



Design and Analysis  
of Algorithms I

# Master Method

## Motivation

# Integer Multiplication Revisited

Motivation : potentially useful algorithmic ideas often need mathematical analysis to evaluate

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

# A Recursive Algorithm

## Recursive approach

Write  $x = 10^{n/2}a + b$        $y = 10^{n/2}c + d$   
[where a,b,c,d are n/2 – digit numbers]

So :

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

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 :  $T(1) \leq$  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

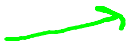
Algorithm #2 (Gauss): recursively compute  $ac^{(1)}$ ,  $bd^{(2)}$ ,  
 $(a+b)(c+d)^{(3)}$  [recall  $ad+bc = (3) - (1) - (2)$  ]

New Recurrence :

Base Case :  $T(1) \leq \text{a constant}$

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 :  $T(1) \leq \text{a constant}$

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 :  $T(n) \leq$  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 ( $\geq 1$ )

$b$  = input size shrinkage factor ( $> 1$ )

$d$  = exponent in running time of “combine step” ( $\geq 0$ )

[ $a, b, d$  independent of  $n$  ]

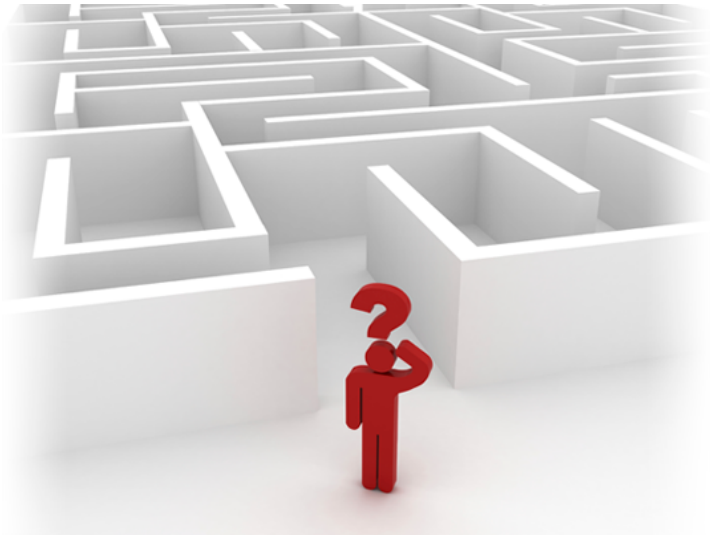
# The Master Method

- 

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

Base doesn't matter (only  
changes leading constants)

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

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

# Example #1

## Merge Sort

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

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

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]

☐ 1, 2, 1 [Case 2]

☐ 2, 2, 0 [Case 3]

☐ 2, 2, 1 [Case 1]

$$a = b^d \Rightarrow T(n) = O(n^d \log n) = O(\log n)$$

# Example #3

## Integer Multiplication Algorithm # 1

$$\left. \begin{array}{l} a = 4 \\ b = 2 \\ d = 1 \end{array} \right\} b^d = 2 < a \text{ (Case 3)}$$

$$\Rightarrow T(n) = O(n^{\log_b a}) = O(n^{\log_2 4})$$
$$= O(n^2)$$

Same as grade-school  
algorithm




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!!!

$$a = 3, \quad b^d = 2 \quad a > b^d \quad (\text{Case 3})$$
$$\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

# Example #5

## Strassen's Matrix Multiplication Algorithm

$$a = 7$$

$$b = 2$$

$$d = 2$$

$$\left. \begin{array}{l} b = 2 \\ d = 2 \end{array} \right\} b^d = 4 < a \quad (\text{Case 3})$$

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

$\Rightarrow$  beats the naïve iterative algorithm !

# Example #6

## Fictitious Recurrence

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

$$\Rightarrow a = 2$$

$$\Rightarrow b = 2$$

$$\Rightarrow d = 2$$

$$\left. \begin{array}{l} \Rightarrow a = 2 \\ \Rightarrow b = 2 \\ \Rightarrow d = 2 \end{array} \right\} b^d = 4 > a \quad (Case\ 2)$$

$$\Rightarrow T(n) = O(n^2)$$