CIS 575. Introduction to Algorithm Analysis
Material for February 26, 2024

Multiplying Large Integers

©2020 Torben Amtoft

1 Multiplying Large Integers

We shall consider the task

> Multiply two n-bit positive integers

One may ask: given that all computers provide multiplication at machine code level, why do we bother about this? But recall that a given computer model uses only a certain number of bits to represent each integer; if $2n$ is bigger than that number then we cannot directly use the multiplication operation, but may have to implement our own. For that purpose, the *divide & conquer* paradigm can be used, as we shall now show. To make the examples more appealing, we shall use base 10 rather than base 2 to represent integers, and thus consider *n-digit* integers.

First recall (from primary school!) that if we know how to multiply 1-digit integers then we can also multiply 2-digit integers: to multiply say **47** and **23**, we perform 4 multiplications of 1-digit integers, suitably align the results, and add them:

```
   7 x 3      21
  4 x 3      12
  7 x 2      14
 4 x 2       8
            ----
            1081
```

Similarly, if we know how to multiply 2-digit integers then we can also multiply 4-digit integers, as illustrated by the following computation of **2043** times **2512**:

```
     43 x 12     516
   20 x 12     240
   43 x 25   1075
 20 x 25    500
            -------
            5132016
```

It should be obvious how to generalize this so as to write an algorithm that multiplies two n-digit integers by 4 multiplications of $\frac{n}{2}$-digit integers. (By adding leading zeros we can assume that the number of digits in an integer is a power of 2.) Since addition can be done in time $\Theta(n)$, for the running time $T(n)$ we get the recurrence

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

and thus (as $\log_2(4) = 2$) we have $T(n) \in \Theta(n^2)$.

But we can do better: in the second example, multiplying **2043** and **2512**, we can do a little conjuring trick:

```
    43 x 12                    516
  (20+43) x (25+12)         2331
                            -516        // the top line
                            -500        // the bottom line
 20 x 25                   500
                           -------
                           5132016
```

While we have increased the number of additions, and thus likely also increased the running time for integers with only few digits, there are now only **3** multiplications[1] and this will improve *asymptotic* running time: we get the recurrence

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) \tag{1}$$

which has solution $T(n) \in \Theta(n^{\lg(3)})$.

To understand why our magic works, observe that: if we have two integers P and Q where

$$\begin{aligned} P &= w \cdot 10^n + y \\ Q &= x \cdot 10^n + z \end{aligned}$$

then

$$P \cdot Q = wx \cdot 10^{2n} + (wz + yx) \cdot 10^n + yz$$

but

$$wz + yx = (w + y)(x + z) - wx - yz$$

and thus the only multiplications needed are: $\mathbf{wx}$, $\mathbf{yz}$, and $(\mathbf{w} + \mathbf{y})(\mathbf{x} + \mathbf{z})$.

It is actually possible to get an even better asymptotic running time (at the price of running slower for "small" numbers), as demonstrated in Exercise 10.4 in the *Howell* textbook (where the context is the multiplication of polynomials). For any $k \geq 2$, it is possible to get the the recurrence (when $k = 2$ this is just (1))

$$T(n) = (2k - 1)T(\frac{n}{k}) + \Theta(n)$$

which has solution $T(n) \in \Theta(n^{\log_k(2k-1)})$ where the exponent $\log_k(2k - 1)$ will approach 1 as k approaches infinity.

We can thus get as close to linear time as we want. But we cannot do better than linear time: any algorithm has to generate at least n digits, and hence must have running time in $\Omega(n)$.

[1] One of which could have involved 3-digit numbers, if we had chosen say 6043 rather than 2043.