CSE 548: Analysis of Algorithms

Lecture 2
(Divide-and-Conquer Algorithms: Integer Multiplication)

Rezaul A. Chowdhury

Department of Computer Science
SUNY Stony Brook

Spring 2014

A Latin Phrase

"Divide et impera"

(meaning: "divide and rule" or "divide and conquer")

— Philip II, king of Macedon (382-336 BC),
describing his policy toward the Greek city-states
(some say the Roman emperor Julius Caesar,
100-44 BC, is the source of this phrase)

The strategy is to break large power alliances into smaller ones that are easier to manage (or subdue).

This is a combination of political, military and economic strategy of gaining and maintaining power.

Unsurprisingly, this is also a very powerful problem solving strategy in computer science.

Divide-and-Conquer

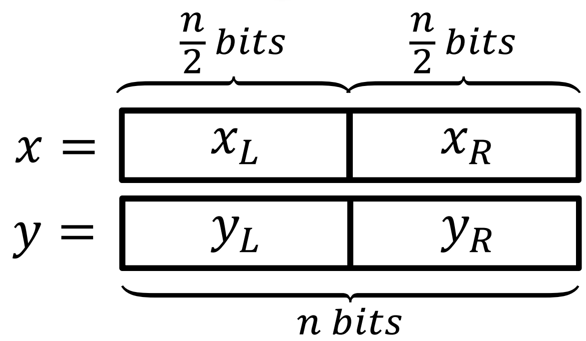
1. **Divide:** divide the original problem into smaller subproblems that are easier are to solve

2. **Conquer:** solve the smaller subproblems (perhaps recursively)

3. **Merge:** combine the solutions to the smaller subproblems to obtain a solution for the original problem

Integer Multiplication

Multiplying Two n-bit Numbers



$$x = 2^{n/2}x_L + x_R$$

$$y = 2^{n/2}y_L + y_R$$

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

So # $\frac{n}{2}$-bit products: 4

bit shifts (by n or $\frac{n}{2}$ bits): 2

additions (at most $2n$ bits long) : 3

We can compute the $\frac{n}{2}$-bit products recursively.

Let $T(n)$ be the overall running time for n-bit inputs. Then

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1, \\ 4T\left(\frac{n}{2}\right) + \mathrm{O}(n) & otherwise. \end{cases} = \mathrm{O}(n^2)$$ (how? derive)

Multiplying Two *n*-bit Numbers Faster (Karatsuba's Algorithm)

$\frac{n}{2}\, bits$ $\quad$ $\frac{n}{2}\, bits$

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R$$

$n\ bits$

$$xy = \left(2^{n/2}x_L + x_R\right)\left(2^{n/2}y_L + y_R\right)$$

$$= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

$$= 2^n x_L y_L + 2^{n/2}\left((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R\right) + x_R y_R$$

So # $\frac{n}{2}$- or $\left(\frac{n}{2} + 1\right)$ -bit products: 3

Then the overall running time for n-bit inputs:

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1, \\ 3T\left(\frac{n}{2}\right) + O(n) & otherwise. \end{cases}$$

$$= O\left(n^{\log_2 3}\right) = O(n^{1.59})$$ (how? derive)

Algorithms for Multiplying Two *n*-bit Numbers

Inventor	Year	Complexity
Classical	—	$\Theta(n^2)$
Anatolii Karatsuba	1960	$\Theta\left(n^{\log_2 3}\right)$
Andrei Toom & Stephen Cook (generalization of Karatsuba's algorithm)	1963 – 66	$\Theta\left(n 2^{\sqrt{2 \log_2 n}} \log n\right)$
Arnold Schönhage & Volker Strassen (Fast Fourier Transform)	1971	$\Theta(n \log n \log \log n)$
Martin Fürer (Fast Fourier Transform)	2005	$n \log n \, 2^{\mathrm{O}(\log^* n)}$

Lower bound: $\Omega(n)$ (why?)