

Because subproblem sizes decrease by a factor of 4 each time we go down one level, we eventually must reach a boundary condition. How far from the root do we reach one? The subproblem size for a node at depth i is $n/4^i$. Thus, the subproblem size hits $n = 1$ when $n/4^i = 1$ or, equivalently, when $i = \log_4 n$. Thus, the tree has $\log_4 n + 1$ levels (at depths $0, 1, 2, \dots, \log_4 n$).

Next we determine the cost at each level of the tree. Each level has three times more nodes than the level above, and so the number of nodes at depth i is 3^i . Because subproblem sizes reduce by a factor of 4 for each level we go down from the root, each node at depth i , for $i = 0, 1, 2, \dots, \log_4 n - 1$, has a cost of $c(n/4^i)^2$. Multiplying, we see that the total cost over all nodes at depth i , for $i = 0, 1, 2, \dots, \log_4 n - 1$, is $3^i c(n/4^i)^2 = (3/16)^i cn^2$. The bottom level, at depth $\log_4 n$, has $3^{\log_4 n} = n^{\log_4 3}$ nodes, each contributing cost $T(1)$, for a total cost of $n^{\log_4 3} T(1)$, which is $\Theta(n^{\log_4 3})$, since we assume that $T(1)$ is a constant.

Now we add up the costs over all levels to determine the cost for the entire tree:

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16} cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \dots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \quad (\text{by equation (A.5)}) . \end{aligned}$$

This last formula looks somewhat messy until we realize that we can again take advantage of small amounts of sloppiness and use an infinite decreasing geometric series as an upper bound. Backing up one step and applying equation (A.6), we have

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2) . \end{aligned}$$

Thus, we have derived a guess of $T(n) = O(n^2)$ for our original recurrence $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. In this example, the coefficients of cn^2 form a decreasing geometric series and, by equation (A.6), the sum of these coefficients

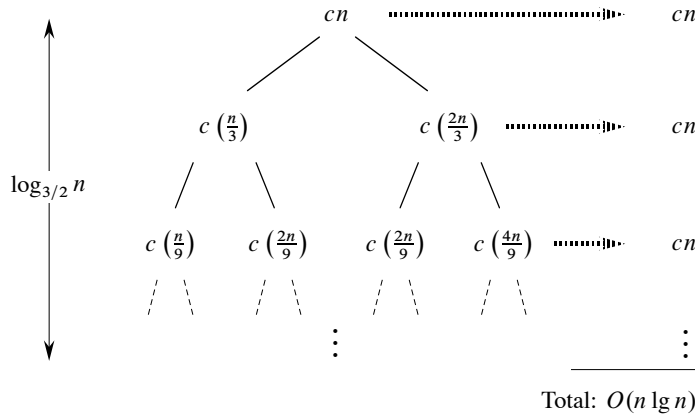


Figure 4.6 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

is bounded from above by the constant $16/13$. Since the root's contribution to the total cost is cn^2 , the root contributes a constant fraction of the total cost. In other words, the cost of the root dominates the total cost of the tree.

In fact, if $O(n^2)$ is indeed an upper bound for the recurrence (as we shall verify in a moment), then it must be a tight bound. Why? The first recursive call contributes a cost of $\Theta(n^2)$, and so $\Omega(n^2)$ must be a lower bound for the recurrence.

Now we can use the substitution method to verify that our guess was correct, that is, $T(n) = O(n^2)$ is an upper bound for the recurrence $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. We want to show that $T(n) \leq dn^2$ for some constant $d > 0$. Using the same constant $c > 0$ as before, we have

$$\begin{aligned}
 T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\
 &\leq 3d \lfloor n/4 \rfloor^2 + cn^2 \\
 &\leq 3d(n/4)^2 + cn^2 \\
 &= \frac{3}{16} dn^2 + cn^2 \\
 &\leq dn^2,
 \end{aligned}$$

where the last step holds as long as $d \geq (16/13)c$.

In another, more intricate, example, Figure 4.6 shows the recursion tree for

$$T(n) = T(n/3) + T(2n/3) + O(n).$$

(Again, we omit floor and ceiling functions for simplicity.) As before, we let c represent the constant factor in the $O(n)$ term. When we add the values across the levels of the recursion tree shown in the figure, we get a value of cn for every level.

The longest simple path from the root to a leaf is $n \rightarrow (2/3)n \rightarrow (2/3)^2 n \rightarrow \dots \rightarrow 1$. Since $(2/3)^k n = 1$ when $k = \log_{3/2} n$, the height of the tree is $\log_{3/2} n$.

Intuitively, we expect the solution to the recurrence to be at most the number of levels times the cost of each level, or $O(cn \log_{3/2} n) = O(n \lg n)$. Figure 4.6 shows only the top levels of the recursion tree, however, and not every level in the tree contributes a cost of cn . Consider the cost of the leaves. If this recursion tree were a complete binary tree of height $\log_{3/2} n$, there would be $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ leaves. Since the cost of each leaf is a constant, the total cost of all leaves would then be $\Theta(n^{\log_{3/2} 2})$ which, since $\log_{3/2} 2$ is a constant strictly greater than 1, is $\omega(n \lg n)$. This recursion tree is not a complete binary tree, however, and so it has fewer than $n^{\log_{3/2} 2}$ leaves. Moreover, as we go down from the root, more and more internal nodes are absent. Consequently, levels toward the bottom of the recursion tree contribute less than cn to the total cost. We could work out an accurate accounting of all costs, but remember that we are just trying to come up with a guess to use in the substitution method. Let us tolerate the sloppiness and attempt to show that a guess of $O(n \lg n)$ for the upper bound is correct.

Indeed, we can use the substitution method to verify that $O(n \lg n)$ is an upper bound for the solution to the recurrence. We show that $T(n) \leq dn \lg n$, where d is a suitable positive constant. We have

$$\begin{aligned}
 T(n) &\leq T(n/3) + T(2n/3) + cn \\
 &\leq d(n/3) \lg(n/3) + d(2n/3) \lg(2n/3) + cn \\
 &= (d(n/3) \lg n - d(n/3) \lg 3) \\
 &\quad + (d(2n/3) \lg n - d(2n/3) \lg(3/2)) + cn \\
 &= dn \lg n - d((n/3) \lg 3 + (2n/3) \lg(3/2)) + cn \\
 &= dn \lg n - d((n/3) \lg 3 + (2n/3) \lg 3 - (2n/3) \lg 2) + cn \\
 &= dn \lg n - dn(\lg 3 - 2/3) + cn \\
 &\leq dn \lg n,
 \end{aligned}$$

as long as $d \geq c/(\lg 3 - (2/3))$. Thus, we did not need to perform a more accurate accounting of costs in the recursion tree.

Exercises

4.4-1

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

4.4-2

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

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

$$T(n) = aT(n/b) + f(n), \quad (4.20)$$

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.

The recurrence (4.20) describes the running time of an algorithm that divides a problem of size n into a subproblems, each of size n/b , where a and b are positive constants. The a subproblems are solved recursively, each in time $T(n/b)$. The function $f(n)$ encompasses the cost of dividing the problem and combining the results of the subproblems. For example, the recurrence arising from Strassen's algorithm has $a = 7$, $b = 2$, and $f(n) = \Theta(n^2)$.

As a matter of technical correctness, the recurrence is not actually well defined, because n/b might not be an integer. Replacing each of the a terms $T(n/b)$ with either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil)$ will not affect the asymptotic behavior of the recurrence, however. (We will prove this assertion in the next section.) We normally find it convenient, therefore, to omit the floor and ceiling functions when writing divide-and-conquer recurrences of this form.

The master theorem

The master method depends on the following theorem.

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. ■

Before applying the master theorem to some examples, let's spend a moment trying to understand what it says. In each of the three cases, we compare the function $f(n)$ with the function $n^{\log_b a}$. Intuitively, the larger of the two functions determines the solution to the recurrence. If, as in case 1, the function $n^{\log_b a}$ is the larger, then the solution is $T(n) = \Theta(n^{\log_b a})$. If, as in case 3, the function $f(n)$ is the larger, then the solution is $T(n) = \Theta(f(n))$. If, as in case 2, the two functions are the same size, we multiply by a logarithmic factor, and the solution is $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$.

Beyond this intuition, you need to be aware of some technicalities. In the first case, not only must $f(n)$ be smaller than $n^{\log_b a}$, it must be *polynomially* smaller.

That is, $f(n)$ must be asymptotically smaller than $n^{\log_b a}$ by a factor of n^ϵ for some constant $\epsilon > 0$. In the third case, not only must $f(n)$ be larger than $n^{\log_b a}$, it also must be polynomially larger and in addition satisfy the “regularity” condition that $af(n/b) \leq cf(n)$. This condition is satisfied by most of the polynomially bounded functions that we shall encounter.

Note that the three cases do not cover all the possibilities for $f(n)$. There is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially smaller. Similarly, there is a gap between cases 2 and 3 when $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger. If the function $f(n)$ falls into one of these gaps, or if the regularity condition in case 3 fails to hold, you cannot use the master method to solve the recurrence.

Using the master method

To use the master method, we simply determine which case (if any) of the master theorem applies and write down the answer.

As a first example, consider

$$T(n) = 9T(n/3) + n.$$

For this recurrence, we have $a = 9$, $b = 3$, $f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply case 1 of the master theorem and conclude that the solution is $T(n) = \Theta(n^2)$.

Now consider

$$T(n) = T(2n/3) + 1,$$

in which $a = 1$, $b = 3/2$, $f(n) = 1$, and $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$. Case 2 applies, since $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$, and thus the solution to the recurrence is $T(n) = \Theta(\lg n)$.

For the recurrence

$$T(n) = 3T(n/4) + n \lg n,$$

we have $a = 3$, $b = 4$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. Since $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon \approx 0.2$, case 3 applies if we can show that the regularity condition holds for $f(n)$. For sufficiently large n , we have that $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$. Consequently, by case 3, the solution to the recurrence is $T(n) = \Theta(n \lg n)$.

The master method does not apply to the recurrence

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

even though it appears to have the proper form: $a = 2$, $b = 2$, $f(n) = n \lg n$, and $n^{\log_b a} = n$. You might mistakenly think that case 3 should apply, since

$f(n) = n \lg n$ 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. $T(n) = 2T(n/4) + n^2$.

4.5-2

Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

4.5-3

Use the master method to show that the solution to the binary-search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(\lg n)$. (See Exercise 2.3-5 for a description of binary search.)

4.5-4

Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not? Give an asymptotic upper bound for this recurrence.

4.5-5 ★

Consider the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, which is part of case 3 of the master theorem. Give an example of constants $a \geq 1$ and $b > 1$ and a function $f(n)$ that satisfies all the conditions in case 3 of the master theorem except the regularity condition.

★ 4.6 Proof of the master theorem

This section contains a proof of the master theorem (Theorem 4.1). You do not need to understand the proof in order to apply the master theorem.

The proof appears in two parts. The first part analyzes the master recurrence (4.20), under the simplifying assumption that $T(n)$ is defined only on exact powers of $b > 1$, that is, for $n = 1, b, b^2, \dots$. This part gives all the intuition needed to understand why the master theorem is true. The second part shows how to extend the analysis to all positive integers n ; it applies mathematical technique to the problem of handling floors and ceilings.

In this section, we shall sometimes abuse our asymptotic notation slightly by using it to describe the behavior of functions that are defined only over exact powers of b . Recall that the definitions of asymptotic notations require that

bounds be proved for all sufficiently large numbers, not just those that are powers of b . Since we could make new asymptotic notations that apply only to the set $\{b^i : i = 0, 1, 2, \dots\}$, instead of to the nonnegative numbers, this abuse is minor.

Nevertheless, we must always be on guard when we use asymptotic notation over a limited domain lest we draw improper conclusions. For example, proving that $T(n) = O(n)$ when n is an exact power of 2 does not guarantee that $T(n) = O(n)$. The function $T(n)$ could be defined as

$$T(n) = \begin{cases} n & \text{if } n = 1, 2, 4, 8, \dots, \\ n^2 & \text{otherwise,} \end{cases}$$

in which case the best upper bound that applies to all values of n is $T(n) = O(n^2)$. Because of this sort of drastic consequence, we shall never use asymptotic notation over a limited domain without making it absolutely clear from the context that we are doing so.

4.6.1 The proof for exact powers

The first part of the proof of the master theorem analyzes the recurrence (4.20)

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

for the master method, under the assumption that n is an exact power of $b > 1$, where b need not be an integer. We break the analysis into three lemmas. The first reduces the problem of solving the master recurrence to the problem of evaluating an expression that contains a summation. The second determines bounds on this summation. The third lemma puts the first two together to prove a version of the master theorem for the case in which n is an exact power of b .

Lemma 4.2

Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b . Define $T(n)$ on exact powers of b by the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ aT(n/b) + f(n) & \text{if } n = b^i, \end{cases}$$

where i is a positive integer. Then

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j). \quad (4.21)$$

Proof We use the recursion tree in Figure 4.7. The root of the tree has cost $f(n)$, and it has a children, each with cost $f(n/b)$. (It is convenient to think of a as being

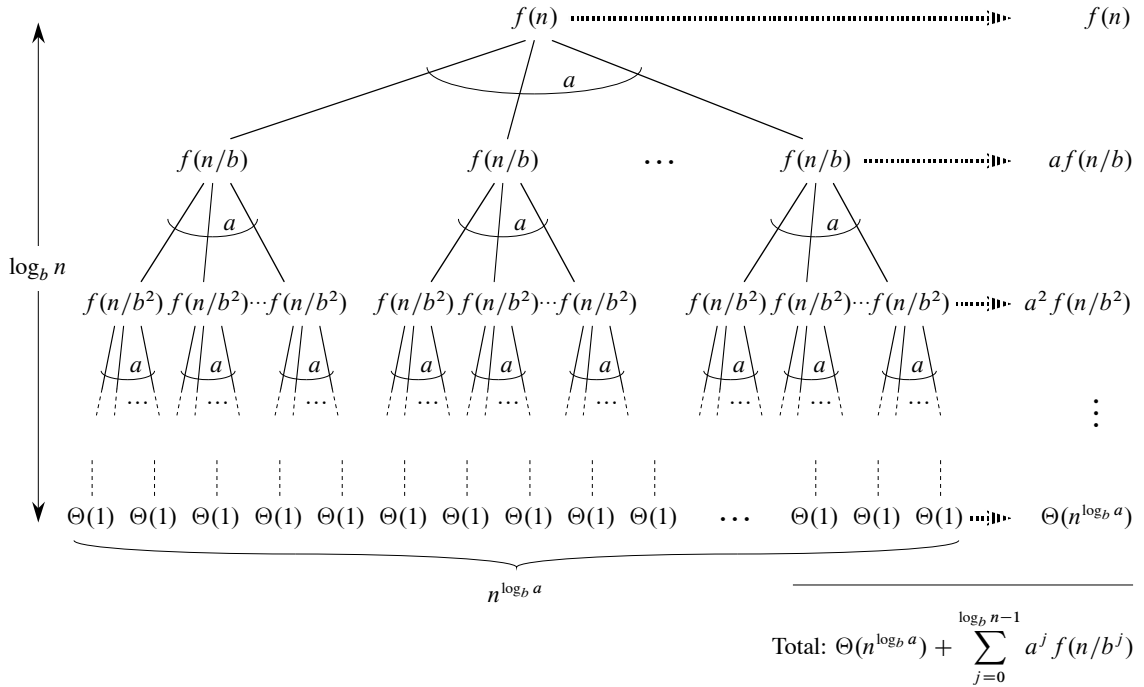


Figure 4.7 The recursion tree generated by $T(n) = aT(n/b) + f(n)$. The tree is a complete a -ary tree with $n^{\log_b a}$ leaves and height $\log_b n$. The cost of the nodes at each depth is shown at the right, and their sum is given in equation (4.21).

an integer, especially when visualizing the recursion tree, but the mathematics does not require it.) Each of these children has a children, making a^2 nodes at depth 2, and each of the a children has cost $f(n/b^2)$. In general, there are a^j nodes at depth j , and each has cost $f(n/b^j)$. The cost of each leaf is $T(1) = \Theta(1)$, and each leaf is at depth $\log_b n$, since $n/b^{\log_b n} = 1$. There are $a^{\log_b n} = n^{\log_b a}$ leaves in the tree.

We can obtain equation (4.21) by summing the costs of the nodes at each depth in the tree, as shown in the figure. The cost for all internal nodes at depth j is $a^j f(n/b^j)$, and so the total cost of all internal nodes is

$$\sum_{j=0}^{\log_b n - 1} a^j f(n/b^j).$$

In the underlying divide-and-conquer algorithm, this sum represents the costs of dividing problems into subproblems and then recombining the subproblems. The

cost of all the leaves, which is the cost of doing all $n^{\log_b a}$ subproblems of size 1, is $\Theta(n^{\log_b a})$. ■

In terms of the recursion tree, the three cases of the master theorem correspond to cases in which the total cost of the tree is (1) dominated by the costs in the leaves, (2) evenly distributed among the levels of the tree, or (3) dominated by the cost of the root.

The summation in equation (4.21) describes the cost of the dividing and combining steps in the underlying divide-and-conquer algorithm. The next lemma provides asymptotic bounds on the summation's growth.

Lemma 4.3

Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b . A function $g(n)$ defined over exact powers of b by

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \quad (4.22)$$

has the following asymptotic bounds for exact powers of b :

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for all sufficiently large n , then $g(n) = \Theta(f(n))$.

Proof For case 1, we have $f(n) = O(n^{\log_b a - \epsilon})$, which implies that $f(n/b^j) = O((n/b^j)^{\log_b a - \epsilon})$. Substituting into equation (4.22) yields

$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right). \quad (4.23)$$

We bound the summation within the O -notation by factoring out terms and simplifying, which leaves an increasing geometric series:

$$\begin{aligned} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j \\ &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j \\ &= n^{\log_b a - \epsilon} \left(\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}\right) \end{aligned}$$