I. Core / B. Divide and Conquer

- General introduction (last week Wednesday)
- Solving recurrences (last week Friday)
- The master method (**today**)
- Quick sort (Wednesday)

Today:

- Brief perusal of Section 4.4, recursion-tree method
- The intuition of the master method
- Using the master method

Formal definitions:

$$\Theta(g(n)) = \{f(n) \mid \exists\ c_1, c_2, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

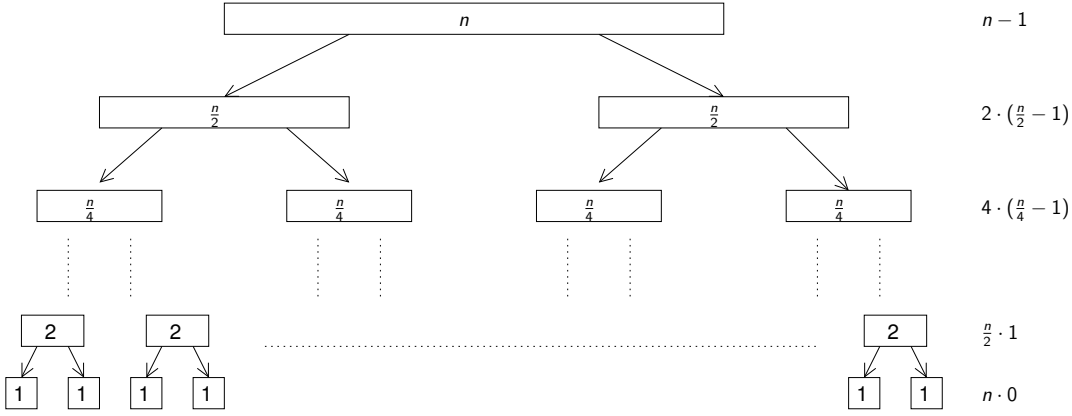
$$O(g(n)) = \{f(n) \mid \exists\ c, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq f(n) \leq c\ g(n)\}$$

$$\Omega(g(n)) = \{f(n) \mid \exists\ c, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c\ g(n) \leq f(n)\ \}$$

$$o(g(n)) = \{f(n) \mid \forall\ c \in \mathbb{R}^+, \exists\ n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq f(n) < c\ g(n)\ \}$$

$$\omega(g(n)) = \{f(n) \mid \forall\ c \in \mathbb{R}^+, \exists\ n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c\ g(n) < f(n)\ \}$$

$$C_{ms}(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ n - 1 + 2C_{ms}(\frac{n}{2}) & \text{otherwise} \end{cases}$$



A General Method for Solving Divide-and-Conquer Recurrences[1]

Jon Louis Bentley[2]
Dorothea Haken
James B. Saxe
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

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

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

The only well-defined method currently used 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. This method is appropriate both as a classroom technique and as a tool for practicing algorithm designers.

7. Conclusions

To conclude this paper we will briefly review its contents. The primary contribution is the *method* that we presented to solve the various recurrences--rewriting the recurrence by decomposing the additive term into some function times a solution of the homogeneous system. We described this method both in the very general terms of a recursion tree, and in the specific framework of divide-and-conquer recurrences. This method can be applied by memorizing a simple template (or the recursion tree formulation) and a table of three entries. While the novice can use this framework to solve (approximately) divide-and-conquer recurrences, the more experienced algorithm designer can use it in a more advanced fashion: to prune his search for an efficient algorithm. For example, if he were trying to find an $O(n \lg^3 n)$ algorithm by marrying together the solution to two problems of size n/2 each, his marriage step must require $O(n \lg^2 n)$ time.

Much work remains to be done in developing general methods for rapidly solving equations that describe the approximate complexity of algorithms; we mentioned some areas for further work in Section 6. Such methods will never be sufficient for the detailed "Knuthian" analysis of algorithms, but they can free algorithm designers from mundane analyses to let them work on more interesting problems.

Acknowledgements

The authors would like to thank Professor D. E. Knuth for (properly) chiding us about the "cook book" flavor of an earlier draft, and Professor V. Pratt for suggesting the recursion tree formulation of Section 6.

The Master theorem (CLRS pg 94). Let $T : \mathbb{N} \to \mathbb{N}$ be defined by the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$.

- If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
- If $f(n) = \Theta(n^{\log_b a})$ for some $\epsilon > 0$, then $f(n) = \Theta(n^{\log_b a} \lg n)$
- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $a \cdot f\left(\frac{n}{b}\right) = O(f(n))$, then $T(n) = \Theta(f(n))$

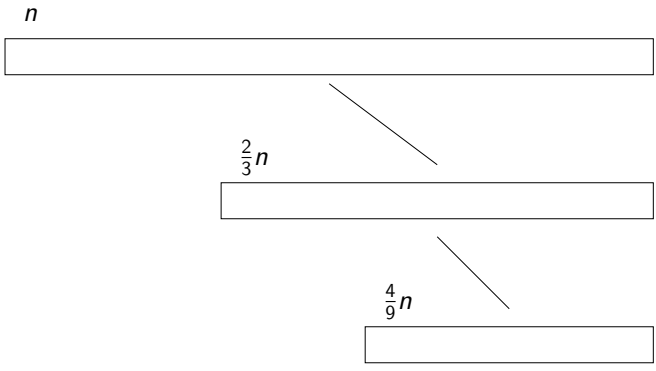
Understanding the Master theorem:

The work done at the leaves is inherently $\Theta(n^{\log_b a})$

binary search	merge sort
$a = 1 \quad b = 2$	$a = 2 \quad b = 2$
1 leaf	n leaves
$n^{\log_2 1} = n^0 = 1$	$n^{\log_2 2} = n^1$

Understanding the Master theorem:

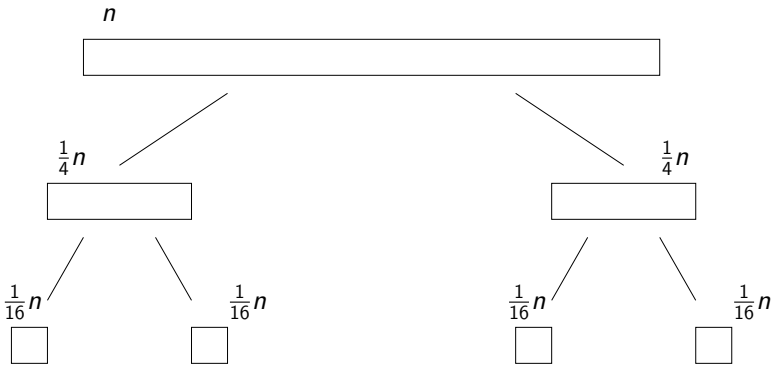
Imagine throwing away $\frac{1}{3}$ of the problem each time.



$a = 1$, $b = \frac{3}{2}$, number of leaves: $n^{\log_{\frac{3}{2}} 1} = 1$.

Understanding the Master theorem:

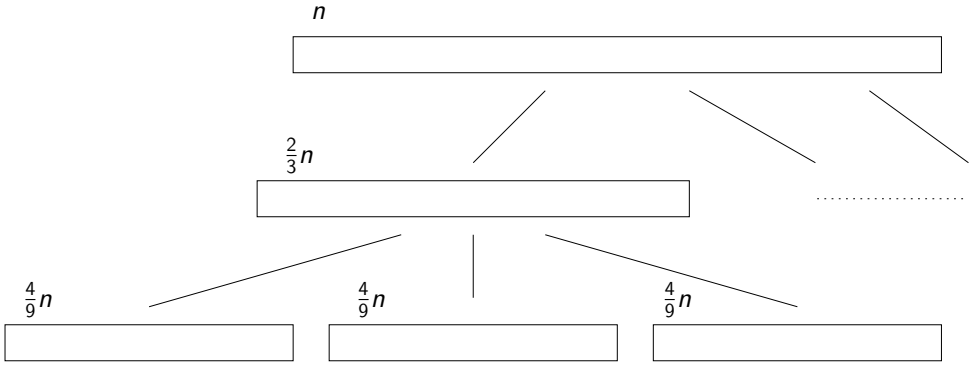
Imagine throwing away two quarters of the problem each time, keeping two (independent) quarters.



$a = 2$, $b = 4$, number of leaves: $n^{\log_4 2} = n^{\frac{1}{2}}$.

Understanding the Master theorem:

Imagine three *overlapping* subproblems, each with size $\frac{2}{3}$



$a = 3$, $b = \frac{3}{2}$, number of leaves: $n^{\log_{\frac{3}{2}} 3} \approx n^{2.7}$.

Understanding the Master theorem:

Let a be the number of subproblems and b be the factor by which the subproblems are decreasing in size (size of subproblems are $\frac{n}{b}$). Then

- The number of leaves is $\Theta(n^{\log_b a})$.
- Assuming a constant amount of work for each leaf, the work done at the leaves is $\Theta(n^{\log_b a})$.
- Thus the total work done by the algorithm is $\Omega(n^{\log_b a})$.

Understanding the Master theorem:

In the recursion tree, what dominates—the work at the *root* or at the *leaves*?

- If one clearly (*polynomially*) dominates, then the whole work is Θ of whichever it is.
- If they're asymptotically equivalent, then multiply the work at each level by the height of the tree, which is $\Theta(\lg n)$.
- Otherwise, you're out of luck.

Understanding the Master theorem—a less-formal, big-oh-only version:

If $T(n) \leq aT(\frac{n}{b}) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d \lg n) & \text{if } a = b^d \quad \text{(same work at each level)} \\ O(n^d) & \text{if } a < b^d \quad \text{(root dominates)} \\ O(n^{\lg_b a}) & \text{if a} > b^d \quad \text{(leaves dominate)} \end{cases}$$

The Master theorem (CLRS pg 94). Let $T : \mathbb{N} \to \mathbb{N}$ be defined by the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$.

- If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
- If $f(n) = \Theta(n^{\log_b a})$ for some $\epsilon > 0$, then $f(n) = \Theta(n^{\log_b a} \lg n)$
- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $a \cdot f\left(\frac{n}{b}\right) = O(f(n))$, then $T(n) = \Theta(f(n))$

The "regularity" condition, that is there exists c such that

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

The amount of work at the next level needs to be "smaller" (asymptotically no bigger than) the work at the current level.

Ex. 4.5-1.

In each of these, $a = 2$, $b = 4$. $n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}$

a. $T(n) = 2T(\frac{n}{4}) + 1$

$$f(n) = 1 = O(n^{\log_4 2 - \epsilon}) = O(n^{\frac{1}{2} - \epsilon})$$

where $\epsilon = \frac{1}{4}$, for example. hence $\Theta(n^{\frac{1}{2}})$.

b. $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

$$f(n) = \sqrt{n} = n^{\frac{1}{2}} = \Theta(n^{\frac{1}{2}})$$

So $\Theta(n^{\frac{1}{2}} \lg n)$

Ex. 4.5-1.

$a = 2,\ b = 4.\ n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}}$

c. $T(n) = 2T(\frac{n}{4}) + n$

$$f(n) = n = \Omega(n^{\frac{1}{2}+\epsilon})$$

where $\epsilon = \frac{1}{4}$, for example. So $\Theta(n)$.

d. $T(n) = 2T(\frac{n}{4}) + n^2$

$$f(n) = n^2 = \Omega(n^{\frac{1}{2}+\epsilon})$$

where $\epsilon = 1$, for example. So $\Theta(n^2)$.

4-1.

a. $T(n) = 2T(\frac{n}{2}) + n^4$

Using the master method, $a = 2$, $b = 2$, and $f(n) = n^4 = \Omega(n^{1+\epsilon})$ where $\epsilon = 2$. So, $\Theta(n^4)$.

Using the substitution method, guess cn^4. Then

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + n^4 \\ &= 2c(\tfrac{n}{2})^4 + n^4 \\ &= \tfrac{c}{8}n^4 + n^4 \\ &= (\tfrac{c+8}{8})n^4 \end{aligned}$$

We need $\frac{c+8}{8} = c$, so $c = \frac{8}{7}$.

4-1.

b. Using the master method, $a = 1$, $b = \frac{10}{7}$. Note that $\log_{\frac{10}{7}} 1 = 0$.
$f(n) = n = \Omega(n^{0+\epsilon})$ where $\epsilon = \frac{1}{2}$. So, $\Theta(n)$.
Using the substitution method, guess cn. Then

$$\begin{aligned} T(n) &= T(\tfrac{7n}{10}) + n \\ &= c(\tfrac{7}{10}n) + n \\ &= \tfrac{7c+10}{10}n \\ \tfrac{7c+10}{10} &= c \\ 7c + 10 &= 10c \\ c &= \tfrac{10}{3} \end{aligned}$$

4.5-4. Can we use the Master method on $T(n) = 4T(\frac{n}{2}) + n^2 \lg n$?
No. $a = 4$, $b = 2$, $\log_2 4 = 2$. Note that $n^2 \lg n = \Omega(n^2)$, but there does not exist ϵ such that $n^2 \lg n = \Omega(n^{2-\epsilon})$

For next time

Read sections 7.(1-3)

Do Ex 7.1-(2,3,4) and 7.2-(3 & 4)