Master method

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

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

with \( T(0) = 0 \) and \( T(1) = \Theta(1) \).

**Terms.**
- \( a \geq 1 \) is the (integer) number of subproblems.
- \( b \geq 2 \) is the (integer) factor by which the subproblem size decreases.
- \( f(n) = \) work to divide and 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 \( i \).

Geometric series

**Fact 1.** For \( r \neq 1 \),
\[
1 + r + r^2 + r^3 + \ldots + r^{k-1} = \frac{1 - r^k}{1 - r}
\]

**Fact 2.** For \( r = 1 \),
\[
1 + r + r^2 + r^3 + \ldots + r^{k-1} = k
\]

**Fact 3.** For \( r < 1 \),
\[
1 + r + r^2 + r^3 + \ldots = \frac{1}{1 - r}
\]

\[ 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 2 \]
Case 1: total cost dominated by cost of leaves

Ex 1. If $T(n)$ satisfies $T(n) = 3T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log_2 n)$.

\[
T(n) = 3T(n/2) + n
\]

\[
T(n/2) = 3T(n/4) + n/2
\]

\[
T(n/4) = 3T(n/8) + n/4
\]

\[
\vdots
\]

\[
r = 3/2 > 1 \quad T(n) = \left(1 + r + r^2 + r^3 + \ldots + r^{\log_2 n}\right) n = \frac{r^{\log_2 n} - 1}{r - 1} n = 3n^{\log_2 3} - 2n
\]

Case 2: total cost evenly distributed among levels

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

\[
T(n) = 2T(n/2) + n
\]

\[
T(n/2) = 2T(n/4) + n/2
\]

\[
T(n/4) = 2T(n/8) + n/4
\]

\[
\vdots
\]

\[
r = 1 \quad T(n) = \left(1 + r + r^2 + r^3 + \ldots + r^{\log_2 n}\right) n = n (\log_2 n + 1)
\]

Case 3: total cost dominated by cost of root

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

\[
T(n) = 3T(n/4) + n^5
\]

\[
T(n/4) = 3T(n/16) + n^{45}
\]

\[
T(n/16) = 3^2T(n/64) + n^{15}
\]

\[
\vdots
\]

\[
r = 3/4^5 < 1 \quad n^5 \leq T(n) \leq \left(1 + r + r^2 + r^3 + \ldots\right) n^5 \leq \frac{1}{1 - r} n^5
\]

Master theorem

**Master theorem.** Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T \left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

**Case 1.** If $f(n) = O(n^d)$ for some constant $k < \log_b a$, then $T(n) = \Theta(n^d)$.

Ex. $T(n) = 3T(n/2) + 5n$.

- $a = 3$, $b = 2$, $f(n) = 5n$, $k = 1$, $\log_b a = 1.58…$
- $T(n) = \Theta(n^{\log_2 3})$. 
Master theorem

Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T \left( \frac{n}{b} \right) + f(n)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

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

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

- $a = 2$, $b = 2$, $f(n) = 17n$, $k = 1$, $\log_b a = 1$, $p = 1$.
- $T(n) = \Theta(n \log^2 n)$.

---

Master theorem need not apply

Suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a T \left( \frac{n}{b} \right) + f(n)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

**Case 3.** If $f(n) = \Omega(n^k)$ for some constant $k > \log_b a$, 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. $T(n) = 3T(n/2) + n^2$.

- $a = 3$, $b = 2$, $f(n) = n^2$, $k = 2$, $\log_b a = 1.58$...
- Regularity condition: $3 (n/2)^2 \leq c n^2$ for $c = 3/4$.
- $T(n) = \Theta(n^2)$.

---

Gaps in master theorem.

- Number of subproblems must be a constant.
  $$T(n) = \Theta(T(n/2)) + n^2$$
- Number of subproblems must be $\geq 1$.
  $$T(n) = \frac{1}{2} T(n/2) + n^2$$
- Non-polynomial separation between $f(n)$ and $n^{\log_b a}$.
  $$T(n) = 2T(n/2) + \frac{n}{\log n}$$
- $f(n)$ is not positive.
  $$T(n) = 2T(n/2) - n^2$$
- Regularity condition does not hold.
  $$T(n) = T(n/2) + n(2 - \cos n)$$

Pf sketch.

- Use recursion tree to sum up terms (assuming $n$ is an exact power of $b$).
- Three cases for geometric series.
- Deal with floors and ceilings.
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 \( g(n) = O(n^c) \), if the function \( T(n) \) satisfies the recurrence:

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

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. \( T(n) = 7/4 \times T([n/2]) + T([3/4 n]) + n^2 \), with \( T(0) = 0 \) and \( T(1) = 1 \).

* \( a_1 = 7/4, b_1 = 1/2, a_2 = 1, b_2 = 3/4 \Rightarrow p = 2 \).
* \( h_1(n) = \lceil 1/2 n \rceil - 1/2 n, h_2(n) = \lfloor 3/4 n \rfloor - 3/4 n \).
* \( g(n) = n^2 \Rightarrow T(n) = \Theta(n^2 \log n) \).

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.

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

Integer multiplication

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

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

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

Theorem. [Karatsuba 1960] Conjecture is wrong.
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.

\[
\begin{align*}
  m &= \left\lceil \frac{n}{2} \right\rceil \\
  a &= \left\lceil \frac{x}{2^m} \right\rceil & b &= x \mod 2^m \\
  c &= \left\lceil \frac{y}{2^m} \right\rceil & d &= y \mod 2^m \\

  (2^m a + b) (2^m c + d) &= 2^{2m} ac + 2^m (bc + ad) + bd \\
\end{align*}
\]

Ex. $x = 10001101$  \hspace{1cm} $y = 11100001$

\[
\begin{array}{c|c}
  a & 10001101 \\
  b & 1100001 \\
  c & 11100001 \\
  d & 11100001 \\
\end{array}
\]

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:

\[
T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)
\]

Karatsuba trick

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

\[
bc + ad = ac + bd - (a - b)(c - d)
\]

\[
\begin{align*}
  m &= \left\lceil \frac{n}{2} \right\rceil \\
  a &= \left\lceil \frac{x}{2^m} \right\rceil & b &= x \mod 2^m \\
  c &= \left\lceil \frac{y}{2^m} \right\rceil & d &= y \mod 2^m \\

  (2^m a + b) (2^m c + d) &= 2^{2m} ac + 2^m (bc + ad) + bd \\

  &= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd \\
\end{align*}
\]

**Bottom line.** Only three multiplications of $n/2$-bit integers.
Karatsuba multiplication

Karatsuba-Multiply \((x, y, n)\)

**IF** \((n = 1)\)
**RETURN** \(x \times y\).

**ELSE**
- \(m \leftarrow \lceil n / 2 \rceil\).
- \(a \leftarrow \lfloor x / 2^m \rfloor\); \(b \leftarrow x \mod 2^m\).
- \(c \leftarrow \lfloor y / 2^m \rfloor\); \(d \leftarrow y \mod 2^m\).
- \(e \leftarrow \text{Karatsuba-Multiply} (a, c, m)\).
- \(f \leftarrow \text{Karatsuba-Multiply} (b, d, m)\).
- \(g \leftarrow \text{Karatsuba-Multiply} (a - b, c - d, m)\).
**RETURN** \(2^m e + 2^m (e + f - g) + f\).

---

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:

\[
T(n) = 3T(n/2) + \Theta(n) \implies T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})
\]

**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>(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 \(M(n)\) 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>1962</td>
<td>Karatsuba–Ofman</td>
<td>(O(n^{1.585}))</td>
</tr>
<tr>
<td>1963</td>
<td>Toom–3, Toom–4</td>
<td>(O(n^{1.465}), O(n^{1.404}))</td>
</tr>
<tr>
<td>1966</td>
<td>Toom–Cook</td>
<td>(O(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.
DIVIDE AND CONQUER II

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

SECTION 4.2

Matrix multiplication

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

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

\[
C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}
\]

\[
\begin{bmatrix}
0.59 & 0.32 & 0.41 \\
0.31 & 0.36 & 0.25 \\
0.45 & 0.31 & 0.42
\end{bmatrix}
\begin{bmatrix}
0.70 & 0.20 & 0.10 \\
0.30 & 0.60 & 0.10 \\
0.50 & 0.10 & 0.40
\end{bmatrix}
\begin{bmatrix}
0.80 & 0.30 & 0.50 \\
0.10 & 0.40 & 0.10 \\
0.10 & 0.30 & 0.40
\end{bmatrix}
\]

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

Dot product

Dot product. Given two length-\( n \) vectors \( a \) and \( b \), compute \( c = a \cdot b \).

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

\[
a = \begin{bmatrix}
0.70 & 0.20 & 0.10 \\
0.30 & 0.40 & 0.30
\end{bmatrix}
b = \begin{bmatrix}
0.10 & 0.20 & 0.10 \\
0.20 & 0.40 & 0.30
\end{bmatrix}
\]

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

Remark. Grade-school dot product algorithm is asymptotically optimal.

Block matrix multiplication

\[
C_{i1} = A_{i1} \times B_{i1} + A_{i2} \times B_{21}
\]

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

\[
C_{i1} = A_{i1} \times B_{i1} + A_{i2} \times B_{21} = \begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{bmatrix}
\begin{bmatrix}
16 & 17 & 18 & 19 \\
20 & 21 & 22 & 23 \\
24 & 25 & 26 & 27 \\
28 & 29 & 30 & 31
\end{bmatrix}
\]

\[
C_{i1} = \begin{bmatrix}
152 & 158 \\
504 & 526
\end{bmatrix}
\]
Matrix multiplication: warmup

To multiply two \( n \times n \) matrices \( A \) and \( B \):

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

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

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21})
\]
\[
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22})
\]
\[
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21})
\]
\[
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

Running time. Apply case 1 of Master Theorem.

\[
T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^3)
\]

Strassen’s trick

Key idea. multiply 2-by-2 blocks with only 7 multiplications.
(plus 11 additions and 7 subtractions)

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

\[
P_1 \leftarrow A_{11} \times (B_{12} - B_{22})
\]
\[
P_2 \leftarrow (A_{11} + A_{12}) \times B_{22}
\]
\[
P_3 \leftarrow (A_{21} + A_{22}) \times B_{11}
\]
\[
P_4 \leftarrow (A_{12} - A_{22}) \times (B_{21} - B_{11})
\]
\[
P_5 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22})
\]
\[
P_6 \leftarrow (A_{12} - A_{21}) \times (B_{11} + B_{22})
\]
\[
P_7 \leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12})
\]

Pf. \[
C_{12} = P_1 + P_2 - P_3 + P_4 + P_5 - P_6 + P_7
\]
\[
C_{11} = P_5 + P_4 - P_2 + P_6
\]
\[
C_{11} = A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} = A_{11} \times B_{12} + A_{12} \times B_{22}
\]

Analysis of Strassen’s algorithm

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

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

\[
T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]

Q. What if \( n \) is not a power of 2?
A. Could pad matrices with zeros.

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

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm when $n$ is “small.”

Common misperception. “Strassen’s algorithm is only a theoretical curiosity.”
- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2,048$.
- Range of instances where it’s useful is a subject of controversy.

Fast matrix multiplication: theory

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

\[ \Theta(2 \log_2 7) = O(n^{2.807}) \]

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

\[ \Theta(2 \log_2 6) = O(n^{2.59}) \]

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

\[ \Theta(3 \log_2 21) = O(n^{2.77}) \]

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.
  \[ O(n^{2.7801}) \]
- A year later.
- December 1979.
- January 1980.

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>$\text{MM}(n)$</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>$A^{-1}$</td>
<td>$\Theta(\text{MM}(n))$</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(\text{MM}(n))$</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>$A = LU$</td>
<td>$\Theta(\text{MM}(n))$</td>
</tr>
<tr>
<td>least squares</td>
<td>$\min |Ax - b|_2$</td>
<td>$\Theta(\text{MM}(n))$</td>
</tr>
</tbody>
</table>

Numerical linear algebra problems with the same complexity $\text{MM}(n)$ as matrix multiplication

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>$O(n^{2.578})$</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>$O(n^{2.575})$</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>$O(n^{2.5727})$</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
**DIVIDE AND CONQUER II**

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

---

**Fourier analysis**

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

\[
y(t) = \frac{2}{\pi} \sum_{k=1}^{N} \frac{\sin kt}{k} \quad N = 100
\]

---

**Euler’s identity**

**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.**

Reference: Cleve Moler, Numerical Computing with MATLAB
Time domain vs. frequency domain

**Signal.** [recording, 8192 samples per second]

Magnitude of discrete Fourier transform.

Reference: Cleve Moler, Numerical Computing with MATLAB

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

Fast Fourier transform: brief history

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

**Runge-König (1924).** Laid theoretical groundwork.

**Danielson-Lanczos (1942).** Efficient algorithm, x-ray crystallography.

**Cooley-Tukey (1965).** Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

An Algorithm for the Machine Calculation of Complex Fourier Series

Paper published only after IBM lawyers decided not to set precedent of patenting numerical algorithms (conventional wisdom: nobody could make money selling software)

**Importance** not fully realized until advent of digital computers.
Polynomials: coefficient representation

**Polynomial.** [coefficient representation]

\[ A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \]

\[ B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1} \]

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

\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1} \]

**Evaluate.** \(O(n)\) using Horner’s method.

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

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

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

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 \(x\).

"New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree."

"NEW PROOF OF THE THEOREM THAT EVERY ALGEBRAIC RATIONAL INTEGRAL FUNCTION IN ONE VARIABLE CAN BE RESOLVED INTO REAL FACTORS OF THE FIRST OR THE SECOND DEGREE."

Polynomials: point-value representation

**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.

\[ A(x) = \sum_{k=0}^{n-1} y_k \prod_{j=k}^{n-1} \frac{x - x_j}{x_k - x_j} \]
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 ⇒ all ops fast.

\[ a_0, a_1, ..., a_{n-1} \rightarrow (x_0, y_0), ..., (x_{n-1}, y_{n-1}) \]

Converting between two representations: brute force

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

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

**Running time.** \( O(n^2) \) for matrix-vector multiply (or \( n \) Horner’s).

Divide-and-conquer

**Decimation in frequency.** Break up polynomial into low and high powers.

- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \).
- \( A_{\text{low}}(x) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \).
- \( A_{\text{high}}(x) = a_2 + a_4 x + a_6 x^6 + a_7 x^7 \).
- \( A(x) = A_{\text{low}}(x) + x^4 A_{\text{high}}(x) \).

**Decimation in time.** Break up polynomial into even and odd powers.

- \( A(x) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \).
- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^4 + a_6 x^6 \).
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^5 + a_7 x^7 \).
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2) \).
Coefficient to point-value representation: intuition

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 \( x_0, \ldots, x_{n-1} \).

Divide. Break up polynomial into even and odd powers.
- \( A(x) = a_0 + a_1 x + a_3 x^2 + a_4 x^3 + a_5 x^4 + a_6 x^5 + a_7 x^6 \)
- \( A_{even}(x) = a_0 + a_1 x + a_3 x^2 + a_5 x^4 + a_7 x^6 \)
- \( A_{odd}(x) = a_1 + a_3 x^2 + a_5 x^4 + a_7 x^6 \)
- \( A(x) = A_{even}(x^2) + x A_{odd}(x^2) \)
- \( A(-x) = A_{even}(x^2) - x A_{odd}(x^2) \)

Intuition. Choose two points to be \( \pm 1 \).
- \( A(1) = A_{even}(1) + 1 A_{odd}(1) \)
- \( A(-1) = A_{even}(1) - 1 A_{odd}(1) \)

Can evaluate polynomial of degree \( \leq n \) at 2 points by evaluating two polynomials of degree \( \leq \frac{1}{2} n \) at 1 point.

Discrete Fourier transform

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 \( x_0, \ldots, x_{n-1} \).

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

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3 \\
\vdots \\
y_{n-1}
\end{bmatrix} = 
\begin{bmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega^1 & \omega^2 & \omega^3 & \ldots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \ldots & \omega^{2(n-1)} \\
1 & \omega^3 & \omega^6 & \omega^9 & \ldots & \omega^{3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \ldots & \omega^{(n-1)(n-1)} \\
\end{bmatrix} 
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
\vdots \\
a_{n-1}
\end{bmatrix} = 
\begin{bmatrix}
y_{0} \\
y_{1} \\
y_{2} \\
y_{3} \\
\vdots \\
y_{n-1}
\end{bmatrix}
\]

\( y_k \) is the DFT of \( a_k \).

Roots of unity

Def. An \( n \)-th root of unity is a complex number \( x \) such that \( x^n = 1 \).

Fact. The \( n \)-th roots of unity are: \( \omega^0, \omega^1, \ldots, \omega^{n-1} \) where \( \omega = e^{2\pi i/n} \).

Pf. \( (\omega^n)^n = (e^{2\pi i/n})^n = e^{2\pi n i} = (-1)^n = 1 \).

Fact. The \( \frac{1}{2} n \)-th roots of unity are: \( \sqrt[n]{\omega}, \sqrt[n]{\omega^2}, \ldots, \sqrt[n]{\omega^{n-1}} \) where \( \sqrt[n]{\omega} = e^{4\pi i/n} \).
**Fast Fourier transform**

**Goal.** Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ at its $n$th roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

**Divide.** Break up polynomial into even and odd powers.
- $A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \ldots + a_{n-2}x^{n-2-1}$.
- $A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \ldots + a_{n-1}x^{n-2-1}$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.

**Conquer.** Evaluate $A_{\text{even}}(x)$ and $A_{\text{odd}}(x)$ at the $\frac{1}{2}n$th roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n/2-1}$.

**Combine.**
- $A(\omega^k) = A_{\text{even}}(\omega^{2k}) + \omega^k A_{\text{odd}}(\omega^{2k})$, $0 \leq k < n/2$
- $A(\omega^{k+n/2}) = A_{\text{even}}(\omega^{2k}) - \omega^k A_{\text{odd}}(\omega^{2k})$, $0 \leq k < n/2$

**FFT: summary**

**Theorem.** The FFT algorithm evaluates a degree $n - 1$ polynomial at each of the $n$th roots of unity in $O(n \log n)$ steps and $O(n)$ extra space.

**Pf.** $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$ assumes $n$ is a power of 2.

**FFT: implementation**

```plaintext
FFT (n, a0, a1, a2, ..., an-1)
IF (n = 1) RETURN a0.
(en, e1, ..., en/2-1) ← FFT (n/2, a0, a2, a4, ..., an-2).
d0, d1, ..., dn/2-1) ← FFT (n/2, a1, a3, ..., an-1).
FOR k = 0 TO n/2 - 1.
    ek ← e2\pi i n k.
    yk ← ek + \omega k dk.
    yk+n/2 ← ek - \omega k dk.
RETURN (y0, y1, y2, ..., yn-1).
```

**FFT: recursion tree**

The recursion tree illustrates the divide-and-conquer strategy of the FFT algorithm, showing how the problem is broken down into smaller subproblems.

**Coefficient representation**:
- $a_0, a_1, ..., a_{n-1}$

**Point-value representation**:
- $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

The diagram shows the process of dividing the problem into even and odd parts, recursively applying the FFT to each, and combining the results using the Cooley-Tukey algorithm.
Inverse discrete Fourier transform

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

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

Inverse DFT

Similarly, there is a matrix inverse (\( F_n^{-1} \)) for the Discrete Fourier Transform.

**Inverse FFT: proof of correctness**

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

**Pf.**

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

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

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

**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: implementation**

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

\[
F_n \leftarrow e^{-2\pi i / n} \quad \text{as principal } n\text{th} \text{ root of unity (and divide the result by } n)\]

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

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

- Assumes $n$ is a power of 2

**Polynomial 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

**FFT in practice**

**Fastest Fourier transform in the West.** [Frigo–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.
**Integer multiplication, redux**

**Integer multiplication.** Given two $n$-bit integers $a = a_{n-1} \ldots a_1a_0$ and $b = b_{n-1} \ldots b_1b_0$, compute their product $a \cdot b$.

**Convolution algorithm.**
- Form two polynomials. $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$
- 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 $n$. 