1 Unequal Divide and Conquer (revised from last time)

1.1 Guidelines

In these problems, you are not required to find c and n_0 and prove the result by induction. It is sufficient to give a proof that appeals to results in the chapter. For example, you can say, without further comment, "$5n^2 + 7n \leq O(n^2)$" or "$\sum_{j=1}^{n} j^3 = \Theta(\int_0^n x^3\,dx) = \Theta(n^4)$" or "a k-ary tree with height h has $\Theta(k^h)$ leaves and nodes" ($k > 1$). In fact, I'd prefer that you get some practice in this "bigger picture" way of thinking, and not always get bogged down with induction details.

1.2 The Problem

Exercises CLRS 4.2-4 and CLRS 4.2-5 illustrate that, if we use a divide-and-conquer approach to a problem, it is typically better to divide the problem into nearly equal-sized subproblems. The above and the last example of CLRS Section 4.2 also illustrate that it is not necessary that the subproblems be *exactly* the same size.

We now switch from *analyzing* to *designing* algorithms. Given a problem of size n, we will break it into two problems, of size $a(n)$ and $n - a(n)$, where $a(n)$ is a "common" function satisfying $1 \leq a(n) \leq n/2$. We assume the Divide and Recombine cost is linear, so we have recurrence

$$T(n) = T(a(n)) + T(n - a(n)) + cn.$$

All other things being equal, we'd want $a(n) = n/2$ to minimize $T(n)$. But sometimes there is a separate (direct or indirect) cost involved in making $a(n)$ exactly $n/2$ and it's easier to choose other $a()$'s. Below we investigate which $a()$'s are acceptable while still meeting certain overall cost requirements. Problem:

- How slowly-growing can $a(n)$ be and still make $T(n) \leq O(n \log^2(n))$?

Restrict attention to functions of the form $n^r(\log(n))^s$, where r and s are constant real numbers, and you only need to find s up to 1, additively. That is, your answer should be numbers r and s with

$$n^r(\log(n))^s \leq a(n) \leq n^r(\log(n))^{s+1}.$$

(If you've already found exactly the right s, please turn it in, but don't attempt this if you haven't started.)

2 Solution

First we show that, if $a(n) = n/\log(n)$, then the requirement $T(n) \leq O(n \log^2(n))$ is satisfied. That is, $T(n) \leq O(n \log^2(n))$. To do this, we show that the height of the recursion tree (longest path from root to leaf) is at most $O(\log^2(n))$. Each level of the recursion tree contributes at most cn, so the total cost is $O(n \log^2(n))$.

Write $b(n) = a(n)/n$, so that $b(n) = 1/\log(n)$. The recurrence becomes

$$T(n) = T(nb(n)) + T(n(1 - b(n))) + cn.$$

The longest path in the recurrence tree occurs along the $n \to n(1 - b(n))$ split (the "right spine" of the tree). Along this path, let m_i denote the size of the problem at depth i, so that $m_{i+1} = m_i(1 - b(m_i))$, *i.e.*,

$$\begin{aligned}
m_0 &= n \\
m_1 &= n(1 - b(m_0)) = n(1 - b(n)) \\
m_2 &= n(1 - b(m_1)) = n(1 - b(n(1 - b(n)))) \\
&\vdots
\end{aligned}$$

We first consider i with $n/2 \leq m_i \leq n$. Because

$$\frac{1}{\log(n)} = b(n) \leq b(m_i) \leq b(n/2) = \frac{1}{\log(n) - 1} \leq \frac{2}{\log(n)},$$

we get simplified expressions for m_i, namely, $m_i(1 - 2/\log(n)) \leq m_{i+1} \leq m_i(1 - 1/\log(n))$, or solving the simple recurrence,

$$n(1 - 2/\log(n))^i = m_0(1 - 2/\log(n))^i \leq m_{i+1} \leq m_0(1 - 1/\log(n))^i = n(1 - 1/\log(n))^i.$$

Using the approximation $1 + x \approx e^x$ if $|x|$ is small, we get $m_i = ne^{\Theta(i/\log(n))}$. If m_i is deepest, $m_i \approx n/2$, we need $e^{\Theta(i/\log(n))} \approx 1/2$, which happens at $i = \Theta(\log(n))$. In particular, we need that there is some constant c_1 such that, for $i \geq c_1 \log(n)$, we have $m_i \leq n/2$.

By repeating the process on m_i instead of $m_0 = n$, we get that, after $\log(m_i)$ additional levels in the tree, the problem size drops by half again, to $m_i/2 \leq n/4$. After $\log(n)$ repetitions of $n \mapsto n/2$, we have reduced the problem size to 1. Since all the m_i's are at most n, the length of this path is $\log^2(n)$. This is what we needed to show.

Next, we show that if $T(n) = T(n/\log^{2+\epsilon}(n)) + T(n(1 - 1/\log^{2+\epsilon}(n))) + cn$, then $T(n) > \omega(n\log^2(n))$. Using an argument similar to the above, let m_i be the size of the largest problem at depth i in the recursion tree. If $m_i \approx n/2$, then $i \geq \Omega(\log^{2+\epsilon}(n))$, using an argument similar to the above. Each problem along the right spine at depth less than i has size at least $n/2$, which gives a total cost of $\Omega(n\log^{2+\epsilon}(n))$.

Thus it follows that $a(n) = n/\log(n)$ or faster growing gives $T(n) \leq O(n\log^2(n))$ and $a(n) = nlog^s(n)$ for $s < -2$ gives $T(n) > \omega(n\log^2(n))$. So $r = 1$ and $-2 \leq s \leq -1$.

Note: It is not necessarily the case that there is a slowest-growing function $a(n) = n^r \log^s(n)$ that satisfies the requirement. There could be some s' such that $n\log^{s'}(n)$ grows too slowly but, for any $s > s'$, the requirement is satisfied. For a simpler example of this phenomenon, suppose we insist that $s = -3$ and ask about r. We know that $a(n) = n/\log^3(n)$ will not satisfy the requirement but for any slower-growing function $n^{1-\epsilon}/\log^3(n)$, *i.e.*, for any $\epsilon > 0$, the requirement will be satisfied.

In this case, $s < -1$ gives $T(n) > \omega(n\log^2(n))$, so the answer is $r = 1$ and $s = -1$, *i.e.*, $a(n) = n/\log(n)$.