Algorithms – CMSC-272XX
Divide and Conquer: *The Karatsuba algorithm*
(multiplication of large integers)

Instructor: László Babai

Updated 01-13-2021

NOTATION. In this note, log will always mean $\log_2$ (base-2 logarithm).

The Karatsuba algorithm provides a striking example of how the "Divide and Conquer" technique can achieve an asymptotic speedup over an ancient algorithm.

The classroom method of multiplying two $n$-digit integers requires $\Theta(n^2)$ digit operations. We shall show that a simple recursive algorithm solves the problem in $O(n^{\log 3})$ digit operations. (Note: $\log 3 \approx 1.58$.) This represents considerable savings in the asymptotic rate of growth of the number of digit-operations.

We describe the procedure in *pseudocode.* We assume that the number of digits is a power of 2 so we will not need to worry about rounding. This assumption is justified by padding the number with initial zeros; this will increase the vaule of $n$ by less than a factor of 2, immaterial for our estimates.

*Procedure* Karatsuba$(X, Y)$

*Input:* $X, Y$: $n$-digit integers.
*Output:* the product $P := XY$.
*Comment:* We assume $n$ is a power of 2.

1. **if** $n = 1$ **then** use multiplication table to find $P := XY$
2. **else** split $X, Y$ in half:
3. $\quad X =: 10^{n/2}X_1 + X_2$
4. $\quad Y =: 10^{n/2}Y_1 + Y_2$
5. $\quad$ *Comment:* $X_1, X_2, Y_1, Y_2$ each have $n/2$ digits.
6. $\quad U :=$ Karatsuba$(X_1, Y_1)$
7. $\quad V :=$ Karatsuba$(X_2, Y_2)$

8. $W := \text{Karatsuba}(X_1 - X_2, Y_1 - Y_2)$

9. $Z := U + V - W$

10. $P := 10^n U + 10^{n/2} Z + V$

11. *Comment:* So $U = X_1 Y_1$, $V = X_2 Y_2$, $W = (X_1 - X_2)(Y_1 - Y_2)$, and therefore $Z = X_1 Y_2 + X_2 Y_1$. Finally we conclude that $P = 10^n X_1 Y_1 + 10^{n/2}(X_1 Y_2 + X_2 Y_1) + X_2 Y_2 = XY$.

12. **return** $P$

*Analysis.* This is a *recursive* algorithm: during execution, it calls smaller instances of itself.

Let $M(n)$ denote the number of *digit-multiplications* (line 1) required by the Karatsuba algorithm when multiplying two $n$-digit integers ($n = 2^k$). In lines 6,7,8 the procedure calls itself three times on $n/2$-digit integers; therefore

$$M(n) = 3M(n/2). \tag{1}$$

This equation is a simple *recurrence* which we may solve directly as follows. Applying equation (1) to $M(n/2)$ we obtain $M(n/2) = 3M(n/4)$; therefore $M(n) = 9M(n/4)$. Continuing similarly we see that $M(n) = 27M(n/8)$, and it follows by induction on $i$ that for every $i$ ($i \leq k$),

$$M(n) = 3^i M(n/2^i).$$

Setting $i = k$ we find that $M(n) = 3^k M(n/2^k) = 3^k M(1) = 3^k$. Notice that $k = \log n$ (recall: in this note, log refers to base-2 logarithm), therefore $\log M(n) = k \log 3$ and hence $M(n) = 2^{\log M(n)} = 2^{k \log 3} = (2^k)^{\log 3} = n^{\log 3}$.

It would seem that we reduced the number of digit-multiplications to $n^{\log 3}$ at the cost of an increased number of additions (lines 9, 10). Appearances are deceptive: actually, the procedure achieves similar savings in terms of the total number of digit-operations (additions as well as multiplications).

To see this, let $T(n)$ be the total number of digit-operations (additions, multiplications, bookkeeping (copying digits, maintaining links)) required by the Karatsuba algorithm. Then

$$T(n) = 3T(n/2) + O(n) \tag{2}$$

where the term $3T(n/2)$ comes, as before, from lines 6,7,8; the additional $O(n)$ term is the number of digit-additions required to perform the additions and subtractions in lines 9 and 10. The $O(n)$ term also includes bookkeeping costs.

We shall learn later how to analyse recurrences of the form (2). It turns out that the additive $O(n)$ term does not change the rate of growth, and the result will still be

$$T(n) = O(n^{\log 3}). \tag{3}$$

**Theorem 1** (Karatsuba)**.** *Multiplication of $n$-digits integers can be performed at the cost of $O(n^{\log 3})$ digit operations and bookkeeping operations.*

Comment: While this was not the last word on the complexity of integer multiplication (methods based on Fast Fourier Transform are even faster), it inspired a lot of further progress, including Strassen's fast matrix multiplication.

---

## Multiplication of polynomials of large degree

The same method works for the multiplication of polynomials.

The model:
*Input:* a pair of polynomials of degree $\leq n-1$, $X(t) = a_0+a_1t+\cdots+a_{n-1}t^{n-1}$ and $Y(t) = b_0+b_1t+\cdots+b_{n-1}t^{n-1}$, each with $n$ coefficient (zero coefficients permitted). The coefficients are real or complex numbers; we refer to these domains as the domain of *scalars*. Each polynomial is given as an array of coefficients.
*Output:* the product of the two polynomials, as an array of its $2n-1$ coefficients.
*Cost:* arithmetic operations with scalars (addition, subtraction, multiplication) and copying scalars at unit cost. Bookkeeping operations (copying links, arithmetic with addresses) at unit cost.

**Exercise 2.** Multiplication of polynomials of degree $\leq n$ can be performed at a cost of $O(n^{\log 3})$ arithmetic operations on scalars, and bookkeeping operations.