Gauss' Complex Puzzle

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?

Can you do better than $4.02?

Gauss' $3.05 Method

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

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

The Gauss optimization saves one multiplication out of four. It requires 25% less work.
**Time complexity of grade school addition**

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

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

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

**Time complexity of grade school multiplication**

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

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

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

---

**Grade School Addition: Linear time**

**Grade School Multiplication: Quadratic time**

---

**Is there a sub-linear time method for addition?**

---

**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

---

<table>
<thead>
<tr>
<th>A did not read this bit at position i</th>
</tr>
</thead>
</table>

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, Carnegie Mellon will give you a PhD!

Can we even break the quadratic time barrier?

In other words, can we do something very different than grade school multiplication?

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

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

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

MULT($X,Y$):

If $|X| = |Y| = 1$ then return $XY$
else break $X$ into $a;b$ and $Y$ into $c;d$
return $\text{MULT}(a,c) 2^n + (\text{MULT}(a,d) + \text{MULT}(b,c)) 2^{n/2} + \text{MULT}(b,d)$
Same thing for numbers in decimal!

\[
\begin{align*}
X &= a \cdot 10^{n/2} + b \\
Y &= c \cdot 10^{n/2} + d \\
X \times Y &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd
\end{align*}
\]

Multiplying (Divide & Conquer style)

\[
\begin{align*}
12345678 \times 21394276 \\
1234 \times 2139 & 1234 \times 4276 & 5678 \times 2139 & 5678 \times 4276 \\
& 252 & 468 & 714 & 1326 \\
& \times 10^4 & + & \times 10^4 & + & \times 10^4 & + & \times 1 \\
= 2639526
\end{align*}
\]

Divide, Conquer, and Glue

\[
\text{MULT}(X,Y)
\]
if |X| = |Y| = 1 then return XY, else...

Divide, Conquer, and Glue

\[ \text{MULT}(X, Y): \]

\[ X = \begin{pmatrix} a & b \end{pmatrix}, Y = \begin{pmatrix} c & d \end{pmatrix} \]

Divide, Conquer, and Glue

\[ \text{MULT}(X, Y): \]

\[ \text{Mult}(a, c) \]
\[ \text{Mult}(a, d) \]
\[ \text{Mult}(b, c) \]
\[ \text{Mult}(b, d) \]
Divide, Conquer, and Glue

\[ \text{MULT}(X, Y): \quad X = a \; b \quad Y = c \; d \]

\[ \begin{align*}
    & ac \\
    & ad \\
    & \text{Mult}(b, c) \\
\end{align*} \]

\[ \begin{align*}
    & ac \\
    & ad \\
    & bc \\
    & \text{Mult}(b, d) \\
\end{align*} \]

Divide, Conquer, and Glue

\[ \text{MULT}(X, Y): \quad X = a \; b \quad Y = c \; d \]

\[ \begin{align*}
    & ac \\
    & ad \\
    & bc \\
    & \text{Mult}(b, c) \\
\end{align*} \]

\[ \begin{align*}
    & ac \\
    & ad \\
    & bc \\
    & \text{Mult}(b, d) \\
\end{align*} \]

Time required by \text{MULT}

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

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

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

T(n) = 4 T(n/2) + (k'n + k'')

Recurrence Relation

T(1) = k for some constant k

T(n) = 4 T(n/2) + k'n + k'' for constants k' and k''
Simplified Recurrence Relation

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

conquering
divide and glue

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

Level \( i \) is the sum of \( 4^i \) copies of \( n/2^i \)

\[ n(1 + 2 + 4 + 8 + \ldots + n) = n(2n-1) = 2n^2 - n \]
Divide and Conquer MULT: $\Theta(n^2)$ time
Grade School Multiplication: $\Theta(n^2)$ time

Bummer!

MULT revisited
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)) $2^{n/2} + MULT(b,d)$

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

c $X_1 = a + b$
c $X_2 = c + d$
$X_3 = X_1 X_2 = ac + ad + bc + bd$
$X_4 = ac$
$X_5 = bd$
c $X_6 = X_4 - X_5 = ac - bd$
cc $X_7 = X_3 - X_4 - X_5 = bc + ad$

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: $T(n) = 2 T(n/2) + T(n/2 + 1) + kn$
$T(n) = \frac{n}{2} + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$

$T(n) = \frac{n}{2} + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right)$

Level $i$ is the sum of $3^i$ copies of $n/2^i$

$n = 3n^{\log_2 3} - 2n$

$= O(n^{\log_2 3})$

$= O(n^{1.58...})$

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

<table>
<thead>
<tr>
<th>Kindergarten</th>
<th>$n^2$</th>
</tr>
</thead>
<tbody>
<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>

A case study

Anagram Programming Task.

You are given a 70,000 word dictionary. Write an anagram utility that given a word as input returns all anagrams of that word appearing in the dictionary.

Examples

Input: CAT
Output: ACT, CAT, TAC

Input: SUBESSENTIAL
Output: SUITABLENESS

(Novice Level Solution)

Loop through all possible ways of rearranging the input word

Use binary search to look up resulting word in dictionary.

If found, output it
Performance Analysis
Counting without executing

On the word “microphotographic”,
we loop $17! \approx 3 \times 10^{14}$ times.

Even at 1 microsecond per iteration,
this will take $3 \times 10^8$ seconds.

Almost a decade!
(There are about $\pi$ seconds in a nanocentury.)

“Expert” Level Solution

Module ANAGRAM(X,Y) returns true exactly when X and Y are anagrams.
(Works by sorting the letters in X and Y)

Input: X
Loop through all dictionary words Y
If ANAGRAM(X,Y) output Y

The hacker is satisfied and reflects no further

Comparing an input word with each of 70,000 dictionary entries takes about 15 seconds

The master keeps trying to refine the solution

The master’s program runs in less than 1/1000 seconds.

Master Solution

Don’t just keep the dictionary in sorted order!

Rearranging the dictionary into “anagram classes” makes the original problem simpler.

Suppose the dictionary was the list below.

ASP
DOG
LURE
GOD
NICE
RULE
SPA
After each word, write its “signature” (sort its letters)

<table>
<thead>
<tr>
<th>Word</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>APS</td>
</tr>
<tr>
<td>DOG</td>
<td>DGO</td>
</tr>
<tr>
<td>LURE</td>
<td>ELRU</td>
</tr>
<tr>
<td>GOD</td>
<td>DGO</td>
</tr>
<tr>
<td>NICE</td>
<td>CEIN</td>
</tr>
<tr>
<td>RULE</td>
<td>ELRU</td>
</tr>
<tr>
<td>SPA</td>
<td>APS</td>
</tr>
</tbody>
</table>

Sort by the signatures

<table>
<thead>
<tr>
<th>Word</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>APS</td>
</tr>
<tr>
<td>SPA</td>
<td>APS</td>
</tr>
<tr>
<td>NICE</td>
<td>CEIN</td>
</tr>
<tr>
<td>DOG</td>
<td>DGO</td>
</tr>
<tr>
<td>GOD</td>
<td>DGO</td>
</tr>
<tr>
<td>LURE</td>
<td>ELRU</td>
</tr>
<tr>
<td>RULE</td>
<td>ELRU</td>
</tr>
</tbody>
</table>

The Master’s Program

Input word W
X := signature of W (sort the letters)

Use binary search to find the anagram class of W and output it.

A useful tool: preprocessing...

Of course, it takes about 30 seconds to create the dictionary, but it is perfectly fair to think of this as programming time. The building of the dictionary is a one-time cost that is part of writing the program.

Here’s What You Need to Know...

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