Information Processing Letters 26 (1987/88) 321–324
North-Holland

25 January 1988

ON ESTIMATING THE COMPLEXITY OF LOGARITHMIC DECOMPOSITIONS

Marc BEZEM

Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Jan VAN LEEUWEN

Department of Computer Science, University of Utrecht, P.O. Box 80012, 3508 TA Utrecht, The Netherlands

Communicated by T. Lengauer
Received 11 April 1987

In the theory of dynamic data structures, one frequently encounters estimates of the type $[f](n) = \Sigma_{i \geq 0} b_i f(2^i)$, where $\ldots b_2 b_1 b_0$ is the binary representation of n and f is a (nondecreasing) function. We argue that 'smoothness' of f, i.e., $f(O(n)) = O(f(n))$, does not play a role in estimating $[f](n)$, contrary to the suggestion in some references. Moreover, we give a number of useful general bounds.

Keywords: Analysis of algorithms, computational complexity, data structure

1. Introduction

A data structure S(n) is generally termed *static* if it only supports insertions and deletions at high cost. In order to dynamize S(n), a number of techniques have been developed, which are based on the idea that it can be a lot cheaper to maintain a dynamic system of smaller data structures $S(n_i)$ with $\Sigma_{i \geq 0} n_i = n$. Insertions and deletions now operate on much smaller component structures and, presumably, the overhead required for maintaining the system of data structures is on the average very limited per instruction. If a searching problem is *decomposable* (as defined by Bentley and Saxe [2]), then it is often sufficient to design a good static data structure for it and rely on one of the available techniques to immediately obtain a competitive dynamic data structure (see [2,4,3] for a detailed account of the theory).

One common technique of dynamizing static data structures is known as the *logarithmic decomposition* (after Bentley [1]). It is based on decomposing a set of n elements into a collection of $1 + \lfloor \log n \rfloor$ subsets, each statically structured and containing $n_i = b_i 2^i$ elements ($0 \leq i \leq \lfloor \log n \rfloor$), where $\ldots b_2 b_1 b_0$ is the binary representation of n. In estimating the efficiency of operations which take f(n) time on S(n), one naturally arrives at expressions of the form $[f](n) = \Sigma_{i \geq 0} b_i f(2^i)$ in estimating the efficiency of these operations on the logarithmic structure. Assuming only that f is nondecreasing, it is clear that $[f](n) = O(f(n) \log n)$, but sometimes better bounds can be derived. In the literature on dynamization, several conditions for f can be found which supposedly imply the (interesting) bound of $[f](n) = O(f(n))$.

In this article we discuss some common fallacies in estimating [f](n), which hopefully leads to greater care in deriving bounds. We also give a number of useful techniques for estimating [f](n), which improve on the O(f(n) log n) bound in some cases.

2. Simple bounds

Let us first consider the (simple) problem of bounding $[f](n)$ for all 'common' functions. It is helpful to make some general observations first. We only consider functions with positive real values.

2.1. Lemma. *For all functions* $f(n)$ *and* $g(n)$:

(i) *If* $f(n) \leqslant g(n)$ *for all* n, *then* $[f](n) \leqslant [g](n)$.

(ii) *If* $g(n) = \Omega(1)$ *and* $f(n) = O(g(n))$, *then* $[f](n) = O([g](n))$.

Proof. (i) is trivial

To prove (ii), note that, by definition, there are a constant c and an integer n_0 such that $f(n) \leqslant cg(n)$ for all $n \geqslant n_0$. Thus, for $n \geqslant n_0$ one has:

$$[f](n) \leqslant \sum_{0 \leqslant i \leqslant \lfloor \log n_0 \rfloor} b_i f(2^i) + c \sum_{i \geqslant 0} b_i g(2^i)$$
$$= O(1) + c[g](n) = O([g](n)). \quad \square$$

2.2. Lemma. *For all functions* $f(n)$ *and* $g(n)$:

(i) $[fg](n) \leqslant [f](n)[g](n)$.

(ii) *If* $g(n)$ *is nondecreasing, then* $[fg](n) \leqslant [f](n)g(n)$.

Proof. (ii) is trivial, by term-wise estimating $[fg](n)$.

To prove (i), observe that

$$[fg](n) = \sum_{i \geqslant 0} b_i f(2^i) g(2^i) = \sum_{i \geqslant 0} \{b_i f(2^i) b_i g(2^i)\}$$

because the b_i's are 0–1 valued, and apply the common product inequality for sums with nonnegative terms. $\square$

It so happens that many bounding functions which arise in the analysis of algorithms are products of a limited collection of 'standard' functions. Products are handled by Lemma 2.2, and a representative set of standard functions is given in Fig. 1. (By Lemma 2.1 we do not have to worry about constant factors.)

Note that types I, II, and III and types VI, V, and IV (in this order) are symmetrically related by function 'inversion': if f belongs to one type, then $\tilde{f}$ defined by

$$\tilde{f}(n) = \min\{m \mid f(m) \geqslant n\}$$

belongs to the related type.

I. Super-exponential functions: 2^{2^n}, $2^{2^{2^n}}, \ldots, 2{*}^n$, where $2{*}^n$ is defined recursively by $2{*}^0 = 1$, $2{*}^{(n+1)} = 2^{(2{*}^n)}$, and Ackermann's function $\mathrm{Ack}(n, n, n)$.

II. Exponential functions: $c^{(n^\delta)}$, for constants $c > 1$ and $\delta > 0$.

III. Polynomial functions: n^k, for constant $k \geqslant 1$.

IV. Sub-polynomial functions: n^ε, for constant ε with $0 < \varepsilon < 1$.

V. Polylogarithmic functions: $(\log n)^\delta$, for constant $\delta > 0$.

VI. Sub-logarithmic functions: $\log\log n$, $\log\log\log n, \ldots, \log^* n$, where $\log^* n$ is defined by $\log^* n = \min\{m \mid 2{*}^m \geqslant n\}$, and $\alpha(n)$ is defined by $\alpha(n) = \min\{m \mid \mathrm{Ack}(m, m, m) \geqslant n\}$.

Fig. 1. Some standard bounding functions encountered in the analysis of algorithms.

We need the following useful concept.

2.3. Definition. A function f is said to have *cost factor* ϕ, with ϕ some function, if $[f](n) = O(f(n)\phi(n))$.

2.4. Lemma. *If* $f(n)/g(n)$ *is nondecreasing and* g *has cost factor* ϕ, *then* f *has cost factor* ϕ *as well.*

Proof. Write $f(n)$ as $g(n)f(n)/g(n)$ and apply Lemma 2.2(ii). $\square$

2.5. Lemma. *Let* f *be a standard function from* Fig. 1.

(i) *If* f *is of type* I, II, III *or* IV, *then* f *has cost factor* 1.

(ii) *If* f *is of type* V *or* VI, *or is a constant* > 0, *then* f *has cost factor* $\log n$.

Proof. (i) First suppose $f(n) = n^\varepsilon$, for constant ε with $0 < \varepsilon < 1$. Then

$$[f](n) = \sum_{i \geqslant 0} b_i (2^i)^\varepsilon \leqslant \sum_{0 \leqslant i \leqslant \lfloor \log n \rfloor} (2^\varepsilon)^i$$
$$= (2^{\varepsilon(1 + \lfloor \log n \rfloor)} - 1)/(2^\varepsilon - 1)$$
$$\leqslant (2^\varepsilon n^\varepsilon - 1)/(2^\varepsilon - 1)$$
$$= O(n^\varepsilon) = O(f(n)).$$

For functions f of type I, II, or III we can apply Lemma 2.4, since $f(n)/n^{\varepsilon}$ is nondecreasing for suitable ε with $0 < \varepsilon < 1$.

(ii) Trivial. □

Clearly, the functions with cost factor 1 are the most interesting in the theory of dynamic data structures, because they suggest that no increase in complexity is incurred by the logarithmic decomposition. For other functions it is of interest to try and improve on the worst-case bound of log n on the cost factor, for a similar reason. (One easily verifies that for the standard functions of type V and VI the cost factor of log n is attained infinitely often.) The following result, easily derived from Lemmas 2.4 and 2.5, seems to be the only result that is actually applied.

2.6. Corollary. *Let* f *be nondecreasing. If* $f(n)/n^{\delta}$ *is nondecreasing for some constant* $\delta > 0$, *then* $[f](n) = O(f(n))$, *otherwise* $[f](n) = O(f(n) \log n)$.

We shall see in the next section that log n is not the only alternative to a cost factor of 1, although it clearly is for the standard bounding functions.

While the proof of Corollary 2.6 is merely an exercise, its precise formulation is not apparent in various source references on dynamization (cf. [4]). At least in a number of cases one finds formulations like: if f is a 'well-behaved' function, then $[f](n) = O(f(n))$ if $f(n) = \Omega(n^{\delta})$ for some $\delta > 0$ and, otherwise, $[f](n) = O(f(n) \log n)$. We shall argue that these formulations are not correct under the assumptions usually made for 'well-behaved' functions, summarized in the following definition (cf. [3, p.9] or [4, p.6]).

2.7. Definition. A function f is called *smooth* if f(n) is nondecreasing and $f(O(n)) = O(f(n))$.

We show that smooth functions which satisfy $f(n) = \Omega(n^{\delta})$ for some $\delta > 0$ can 'accumulate' a surprisingly large cost factor in some cases, contrary to the suggestion in some references.

2.8. Proposition. *There exist smooth functions* f *with* $f(n) = \Omega(n)$ *such that* $[f](n) \neq O(f(n))$.

Proof. For $k \geqslant 0$ let $t(k) = \frac{1}{2}k(k+1)$ denote the *kth* triangular number. Let lt(n) be the unique k such that $2^{t(k-1)} \leqslant n < 2^{t(k)}$. Consider the function f defined as follows:

$$f(n) = \begin{cases} 2^{t(k)} & \text{if } \mathrm{lt}(n) = k \text{ is even,} \\ 2^{t(k-1)+2(\lfloor \log n \rfloor + 1 - t(k-1))} & \\ & \text{if } \mathrm{lt}(n) = k \text{ is odd.} \end{cases}$$

Note that f has the (constant) value $2^{t(k)}$ for n from $2^{t(k-1)}$ to $2^{t(k)}$ and k even, but climbs from $2^{t(k)+2}$ up to and including $2^{t(k+2)-1}$ for n from $2^{t(k)}$ to $2^{t(k+1)}$ in the next interval of triangular powers of 2. One verifies that f is nondecreasing and that $f(n) \leqslant f(2n) \leqslant 4f(n)$. It follows that f is smooth. Clearly, $f(n) = \Omega(n)$. To prove that $[f](n) \neq O(f(n))$ we evaluate $[f](n)$ for $n = 2^{t(k)} - 1$, k even:

$$\begin{aligned} [f](n) &= \sum_{0 \leqslant i \leqslant t(k)-1} f(2^i) \\ &\geqslant (t(k) - t(k-1)) f(2^{t(k)-1}) = k f(2^{t(k)} - 1) \\ &= k f(n) = \Omega\left(f(n)\sqrt{\log n}\right). \end{aligned}$$

As this holds for infinitely many n, it is clear that $[f](n) \neq O(f(n))$. □

The function f constructed in the proof above is also interesting because one can show that $[f](n) = O(f(n)\sqrt{\log n})$, which means it has cost factor $\sqrt{\log n}$ (see Corollary 3.2 below).

3. A general bounding technique

We now present a general technique of estimating [f] in terms of other characteristics of the function f. Let f be nondecreasing and $f(n) \neq O(1)$. It follows that for all m there exists an $n > m$ such that $f(n) > f(m)$. Define the sequence of 'jump' points $\{j_i\}_{i \geqslant 1}$ as follows:

$j_1 = 1,$

$j_{i+1} = \min\{n \mid f(n) > f(j_i)\}$ for $i \geqslant 1$.

The sequence $\{j_i\}_{i \geqslant 1}$ is strictly increasing and infinite, and consists precisely of the arguments where f 'increases' in value. Define the sequence of consecutive increments $\{u_i\}_{i \geqslant 1}$ as follows:

$$u_1 = f(j_1),$$
$$u_{i+1} = f(j_{i+1}) - f(j_i) \quad \text{for } i \geqslant 1.$$

For $i \geqslant 1$ we also need the quantities p_i defined by: p_i = "the number of powers of 2 among the integers from j_i to j_{i+1}". One easily verifies that

$$p_i = \lfloor \log(j_{i+1} - 1) \rfloor - \lceil \log j_i \rceil + 1.$$

The crucial point to observe is that

$$[f](n) \leqslant p_1 u_1 + p_2(u_1 + u_2) + \cdots + p_k(u_1 + \cdots + u_k),$$

where $k = k(n)$ is such that $j_k \leqslant n < j_{k+1}$ (i.e., $k(n) = \max\{i \mid j_i \leqslant n\}$).

3.1. Theorem. *If there are a constant* $c > 1$ *and a function* ϕ *such that* $u_{i+1}/u_i \geqslant c$ *for all* i *and* $p_i \leqslant \phi(n)$ *for all* n *and* $1 \leqslant i \leqslant k(n)$, *then* f *has cost factor* ϕ.

Proof. We have

$$\begin{aligned}[f](n) &\leqslant p_1 u_1 + p_2(u_1 + u_2) + \cdots + p_k(u_1 + \cdots + u_k) \\ &= u_1(p_1 + \cdots + p_k) + u_2(p_2 + \cdots + p_k) + \cdots + u_k p_k.\end{aligned}$$

Using that $u_i \leqslant c^{-1} u_{i+1}$ for $i \geqslant 1$, whence $u_i \leqslant c^{i-k} u_k$ for $1 \leqslant i \leqslant k = k(n)$, and $p_i \leqslant \phi(n)$ for the same range, we obtain

$$\begin{aligned}[f](n) &\leqslant u_k \phi(n) \sum_{1 \leqslant i \leqslant k} (k - i + 1) c^{i-k} \\ &\leqslant f(n)\phi(n) \sum_{i \geqslant 1} i c^{-i} = O(f(n)\phi(n)). \quad \square\end{aligned}$$

3.2. Corollary. *If there are a constant* $c > 1$ *and a function* ϕ *such that* $u_{i+1}/u_i \geqslant c$ *for all* i *and* $j_{i+1}/j_i \leqslant 2^{\phi(n)}$ *for all* n *and* $1 \leqslant i \leqslant k(n)$, *then* f *has cost factor* $\phi + 1$.

Proof. The proof immediately follows from Theorem 3.1 by observing that

$$p_i \leqslant 1 + \log(j_{i+1}/j_i) \leqslant 1 + \phi(n). \quad \square$$

Both Theorem 3.1 and Corollary 3.2 show the relation between the 'growth' of a function and its cost factor, and can be used for instance to prove that the function f constructed in the proof of Proposition 2.8 has cost factor $\sqrt{\log n}$ (take $c = \frac{4}{3}$ and $\phi(n) = 1 + \sqrt{2 \log n}$). The condition on the sequence $\{u_i\}_{i \geqslant 1}$ clearly restricts the application to functions f with increments which grow at least exponentially.

There are various other bounds which can be derived along similar lines, starting from the basic estimate

$$[f](n) \leqslant p_1 u_1 + p_2(u_1 + u_2) + \cdots + p_k(u_1 + \cdots + u_k)$$

or, equivalently,

$$[f](n) \leqslant u_1(p_1 + \cdots + p_k) + u_2(p_2 + \cdots + p_k) + \cdots + u_k p_k$$

with $k = k(n)$. We leave this to the reader.

References

[1] J.L. Bentley, Decomposable searching problems, Inform. Process. Lett. 8 (1979) 244–251.

[2] J.L. Bentley and J.B. Saxe, Decomposable searching problems, Part I: Static-to-dynamic transformations, J. Algorithms 1 (1980) 301–358.

[3] K. Mehlhorn, Data Structures and Algorithms, Vol. 3: Multi-Dimensional Searching and Computational Geometry (Springer, Berlin, 1984).

[4] M.H. Overmars, The Design of Dynamic Data Structures, Lecture Notes in Computer Science, Vol. 156 (Springer Berlin, 1983).