Remember how to multiply two complex numbers $a + bi$ and $c + di$?

$$(a+bi)(c+di) = (ac - bd) + (ad + bc)i$$

Input: $a, b, c, d$
Output: $ac - bd, ad + bc$

If multiplying two real numbers costs $1$ and adding them costs a penny, what is the cheapest way to obtain the output from the input?

The above method costs $4.02$. 

$$ (a+bi)(c+di) = (ac - bd) + (ad + bc)i $$

Input: $a, b, c, d$
Output: $ac - bd, ad + bc$
Can we do better?
Take out a piece of paper and try…

Hint:
Try doing a+b and c+d first

Gauss’ $3.05$ Method

Input:  a,b,c,d
Output:  ac-bd, ad+bc

\[
\begin{align*}
X_1 &= a + b \\
X_2 &= c + d \\
X_3 &= X_1 X_2 = ac + ad + bc + bd \\
X_4 &= ac \\
X_5 &= bd \\
X_6 &= X_4 - X_5 = ac - bd \\
X_7 &= X_3 - X_4 - X_5 = bc + ad
\end{align*}
\]

The Gauss optimization saves one multiplication out of four.
It requires 25% less work.

Time complexity of grade school addition

\[
T(n) = \text{amount of time grade school addition uses to add two n-bit numbers}
\]

We saw that $T(n)$ was linear

$T(n) = \Theta(n)$. 
Time complexity of grade school multiplication

\[ T(n) = \text{The amount of time grade school multiplication uses to add two n-bit numbers} \]

We saw that \( T(n) \) was quadratic

\[ T(n) = \Theta(n^2) \]

Grade School Addition: Linear time
Grade School Multiplication: Quadratic time

No matter how dramatic the difference in the constants, the quadratic curve will eventually dominate the linear curve

Is there a sub-linear time method for addition?

... what would this mean?

Any addition algorithm takes \( \Omega(n) \) time

**Claim:** Any algorithm for addition must read all of the input bits

**Proof:** Suppose there is a mystery algorithm \( A \) that does not examine each bit

Give \( A \) a pair of numbers. There must be some unexamined bit position \( i \) in one of the numbers
Any addition algorithm takes $\Omega(n)$ time

A did not read this bit at position $i$

If $A$ is not correct on the inputs, we found a bug.

If $A$ is correct, flip the bit at position $i$ and give $A$ the new pair of numbers. $A$ gives the same answer as before, which is now wrong.

Grade school addition can't be improved upon by more than a constant factor

Grade School Addition: $\Theta(n)$ time. Furthermore, it is optimal.

Grade School Multiplication: $\Theta(n^2)$ time

Is there a clever algorithm to multiply two numbers in linear time?

Despite years of research, no one knows! If you resolve this question, please let Matt know immediately.

Can we even break the quadratic time barrier?

In other words, can we do something very different than grade school multiplication?
Why is he making us learn this?

Good question!

WHERE'S THE BEEF?

One thing that makes algorithm design “Computer Science” is that solving a problem in the most obvious way from its definitions is often not the best way to get a solution.

... multiplication is a simple example of this.

Divide And Conquer

An approach to faster algorithms:

DIVIDE a problem into smaller subproblems

CONQUER them recursively

GLUE the answers together so as to obtain the answer to the larger problem

Multiplication of 2 n-bit numbers

\[
X = a \cdot 2^{n/2} + b \quad Y = c \cdot 2^{n/2} + d
\]

\[
X \times Y = ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd
\]
Same thing for numbers in decimal!

$$X = a \times 10^{n/2} + b \quad Y = c \times 10^{n/2} + d$$

$$X \times Y = ac \times 10^n + (ad + bc) \times 10^{n/2} + bd$$

Multiplying (Divide & Conquer style)

12345678 * 21394276
1234*2139 1234*4276 5678*2139 5678*4276
12*21 12*39 34*21 34*39
1*2 1*1 2*2 2*1
2 1 4 2

Hence: 12*21 = 2*10^2 + (1 + 4)10^1 + 2 = 252

$$X = a \quad b$$
$$Y = c \quad d$$

$$X \times Y = ac \times 10^n + (ad + bc) \times 10^{n/2} + bd$$

Multiplying (Divide & Conquer style)

12345678 * 21394276
1234*2139 1234*4276 5678*2139 5678*4276
252 468 714 1326
*10^4 + *10^2 + *10^2 + *1 = 2639526

$$X = a \quad b$$
$$Y = c \quad d$$

$$X \times Y = ac \times 10^n + (ad + bc) \times 10^{n/2} + bd$$

Multiplying (Divide & Conquer style)

12345678 * 21394276
1234*2139 1234*4276 5678*2139 5678*4276
2639526 5276584 12145242 24279128
*10^8 + *10^4 + *10^4 + *1

= 264126842539128

$$X = a \quad b$$
$$Y = c \quad d$$

$$X \times Y = ac \times 10^n + (ad + bc) \times 10^{n/2} + bd$$
Divide, Conquer, and Glue

MULT(X,Y):

if \(|X| = |Y| = 1\)
then return XY,
else...

MULT(X,Y):

X=a;b  Y=c;d

MULT(a,c)
MULT(a,d)
MULT(b,c)
MULT(b,d)
**Divide, Conquer, and Glue**

**MULT(X,Y):**

\[
X=a; \quad b \quad Y=c; \quad d
\]

- \(ac\)
- \(b; \quad c\)
- \(d\)
- \(b; \quad d\)

**Divide, Conquer, and Glue**

**MULT(X,Y):**

\[
X=a; \quad b \quad Y=c; \quad d
\]

- \(ac\)
- \(ad\)
- \(b; \quad c\)
- \(b; \quad d\)

**Divide, Conquer, and Glue**

**MULT(X,Y):**

\[
X=a; \quad b \quad Y=c; \quad d
\]

- \(ac\)
- \(ad\)
- \(b; \quad c\)
- \(b; \quad d\)

**Divide, Conquer, and Glue**

**MULT(X,Y):**

\[
X=a; \quad b \quad Y=c; \quad d
\]

- \(ac\)
- \(b; \quad c\)
- \(b; \quad d\)
- \(Mult(b, c)\)
Divide, Conquer, and Glue

MULT(X,Y):

\[
XY = \frac{ac2^n + (ad+bc)2^{n/2} + bd}{2}
\]

Time required by MULT

\[
T(n) = \text{time taken by MULT on two } n\text{-bit numbers}
\]

What is \( T(n) \)? What is its growth rate?

Big Question: Is it \( \Theta(n^2) \)?

\[
T(n) = 4T(n/2) + (k'n + k'')
\]
Recurrence Relation

\[ T(1) = k \quad \text{for some constant } k \]
\[ T(n) = 4 \ T(n/2) + k' n + k'' \quad \text{for constants } k' \text{ and } k'' \]

\textbf{MULT(X,Y):}

If \(|X| = |Y| = 1\) then return \(XY\)
else break \(X\) into \(a;b\) and \(Y\) into \(c;d\)
return \(MULT(a,c) \ 2^n + (MULT(a,d) + MULT(b,c)) \ 2^{n/2} + MULT(b,d)\)

\[ T(1) = 1 \]
\[ T(n) = 4 \ T(n/2) + n \]

\textbf{MULT(X,Y):}

If \(|X| = |Y| = 1\) then return \(XY\)
else break \(X\) into \(a;b\) and \(Y\) into \(c;d\)
return \(MULT(a,c) \ 2^n + (MULT(a,d) + MULT(b,c)) \ 2^{n/2} + MULT(b,d)\)

\textbf{Technique: Labeled Tree Representation}

\[ T(n) = n + 4 \ T(n/2) \]

\[ T(1) = 1 \]

\[ T(n) = 1 \]
\[ T(n) = n \]

Level \( i \) is the sum of \( 4^i \) copies of \( \frac{n}{2^i} \)

<table>
<thead>
<tr>
<th>( \log_2(n) )</th>
<th>( T(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td>( \frac{n}{2} + \frac{n}{2} + \frac{n}{2} + \frac{n}{2} )</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{n}{4} + \frac{n}{4} + \frac{n}{4} + \frac{n}{4} )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>
\[ n = \begin{array}{c|cccc} 1n & \vdots & \vdots & \vdots & \vdots \\ \hline 2n & n/2 & n/2 & n/2 & n/2 \\ \hline 4n & \vdots & \vdots & \vdots & \vdots \\ \hline 2n & \text{Level } i \text{ is the sum of } 4^i \text{ copies of } n/2^i \\ \hline (n)n & \vdots & \vdots & \vdots & \vdots \\ \hline \end{array} \]

\[ n(1+2+4+8+ \ldots + n) = n(2n-1) = 2n^2-n \]

\[ 2n = \begin{array}{c|cccc} 1n & \vdots & \vdots & \vdots & \vdots \\ \hline 2n & n/2 & n/2 & n/2 & n/2 \\ \hline 4n & \vdots & \vdots & \vdots & \vdots \\ \hline 2n & \text{Level } i \text{ is the sum of } 4^i \text{ copies of } n/2^i \\ \hline (n)n & \vdots & \vdots & \vdots & \vdots \\ \hline \end{array} \]

**Divide and Conquer MULT:** $\Theta(n^2)$ time

**Grade School Multiplication:** $\Theta(n^2)$ time

---

**MULT revisited**

**MULT** \((X, Y)\):

- If \(|X| = |Y| = 1\) then return \(XY\)
- else break \(X\) into \(a; b\) and \(Y\) into \(c; d\)
- return \(\text{MULT}(a,c) \cdot 2^n + (\text{MULT}(a,d) + \text{MULT}(b,c)) \cdot 2^{n/2} + \text{MULT}(b,d)\)

**MULT** calls itself 4 times. Can you see a way
to reduce the number of calls?

---

**Gauss’ optimization**

**Input:** \(a, b, c, d\)

**Output:** \(ac-bd, ad+bc\)

\[ \begin{align*}
&c \quad X_1 = a + b \\
&c \quad X_2 = c + d \\
&c \quad X_3 - X_5 = X_1 X_2 = ac + ad + bc + bd \\
&c \quad X_4 = ac \\
&c \quad X_6 = bd \\
&c \quad X_7 - X_5 = X_3 - X_5 = bc + ad \\
\end{align*} \]
Karatsuba, Anatolii Alexeevich (1937-)

Sometime in the late 1950's Karatsuba had formulated the first algorithm to break the n^2 barrier!

Gaussified MULT (Karatsuba 1962)

MULT(X,Y):

If |X| = |Y| = 1 then return XY
else break X into a;b and Y into c;d
    e := MULT(a,c)
    f := MULT(b,d)
return e 2^n + (MULT(a+b,c+d) − e − f) 2^{n/2} + f

T(n) = 3 T(n/2) + n

Actually: T(n) = 2 T(n/2) + T(n/2 + 1) + kn
Dramatic Improvement for Large $n$

$$T(n) = 3n^{\log_2 3} - 2n$$
$$= \Theta(n^{\log_2 3})$$
$$= \Theta(n^{1.58\ldots})$$

A huge savings over $\Theta(n^2)$ when $n$ gets large.
### Multiplication Algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kindergarten</td>
<td>$n^{2^n}$</td>
</tr>
<tr>
<td>Grade School</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>$n^{1.58...}$</td>
</tr>
<tr>
<td>Fastest Known</td>
<td>$n \log(n) \log\log(n)$</td>
</tr>
</tbody>
</table>

The kind of analysis we have been doing is only meaningful for very large numbers.

On a computer, if you are multiplying numbers that fit into the word size, you would do this in hardware that has gates working in parallel.

Here’s What You Need to Know…

- Gauss’s Multiplication Trick
- Proof of Lower bound for addition
- Divide and Conquer
- Solving Recurrences
- Karatsuba Multiplication