Design and Analysis of Algorithms I

Master Method

Motivation

Integer Multiplication Revisited

<u>Motivation </u>: potentially useful algorithmic ideas often need mathematical analysis to evaluate

<u>Recall</u> : grade-school multiplication algorithm uses $\theta(n^2)$ operation to multiply two n-digit numbers

A Recursive Algorithm

Recursive approach

Write $x = 10^{n/2}a + b \qquad y = 10^{n/2}c + d$

[where a,b,c,d are n/2 – digit numbers]

So :

$$x \cdot y = 10^{n}ac + 10^{n/2}(ad + bc) + bd \qquad (*)$$

Algorithm#1 : recursively compute ac,ad,bc,bd, then compute (*) in the obvious way.

A Recursive Algorithm

T(n) = maximum number of operations this algorithm needs to multiply two n-digit numbers

Recurrence : express T(n) in terms of running time of recursive calls.

Base Case : T(1) <= a constant.

For all n > 1 : $$T(n) \leq 4T(n/2) + O(n)$$

Work done here

Work done by recursive calls

A Better Recursive Algorithm

<u>Algorithm #2 (Gauss)</u> : recursively compute ac (1), bd (2), (a+b)(c+d) (3)   [recall ad+bc = (3) – (1) – (2)]

<u>New Recurrence :</u>

<u>Base Case :</u> T(1) <= a constant

Ter
ver

Which recurrence best describes the running time of Gauss's algorithm for integer multiplication?

- $T(n) \leq 2T(n/2) + O(n^2)$
- $3T(n/2) + O(n)$
- $4T(n/2) + O(n)$
- $4T(n/2) + O(n^2)$

A Better Recursive Algorithm

Algorithm #2 (Gauss) : recursively compute ac,[(1)] bd,[(2)] (a+b)(c+d)[(3)] [recall ad+bc = (3) – (1) – (2)]

New Recurrence :

Base Case : T(1) <= a constant

For all n>1 : $T(n) \leq 3T(n/2) + O(n)$

Work done here

Work done by recursive calls

Design and Analysis
of Algorithms I

Master Method

The Precise Statement

The Master Method

Cool Feature : a "black box" for solving recurrences.

Assumption : all subproblems have equal size.

Recurrence Format

1. Base Case : T(n) <= a constant for all sufficiently small n
2. For all larger n :

$$T(n) \leq aT(n/b) + O(n^d)$$

where

a = number of recursive calls (>= 1)

b = input size shrinkage factor (> 1)

d = exponent in running time of "combine step" (>=0)

[a,b,d independent of n]

The Master Method

Base doesn't matter (only changes leading constants)

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Base matters

Design and Analysis of Algorithms I

Master Method

Examples

The Master Method

- If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

$$T(n) = \begin{cases} O\left(n^d \log n\right) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Example #1

<u>Merge Sort</u>

$$\left.\begin{array}{l} a = 2 \\ b = 2 \\ d = 1 \end{array}\right\} \quad b^d = a => Case\ 1$$

$$T(n) = O(n^d \log n) = O(n \log n)$$

Where are the respective values of a, b, d for a binary search of a sorted array, and which case of the Master Method does this correspond to?

- 1, 2, 0 [Case 1] $a = b^d \Rightarrow T(n) = O(n^d \log n) = O(\ \log n)$
- 1, 2, 1 [Case 2]
- 2, 2, 0 [Case 3]
- 2, 2, 1 [Case 1]

Example #3

Integer Multiplication Algorithm # 1

a = 4
b = 2
d = 1

$$b^d = 2 < a \quad (Case \;\; 3)$$

$$=> T(n) = O(n^{\log_b a}) = O(n^{\log_2 4})$$

$$= O(n^2)$$

Same as grade-school algorithm

Ter

ver

Where are the respective values of a, b, d for Gauss's recursive integer multiplication algorithm, and which case of the Master Method does this correspond to?

- 2, 2, 1 [Case 1]
- 3, 2, 1 [Case 1]
- 3, 2, 1 [Case 2]
- 3, 2, 1 [Case 3]

Better than the grade-school algorithm!!!

$$a = 3, \; b^d = 2 \; a > b^d \quad (Case\ 3)$$

$$=> T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Example #5

Strassen's Matrix Multiplication Algorithm

a = 7

b = 2

d = 2

$$b^d = 4 < a \quad (Case\ 3)$$

$$=> T(n) = O(n^{\log_2 7}) = O(n^{2.81})$$

=> beats the naïve iterative algorithm !

Tim Roughgarden

Example #6

Fictitious Recurrence

$$T(n) \leq 2T(n/2) + O(n^2)$$

$\Rightarrow$ a = 2

$\Rightarrow$ b = 2

$\Rightarrow$ d = 2

$$b^d = 4 > a \quad (Case\ 2)$$

$$=> T(n) = O(n^2)$$