APPENDIX B

Short Tutorial on Recurrence Relations

Sequences and Recurrence Relations

DEFINITION A (numerical) ***sequence*** is an ordered list of numbers.

Examples: 2, 4, 6, 8, 10, 12, . . . (positive even integers)

0, 1, 1, 2, 3, 5, 8, . . . (the Fibonacci numbers)

0, 1, 3, 6, 10, 15, . . . (numbers of key comparisons in selection sort)

A sequence is usually denoted by a letter (such as x or a) with a subindex (such as n or i) written in curly brackets, e.g., $\{x_n\}$. We use the alternative notation $x(n)$. This notation stresses the fact that a sequence is a function: its argument n indicates a position of a number in the list, while the function's value $x(n)$ stands for that number itself. $x(n)$ is called the ***generic term*** of the sequence.

There are two principal ways to define a sequence:

- by an explicit formula expressing its generic term as a function of n, e.g., $x(n) = 2n$ for $n \geq 0$
- by an equation relating its generic term to one or more other terms of the sequence, combined with one or more explicit values for the first term(s), e.g.,

$$x(n) = x(n-1) + n \quad \text{for } n > 0 \qquad \textbf{(B.1)}$$

$$x(0) = 0 \qquad \textbf{(B.2)}$$

It is the latter method that is particularly important for analysis of recursive algorithms (see Section 2.4 for a detailed discussion of this topic).

An equation such as (B1) is called a ***recurrence equation*** or ***recurrence relation*** (or simply a ***recurrence***), while an equation such as (B2) is called its ***initial***

condition. An initial condition can be given for a value of n other than 0 (e.g., for $n = 1$) and for some recurrences (e.g., for recurrence $F(n) = F(n-1) + F(n-2)$ defining the Fibonacci numbers—see Section 2.5), more than one value needs to be specified by initial conditions.

To solve a given recurrence subject to a given initial condition means to find an explicit formula for the generic term of the sequence that satisfies both the recurrence equation and the initial condition or to prove that such a sequence does not exist. For example, the solution to recurrence (B.1) subject to initial condition (B.2) is

$$x(n) = \frac{n(n+1)}{2} \quad \text{for } n \geq 0. \tag{B.3}$$

It can be verified by substituting this formula into (B.1) to check that the equality holds for every $n > 0$, i.e., that

$$\frac{n(n+1)}{2} = \frac{(n-1)(n-1+1)}{2} + n,$$

and into (B.2) to check that $x(0) = 0$, i.e., that

$$\frac{0(0+1)}{2} = 0.$$

Sometimes it is convenient to distinguish between a general solution and a particular solution to a recurrence. Recurrence equations typically have an infinite number of sequences that satisfy them. A ***general solution*** to a recurrence equation is a formula that specifies all such sequences. Typically, a general solution involves one or more arbitrary constants. For example, for recurrence (B.1), the general solution can be specified by the formula

$$x(n) = c + \frac{n(n+1)}{2}, \tag{B.4}$$

where c is such an arbitrary constant. By assigning different values to c, we can get all the solutions to equation (B.1) and only these solutions.

A ***particular solution*** is a specific sequence that satisfies a given recurrence equation. Usually we are interested in a particular solution that satisfies a given initial condition. For example, sequence (B.3) is a particular solution to (B.1)–(B.2).

Methods for Solving Recurrence Relations

No universal method exists that would enable us to solve every recurrence relation. (This is not surprising, because we do not have such a method even for solving much simpler equations in one unknown $f(x) = 0$ for an arbitrary function $f(x)$.) There are several techniques, however, some more powerful than others, that can solve a variety of recurrences.

Method of forward substitutions Starting with the initial term (or terms) of the sequence given by the initial condition(s), we can use the recurrence equation to generate the first few terms of its solution in the hope of seeing a pattern that can be expressed by a closed-end formula. If such a formula is found, its validity should be either checked by direct substitution into the recurrence equation and the initial condition (as we did for (B.1)–(B.2)) or proved by mathematical induction.

For example, consider the recurrence

$$x(n) = 2x(n-1) + 1 \quad \text{for } n > 1 \qquad \textbf{(B.5)}$$

$$x(1) = 1. \qquad \textbf{(B.6)}$$

We obtain the first few terms as follows:

$$\begin{aligned} x(1) &= 1 \\ x(2) &= 2x(1) + 1 = 2 \cdot 1 + 1 = 3 \\ x(3) &= 2x(2) + 1 = 2 \cdot 3 + 1 = 7 \\ x(4) &= 2x(3) + 1 = 2 \cdot 7 + 1 = 15. \end{aligned}$$

It is not difficult to note that these numbers are one less than consecutive powers of 2:

$$x(n) = 2^n - 1 \quad \text{for } n = 1, 2, 3, \text{ and } 4.$$

We can prove the hypothesis that this formula yields the generic term of the solution to (B.5)–(B.6) either by direct substitution of the formula into (B.5) and (B.6) or by mathematical induction.

As a practical matter, the method of forward substitutions works in a very limited number of cases because it is usually very difficult to recognize the pattern in the first few terms of the sequence.

Method of backward substitutions This method of solving recurrence relations works exactly as its name implies: using the recurrence relation in question, we express $x(n-1)$ as a function of $x(n-2)$ and substitute the result into the original equation to get $x(n)$ as a function of $x(n-2)$. Repeating this step for $x(n-2)$ yields an expression of $x(n)$ as a function of $x(n-3)$. For many recurrence relations, we will then be able to see a pattern and express $x(n)$ as a function of $x(n-i)$ for an arbitrary $i = 1, 2, \ldots$. Selecting i to make $n - i$ reach the initial condition and using one of the standard summation formulas often leads to a closed-end formula for the solution to the recurrence.

As an example, let us apply the method of backward substitutions to recurrence (B.1)–(B.2). Thus, we have the recurrence equation

$$x(n) = x(n-1) + n.$$

Replacing n by $n-1$ in the equation yields $x(n-1) = x(n-2) + n - 1$; after substituting this expression for $x(n-1)$ in the initial equation, we obtain

$$x(n) = [x(n-2) + n - 1] + n = x(n-2) + (n-1) + n.$$

Replacing n by $n-2$ in the initial equation yields $x(n-2)=x(n-3)+n-2$; after substituting this expression for $x(n-2)$, we obtain

$$x(n)=[x(n-3)+n-2]+(n-1)+n=x(n-3)+(n-2)+(n-1)+n.$$

Comparing the three formulas for $x(n)$, we can see the pattern that arises after i such substitutions:[1]

$$x(n)=x(n-i)+(n-i+1)+(n-i+2)+\cdots+n.$$

Since initial condition (B.2) is specified for $n=0$, we need $n-i=0$, i.e., $i=n$, to reach it:

$$x(n)=x(0)+1+2+\cdots+n=0+1+2+\cdots+n=n(n+1)/2.$$

The method of backward substitutions works surprisingly well for a wide variety of simple recurrence relations. You can find many examples of its successful applications throughout this book (see, in particular, Section 2.4 and its exercises).

Linear second-order recurrences with constant coefficients An important class of recurrences that can be solved by neither forward nor backward substitutions are recurrences of the type

$$ax(n)+bx(n-1)+cx(n-2)=f(n), \qquad \textbf{(B.7)}$$

where a, b, and c are real numbers, $a\neq 0$. Such a recurrence is called ***second-order linear recurrence with constant coefficients***. It is ***second-order*** because elements $x(n)$ and $x(n-2)$ are two positions apart in the unknown sequence in question; it is ***linear*** because the left-hand side is a linear combination of the unknown terms of the sequence; it has ***constant coefficients*** because of the assumption that a, b, and c are some fixed numbers. If $f(n)=0$ for every n, the recurrence is said to be ***homogeneous***; otherwise, it is called ***inhomogeneous***.

Let us consider first the homogeneous case:

$$ax(n)+bx(n-1)+cx(n-2)=0. \qquad \textbf{(B.8)}$$

Except for the degenerate situation of $b=c=0$, equation (B.8) has infinitely many solutions. All these solutions, which make up the general solution to (B.8), can be obtained by one of the three formulas that follow. Which of the three formulas applies to a particular case depends on the roots of the quadratic equation with the same coefficients as recurrence (B.8):

$$ar^2+br+c=0. \qquad \textbf{(B.9)}$$

1. Strictly speaking, the validity of the pattern's formula needs to be proved by mathematical induction on i. It is often easier, however, to get the solution first and then verify it (e.g., as we did earlier for $x(n)=n(n+1)/2$).

Quadratic equation (B.9) is called the ***characteristic equation*** for recurrence equation (B.8).

THEOREM 1 Let r_1, r_2 be two roots of characteristic equation (B.9) for recurrence relation (B.8).

Case 1 If r_1 and r_2 are real and distinct, the general solution to recurrence (B.8) is obtained by the formula

$$x(n) = \alpha r_1^n + \beta r_2^n,$$

where α and β are two arbitrary real constants.

Case 2 If r_1 and r_2 are equal to each other, the general solution to recurrence (B.8) is obtained by the formula

$$x(n) = \alpha r^n + \beta n r^n,$$

where $r = r_1 = r_2$ and α and β are two arbitrary real constants.

Case 3 If $r_{1,2} = u \pm iv$ are two distinct complex numbers, the general solution to recurrence (B.8) is obtained as

$$x(n) = \gamma^n[\alpha \cos n\theta + \beta \sin n\theta],$$

where $\gamma = \sqrt{u^2 + v^2}$, $\theta = \arctan v/u$, and α and β are two arbitrary real constants.

Case 1 of this theorem arises, in particular, in deriving the explicit formula for the nth Fibonacci number (Section 2.5). As another example, let us solve the recurrence

$$x(n) - 6x(n-1) + 9x(n-2) = 0.$$

Its characteristic equation

$$r^2 - 6r + 9 = 0$$

has two equal roots $r_1 = r_2 = 3$. Hence, according to Case 2 of Theorem 1, its general solution is given by the formula

$$x(n) = \alpha 3^n + \beta n 3^n.$$

If we want to find its particular solution for which, say, $x(0) = 0$ and $x(1) = 3$, we substitute $n = 0$ and $n = 1$ into the last equation to get a system of two linear equations in two unknowns. Its solution is $\alpha = 0$ and $\beta = 1$, and hence the particular solution is

$$x(n) = n3^n.$$

Let us now turn to the case of inhomogeneous linear second-order recurrences with constant coefficients.

THEOREM 2 The general solution to inhomogeneous equation (B.7) can be obtained as the sum of the general solution to the corresponding homogeneous equation (B.8) and a particular solution to inhomogeneous equation (B.7).

Since Theorem 1 gives a complete recipe for finding the general solution to a homogeneous second-order linear equation with constant coefficients, Theorem 2 reduces the task of finding all solutions to equation (B.7) to finding just one particular solution to it. For an arbitrary function $f(n)$ in the right-hand side of equation (B.7), it is still a difficult task with no general help available. For a few simple classes of functions, however, a particular solution can be found. Specifically, if $f(n)$ is a nonzero constant, we can look for a particular solution that is a constant as well.

As an example, let us find the general solution to the inhomogeneous recurrence

$$x(n) - 6x(n-1) + 9x(n-2) = 4.$$

If $x(n) = c$ is its particular solution, constant c must satisfy the equation

$$c - 6c + 9c = 4,$$

which yields $c = 1$. Since we have already found the general solution to the corresponding homogeneous equation

$$x(n) - 6x(n-1) + 9x(n-2) = 0,$$

the general solution to $x(n) - 6x(n-1) + 9x(n-2) = 4$ is obtained by the formula

$$x(n) = \alpha 3^n + \beta n 3^n + 1.$$

Before leaving this topic, we should note that the results analogous to those of Theorems 1 and 2 hold for the ***general linear k-th degree recurrence with constant coefficients***

$$a_k x(n) + a_{k-1} x(n-1) + \cdots + a_0 x(n-k) = f(n). \quad \textbf{(B.10)}$$

The practicality of this generalization is limited, however, by the necessity of finding roots of the k-th degree polynomial

$$a_k r^k + a_{k-1} r^{k-1} + \cdots + a_0 = 0, \quad \textbf{(B.11)}$$

which is the characteristic equation for recurrence (B.10).

Finally, there are several other, more sophisticated techniques for solving recurrence relations. Purdom and Brown [Pur85] provide a particularly thorough discussion of this topic from the analysis of algorithms perspective.

Common Recurrence Types in Algorithm Analysis

There are a few recurrence types that arise in the analysis of algorithms with remarkable regularity. This happens because they reflect one of the fundamental design techniques.

Decrease-by-one A decrease-by-one algorithm solves a problem by exploiting a relationship between a given instance of size n and a smaller instance of size $n-1$. Specific examples include recursive evaluation of $n!$ (Section 2.4) and insertion sort (Section 5.1). The recurrence equation for investigating the time efficiency of such algorithms typically has the form

$$T(n) = T(n-1) + f(n), \tag{B.12}$$

where function $f(n)$ accounts for the time needed to reduce an instance to a smaller one and to extend the solution of the smaller instance to a solution of the larger instance. Applying backward substitutions to (B.12) yields

$$\begin{aligned} T(n) &= T(n-1) + f(n) \\ &= T(n-2) + f(n-1) + f(n) \\ &= \dots \\ &= T(0) + \sum_{j=1}^{n} f(j). \end{aligned}$$

For a specific function $f(x)$, the sum $\sum_{j=1}^{n} f(j)$ can usually be either computed exactly or its order of growth ascertained. For example, if $f(n) = 1$, $\sum_{j=1}^{n} f(j) = n$; if $f(n) = \log n$, $\sum_{j=1}^{n} f(j) \in \Theta(n \log n)$; if $f(n) = n^k$, $\sum_{j=1}^{n} f(j) \in \Theta(n^{k+1})$. The sum $\sum_{j=1}^{n} f(j)$ can also be approximated by formulas involving integrals (see, in particular, the appropriate formulas in Appendix A).

Decrease-by-a-constant-factor A decrease-by-a-constant-factor algorithm solves a problem by reducing its instance of size n to an instance of size n/b ($b = 2$ for most but not all such algorithms), solving the smaller instance recursively, and then, if necessary, extending the solution of the smaller instance to a solution of the given instance. The most important example is binary search; other examples include exponentiation by squaring (introduction to Chapter 5), multiplication à la russe, and the fake-coin problem (Section 5.5).

The recurrence equation for investigating the time efficiency of such algorithms typically has the form

$$T(n) = T(n/b) + f(n), \tag{B.13}$$

where $b > 1$ and function $f(n)$ accounts for the time needed to reduce an instance to a smaller one and to extend the solution of the smaller instance to a solution of the larger instance. Strictly speaking, equation (B.13) is valid only for $n = b^k$,

$k = 0, 1, \ldots$. For values of n that are not powers of b, there is typically some round-off, usually involving the floor and/or ceiling functions. The standard approach to such equations is to solve them for $n = b^k$ first. Afterward, either the solution is tweaked to make it valid for all n's (see, for example, Section 4.3 and Problem 3 in Exercises 4.3), or the order of growth of the solution is established based on the ***smoothness rule*** (Theorem 4 in this appendix).

By considering $n = b^k$, $k = 0, 1, \ldots$, and applying backward substitutions to (B.13), we obtain the following:

$$\begin{aligned} T(b^k) &= T(b^{k-1}) + f(b^k) \\ &= T(b^{k-2}) + f(b^{k-1}) + f(b^k) \\ &= \ldots \\ &= T(1) + \sum_{j=1}^{k} f(b^j). \end{aligned}$$

For a specific function $f(x)$, the sum $\sum_{j=1}^{k} f(b^j)$ can usually be either computed exactly or its order of growth ascertained. For example, if $f(n) = 1$,

$$\sum_{j=1}^{k} f(b^j) = k = \log_b n.$$

If $f(n) = n$, to give another example,

$$\sum_{j=1}^{k} f(b^j) = \sum_{j=1}^{k} b^j = b\frac{b^k - 1}{b - 1} = b\frac{n - 1}{b - 1}.$$

Also, recurrence (B.13) is a special case of recurrence (B.14) covered by the ***Master Theorem*** (Theorem 5 in this appendix). According to this theorem, in particular, if $f(n) \in \Omega(n^d)$ where $d > 0$, then $T(n) \in \Omega(n^d)$ as well.

Divide-and-conquer A divide-and-conquer algorithm solves a problem by dividing its given instance into several smaller instances, solving each of them recursively, and then, if necessary, combining the solutions to the smaller instances into a solution to the given instance. Assuming that all smaller instances have the same size n/b, with a of them being actually solved, we get the following recurrence valid for $n = b^k$, $k = 1, 2, \ldots$:

$$T(n) = aT(n/b) + f(n), \qquad \textbf{(B.14)}$$

where $a \geq 1$, $b \geq 2$, and $f(n)$ is a function that accounts for the time spent on dividing the problem into smaller ones and combining their solutions. Recurrence (B.14) is called the ***general divide-and-conquer recurrence.***[2]

2. In our terminology, for $a = 1$, it covers decrease-by-a-constant-factor, not divide-and-conquer, algorithms.

Applying backward substitutions to (B.14) yields the following:

$$\begin{aligned}
T(b^k) &= aT(b^{k-1}) + f(b^k)\\
&= a[aT(b^{k-2}) + f(b^{k-1})] + f(b^k) = a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k)\\
&= a^2[aT(b^{k-3}) + f(b^{k-2})] + af(b^{k-1}) + f(b^k)\\
&= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k)\\
&= \ldots\\
&= a^kT(1) + a^{k-1}f(b^1) + a^{k-2}f(b^2) + \cdots + a^0f(b^k)\\
&= a^k[T(1) + \sum_{j=1}^{k} f(b^j)/a^j].
\end{aligned}$$

Since $a^k = a^{\log_b n} = n^{\log_b a}$, we get the following formula for the solution to recurrence (B.14) for $n = b^k$:

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j]. \quad \textbf{(B.15)}$$

Obviously, the order of growth of solution $T(n)$ depends on the values of the constants a and b and the order of growth of the function $f(n)$. Under certain assumptions about $f(n)$ discussed in the next section, we can simplify formula (B.15) and get explicit results about the order of growth of $T(n)$.

Smoothness Rule and the Master Theorem We mentioned earlier that the time efficiency of decrease-by-a-constant-factor and divide-and-conquer algorithms is usually investigated first for n's that are powers of b. (Most often, $b = 2$, as it is in binary search and mergesort; sometimes $b = 3$, as it is in the better algorithm for the fake-coin problem of Section 5.5, but it can be any integer greater than or equal to 2.) The question we are going to address now is when the order of growth observed for n's that are powers of b can be extended to all its values.

DEFINITION Let $f(n)$ be a nonnegative function defined on the set of natural numbers. $f(n)$ is called ***eventually nondecreasing*** if there exists some nonnegative integer n_0 so that $f(n)$ is nondecreasing on the interval $[n_0, \infty)$, i.e.,

$$f(n_1) \le f(n_2) \quad \text{for any } n_2 > n_1 \ge n_0.$$

For example, the function $(n - 100)^2$ is eventually nondecreasing, although it is decreasing on the interval [0, 100], and the function $\sin^2 \frac{\pi n}{2}$ is a function that is not eventually nondecreasing. The vast majority of functions we encounter in the analysis of algorithms *are* eventually nondecreasing. (Most of them are, in fact, nondecreasing on their entire domains.)

DEFINITION Let $f(n)$ be a nonnegative function defined on the set of natural numbers. $f(n)$ is called ***smooth*** if it is eventually nondecreasing and

$$f(2n) \in \Theta(f(n)).$$

It is easy to check that functions which do not grow too fast, including $\log n$, n, $n \log n$, and n^α where $\alpha \geq 0$, are smooth. For example, $f(n) = n \log n$ is smooth because

$$f(2n) = 2n \log 2n = 2n(\log 2 + \log n) = (2 \log 2)n + 2n \log n \in \Theta(n \log n).$$

Fast-growing functions, such as a^n where $a > 1$ and $n!$, are not smooth. For example, $f(n) = 2^n$ is not smooth because

$$f(2n) = 2^{2n} = 4^n \notin \Theta(2^n).$$

THEOREM 3 Let $f(n)$ be a smooth function as just defined. Then, for any fixed integer $b \geq 2$,

$$f(bn) \in \Theta(f(n)),$$

i.e., there exist positive constants c_b and d_b and a nonnegative integer n_0 such that

$$d_b f(n) \leq f(bn) \leq c_b f(n) \quad \text{for } n \geq n_0.$$

(The same assertion, with obvious changes, holds for the O and Ω notations.)

PROOF We will prove the theorem for the O notation only; the proof of the Ω part is the same. First, it is easy to check by induction that if $f(2n) \leq c_2 f(n)$ for $n \geq n_0$, then

$$f(2^k n) \leq c_2^k f(n) \quad \text{for } k = 1, 2, \ldots \text{ and } n \geq n_0.$$

The induction basis for $k = 1$ checks out trivially. For the general case, assuming that $f(2^{k-1}n) \leq c_2^{k-1} f(n)$ for $n \geq n_0$, we obtain

$$f(2^k n) = f(2 \cdot 2^{k-1} n) \leq c_2 f(2^{k-1} n) \leq c_2 c_2^{k-1} f(n) = c_2^k f(n).$$

(This proves the theorem for $b = 2^k$.) Consider now an arbitrary integer $b \geq 2$. Let k be a positive integer such that $2^{k-1} \leq b < 2^k$. We can estimate $f(bn)$ above by assuming without loss of generality that $f(n)$ is nondecreasing for $n \geq n_0$:

$$f(bn) \leq f(2^k n) \leq c_2^k f(n).$$

Hence, we can use c_2^k as a required constant for this value of b to complete the proof. ■

The importance of the notions introduced above stems from the following theorem.

THEOREM 4 ***(Smoothness Rule)*** Let $T(n)$ be an eventually nondecreasing function and $f(n)$ be a smooth function. If

$$T(n) \in \Theta(f(n)) \text{ for values of } n \text{ that are powers of } b,$$

where $b \geq 2$, then

$$T(n) \in \Theta(f(n)).$$

(The analogous results hold for the cases of O and Ω as well.)

PROOF We will prove just the O part; the Ω part can be proved by the analogous argument. By the theorem's assumption, there exist a positive constant c and a positive integer $n_0 = b^{k_0}$ such that

$$T(b^k) \leq cf(b^k) \quad \text{for } b^k \geq n_0,$$

$T(n)$ is nondecreasing for $n \geq n_0$, and $f(bn) \leq c_b f(n)$ for $n \geq n_0$ by Theorem 3. Consider an arbitrary value of $n, n \geq n_0$. It is bracketed by two consecutive powers of b: $n_0 \leq b^k \leq n < b^{k+1}$. Therefore,

$$T(n) \leq T(b^{k+1}) \leq cf(b^{k+1}) = cf(bb^k) \leq cc_b f(b^k) \leq cc_b f(n).$$

Hence, we can use the product cc_b as a constant required by the $O(f(n))$ definition to complete the O part of the theorem's proof. ■

Theorem 4 allows us to expand the information about the order of growth established for $T(n)$ on a convenient subset of values (powers of b) to its entire domain. Here is one of the most useful assertions of this kind.

THEOREM 5 ***(Master Theorem)*** Let $T(n)$ be an eventually nondecreasing function that satisfies the recurrence

$$T(n) = aT(n/b) + f(n) \quad \text{for } n = b^k,\ k = 1, 2, \ldots$$
$$T(1) = c,$$

where $a \geq 1$, $b \geq 2$, $c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if} \quad a < b^d \\ \Theta(n^d \log n) & \text{if} \quad a = b^d \\ \Theta(n^{\log_b a}) & \text{if} \quad a > b^d. \end{cases}$$

(Similar results hold for the O and Ω notations, too.)

PROOF We will prove the theorem for the principal special case of $f(n) = n^d$. (A proof of the general case is a minor technical extension of the same argument—see, e.g., [Cor01].) If $f(n) = n^d$, equality (B.15) yields for $n = b^k$, $k = 0, 1, \ldots,$

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} b^{jd}/a^j] = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j].$$

The sum in this formula is that of a geometric series, and therefore

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = (b^d/a)\frac{(b^d/a)^{\log_b n} - 1}{(b^d/a) - 1} \quad \text{if } b^d \neq a$$

and

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = \log_b n \text{ if } b^d = a.$$

If $a < b^d$, then $b^d/a > 1$, and therefore

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = (b^d/a)\frac{(b^d/a)^{\log_b n} - 1}{(b^d/a) - 1} \in \Theta((b^d/a)^{\log_b n}).$$

Hence,

$$\begin{aligned} T(n) &= n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j] \in n^{\log_b a}\Theta((b^d/a)^{\log_b n}) \\ &= \Theta(n^{\log_b a}(b^d/a)^{\log_b n}) = \Theta(a^{\log_b n}(b^d/a)^{\log_b n}) \\ &= \Theta(b^{d \log_b n}) = \Theta(b^{\log_b n^d}) = \Theta(n^d). \end{aligned}$$

If $a > b^d$, then $b^d/a < 1$, and therefore

$$\sum_{j=1}^{\log_b n} (b^d/a)^j = (b^d/a)\frac{(b^d/a)^{\log_b n} - 1}{(b^d/a) - 1} \in \Theta(1).$$

Hence,

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j] \in \Theta(n^{\log_b a}).$$

If $a = b^d$, then $b^d/a = 1$, and therefore

$$T(n) = n^{\log_b a}[T(1) + \sum_{j=1}^{\log_b n} (b^d/a)^j] = n^{\log_b a}[T(1) + \log_b n]$$

$$\in \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b b^d} \log_b n) = \Theta(n^d \log_b n).$$

Since $f(n) = n^d$ is a smooth function for any d, a reference to Theorem 4 completes the proof. ■

Theorem 5 provides a very convenient tool for a quick efficiency analysis of divide-and-conquer and decrease-by-a-constant-factor algorithms. You can find examples of such applications throughout the book.