The Master Method

COMP215: Design & Analysis of Algorithms

Today

- Integer Multiplication Revisited
- Formal Statement
- Examples

Integer Multiplication Revisited

$x * y = 2698 * 4263 = ?$

- The problem is to multiply two n-digit numbers
- The primitive operations are the addition or multiplication of two single-digit numbers.
- The iterative grade-school algorithm requires $\Theta(n^2)$ operations to multiply two n-digit numbers.

Integer Multiplication Revisited

x * y= 26 98 * 42 63

a b c d

- The ***RecIntMult*** Algorithm:

$$x \cdot y = 10^{n} \cdot (a \cdot c) + 10^{n/2} \cdot (a \cdot d + b \cdot c) + b \cdot d$$

- To describe this formally is by a **recurrence:**
  - Let T(n) denote the maximum number of operations used by this recursive algorithm to multiply two n-digit numbers
  - A recurrence expresses a running time bound T(n) in terms of the number of operations performed by recursive calls. The recurrence for the ***RecIntMult*** algorithm is

$$T(n) \leq \underbrace{4 \cdot T\left(\frac{n}{2}\right)}_{\text{work done by recursive calls}} + \underbrace{O(n)}_{\text{work done outside recursive calls}} .$$

Integer Multiplication Revisited

$$x * y = \underbrace{26}_{a}\,\underbrace{98}_{b} * \underbrace{42}_{c}\,\underbrace{63}_{d}$$

- ***Karatsuba's recursive*** algorithm for integer multiplicat

$$x * y = 10^{n} * (a * c) + 10^{n/2} * (a * d + b * c) + b * d$$

  - Instead of recursively computing **a*d or b*c**, we recursively compute the product of a + b and c + d.
  - Compute **a + b** and **c + d** using grade-school addition, and recursively compute **(a + b) * (c + d)**

Quiz 4.1

Which recurrence best describes the running time of the `Karatsuba` algorithm for integer multiplication?

a) $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + O(n^2)$

b) $3 \cdot T\left(\frac{n}{2}\right) + O(n)$

c) $3 \cdot T\left(\frac{n}{2}\right) + O(n^2)$

d) $4 \cdot T\left(\frac{n}{2}\right) + O(n)$

Formal Statement

Step 1

Standard Recurrence Format

Base case: $T(n)$ is at most a constant for all sufficiently small n.[4]

General case: for larger values of n,

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d).$$

Parameters:

- $a =$ number of recursive calls
- $b =$ input size shrinkage factor
- $d =$ exponent in running time of the "combine step"

Step 2

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3]}. \end{cases} \tag{4.2}$$

Examples:

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3].} \end{cases} \tag{4.2}$$

- MergeSort: O(n log n)
  - a? 2
  - b? 2
  - d? 1

MergeSort

Input: array A of n distinct integers.
Output: array with the same integers, sorted from smallest to largest.

```
// ignoring base cases
C := recursively sort first half of A
D := recursively sort second half of A
return Merge (C,D)
```

Examples:

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3].} \end{cases} \tag{4.2}$$

- Binary search

O(log n)

  - a? 1
  - b? 2
  - d? 0

Pseudocode: A Recursive Binary Search Algorithm.

```
procedure binary search(i, j, x: integers, 1≤i≤j≤n)
m := ⌊(i+j)/2⌋
if x=a_m then
    return m
else if (x<a_m and i < m) then
    return binary search(i, m − 1, x)
else if (x>a_m and j> m) then
    return binary search(m + 1, j, x)
else return 0
```

Examples:

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3].} \end{cases} \tag{4.2}$$

- RecIntMult

$O(n^{\log_2 4})$

$O(n^2)$

  - a?

4

  - b?

2

  - d?

1

Examples:

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3].} \end{cases} \tag{4.2}$$

- Karatsuba's recursive algorithm

$O(n^{\log_3 4})$

$O(n^{1.59})$

  - a? 3
  - b? 2
  - d? 1

Examples:

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \text{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \text{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{[Case 3].} \end{cases} \tag{4.2}$$

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 7T(n/3) + n^2$
- $T(n) = 9T(n/3) + n^2$
- $T(n) = 7T(n/4) + n$