cost of every step is 3 times the cost of splitting the list into (much) smaller pieces, and recursing on the pieces. The fact that we're splitting and recursing implies a tree, and the total cost of running the algorithm relates to the depth of the tree which, in turn, is the log of the number of inputs. Because we incur a cost of "+1" at every step, we expect the answer to be 1 times the number of levels in the tree.

This was a challenging problem. By following this, you should be able to apply the techniques to related problems. I hope this helped.

Note

There may be more efficient ways to solve this, and that is okay. If you see a mistake in this instruction, or some way to make it better, please reach out! For other problems, know that I am here for you, and available to help during office hours and by email. – Russ