CMU-CS-78-154

A General Method for Solving Divide-and-Conquer Recurrences

Jon Louis Bentley[1]
Dorothea Haken
James B. Saxe
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

13 December 1978

Abstract

The complexity of divide-and-conquer algorithms is often described by recurrence relations of the form

$$T(n) = kT(n/c) + f(n).$$

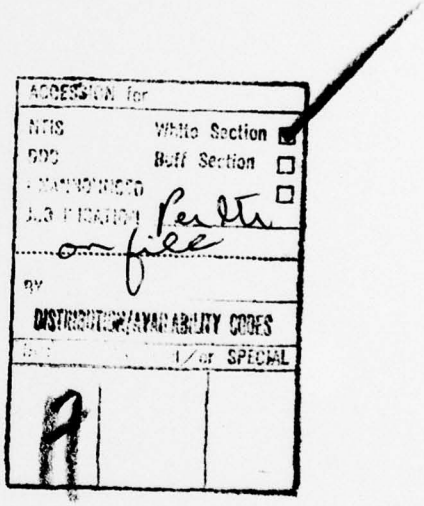
The only method currently available for solving such recurrences consists of solution tables for fixed functions f and varying k and c. In this note we describe a unifying method for solving these recurrences that is both general in applicability and easy to apply without the use of large tables.

---

1. Also with the Department of Mathematics.

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370.

Table of Contents



1. Introduction

The running times of recursive algorithms can often be analyzed by the use of recurrence relations. *Divide-and-conquer algorithms* are an important subclass of recursive algorithms (see Aho, Hopcroft and Ullman [1974, Section 2.6] for a discussion of this class). The recurrence relations describing the complexities of these algorithms often have the form[1]

$$T(1) \text{ given,}$$
$$T(n) = kT(n/c) + f(n).$$

Much work has been done on systematic ways of solving recurrences of this form; see, for example, Aho, Hopcroft and Ullman [1974, pp. 64-65], Borodin and Munro [1975, p. 80], Keller [1978], and Stanat and McAllister [1977, pp. 249, 255]. The only methods described by the above authors, however, consist of solution tables for fixed functions f and varying k and c. Although this method is quite satisfactory when applied to recurrences with common f, there are many times when a function f arises that has not been previously tabulated.

In this note we describe a new method for solving recurrences of the above form. Our method is based on rewriting the recurrence into a standard *template*, which can then be solved easily. This template is discussed in Section 2 for the particular case that $c = 2$. In Section 3 we illustrate the use of the template by solving a number of particular recurrences. The extension of the template to the case that $c \neq 2$ is the subject of Section 4. Further work that has already been done on solving a broader class of recurrences is then described in Section 5. The primary contribution of this paper is not in solving any particular new recurrences but rather in providing a general and succinct method by which a large class of recurrences may be solved. It has already been the experience of the authors that this method is an excellent didactic tool in the classroom context.

---

[1] Note that T(n) is defined only when n is a power of c; the implications of this restriction are discussed by Aho, Hopcroft and Ullman [1974, p. 65].

2. The Template

In this section we will investigate recurrences of the form

$$T(1) \text{ given}, \qquad (1)$$
$$T(n) = kT(n/2) + f(n).$$

As mentioned before, this recurrence is defined only for n a power of two. To solve the recurrence we will rewrite it into the *template*

$$T(1) \text{ given}, \qquad (2)$$
$$T(n) = 2^p T(n/2) + n^p g(n),$$

where $p = \lg k$ and $g(n) = f(n)/n^p$.[1] This new recurrence is easily solved; it has the unique solution

$$T(n) = n^p[T(1) + g(2) + g(4) + \ldots + g(n)],$$

which we abbreviate as

$$T(n) = n^p[T(1) + \tilde{g}(n)], \qquad (3)$$

where $\tilde{g}$ is defined as

$$\tilde{g}(n) = \sum_{1 \le i \le \lg n} g(2^i).$$

Mathematical induction on the powers of two can be used to show that Equation 3 is indeed the unique solution to the recurrence of Equation 2.

The above facts provide us with a method for solving any recurrence in the form of Equation 1. We cast it in the template of Equation 2 by doing a division and taking a logarithm, and then the solution to the recurrence is given by Equation 3. The only complicated part of this process is determining the sum implicit in the function $\tilde{g}$ in Equation 3, and this can usually be done with the aid of the following

---

[1] Throughout the paper we use lg as an abbreviation for $\log_2$ and $\lg^j n$ as an abbreviation for $(\lg n)^j$.

table in which we describe $\tilde{g}$ in relation to g.

Table 1.

g(n)	$\tilde{g}(n)$
$O(n^q)$ $q<0$	$O(1)$
$\lg^j n$ $j\geq 0$	$(\lg^{j+1} n)/(j+1) + (\lg^j n)/2 + \theta(\lg^{j-1} n)$
$\Omega(n^q)$ $q>0$	$\theta(g(n))$

For the third entry we use Ω in the following restricted sense: we write $g(n) = \Omega(n^q)$ if there exists an n_0 such that $g(cn) \geq c^q g(n)$ for all $c>1$ and all $n>n_0$.

3. Examples

In this section we will study a few common recurrences that have appeared in the literature and show that they can be solved by our method. Throughout this section we will mention where the recurrences arise in applications without giving complete bibliographic references to those applications; this is because our main point in mentioning the recurrences is not the applications themselves but the fact that these are common recurrences. (Further descriptions of and references to the applications may be found in the Appendix, however.) In all of these examples we will assume that T is defined only at powers of two and that some initial value T(1) is given.

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

This recurrence can be used to describe the worst-case cost of performing a binary search in an ordered table. The recurrence can be cast in the template of Equation 2 with $p = 0$ and $g(n) = 1$. By the second entry in Table 1 (setting $j = 0$) we have

$$\begin{aligned} T(n) &= 2^0 T(n/2) + n^0(\lg^0 n) \\ &= n^0[T(1) + \lg n + \theta(1)] \\ &= \lg n + \theta(1). \end{aligned}$$

Example 2. $T(n) = 2T(n/2) + n \lg n$

This recurrence arises in a multitude of applications, including algorithms for finding the maximal elements in a four-dimensional vector set, for evaluating normalized derivatives, for finding all nearest-neighbor pairs in three-dimensional point sets, and in Batcher's odd-even merge sort. To cast this recurrence in the template of Equation 2 we let $p = 1$ and $g(n) = \lg n$. We then use the second entry in Table 1 (with $j = 1$) which yields

$$\begin{aligned} T(n) &= 2^1 T(n/2) + n^1 \lg n \\ &= n^1[T(1) + (\lg^2 n)/2 + \theta(\lg n)] \\ &= (n \lg^2 n)/2 + \theta(n \lg n). \end{aligned}$$

Example 3. $T(n) = 7T(n/2) + \theta(n^2)$

This recurrence describes the running time of Strassen's matrix multiplication algorithm. We cast this recurrence into the template of Equation 2 by letting $p = \lg 7$ and $g(n) = n^{2 - \lg 7}$. Since $2 - \lg 7 < 0$ we use the first entry in Table 1 to conclude that

$$\begin{aligned} T(n) &= 2^{\lg 7} T(n/2) + n^{\lg 7} \theta(n^{2 - \lg 7}) \\ &= n^{\lg 7}[T(1) + O(1)] \\ &= \theta(n^{\lg 7}). \end{aligned}$$

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

This recurrence arises in algebraic complexity whenever Newton iteration is used to accomplish "extrapolative recursion"; it is also used in the analysis of the Ford-Johnson sorting algorithm. To cast this recurrence in the template of Equation 2 we let $p = 0$ and $g(n) = n \lg n$. Since $n \lg n = \Omega(n^1)$ we can use the third entry in Table 1 to deduce that

$$T(n) = 2^0 T(n/2) + n^0(n \lg n)$$

$= n^0[T(1) + \theta(n \lg n)]$
$= \theta(n \lg n)$.

4. A More General Template

The template developed in Section 2 can easily be generalized to solve recurrences of the form

$T(1)$ given,
$T(n) = kT(n/c) + f(n)$.

(Note that this generalizes Equation 1 of Section 2 from division by two to division by any constant.) To solve the recurrence we cast it into the template

$$T(n) = c^p T(n/c) + n^p g(n),$$

where $p = \log_c k$ and $g(n) = f(n)/n^p$. The solution to this modified recurrence is

$$T(n) = n^p[T(1) + \sum_{1 \le i \le \log_c n} g(c^i)].$$

As in Section 2, the proof that this is the unique solution to the recurrence can be performed by mathematical induction on the powers of c. For monotone functions g it can be proved that the above sum differs from $\tilde{g}(n)$ by at most a factor of lg c, so Table 1 can be used to solve the recurrence to within a constant factor.

This template can be used, for example, to analyze the running time of Pan's matrix multiplication algorithm, which satisfies the recurrence

$$T(n) = 143640\ T(n/70) + \theta(n^2).$$

This recurrence is rewritten into the template as

$$T(n) = 70^p T(n/70) + n^p g(n),$$

where p is defined as $\log_{70} 143640$, or approximately 2.79, and $g(n) = \theta(n^{2-p})$. The first entry of Table 1 tells us that $\tilde{g}(n) = O(1)$, so the solution of the recurrence

is

$$\begin{aligned} T(n) &= n^p[T(1) + \theta(\tilde{g}(n))] \\ &= n^p[T(1) + O(1)] \\ &= \theta(n^p), \end{aligned}$$

or approximately $\theta(n^{2.79})$. This method can also be employed to analyze other algorithms based on Pan's approach, in which 143640 and 70 are replaced by other constants.

5. Extensions

The work that we have described so far in this paper is a specialization of a more general theory for solving classes of recurrences. In this section we will sketch parts of the more general theory, which the authors will describe in detail in a future paper. It is the authors' contention, however, that the special theory is all that is necessary to solve many of the recurrences that arise in practice.

The special template that we have described in this note has been extended in several ways. In the future paper, Table 1 will be expanded to include many functions besides the three described in Section 2. We will also describe a method for "interpolating" to find values of the ~ operator for functions not in the table: if $g_2(n)$ is between $g_1(n)$ and $g_3(n)$, then $\tilde{g}_2(n)/g_2(n)$ is between $\tilde{g}_3(n)/g_3(n)$ and $\tilde{g}_1(n)/g_1(n)$. (We use "between" in a formal sense and assume "smoothness" properties of the function to show this result.) We will also show in the future paper how the linearity of the ~ operator (which follows from the linearity of $\sum$) can be used to find the second and higher order terms of the solution to the recurrence.

In the future paper the authors will show how to define templates to solve classes of recurrences of the form

$$\begin{aligned} &T(n_0) \text{ given}, \\ &T(n) = a(n)T(b(n)) + f(n). \end{aligned}$$

This will allow us to solve such recurrences as

$T(2)$ given,
$T(n) = T(n^{1/2}) + 1,$

which arises in the expected-time analysis of interpolation search, a $\theta(\lg \lg n)$ searching algorithm.

References

Aho, A. V., J. E. Hopcroft and J. D. Ullman [1974]. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Borodin, A. and I. Munro [1975]. *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, N. Y.

Keller, K. [1978]. "Computation cost functions for divide-and-conquer algorithms", to appear in *Journal of Undergraduate Mathematics*.

Stanat, D. F. and D. F. McAllister [1977]. *Discrete Mathematics in Computer Science*, Prentice-Hall, Englewood Cliffs, N. J.

Appendix

When we presented examples in the text we only mentioned the applications that motivated the recurrences and did not discuss them in any detail. In this appendix we will further describe the applications mentioned above and give bibliographic references.

The recurrence of Example 1 describes the worst-case complexity of binary search; see Knuth [1973, Section 6.2.1] for a description of that algorithm.

We mentioned that the recurrence of Example 2 arises in a number of applications. Kung, Luccio and Preparata [1975] describe the problem of computing all the *maximal* elements in a set of vectors. (A maximal vector is one which is not less than any other vector in all components.) They give algorithms for finding the maxima of k-dimensional vector sets; their algorithm for the case k=4 has running time modelled by this recurrence. Kung [1973] describes an algorithm for computing the n normalized derivatives $P^{(i)}(t)/i!$ for $i = 1,...,n$, where P is an n-th degree polynomial; the running time of his algorithm is also described by this recurrence. Given a set of n points in 3-space, the "All Nearest Neighbors" problem calls for finding the nearest neighbor for each point in the set among the rest of the points; Bentley and Shamos' [1976] algorithm for this problem has running time described by the recurrence. This recurrence also describes the number of comparators needed to implement Batcher's nonadaptive odd-even merge sort (which is described by Liu [1977, pp. 200-203]).

The recurrence of Example 3 describes the complexity of Strassen's [1969] sub-cubic matrix multiplication algorithm; this algorithm is also discussed by Aho, Hopcroft and Ullman [1974, pp. 230-232].

The recurrence of Example 4 arises in a number of algorithms based on "extrapolative recursion". This term is described in more detail by Borodin and Munro [1975, p. 80]; many examples of such algorithms can be found later in their

book. The same recurrence also arises in the analysis of the Ford-Johnson sorting algorithm (see Ford and Johnson [1959] or Knuth [1973]) which was the best-known sorting algorithm (in terms of using minimal comparisons) for almost twenty years--an algorithm faster for some values of n was recently obtained by Manacher [1977].

Pan's [1978] matrix multiplication algorithm was mentioned in Section 4; the authors suspect that many other descriptions (and modifications) of that algorithm will be available in the near future.

At the end of Section 5 we mentioned the analysis of interpolation search in a sorted table. An elegant analysis of this algorithm can be found in Perl and Reingold [1977].

References for Appendix

Aho, A. V., J. E. Hopcroft and J. D. Ullman [1974]. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Bentley, J. L. and M. I. Shamos [1976]. "Divide and conquer in multidimensional space", *Proceedings of the Eighth Symposium on the Theory of Computing*, ACM, May 1976, pp. 220-230.

Borodin, A. and I. Munro [1975]. *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, N.Y.

Ford, L. and S. Johnson [1959]. "A tournament problem", *American Mathematical Monthly* 66, pp. 391-395.

Knuth, D. E. [1973]. *The Art of Computer Programming, volume 3: Sorting and Searching*, Addison-Wesley, Reading, Mass.

Kung, H. T. [1973]. *A New Upper Bound on the Complexity of Derivative Evaluation*, Carnegie-Mellon University Computer Science Report, September 1973.

Kung, H. T., F. Luccio and F. P. Preparata [1975]. "On finding the maxima of a set of vectors", *JACM 22*, 4, October 1975, pp. 469-476.

Liu, C. L. [1977]. *Elements of Discrete Mathematics*, McGraw-Hill, New York, N. Y.

Manacher, G. K. [1977]. "The Ford-Johnson sorting algorithm is not optimal", *Proceedings of the Fifteenth Annual Allerton Conference on Communications, Control and Computing*, Sepember 1977, pp. 390-397.

Pan, V. Ya. [1978]. "An introduction to the trilinear technique of aggregating, uniting and canceling and applications of the technique for constructing fast algorithms for matrix operations", *Proceedings of the Nineteenth Annual Symposium on the Foundations of Computer Science*, IEEE, 1978.

Perl, Y. and E. M. Reingold [1977]. "Understanding the complexity of interpolation search", *Information Processing Letters* 6, December 1977, pp. 219-222.

Strassen, V. [1969]. "Gaussian elimination is not optimal", *Numerische Mathematik 13*, pp. 354-356.