Prof. Sergio A. Alvarez
Fulton Hall 410–B
Computer Science Department
Boston College
Chestnut Hill, MA 02467 USA

http://www.cs.bc.edu/∼alvarez/
alvarez@cs.bc.edu
voice: (617) 552-4333
fax: (617) 552-6790

CS383, Algorithms
Notes on Divide and Conquer Recurrences

Divide and conquer algorithms will typically deal with an input instance of size n by dividing it into a smaller instances, each of size approximately n/b, recursively construct a solution for each of these smaller instances, and then combine these solutions to construct a solution for the original instance. Letting $T(n)$ denote the total running time of the algorithm for an input instance of size n, and assuming that combination of solutions takes time $c(n)$, this leads to the following relationship satisfied by T:

$$T(n) = aT(n/b) + f(n) \tag{1}$$

We discuss the solution of a class of recurrences of this form. Throughout, we assume that a base case for the recursion is provided by specifying the running time for some instance size n_0.

1 Series Expansion of the Basic Divide and Conquer Recurrence

Expanding the right-hand side of Eq. 1 by applying the equation a second time (for n/b instead of n) yields the following version of the recurrence

$$T(n) = a(aT(n/b^2) + f(n/b)) + f(n)$$

Iterating this process k times produces:

$$T(n) = a^k T(n/b^k) + \sum_{j=0}^{k} a^j f(n/b^j)$$

Since recursion stops when n reaches some base level $n = n_0$, the repeated expansion technique can only be applied for k as long as $n/b^k \geq n_0$, that is, $k \leq \log_b n/n_0$. For $k_0 = \log_b n/n_0$, we obtain:

$$T(n) = C_0 + \sum_{j=0}^{k_0} a^j f(n/b^j) \tag{2}$$

where

$$C_0 = a^{k_0} T(n_0)$$

2 Solution of the Recurrence for Polynomial Combination Times

Explicit solution of Eq. 2 is possible only in certain cases. We will assume specifically that the combination time $f(n)$ grows at a polynomial rate:

$$f(n) = O(n^p)$$

Suppose concretely that a finite constant C exists such that:

$$f(n) \leq Cn^p \text{ for all } n$$

Eq. 2 then becomes:

$$T(n) \leq C_0 + C \sum_{j=0}^{\log_b n/n_0} a^j (n/b^j)^p = C_0 + Cn^p \sum_{j=0}^{\log_b n/n_0} (a/b^p)^j \tag{3}$$

The summation on the right-hand side is a partial sum of a geometric series. Recall the formula for such a geometric sum with a positive base, r:

$$\sum_{j=0}^{m} r^j = \begin{cases} \frac{r^{m+1}-1}{r-1}, & \text{if } r \neq 1 \\ m+1, & \text{if } r = 1 \end{cases}$$

We are interested in the asymptotic behavior of such a sum for large values of m. Note that r^m grows exponentially with m if $r > 1$ and decays exponentially if $r < 1$. Therefore, we have:

$$\sum_{j=0}^{m} r^j = \begin{cases} O(r^m), & \text{if } r > 1 \\ O(m), & \text{if } r = 1 \\ O(1), & \text{if } r < 1 \end{cases}$$

In the case of Eq. 3, the base r is a/b^p and the upper summation limit m is $\log_b n/n_0$. Asymptotically, $\log_b n/n_0 = \log_b n - \log_b n_0 = O(\log_b n)$. Also,

$$(a/b^p)^{\log_b n} = a^{\log_b n}/n^p = (b^{\log_b a})^{\log_b n} = n^{\log_b a}/n^p$$

Hence, using the formula for the asymptotic behavior of a geometric sum, Eq. 3 yields:

$$T(n) = \begin{cases} O(n^{\log_b a}), & \text{if } a > b^p \\ O(n^p \log_b n), & \text{if } a = b^p \\ O(n^p), & \text{if } a < b^p \end{cases} \tag{4}$$

This result is equivalent to what the book calls the Master Theorem.

Algorithm 1: Divide and Conquer Multiplication
Input: Bit strings $a[1...d]$ and $b[1...d]$ that represent non-negative integers.
Output: The product $a * b$ of the corresponding integers.
$\textsc{recMult}(a,\ b)$
if a and b occupy one machine word or less **then return** $a * b$
$\text{high}(a) = a[1...\lfloor d/2 \rfloor]$, $\text{high}(b) = b[1...\lfloor d/2 \rfloor]$
$\text{low}(a) = a[\lfloor d/2 \rfloor + 1...d]$, $\text{low}(b) = b[\lfloor d/2 \rfloor + 1...d]$
$x = \textsc{recMult}(\text{high}(a), \text{high}(b))$
$y = \textsc{recMult}(\text{low}(a), \text{low}(b))$
$z = \textsc{recMult}(\text{high}(a) + \text{low}(a), \text{high}(b) + \text{low}(b))$
return $2^d x + 2^{d/2}(z - x - y) + y$

2.1 Examples of Divide and Conquer Recurrence Analysis

1. Consider the divide and conquer approach to multiplication that we discussed in class, which appears in Algorithm 1.

   Let $T(d)$ be the running time of recMult for d-digit inputs. The time required for extracting high and low parts, adding, subtracting, and multiplying by powers of 2 (left shift) is $O(d)$. Since the number of recursive calls is 3, the recurrence relation for the running time is as follows:

   $$T(d) = 3T(d/2) + O(d)$$

   We apply the Master Theorem discussed above, with $a = 3,\ b = 2,\ p = 1$. In this case, $a > b^p$, so the running time satisfies:

   $$T(d) = O(d^{\log_2 3})$$

2. Design a divide and conquer algorithm for finding the largest element of an unsorted array of positive integers, and analyze its running time.

Algorithm 2: Divide and Conquer Maximum
Input: An array $a[1...n]$ of strictly positive integers.
Output: The largest value among all elements of a.
$\textsc{dcMax}(a)$
if a is empty **then return** 0
if a has length 1 **then return** $a[1]$
return $\textsc{max}(\textsc{dcMax}(a[1...\lfloor n/2 \rfloor]), \textsc{dcMax}(a[1 + \lfloor n/2 \rfloor...n]))$

We will assume that the integers stored in the input array fit into a single machine word. The maximum of two such integers may be computed in time $O(1)$. Hence, the recurrence relation for the running time is:

$$T(n) = 2T(n/2) + O(1)$$

Applying the Master Theorem with $a = 2,\ b = 2,\ p = 0$ (note that $a > b^p$), we find that the running time satisfies $T(n) = O(n)$. This is not surprising. After all, this particular divide and conquer algorithm is just a recursive recast of an exhaustive search algorithm.