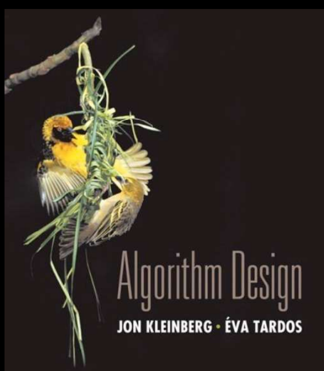
How to Multiply

integers, matrices, and polynomials



Complex Multiplication

Complex multiplication. $(a + bi)(c + di) = x + yi.$

Grade-school. $x = ac - bd,\ y = bc + ad.$

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

Complex Multiplication

Complex multiplication. $(a + bi)(c + di) = x + yi.$

Grade-school. $x = ac - bd,\ y = bc + ad.$

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

A. Yes. [Gauss] $x = ac - bd,\ y = (a + b)(c + d) - ac - bd.$

3 multiplications, 5 additions

Remark. Improvement if no hardware multiply.

5.5 Integer Multiplication

Integer Addition

Addition. Given two n-bit integers a and b, compute $a + b$.
Grade-school. $\Theta(n)$ bit operations.

```
  1 1 1 1 1 1 0 1
    1 1 0 1 0 1 0 1
+   0 1 1 1 1 1 0 1
  1 0 1 0 1 0 0 1 0
```

Remark. Grade-school addition algorithm is optimal.

Integer Multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.
Grade-school. $\Theta(n^2)$ bit operations.

```
                1 1 0 1 0 1 0 1
              × 0 1 1 1 1 1 0 1
                1 1 0 1 0 1 0 1
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```

Q. Is grade-school multiplication algorithm optimal?

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers a and b:
- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$\begin{aligned} a &= 2^{n/2} \cdot a_1 + a_0 \\ b &= 2^{n/2} \cdot b_1 + b_0 \\ ab &= \left(2^{n/2} \cdot a_1 + a_0\right)\left(2^{n/2} \cdot b_1 + b_0\right) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0 \end{aligned}$$

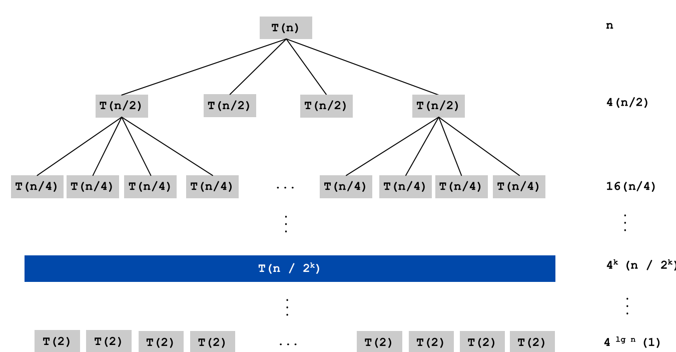
Ex. $a = \underbrace{1000}_{a_1}\underbrace{1101}_{a_0}$ $\quad b = \underbrace{1110}_{b_1}\underbrace{0001}_{b_0}$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n\, 2^k = n\left(\frac{2^{1+\lg n} - 1}{2 - 1}\right) = 2n^2 - n$$



Karatsuba Multiplication

To multiply two n-bit integers a and b:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$
\begin{aligned}
a &= 2^{n/2}\cdot a_1 + a_0 \\
b &= 2^{n/2}\cdot b_1 + b_0 \\
ab &= 2^n\cdot a_1b_1 + 2^{n/2}\cdot\left(a_1b_0 + a_0b_1\right) + a_0b_0 \\
&= 2^n\cdot \underbrace{a_1b_1}_{1} + 2^{n/2}\cdot\left(\underbrace{(a_1+a_0)(b_1+b_0)}_{2} - \underbrace{a_1b_1}_{1} - \underbrace{a_0b_0}_{3}\right) + \underbrace{a_0b_0}_{3}
\end{aligned}
$$

Karatsuba Multiplication

To multiply two n-bit integers a and b:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$
\begin{aligned}
a &= 2^{n/2}\cdot a_1 + a_0 \\
b &= 2^{n/2}\cdot b_1 + b_0 \\
ab &= 2^n\cdot a_1b_1 + 2^{n/2}\cdot\left(a_1b_0 + a_0b_1\right) + a_0b_0 \\
&= 2^n\cdot \underbrace{a_1b_1}_{1} + 2^{n/2}\cdot\left(\underbrace{(a_1+a_0)(b_1+b_0)}_{2} - \underbrace{a_1b_1}_{1} - \underbrace{a_0b_0}_{3}\right) + \underbrace{a_0b_0}_{3}
\end{aligned}
$$

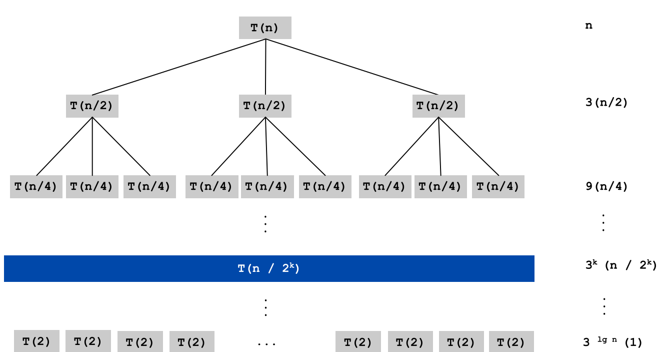
Theorem. [Karatsuba-Ofman 1962] Can multiply two n-bit integers in $O(n^{1.585})$ bit operations.

$$T(n) \le \underbrace{T\left(\lfloor n/2\rfloor\right) + T\left(\lceil n/2\rceil\right) + T\left(1+\lceil n/2\rceil\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})$$

Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n\left(\tfrac{3}{2}\right)^k = n\left(\frac{\left(\frac{3}{2}\right)^{1+\lg n} - 1}{\frac{3}{2} - 1}\right) = 3n^{\lg 3} - 2n$$



Fast Integer Division Too (!)

Integer division. Given two n-bit (or less) integers s and t, compute quotient $q = s / t$ and remainder $r = s \bmod t$.

Fact. Complexity of integer division is same as integer multiplication.
To compute quotient q:

- Approximate $x = 1 / t$ using Newton's method: $x_{i+1} = 2x_i - t\,x_i^2$ ← using fast multiplication
- After $\log n$ iterations, either $q = \lfloor s\,x\rfloor$ or $q = \lceil s\,x\rceil$.