CSE 548: Analysis of Algorithms

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

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2012
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.

**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)
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

\[
x = \begin{array}{c|c}
\frac{n}{2}\text{ bits} & \frac{n}{2}\text{ bits} \\
\hline
x_L & x_R \\
\end{array} = 2^{n/2}x_L + x_R
\]

\[
y = \begin{array}{c|c}
\frac{n}{2}\text{ bits} \\
\hline
y_L & y_R \\
\end{array} = 2^{n/2}y_L + y_R
\]

\[
xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R
\]

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

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
4T\left(\frac{n}{2}\right) + O(n) & \text{otherwise.}
\end{cases} = O(n^2) \ \ (\text{how? derive})
\]
Multiplying Two $n$-bit Numbers Faster
(Karatsuba’s Algorithm)

$$x = \begin{array}{c|c}
\frac{n}{2} \text{ bits} & \frac{n}{2} \text{ bits} \\
\hline
x_L & x_R \\
\end{array} = 2^{n/2}x_L + x_R$$

$$y = \begin{array}{c|c}
\frac{n}{2} \text{ bits} & \frac{n}{2} \text{ bits} \\
\hline
y_L & y_R \\
\end{array} = 2^{n/2}y_L + y_R$$

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

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

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

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

Then the overall running time for $n$-bit inputs:

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

$$= O(n^{\log_2 3}) = O(n^{1.59}) \text{ (how? derive)}$$
## Algorithms for Multiplying Two $n$-bit Numbers

<table>
<thead>
<tr>
<th>Inventor</th>
<th>Year</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>—</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Anatolii Karatsuba</td>
<td>1960</td>
<td>$\Theta(n^{\log_2 3})$</td>
</tr>
<tr>
<td>Andrei Toom &amp; Stephen Cook</td>
<td>1963 – 66</td>
<td>$\Theta(n2^{\sqrt{2\log_2 n}\log n})$</td>
</tr>
<tr>
<td>Arnold Schönhage &amp; Volker Strassen (Fast Fourier Transform)</td>
<td>1971</td>
<td>$\Theta(n \log n \log \log n)$</td>
</tr>
<tr>
<td>Martin Fürer (Fast Fourier Transform)</td>
<td>2005</td>
<td>$n \log n 2^{O(\log^* n)}$</td>
</tr>
</tbody>
</table>

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