Karatsuba's algorithm is a method for efficient multiplication of large integers. We assume that large integers exceed the native word size of the computer used for implementation; large integers represented by a list of digits in some base B.

A highly successful library for computer algebra (SACLIB) originally used $B = 2^{29}$. On a computer with a 32-bit word size, 29 bits of the large integer was stored in the low-order bits. The remaining three bits were used as follows:

- One bit is used to indicate the sign of the large integer.
- One bit is used by a mark-and-sweep style garbage collector.
- One bit is used to tell whether this is a list cell, or is an atom.

From Wikipedia:

> Karatsuba's algorithm is a fast multiplication algorithm. It was discovered by Anatoly Karatsuba in 1960 and published in 1962. It reduces the multiplication of two n-digit numbers to at most $n^{\log_2 3} \approx n^{1.585}$ single-digit multiplications in general (and exactly $n^{\log_2 3}$ when n is a power of 2). It is therefore faster than the classical algorithm, which requires n^2 single-digit products. For example, the Karatsuba algorithm requires $3^{10} = 59,049$ single-digit multiplications to multiply two 1024-digit numbers, whereas the classical algorithm requires $(2^{10})^2 = 1,048,576$ single-digit multiplications.
>
> The Karatsuba algorithm was the first multiplication algorithm asymptotically faster than the quadratic "grade school" algorithm. The Toom–Cook algorithm is a faster generalization of Karatsuba's method, and the Schönhage–Strassen algorithm is even faster, for sufficiently large n.
>
> **Algorithm Basic step:**
>
> The basic step of Karatsuba's algorithm is a formula that allows one to compute the product of two large numbers x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts.
>
> Let x and y be represented as n-digit strings in some base B. For any positive integer m where $m < n$, one can write the two given numbers as
>
> $$\begin{aligned} x &= x_1 B^m + x_0 \\ y &= y_1 B^m + y_0 \end{aligned}$$
>
> where x_0 and y_0 are less than B^m. The product is then:
>
> $$\begin{aligned} xy &= (x_1 B^m + x_0)(y_1 B^m + y_0) \\ &= z_2 B^{2m} + z_1 B^m + z_0 \end{aligned} \tag{1}$$
>
> where
>
> $$\begin{aligned} z_2 &= x_1 y_1 \\ z_0 &= x_0 y_0 \\ z_1 &= (x_1 + x_0)(y_1 + y_0) - z_2 - z_0 \end{aligned} \tag{2}$$

Analysis

If we suppose $n = 2^k$ for some $k > 0$, then our n-digit numbers x and y can be partitioned into two parts:

$$x = \boxed{x_1 \mid x_0}$$
$$y = \boxed{y_1 \mid y_0}$$

where $x_1, x_0, y_1,$ and y_0 are $\frac{n}{2}$ digits in length.

We apply equations (1) and (2) recursively. Equation (2) requires 3 multiplication of integers length $n/2$, plus two subtract operations. Equation (1) requires two shift operations and two add operations. We can easily establish that the time for the add, subtract and shift operations is $\mathcal{O}(n)$. Let $T(n)$ denote the total time to multiply two n-digit numbers. We have the following equation:

$$\begin{aligned} T(1) &= c \\ T(n) &= 3T(n/2) + cn \end{aligned} \tag{3}$$

Equation (3) has solution $T(n) \in \mathcal{O}(n^{\log_2 3})$.

Almost, But Not Quite Correct

In the discussion and analysis above, we have overlooked one subtle, but important point. In the computation of z_1 in equation (2), we have overlooked the fact that the additions $(x_1 + x_0)$ and $(y_1 + y_0)$ may involve a "carry" in the high order digit. The assumption that $(x_1 + x_0)$ and $(y_1 + y_0)$ are $n/2$-digit numbers is not correct. In the worst (and probable) case, they are $(n/2 + 1)$-digit numbers. That means that the product $(x_1 + x_0)(y_1 + y_0)$ may be between two $(n/2 + 1)$-digit numbers. This contradicts one of the key assumptions made by equation (3).

Fortunately, we can work-around the carry problem by partitioning the $(n/2 + 1)$-digit numbers into a single digit and the remaining $n/2$ digits.

Let $u = (x_1 + x_0)$ and $v = (y_1 + y_0)$ and assume they are both $(n/2 + 1)$-digit numbers. We partition u and v into their high-order digit and the $n/2$ remaining low order digits as follows:

$$u = \boxed{u_1 \mid u_0}$$
$$v = \boxed{v_1 \mid v_0}$$

Note that u_1 and v_1 are single-digit numbers. We then compute the product:

$$uv = (u_1 B^{n/2} + u_0)(v_1 B^{n/2} + v_0) \tag{4}$$

and therefore:

$$uv = u_1 v_1 B^n + u_1 v_0 B^{n/2} + u_0 v_1 B^{n/2} + u_0 v_0 \tag{5}$$

Equation (5) contains three additions involving values which are shorter or equal to $(n+2)$ digits in length; these additions can be done in $\mathcal{O}(n)$ time.

The product $u_1 v_1$ involves only single-digit numbers and can be done in $\mathcal{O}(1)$ time. The additional shift involved in forming $u_1 v_1 B^n$ can be done in $\mathcal{O}(n)$ time.

The product $u_0 v_0$ involves only $n/2$-digit numbers and can be counted as one of the sub-problems size $n/2$ in equation (3). I.e., the product $u_0 v_0$ can be done in time $T(n/2)$.[1]

[1] Recall the point of this section is to resolve the question of how to compute z_1 in equation (2) in time $T(n/2) + \mathcal{O}(n)$.

The remaining two terms in equation (5), $u_1 v_0 B^{n/2}$ and $u_0 v_1 B^{n/2}$, involve a product of a single digit with an $n/2$-digit number and a shift of $n/2$ positions. Both these computations can also be done in $\mathcal{O}(n)$ time.

Our analysis of equation (5) confirms that z_1 in equation (2) can be computed in $T(n/2) + \mathcal{O}(n)$ time. Combining this result with the computation of z_2 and z_0 in equation (2) confirms the validity of equation (3), with the caveat that the constant c is somewhat larger than originally stated.

The form of equation (3) remains unchanged, as well as our conclusion that

$$T(n) \in \mathcal{O}(n^{\log_2 3}) .$$