The changing $\times$: Multiplication algorithms, new and old

Ricky Liu

University of Washington Math Hour

May 22, 2022
Addition is easy.

\[
\begin{array}{ccccccc}
1 & 1 & 1 & 1 \\
2 & 3 & 8 & 4 & 7 & 6 & 2 & 4 & 5 & 1 \\
+ & 8 & 4 & 2 & 1 & 7 & 9 & 3 & 6 & 2 & 7 \\
\hline
1 & 0 & 8 & 0 & 6 & 5 & 5 & 6 & 0 & 7 & 8
\end{array}
\]

Multiplication is hard.

\[
\begin{array}{ccccccc}
2 & 3 & 8 & 4 & 7 & 6 & 2 & 4 & 5 & 1 \\
\times & 8 & 4 & 2 & 1 & 7 & 9 & 3 & 6 & 2 & 7 \\
\hline
1 & 6 & 6 & 9 & 3 & 3 & 3 & 7 & 1 & 5 & 7 \\
4 & 7 & 6 & 9 & 5 & 2 & 4 & 9 & 0 & 2 \\
1 & 4 & 3 & 0 & 8 & 5 & 7 & 4 & 7 & 0 & 6 \\
7 & 1 & 5 & 4 & 2 & 8 & 7 & 3 & 5 & 3 \\
2 & 1 & 4 & 6 & 2 & 8 & 6 & 2 & 0 & 5 & 9 \\
1 & 6 & 6 & 9 & 3 & 3 & 3 & 7 & 1 & 5 & 7 \\
2 & 3 & 8 & 4 & 7 & 6 & 2 & 4 & 5 & 1 \\
4 & 7 & 6 & 9 & 5 & 2 & 4 & 9 & 0 & 2 \\
9 & 5 & 3 & 9 & 0 & 4 & 9 & 8 & 0 & 4 \\
1 & 9 & 0 & 7 & 8 & 0 & 9 & 9 & 6 & 0 & 8 \\
\hline
2 & 0 & 0 & 8 & 3 & 9 & 7 & 7 & 2 & 1 & 1 & 7 & 4 & 0 & 6 & 9 & 9 & 7 & 7 & 7
\end{array}
\]

**Question**

What is the fastest way to multiply?
The Standard Algorithm

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\times & 4 & 5 & 6 \\
\hline
 & 7 & 3 & 8 \\
 & 6 & 1 & 5 \\
4 & 9 & 2 \\
\hline
5 & 6 & 0 & 8 & 8 \\
\end{array}
\]

Requires:
- multiplying every digit in the first number by every digit in the second number;
- knowledge of a $10 \times 10$ multiplication table.
The lattice or grid method:

\[
\begin{array}{ccc}
0 & 4 & 8 \\
0 & 1 & 0 \\
0 & 1 & 2 \\
6 & 2 & 8 \\
\end{array}
\]

\[
123 \times 456 = 56088
\]

The underlying process is the same as the standard algorithm (the same multiplications and additions are done but in a slightly different order).
The lattice method was used in various historical computing devices.

Napier’s bones (1617)  Genaille-Lucas rulers (1891)
What if you don’t have a multiplication table memorized? Enter **Russian peasant multiplication**, based on doubling and halving.

\[
\begin{align*}
41 & \times 23 \\
20 & \times 46 \\
10 & \times 92 \\
5 & \times 184 \\
2 & \times 368 \\
1 & \times 736 \\
\end{align*}
\]

\[943\]

Requires:
- Knowledge of addition and halving;
- More steps than the standard algorithm, but the steps are simpler.
A similar method involving only doubling was used by the ancient Egyptians.

\[ 41 \times 23 = (32 + 8 + 1) \times 23 \]

\[
\begin{array}{cc}
1 & 23 \\
2 & 46 \\
4 & 92 \\
8 & 184 \\
16 & 368 \\
32 & 736 \\
\hline
41 & 943 \\
\end{array}
\]
Actually, this is essentially the standard algorithm in binary!

\[
\begin{array}{ccccccc}
1 & 0 & 1 & 1 & 1 & \times & 23 \\
\times & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 & 1 & \times & 41 \\
1 & 0 & 1 & 1 & 1 & & 23 \\
1 & 0 & 1 & 1 & 1 & & 184 \\
1 & 0 & 1 & 1 & 1 & & 736 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & & 943 \\
\end{array}
\]
Table-based methods

Instead of a multiplication table, some methods used other tables.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>6</th>
<th>9</th>
<th>11</th>
<th>30</th>
<th>16</th>
<th>64</th>
<th>156</th>
<th>6084</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>12</td>
<td>12</td>
<td>36</td>
<td>17</td>
<td>72</td>
<td>157</td>
<td>6162</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>16</td>
<td>13</td>
<td>42</td>
<td>18</td>
<td>81</td>
<td>158</td>
<td>6241</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>9</td>
<td>20</td>
<td>14</td>
<td>49</td>
<td>19</td>
<td>90</td>
<td>159</td>
<td>6320</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>10</td>
<td>25</td>
<td>15</td>
<td>56</td>
<td>20</td>
<td>100</td>
<td>160</td>
<td>6400</td>
</tr>
</tbody>
</table>

To multiply $83 \times 74$:

\[
\begin{align*}
83 + 74 &= 157 & \rightarrow & & 6162 \\
83 - 74 &= 9 & \rightarrow & & 20 \\
\end{align*}
\]

\[
6142
\]
This is the Babylonian quarter-square method. It uses the identity

\[
\frac{(x + y)^2}{4} - \frac{(x - y)^2}{4} = xy.
\]

To multiply numbers up to \( n \), you need \( 2n \) quarter-squares (as opposed to \( n^2 \) entries in a multiplication table).

\[\text{Table of Quarter-Squares}
\]
A different table-based method was introduced by John Napier in 1614.

It was turned into a computing device by William Oughtred in 1622.
A simple slide rule:

What number should go here?

In general, the number located $d$ units from the left is $2^d$. 

$$\sqrt{32} = 2^{2.5} \approx 5.657$$
If $x = 2^d$, then $d = \log_2 x$ is the (base-2) logarithm of $x$.

With a table of logarithms, you can do multiplication with just addition and a few lookups.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$\log_{10} x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>.00000</td>
</tr>
<tr>
<td>1.1</td>
<td>.04139</td>
</tr>
<tr>
<td>1.2</td>
<td>.07918</td>
</tr>
<tr>
<td>1.3</td>
<td>.11394</td>
</tr>
<tr>
<td>1.4</td>
<td>.14613</td>
</tr>
<tr>
<td>1.5</td>
<td>.17609</td>
</tr>
<tr>
<td>1.6</td>
<td>.20412</td>
</tr>
<tr>
<td>1.7</td>
<td>.23045</td>
</tr>
<tr>
<td>1.8</td>
<td>.25527</td>
</tr>
<tr>
<td>1.9</td>
<td>.27875</td>
</tr>
<tr>
<td>2.0</td>
<td>.30103</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
13 &= 1.3 \times 10^1 \rightarrow 1.11394 \\
\times 14 &= 1.4 \times 10^1 \rightarrow + 1.14613 \\
\approx 180 &= 1.8 \times 10^2 \leftarrow 2.26007
\end{align*}
\]

It can be a bit imprecise...
Question
What is the fastest way to multiply?

Question
How do we judge the speed of an algorithm?

A: Count the number of operations required with \( n \) digit numbers as inputs.
For example, adding two \( n \) digit numbers requires \( n \) one-digit additions and potentially \( n \) carries, for a total of \( 2n \) operations.
What about multiplication?

For the standard method:

- $n^2$ one-digit multiplications,
- $\approx n^2$ 2-digit additions (equivalent to $\approx 2n^2$ one-digit additions)

for a total of about $3n^2$ operations.

For peasant multiplication:

- There are $\approx 3.3n$ rows.
- For each row, we may have to do an $\approx n$-digit halving, doubling, and addition ($\approx 3n$ operations),

for a total of about $10n^2$ operations.

We say that both algorithms run in $O(n^2)$ operations.
$O$ is ‘Big O’ notation meaning roughly, “on the order of” or “up to a constant factor.” Thus $O(n)$ could mean $2n$ or $999n + 7$.

Why don’t we care about constant factors?

For really really big $n$, the constant is not important: any $O(n)$ algorithm will be faster than any $O(n^2)$ algorithm for $n \gg 0$, even if it is slower for small $n$ due to more “overhead.”

For instance, $999n < n^2$ when $n > 999$. 
In 1960, Russian mathematician Andrey Kolmogorov made the following conjecture at a conference.

**Conjecture**

*Any algorithm to multiply two \( n \)-digit numbers requires at least \( O(n^2) \) steps.*
This conjecture intrigued 23-year-old student Anatoly Karatsuba. Within a week, Karatsuba had disproved the conjecture by finding a way to multiply two \( n \)-digit numbers using \( O(n^{1.58}) \) operations! Kolmogorov was so pleased by the result that he wrote it up and had it published on Karatsuba’s behalf.
Consider multiplying two-digit numbers using the standard method (before performing carries).

\[
\begin{array}{c}
\times \\
\hline
a & b \\
\hline
a \times c & b \times c \\
\hline
a \times c & (a \times d) + (b \times c) & b \times d
\end{array}
\]
To multiply $ab \times cd$, we need to find:

- $X = a \times c$
- $Y = b \times d$
- $Z = (a \times d) + (b \times c)$

It seems like we need to do 4 multiplications.

But there is another way: note that

\[(a + b) \times (c + d) = (a \times c) + (a \times d) + (b \times c) + (b \times d)\]
\[= X + Z + Y\]

Thus

\[Z = (a + b) \times (c + d) - X - Y.\]

Then we can find $X$, $Y$, and $Z$ using only 3 multiplications instead of 4 at the expense of more additions/subtractions.
\[
\begin{array}{cc}
5 & 3 \\
\times & 2 & 7 \\
\hline
2 & 1 \\
4 & 1 \\
1 & 0 \\
\hline
1 & 4 & 3 & 1 \\
\end{array}
\]

\[
X = 5 \times 2 = 10 \\
Y = 3 \times 7 = 21 \\
Z = (5 + 3) \times (2 + 7) - X - Y \\
= 8 \times 9 - 10 - 21 = 41 
\]
We traded a multiplication for a bunch of additions. Is this really faster?

Not for two-digit numbers...

But we can also use this idea for numbers with more digits!

\[
\begin{array}{cc}
3825 & 4926 \\
\times & 2937 \quad 6328 \\
\hline
3117 & 1728 \\
\end{array}
\]

(standard) \[ Z = 3825 \times 6328 + 4926 \times 2937 \]

(Karatsuba) \[ Z = (3825 + 4926) \times (2937 + 6328) - 31171728 - 11234025 \]

We’re replacing a hard multiplication with easy additions/subtractions, which are much faster!
We can **divide and conquer** to get more savings by using Karatsuba’s algorithm for the smaller multiplications.

To multiply two 16-digit numbers, Karatsuba would do:

- 1 16-digit multiplication $\rightarrow$ 3 8-digit multiplications
- 3 8-digit multiplications $\rightarrow$ 9 4-digit multiplications
- 9 4-digit multiplications $\rightarrow$ 27 2-digit multiplications
- 27 2-digit multiplications $\rightarrow$ 81 1-digit multiplications

Compare with $16^2 = 256$ 1-digit multiplications for the standard algorithm.

For 1000-digit numbers, the standard algorithm needs $1000000$ 1-digit multiplications while Karatsuba needs only $60000$.

In general, Karatsuba’s algorithm uses only

$$O(n^{\log_2 3}) \approx O(n^{1.58})$$

operations.
Can we do better?

- Karatsuba (1960) $O(n^{1.58})$
  - Can multiply 2-digit numbers with 3 multiplications instead of 4
- Toom-Cook (1963) $O(n^{1.46})$
  - Can multiply 3-digit numbers with 5 multiplications instead of 9
  - Can make $1.46$ close to 1 with more pieces but a lot of overhead
- Schönhage and Strassen (1971) $O(n \cdot \log n \cdot \log \log n)$
  - Based on Fast Fourier Transform
  - Faster for numbers $> 10000$ digits
- Fürer (2007) $O(n \cdot \log n \cdot 2^{O(\log^* n)})$
  - Slower for practical applications due to large overhead
- Harvey and van der Hoeven (2021) $O(n \cdot \log n)$

Open question

Is there an algorithm for multiplying $n$-digit numbers that is faster than $O(n \cdot \log n)$?