**DIVIDE AND CONQUER II**

- master theorem
- integer multiplication
- matrix multiplication
- convolution and FFT

---

**Master method**

**Goal.** Recipe for solving common divide-and-conquer recurrences:

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \]

**Terms.**

- \(a \geq 1\) is the number of subproblems.
- \(b > 0\) is the factor by which the subproblem size decreases.
- \(f(n)\) = work to divide/merge subproblems.

**Recursion tree.**

- \(k = \log_b n\) levels.
- \(a^i\) = number of subproblems at level \(i\).
- \(n / b^i\) = size of subproblem at level \(i\).

---

**Case 1: total cost dominated by cost of leaves**

**Ex 1.** If \(T(n)\) satisfies \(T(n) = 3 \cdot T(n/2) + n\), with \(T(1) = 1\), then \(T(n) = \Theta(n \log_3 3)\).

\[
T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_3 n}) \cdot n = \frac{r^{1 + \log_3 n} - 1}{r - 1} \cdot n = \frac{3n^{\log_3 3} - 1}{3} 
\]

\(r = 3/2 > 1\)
Ex 2. If \( T(n) \) satisfies \( T(n) = 2T(n/2) + n \), with \( T(1) = 1 \), then \( T(n) = \Theta(n \log n) \).

**Master theorem.** Suppose that \( T(n) \) is a function on the nonnegative integers that satisfies the recurrence

\[
T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)
\]

where \( n/b \) means either \([n/b]\) or \([n/b]\). Let \( k = \log_b a \). Then,

**Case 1.** If \( f(n) = O(n^{k-\epsilon}) \) for some constant \( \epsilon > 0 \), then \( T(n) = \Theta(n^k) \).

Ex. \( T(n) = 3T(n/2) + n \).

- \( a = 3 \), \( b = 2 \), \( f(n) = n \), \( k = \log_2 3 \).
- \( T(n) = \Theta(n^{\log_2 3}) \).

**Case 3:** total cost dominated by cost of root

Ex. \( T(n) = 3T(n/4) + n^5 \), with \( T(1) = 1 \), then \( T(n) = \Theta(n^5) \).

**Master theorem.** Suppose that \( T(n) \) is a function on the nonnegative integers that satisfies the recurrence

\[
T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)
\]

where \( n/b \) means either \([n/b]\) or \([n/b]\). Let \( k = \log_b a \). Then,

**Case 2.** If \( f(n) = \Theta(n^k \log \log n) \), then \( T(n) = \Theta(n^k \log^{p+1} n) \).

Ex. \( T(n) = 2T(n/2) + \Theta(n \log n) \).

- \( a = 2 \), \( b = 2 \), \( f(n) = 17n \), \( k = \log_2 2 = 1 \), \( p = 1 \).
- \( T(n) = \Theta(n \log^2 n) \).
**Master theorem**

**Master theorem.** Suppose that \( T(n) \) is a function on the nonnegative integers that satisfies the recurrence

\[
T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)
\]

where \( n/b \) means either \([n/b]\) or \((n/b)\). Let \( k = \log_b a \). Then,

\[
T(n) = \Theta(n^k)
\]

regularity condition holds if \( f(n) = O(n^{k+c}) \)

**Case 3.** If \( f(n) = \Omega(n^{k+c}) \) for some constant \( c < 0 \) and if \( a f(n/b) \leq c f(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

**Ex.** \( T(n) = 3 T(n/4) + n^2 \).
- \( a = 3 \), \( b = 4 \), \( f(n) = n^2 \), \( k = \log_4 3 \).
- \( T(n) = \Theta(n^2) \).

**Akra-Bazzi theorem**

**Desiderata.** Generalizes master theorem to divide-and-conquer algorithms where subproblems have substantially different sizes.

**Theorem.** [Akra-Bazzi] Given constants \( a_i > 0 \) and \( 0 < b_i \leq 1 \), functions \( h_i(n) = O(n / \log^2 n) \) and \( g(n) = O(n^p) \), if the function \( T(n) \) satisfies the recurrence:

\[
T(n) = \sum_{i=1}^{k} a_i \cdot T(b_i n + h_i(n)) + g(n)
\]

Then \( T(n) = \Theta\left(n^p \left(1 + \int_{1}^{n} \frac{g(u)}{u^{p+1}} \, du\right)\right) \) where \( p \) satisfies \( \sum_{i=1}^{k} a_i b_i^p = 1 \).

**Ex.** \( T(n) = 7/4 \cdot T([n/2]) + T([3/4 \cdot n]) + n^2 \).
- \( a_1 = 7/4 \), \( b_1 = 1/2 \), \( a_2 = 1 \), \( b_2 = 3/4 \) \( \Rightarrow p = 2 \).
- \( h_1(n) = [1/2 \cdot n] - 1/2 \cdot n \), \( h_2(n) = [3/4 \cdot n] - 3/4 \cdot n \).
- \( g(n) = n^2 \Rightarrow T(n) = \Theta(n^2 \log n) \).
Integer addition

**Addition.** Given two \( n \)-bit integers \( a \) and \( b \), compute \( a + b \).

**Subtraction.** Given two \( n \)-bit integers \( a \) and \( b \), compute \( a - b \).

Grade-school algorithm. \( \Theta(n) \) bit operations.

![Grade-school addition diagram](image)

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

Divide-and-conquer multiplication

To multiply two \( n \)-bit integers \( x \) and \( y \):

- Divide \( x \) and \( y \) into low- and high-order bits.
- Multiply four \( \frac{n}{2} \) bit integers, recursively.
- Add and shift to obtain result.

\[
m = \lceil n / 2 \rceil \\
a = \lfloor x / 2^n \rfloor, \quad b = x \mod 2^n \\
c = \lfloor y / 2^n \rfloor, \quad d = y \mod 2^n \\
(2^n a + b) (2^n c + d) = 2^{2n} ac + 2^n (bc + ad) + bd
\]

**Ex.** \( x = 10001101 \) \( y = 11100001 \)

\[
a \quad b \quad c \quad d
\]

Integer multiplication

**Multiplication.** Given two \( n \)-bit integers \( a \) and \( b \), compute \( a \times b \).

Grade-school algorithm. \( \Theta(n^2) \) bit operations.

![Divide-and-conquer multiplication diagram](image)

**Conjecture.** [Kolmogorov 1952] Grade-school algorithm is optimal.

**Theorem.** [Karatsuba 1960] Conjecture is wrong.
Divide-and-conquer multiplication analysis

**Proposition.** The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two $n$-bit integers.

**Pf.** Apply case 1 of the master theorem to the recurrence:

$$T(n) = 4T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

Karatsuba multiplication

Karatsuba analysis

**Proposition.** Karatsuba’s algorithm requires $O(n^{1.585})$ bit operations to multiply two $n$-bit integers.

**Pf.** Apply case 1 of the master theorem to the recurrence:

$$T(n) = 3T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^{\lg 3}) = O(n^{1.585})$$

**Practice.** Faster than grade-school algorithm for about 320-640 bits.
Integer arithmetic reductions

**Integer multiplication.** Given two $n$-bit integers, compute their product.

<table>
<thead>
<tr>
<th>problem</th>
<th>arithmetic</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer multiplication</td>
<td>$a \times b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer division</td>
<td>$a / b, a \mod b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square</td>
<td>$a^2$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square root</td>
<td>$\lfloor \sqrt{a} \rfloor$</td>
<td>$\Theta(M(n))$</td>
</tr>
</tbody>
</table>

integer arithmetic problems with the same complexity as integer multiplication

---

### History of asymptotic complexity of integer multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</td>
<td>$\Theta(n^{1.385})$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>$\Theta(n^{1.465}), \Theta(n^{2.404})$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom-Cook</td>
<td>$\Theta(n^{1.5})$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage-Strassen</td>
<td>$\Theta(n \log n \log \log n)$</td>
</tr>
<tr>
<td>2007</td>
<td>Fürer</td>
<td>$n \log n 2^{O(\log^* n)}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

number of bit operations to multiply two $n$-bit integers

used in Maple, Mathematica, gcc, cryptography, ...

Remark. **GNU Multiple Precision Library uses one of five different algorithms depending on size of operands.**

---

### Dot product

**Dot product.** Given two length $n$ vectors $a$ and $b$, compute $c = a \cdot b$.

**Grade-school.** $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum a_i b_i$$

$$a = [.70 \quad .20 \quad .10]$$

$$b = [.30 \quad .40 \quad .30]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Remark. **Grade-school dot product algorithm is asymptotically optimal.**
Matrix multiplication

Matrix multiplication. Given two \( n \)-by-\( n \) matrices \( A \) and \( B \), compute \( C = AB \).

Grade-school. \( \Theta(n^3) \) arithmetic operations.

\[
C = \sum a_{ij} b_{ij}
\]

\[
\begin{bmatrix}
  c_{11} & c_{12} & \cdots & c_{1n} \\
  c_{21} & c_{22} & \cdots & c_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix}
= \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\times
\begin{bmatrix}
  b_{11} & b_{12} & \cdots & b_{1n} \\
  b_{21} & b_{22} & \cdots & b_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  .59 & .32 & .41 \\
  .31 & .36 & .25 \\
  .45 & .31 & .42
\end{bmatrix}
= \begin{bmatrix}
  .70 & .20 & .10 \\
  .30 & .60 & .10 \\
  .50 & .10 & .40
\end{bmatrix}
\times
\begin{bmatrix}
  .80 & .30 & .50 \\
  .10 & .40 & .10 \\
  .10 & .30 & .40
\end{bmatrix}
\]

**Q.** Is grade-school matrix multiplication algorithm asymptotically optimal?

Matrix multiplication: warmup

To multiply two \( n \)-by-\( n \) matrices **A** and **B**:

- Divide: partition **A** and **B** into \( \frac{n}{2} \)-by-\( \frac{n}{2} \) blocks.
- Conquer: multiply 8 pairs of \( \frac{n}{2} \)-by-\( \frac{n}{2} \) matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

\[
\begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
\end{bmatrix}
= \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix}
\times
\begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
\end{bmatrix}
\]

\[
\begin{align*}
C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\end{align*}
\]

Running time. Apply case 1 of Master Theorem.

\[
T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)
\]

\[
\Rightarrow T(n) = \Theta(n^3)
\]

**Strassen’s trick**

**Key idea.** multiply 2-by-2 blocks with only 7 multiplications.

(plus 11 additions and 7 subtractions)

\[
\begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
\end{bmatrix}
= \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix}
\times
\begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
\end{bmatrix}
\]

\[
\begin{align*}
P_1 &= A_{11} \times (B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12}) \times B_{22} \\
P_3 &= (A_{21} + A_{22}) \times B_{11} \\
P_4 &= A_{22} \times (B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12}) \times (B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12})
\end{align*}
\]

**Pf.**

\[
C_{12} = P_1 + P_2 \\
= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\
= A_{11} \times B_{12} + A_{12} \times B_{22}.
\]

Block matrix multiplication

\[
\begin{bmatrix}
  152 & 158 & 164 & 170 \\
  504 & 526 & 548 & 570 \\
  856 & 894 & 932 & 970 \\
  1208 & 1262 & 1316 & 1370
\end{bmatrix}
= \begin{bmatrix}
  0 & 1 & 2 & 3 \\
  4 & 5 & 6 & 7 \\
  8 & 9 & 10 & 11 \\
  12 & 13 & 14 & 15
\end{bmatrix}
\times
\begin{bmatrix}
  16 & 17 & 18 & 19 \\
  20 & 21 & 22 & 23 \\
  24 & 25 & 26 & 27 \\
  28 & 29 & 30 & 31
\end{bmatrix}
\]

\[
\begin{bmatrix}
  152 & 158 \\
  504 & 526
\end{bmatrix}
= \begin{bmatrix}
  0 & 1 \\
  4 & 5
\end{bmatrix}
\times
\begin{bmatrix}
  16 & 17 \\
  20 & 21
\end{bmatrix}
\]

\[
\begin{bmatrix}
  0 & 526 \\
  504 & 158
\end{bmatrix}
= \begin{bmatrix}
  2 & 3 \\
  6 & 7
\end{bmatrix}
\times
\begin{bmatrix}
  16 & 17 \\
  20 & 22
\end{bmatrix}
\]

\[
\begin{bmatrix}
  970 \\
  1370
\end{bmatrix}
= \begin{bmatrix}
  8 & 9 \\
  12 & 13
\end{bmatrix}
\times
\begin{bmatrix}
  28 & 29 \\
  30 & 31
\end{bmatrix}
\]
Strassen’s algorithm

**Strassen***(n, A, B)**

1. If \((n = 1)\) return \(A \times B\).
2. Assume \(n\) is a power of 2.
3. Partition \(A\) and \(B\) into 2-by-2 block matrices.
   - \(P_1 \leftarrow \text{Strassen}(n / 2, A_{11}, A_{12}, B_{11}, B_{12})\).
   - \(P_2 \leftarrow \text{Strassen}(n / 2, A_{11} + A_{12}, B_{21} - B_{22})\).
   - \(P_3 \leftarrow \text{Strassen}(n / 2, A_{21} + A_{22}, B_{11})\).
   - \(P_4 \leftarrow \text{Strassen}(n / 2, A_{22}, B_{11} - B_{12})\).
   - \(P_5 \leftarrow \text{Strassen}(n / 2, (A_{11} + A_{22}) \times (B_{11} + B_{22}))\).
   - \(P_6 \leftarrow \text{Strassen}(n / 2, (A_{12} - A_{21}) \times (B_{11} + B_{12}))\).
   - \(P_7 \leftarrow \text{Strassen}(n / 2, (A_{11} - A_{22}) \times (B_{11} + B_{12}))\).
4. Compute \(C_{11} = P_3 + P_4 - P_2 + P_6\).
5. Compute \(C_{12} = P_1 + P_2\).
6. Compute \(C_{21} = P_3 + P_4\).
7. Compute \(C_{22} = P_1 + P_5 - P_3 - P_7\).
8. Return \(C\).

**Analysis of Strassen’s algorithm**

**Theorem.** Strassen’s algorithm requires \(O(n^{2.81})\) arithmetic operations to multiply two \(n\)-by-\(n\) matrices.

**Proof.** Apply case 1 of the master theorem to the recurrence:

\[
T(n) = 7T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n \log_2 7) = \Theta(n^{2.81})
\]

**Q.** What if \(n\) is not a power of 2?

**A.** Could pad matrices with zeros.

\[
\begin{bmatrix}
1 & 2 & 3 & 0 \\
4 & 5 & 6 & 0 \\
7 & 8 & 9 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix}
10 & 11 & 12 & 0 \\
13 & 14 & 15 & 0 \\
16 & 17 & 18 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
= 
\begin{bmatrix}
84 & 90 & 96 & 0 \\
201 & 216 & 231 & 0 \\
318 & 342 & 366 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

**Linear algebra reductions**

**Matrix multiplication.** Given two \(n\)-by-\(n\) matrices, compute their product.

<table>
<thead>
<tr>
<th>problem</th>
<th>linear algebra</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix multiplication</td>
<td>(A \times B)</td>
<td>(\Theta(MM(n)))</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>(A^{-1})</td>
<td>(\Theta(MM(n)))</td>
</tr>
<tr>
<td>determinant</td>
<td>(</td>
<td>A</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>(Ax = b)</td>
<td>(\Theta(MM(n)))</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>(A = LU)</td>
<td>(\Theta(MM(n)))</td>
</tr>
<tr>
<td>least squares</td>
<td>(\min |Ax - b|_2)</td>
<td>(\Theta(MM(n)))</td>
</tr>
</tbody>
</table>

Numerical linear algebra problems with the same complexity as matrix multiplication.
Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969]

\[ \Theta(n^{\log_2 7}) = O(n^{2.807}) \]

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr 1971]

\[ \Theta(n^{\log_2 6}) = O(n^{2.59}) \]

Q. Multiply two 3-by-3 matrices with 21 scalar multiplications?
A. Unknown.

\[ \Theta(n^{\log_3 21}) = O(n^{2.77}) \]

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]
- Two 20-by-20 matrices with 4,460 scalar multiplications.
  \[ O(n^{2.805}) \]
- Two 48-by-48 matrices with 47,217 scalar multiplications.
  \[ O(n^{2.799}) \]
- A year later.
- December 1979.
- January 1980.

History of asymptotic complexity of matrix multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>(O(n^{2.808}))</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>(O(n^{2.796}))</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>(O(n^{2.780}))</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>(O(n^{2.522}))</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>(O(n^{2.517}))</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>(O(n^{2.496}))</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>(O(n^{2.479}))</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>(O(n^{2.378}))</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>(O(n^{2.373}))</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>(O(n^{2.372}))</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>(O(n^{2 + \varepsilon}))</td>
</tr>
</tbody>
</table>

number of floating-point operations to multiply two \(n\)-by-\(n\) matrices

Fourier analysis

Fourier theorem. [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.

\[
y(t) = \frac{2}{\pi} \sum_{k=1}^{N} \sin(kt) \frac{\sin(kt)}{k} \quad N = 100
\]
**Euler’s identity.** \( e^{ix} = \cos x + i \sin x. \)

**Sinusoids.** Sum of sine and cosines = sum of complex exponentials.

**Time domain vs. frequency domain**

**Signal.** [touch tone button 1] \( y(t) = \frac{1}{2} \sin(2\pi \cdot 697 t) + \frac{1}{2} \sin(2\pi \cdot 1209 t) \)

**Time domain.**

**Frequency domain.**

**Fast Fourier transform**

**FFT.** Fast way to convert between time-domain and frequency-domain.

**Alternate viewpoint.** Fast way to multiply and evaluate polynomials.

“*If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it.*” — Numerical Recipes
Fast Fourier transform:  applications

Applications.
- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

"The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT."
— Charles van Loan

Polynomials: coefficient representation

Polynomial. [coefficient representation]

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^{n-1}$$

Add. $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_n + b_n) x^{n-1}$$

Evaluate. $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + (x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1})\cdots))))$$

Multiply (convolve). $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i,$$ where $c_i = \sum_{j=0}^{i} a_j b_{i-j}$

Fast Fourier transform:  brief history

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.


Importance not fully realized until advent of digital computers.
Polynomials: point-value representation

Fundamental theorem of algebra. A degree \( n \) polynomial with complex coefficients has exactly \( n \) complex roots.

Corollary. A degree \( n - 1 \) polynomial \( A(x) \) is uniquely specified by its evaluation at \( n \) distinct values of \( x \).

Converting between two representations

Tradeoff. Fast evaluation or fast multiplication. We want both!

<table>
<thead>
<tr>
<th>representation</th>
<th>multiply</th>
<th>evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>coefficient</td>
<td>( O(n^2) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>point-value</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Goal. Efficient conversion between two representations \( \Rightarrow \) all ops fast.

<table>
<thead>
<tr>
<th>point-value</th>
<th>( (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>coefficient</td>
<td>( a_0, a_1, \ldots, a_{n-1} )</td>
</tr>
</tbody>
</table>

Polynomials: coefficient representation

\[ A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \]

Polynomial. [point-value representation]

\[ A(x) : (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \]

\[ B(x) : (x_0, z_0), \ldots, (x_{n-1}, z_{n-1}) \]

Add. \( O(n) \) arithmetic operations.

\[ A(x) + B(x) : (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1}) \]

Multiply (convolve). \( O(n) \), but need \( 2n - 1 \) points.

\[ A(x) \times B(x) : (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1}) \]

Evaluate. \( O(n^2) \) using Lagrange's formula.

\[ A(x) = \sum_{k=0}^{n-1} \frac{y_k \prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)} \]

Converting between two representations: brute force

Coefficient \( \Rightarrow \) point-value. Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \).

Running time. \( O(n^2) \) for matrix-vector multiply (or \( n \) Horner's).
Converting between two representations: brute force

Point-value ⇒ coefficient. Given \( n \) distinct points \( x_0, \ldots, x_{n-1} \) and values \( y_0, \ldots, y_{n-1} \) find unique polynomial \( a_0 + a_1x + \ldots + a_{n-1}x^{n-1} \), that has given values at given points.

\[
\begin{bmatrix}
 y_0 \\
 y_1 \\
 y_2 \\
 \vdots \\
 y_{n-1}
\end{bmatrix} = \begin{bmatrix}
 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix} \begin{bmatrix}
 a_0 \\
 a_1 \\
 a_2 \\
 \vdots \\
 a_{n-1}
\end{bmatrix}
\]

Vandermonde matrix is invertible iff \( x \) : distinct

Running time. \( O(n^3) \) for Gaussian elimination.

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial \( a_0 + a_1x + \ldots + a_{n-1}x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \). \rightarrow we get to choose which ones!

Divide. Break up polynomial into even and odd powers.

- \( A(x) = a_0 + a_1x + a_3x^2 + a_5x^3 + a_7x^4 + a_9x^5 + a_3x^6 + a_5x^7 \).
- \( A_{\text{even}}(x) = a_0 + a_3x + a_6x^2 + a_9x^3 \).
- \( A_{\text{odd}}(x) = a_1 + a_5x + a_7x^2 + a_3x^3 \).
- \( A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2) \).

Intuition. Choose two points to be \( \pm 1 \).

- \( A(1) = A_{\text{even}}(1) + xA_{\text{odd}}(1) \) \rightarrow Can evaluate polynomial of degree \( \leq n \) at 2 points by evaluating two polynomials of degree \( \leq \frac{1}{2}n \) at 1 point.
- \( A(-1) = A_{\text{even}}(1) - xA_{\text{odd}}(1) \)

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial \( a_0 + a_1x + \ldots + a_{n-1}x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \). \rightarrow we get to choose which ones!

Divide. Break up polynomial into even and odd powers.

- \( A(x) = a_0 + a_1x + a_3x^2 + a_5x^3 + a_7x^4 + a_9x^5 + a_3x^6 + a_5x^7 \).
- \( A_{\text{even}}(x) = a_0 + a_3x + a_6x^2 + a_9x^3 + a_6x^6 + a_9x^7 \).
- \( A_{\text{odd}}(x) = a_1 + a_5x + a_7x^2 + a_3x^3 \).
- \( A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2) \).

Intuition. Choose four complex points to be \( \pm 1, \pm i \).

- \( A(1) = A_{\text{even}}(1) + iA_{\text{odd}}(1) \) \rightarrow Can evaluate polynomial of degree \( \leq n \) at 4 points by evaluating two polynomials of degree \( \leq \frac{1}{2}n \) at 2 point.
Discrete Fourier transform

**Coefficient ⇒ point-value.** Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \).

**Key idea.** Choose \( x_k = \omega^k \) where \( \omega \) is principal \( n^{th} \) root of unity.

\[
\begin{bmatrix}
\omega^k \\
\omega^{k+1} \\
\omega^{k+2} \\
\vdots \\
\omega^{k+(n-1)}
\end{bmatrix} = \begin{bmatrix}
\omega^k \\
\omega^{k+1} \\
\omega^{k+2} \\
\vdots \\
\omega^{k+(n-1)}
\end{bmatrix}
\]

\( 0 \leq k < n \)

\[
A(\omega^k) = a_0 + a_1 \omega^k + a_2 \omega^{2k} + \ldots + a_{n-1} \omega^{(n-1)k},
\]

\( 0 \leq k < n \)

\[
A(\omega^{k+1}) = a_0 + a_1 \omega^{k+1} + a_2 \omega^{2(k+1)} + \ldots + a_{n-1} \omega^{(n-1)(k+1)},
\]

\( 0 \leq k < n \)

\[
A(\omega^{k+2}) = a_0 + a_1 \omega^{k+2} + a_2 \omega^{2(k+2)} + \ldots + a_{n-1} \omega^{(n-1)(k+2)},
\]

\( 0 \leq k < n \)

\[
A(\omega^{k+(n-1)}) = a_0 + a_1 \omega^{k+(n-1)} + a_2 \omega^{2(k+(n-1))} + \ldots + a_{n-1} \omega^{(n-1)(k+(n-1))},
\]

\( 0 \leq k < n \)

**Fast Fourier transform**

**Goal.** Evaluate a degree \( n - 1 \) polynomial \( A(x) = a_0 + \ldots + a_{n-1} x^{n-1} \) at its \( n^{th} \) roots of unity: \( \omega^0, \omega^1, \ldots, \omega^{n-1} \).

**Divide.** Break up polynomial into even and odd powers.

- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n-2} \).
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n-2} \).
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2) \).

**Conquer.** Evaluate \( A_{\text{even}}(x) \) and \( A_{\text{odd}}(x) \) at the \( \sqrt{n}^{th} \) roots of unity: \( \nu_0, \nu_1, \ldots, \nu_{n/2-1} \).

**Combine.**

- \( A(\nu^0) = A_{\text{even}}(\nu^0) + \nu^0 A_{\text{odd}}(\nu^0), \quad 0 \leq k < n/2 \)
- \( A(\nu^k + \nu^{n-k}) = A_{\text{even}}(\nu^k) - \nu^k A_{\text{odd}}(\nu^k), \quad 0 \leq k < n/2 \)
- \( A(\nu^{n/2}) = A_{\text{even}}(\nu^{n/2}) - \nu^{n/2} A_{\text{odd}}(\nu^{n/2}), \quad 0 \leq k < n/2 \)

**FFT: implementation**

**FFT**

\( \text{FFT}(a_0, a_1, a_2, \ldots, a_{n-1}) \)

**IF** \( n = 1 \) **RETURN** \( a_0 \).

\( \begin{align*}
(e_0, e_1, \ldots, e_{n/2-1}) & \leftarrow \text{FFT}(n/2, a_0, a_2, a_4, \ldots, a_{n-2}). \\
(d_0, d_1, \ldots, d_{n/2-1}) & \leftarrow \text{FFT}(n/2, a_1, a_3, a_5, \ldots, a_{n-1}). \\
\text{FOR} \ k = 0 \text{ TO } n/2 - 1. \\
& \omega^k = e^{2\pi ik/n}.
\end{align*} \)

\( y_k \leftarrow e_k + \omega^k d_k. \)

\( y_k + \omega^{n/2} \leftarrow e_k - \omega^k d_k. \)

**RETURN** \( (y_0, y_1, y_2, \ldots, y_{n-1}). \)
**Theorem.** The FFT algorithm evaluates a degree \( n - 1 \) polynomial at each of the \( n^\text{th} \) roots of unity in \( O(n \log n) \) steps and \( O(n) \) extra space.

**Pf.** \( T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n) \)

---

**Inverse discrete Fourier transform**

**Point-value \( \Rightarrow \) coefficient.** Given \( n \) distinct points \( x_0, \ldots, x_{n-1} \) and values \( y_0, \ldots, y_{n-1} \), find unique polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), that has given values at given points.

\[
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  \vdots \\
  a_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 1 & 1 & 1 & \ldots & 1 \\
  1 & \omega^1 & \omega^2 & \omega^3 & \ldots & \omega^{n-1} \\
  1 & \omega^2 & \omega^4 & \omega^6 & \ldots & \omega^{2(n-1)} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \ldots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1}
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  \vdots \\
  y_{n-1}
\end{bmatrix}
\]

---

**Claim.** Inverse of Fourier matrix \( F_n \) is given by following formula:

\[
G_n = \frac{1}{n}
\begin{bmatrix}
  1 & 1 & 1 & 1 & \ldots & 1 \\
  1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \ldots & \omega^{-(n-1)} \\
  1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \ldots & \omega^{-(2(n-1))} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \ldots & \omega^{-(n-1)(n-1)}
\end{bmatrix}
\]

**Consequence.** To compute inverse FFT, apply same algorithm but use \( \omega^{-1} = e^{-2\pi i / n} \) as principal \( n^\text{th} \) root of unity (and divide the result by \( n \)).
Inverse FFT: proof of correctness

Claim. \( F_n \) and \( G_n \) are inverses.

Pf.

\[
(F_n \circ G_n)_{k k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{k j} \omega^{-j k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}
\]

(summation lemma (below))

Summation lemma. Let \( \omega \) be a principal \( n^{th} \) root of unity. Then

\[
\sum_{j=0}^{n-1} \omega^{k j} = \begin{cases} n & \text{if } k = 0 \mod n \\ 0 & \text{otherwise} \end{cases}
\]

Pf.

• If \( k \) is a multiple of \( n \) then \( \omega^k = 1 \) \( \Rightarrow \) series sums to \( n \).
• Each \( n^{th} \) root of unity \( \omega^k \) is a root of \( x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1}) \).
• If \( \omega^k \neq 1 \) we have: \( 1 + \omega^k + \omega^{2k} + \ldots + \omega^{(n-1)k} = 0 \) \( \Rightarrow \) series sums to 0.

Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree \( n - 1 \) polynomial given values at each of the \( n^{th} \) roots of unity in \( O(n \log n) \) steps.

Pf.

\[ a_0, a_1, \ldots, a_{n-1} \]

\[ O(n \log n) \text{ (FFT)} \]

\[ \omega^0, \omega^1, \ldots, \omega^{n-1} \]

\[ O(n \log n) \text{ (inverse FFT)} \]

assumes \( n \) is a power of 2

Note. Need to divide result by \( n \).

\[
\text{INVERSE-FFT} (n, y_0, y_1, y_2, \ldots, y_{n-1})
\]

\[
\text{IF } (n = 1) \text{ RETURN } y_0.
\]

\[
(a_0, a_1, \ldots, a_{n-1}) \leftarrow \text{INVERSE-FFT} (n/2, y_0, y_2, y_4, \ldots, y_{n-2}).
\]

\[
(b_0, b_1, \ldots, b_{n-1}) \leftarrow \text{INVERSE-FFT} (n/2, y_1, y_3, y_5, \ldots, y_{n-1}).
\]

\[
\text{FOR } k = 0 \text{ TO } n/2 - 1.
\]

\[
\omega^{k} \leftarrow \omega^{-k n/2},
\]

\[
a_k \leftarrow (a_k + \omega^k b_k).
\]

\[
a_k + \omega^{k/2} \leftarrow \omega^{k} b_k.
\]

\[
\text{RETURN } (a_0, a_1, a_2, \ldots, a_{n-1}).
\]

Polyomial multiplication

Theorem. Can multiply two degree \( n - 1 \) polynomials in \( O(n \log n) \) steps.

Pf.

pad with 0s to make \( n \) a power of 2

\[
A(\omega^0), \ldots, A(\omega^{2n-1})
\]

\[
B(\omega^0), \ldots, B(\omega^{2n-1})
\]

\[
C(\omega^0), \ldots, C(\omega^{2n-1})
\]

point-value multiplication \( O(n) \)

inverse FFT \( O(n \log n) \)

2 FFTs \( O(n \log n) \)

coefficient representation

\[ a_0, a_1, \ldots, a_{n-1} \]

\[ b_0, b_1, \ldots, b_{n-1} \]

\[ c_0, c_1, \ldots, c_{2n-1} \]
FFT in practice

Fastest Fourier transform in the West. [Frigo and Johnson]
- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.
- Core algorithm is nonrecursive version of Cooley-Tukey.
- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Runs in $O(n \log n)$ time, even when $n$ is prime.
- Multidimensional FFTs.

FFT in practice

Integer multiplication, redux

Integer multiplication. Given two $n$-bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

Convolution algorithm.
- Form two polynomials. $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Note: $a = A(2)$, $b = B(2)$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ complex arithmetic operations.

Theory. [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.
Theory. [Fürer 2007] $n \log n 2^{O(\log^* n)}$ bit operations.

Practice. [GNU Multiple Precision Arithmetic Library]
It uses brute force, Karatsuba, and FFT, depending on the size of $n$. 

http://www.fftw.org