4.4-3
Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 4T(n/2 + 2) + n$. Use the substitution method to verify your answer.

4.4-4
Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 2T(n - 1) + 1$. Use the substitution method to verify your answer.

4.4-5
Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n-1) + T(n/2) + n$. Use the substitution method to verify your answer.

4.4-6
Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree.

4.4-7
Draw the recursion tree for $T(n) = 4T(\lfloor n/2 \rfloor) + cn$, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

4.4-8
Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(n - a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.

4.4-9
Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant.

4.5 The master method for solving recurrences

The master method provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n)\,, \tag{4.20}$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. To use the master method, you will need to memorize three cases, but then you will be able to solve many recurrences quite easily, often without pencil and paper.

$f(n) = n \lg n$ is asymptotically larger than $n^{\log_b a} = n$. The problem is that it is not *polynomially* larger. The ratio $f(n)/n^{\log_b a} = (n \lg n)/n = \lg n$ is asymptotically less than n^ϵ for any positive constant ϵ. Consequently, the recurrence falls into the gap between case 2 and case 3. (See Exercise 4.6-2 for a solution.)

Let's use the master method to solve the recurrences we saw in Sections 4.1 and 4.2. Recurrence (4.7),

$$T(n) = 2T(n/2) + \Theta(n) ,$$

characterizes the running times of the divide-and-conquer algorithm for both the maximum-subarray problem and merge sort. (As is our practice, we omit stating the base case in the recurrence.) Here, we have $a = 2$, $b = 2$, $f(n) = \Theta(n)$, and thus we have that $n^{\log_b a} = n^{\log_2 2} = n$. Case 2 applies, since $f(n) = \Theta(n)$, and so we have the solution $T(n) = \Theta(n \lg n)$.

Recurrence (4.17),

$$T(n) = 8T(n/2) + \Theta(n^2) ,$$

describes the running time of the first divide-and-conquer algorithm that we saw for matrix multiplication. Now we have $a = 8$, $b = 2$, and $f(n) = \Theta(n^2)$, and so $n^{\log_b a} = n^{\log_2 8} = n^3$. Since n^3 is polynomially larger than $f(n)$ (that is, $f(n) = O(n^{3-\epsilon})$ for $\epsilon = 1$), case 1 applies, and $T(n) = \Theta(n^3)$.

Finally, consider recurrence (4.18),

$$T(n) = 7T(n/2) + \Theta(n^2) ,$$

which describes the running time of Strassen's algorithm. Here, we have $a = 7$, $b = 2$, $f(n) = \Theta(n^2)$, and thus $n^{\log_b a} = n^{\log_2 7}$. Rewriting $\log_2 7$ as $\lg 7$ and recalling that $2.80 < \lg 7 < 2.81$, we see that $f(n) = O(n^{\lg 7 - \epsilon})$ for $\epsilon = 0.8$. Again, case 1 applies, and we have the solution $T(n) = \Theta(n^{\lg 7})$.

Exercises

4.5-1

Use the master method to give tight asymptotic bounds for the following recurrences.

a. $T(n) = 2T(n/4) + 1$.

b. $T(n) = 2T(n/4) + \sqrt{n}$.

c. $T(n) = 2T(n/4) + n$.

d. $T(n) = 2T(n/4) + n^2$.