Algorithm II

5. Divide and Conquer II

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Master theorem

DnC recurrences

Goal. Recipe for solving general divide-and-conquer recurrences:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

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

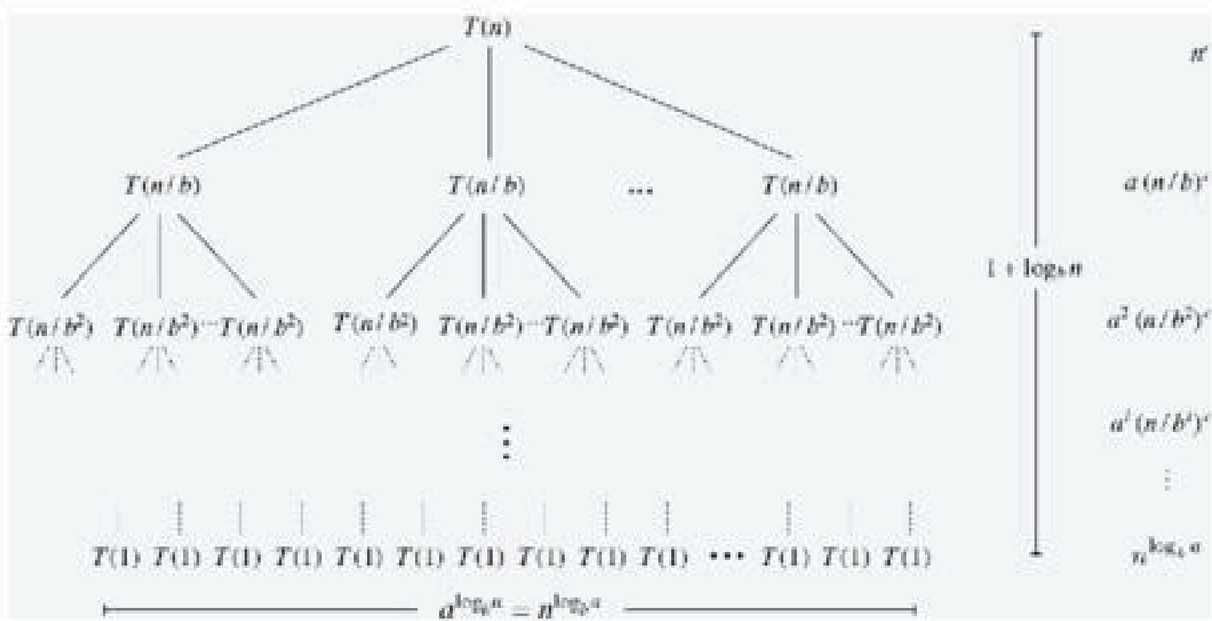
Terms.

- $a \geq 1$: number of sub-problems.
- $b \geq 2$: factor by which the subproblem size decreases.
- $f(n) \geq 0$: work to divide and combine sub-problems.

Recursion tree

Suppose $T(n) = aT(n/b) + n^c$, with $T(1) = 1$ and n a power of b.

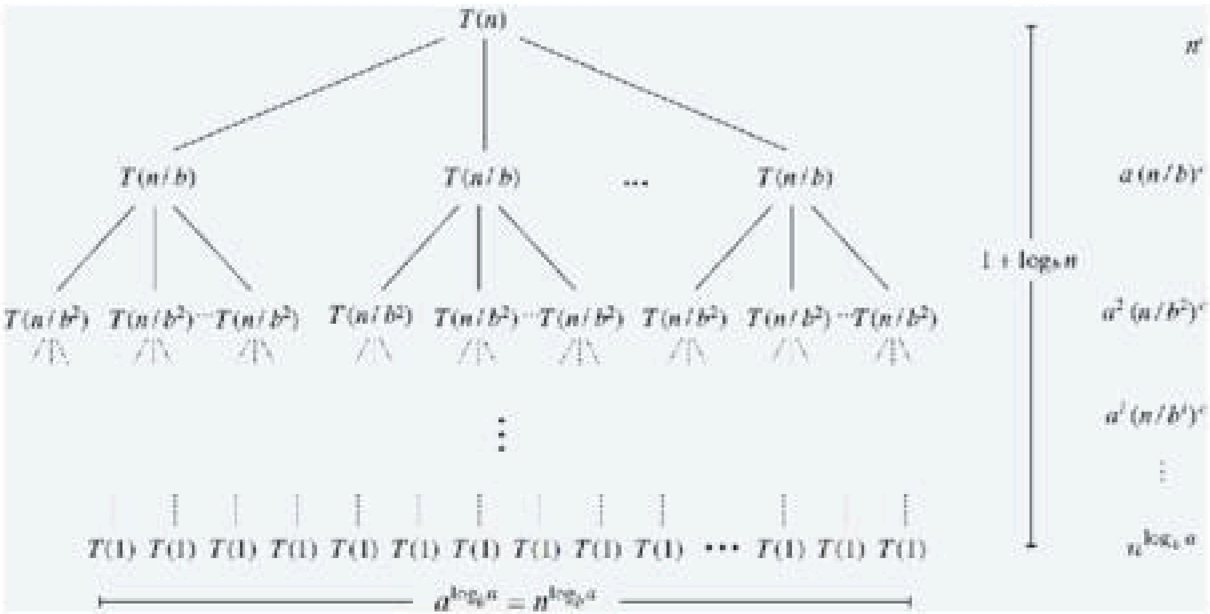
- a: branching factor.
- a^i: number of sub-problems at level i.
- n/b^i: size of subproblem at level i.
- $1 + \log_b n$ levels.



Recursion tree (cont.)

Suppose $T(n) = aT(n/b) + n^c$, with $T(1) = 1$ and n a power of b.

- Let $r = a/b^c$, then $T(n) = n^c \sum_{i=0}^{\log_b n} r^i$.



Note the last one on the right: $a^{\log_b n}(\frac{n}{b^{\log_b n}})^c = n^{\log_b a}(\frac{n}{n})^c = n^{\log_b a}$.

Recursion tree: analysis

Suppose $T(n) = aT(n/b) + n^c$, with $T(1) = 1$ and n a power of b.

- Let $r = a/b^c$. Note that $r < 1$ iff $c > log_b a$.

$$T(n) = n^c \sum_{i=0}^{\log_b n} r^i = \begin{cases} \Theta(n^c) & \text{if} \quad r < 1, \text{ie. cost dominated by root} \\ \Theta(n^c \log n) & \text{if} \quad r = 1, \text{ie. cost evenly distributed} \\ \Theta(n^{\log_b a}) & \text{if} \quad r > 1, \text{ie. cost dominated by leaves} \end{cases}$$

Geometric series.

- If $0 < r < 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = 1/(1-r)$.
- If $r = 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = k + 1$.
- If $r > 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = (r^{k+1} - 1)/(r - 1)$.

DnC: Master theorem

Master theorem

Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on non-negative integers that satisfies the recurrence $T(n) = aT(n/b) + \Theta(n^c)$ with $T(0) = 0, T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Pf sketch.

- Prove when b is an integer and n is an exact power of b.
- Extend domain of recurrences to reals (or rationals).
- Deal with floors and ceilings.

DnC: Master theorem extensions

Master theorem

Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on non-negative integers that satisfies the recurrence $T(n) = aT(n/b) + \Theta(n^c)$ with $T(0) = 0, T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Extensions.

- Can replace Θ with O everywhere.
- Can replace Θ with Ω everywhere.
- Can replace initial conditions so $T(n) = \Theta(1)$ for all $n \leq n_0$ and require recurrence to hold only for all $n > n_0$.

DnC: Master theorem - case 1

Master theorem

Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on non-negative integers that satisfies the recurrence $T(n) = aT(n/b) + \Theta(n^c)$ with $T(0) = 0, T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 1] $T(n) = 48T(\lfloor n/4 \rfloor) + n^3$.

- $a = 48, b = 4, c = 3 > \log_b a = 2.7924\ldots$
- $T(n) = \Theta(n^3)$.

DnC: Master theorem - case 2

Master theorem

Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on non-negative integers that satisfies the recurrence $T(n) = aT(n/b) + \Theta(n^c)$ with $T(0) = 0, T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 2] $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 17n$.

- $a = 2, b = 2, c = 1 = \log_b a$.
- $T(n) = \Theta(n \log n)$.

DnC: Master theorem - case 3

Master theorem

Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on non-negative integers that satisfies the recurrence $T(n) = aT(n/b) + \Theta(n^c)$ with $T(0) = 0, T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Ex. [Case 3] $T(n) = 3T(\lfloor n/2 \rfloor) + 5n$.

- $a = 3, b = 2, c = 1 < \log_b a = 1.5849....$
- $T(n) = \Theta(n^{log_2 3}) = O(n^{1.58})$.

DnC: Master theorem - exceptions

Gaps in master theorem.

- Number of sub-problems is not a constant.
  - $T(n) = nT(n/2) + n^2$
- Number of sub-problems is less than 1.
  - $T(n) = \frac{1}{2}T(n/2) + n^2$
- Work to divide and combine sub-problems is not $\Theta(n^c)$.
  - $T(n) = 2T(n/2) + n \log n$

Akra–Bazzi theorem

Theorem. [Akra–Bazzi 1998]

Let $a_1 > 0, 0 < b_i < 1$, functions $|h_i(n)| = O(n/\log^2 n)$ and $g(n) = O(n^c)$. If $T(n)$ satisfies the recurrence:

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

then, $T(n) = (n^p(1 + \int_1^n \frac{g(u)}{u^{p+1}}\, du))$, where p satisfies $\sum_{i=1}^{k} a_i b_i^p = 1$.

Ex. $T(n) = T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5n$, with $T(0) = 0, T(1) = 0$.

- $a_1 = 1, b_1 = 1/5, a_2 = 1, b2 = 7/10 \Rightarrow p = 0.83978... < 1$.
- $h_1(n) = \lfloor n/5 \rfloor - n/5, h_2(n) = 3/10n - 3\lfloor n/10 \rfloor$.
- $g(n) = 11/5n \Rightarrow T(n) = \Theta(n)$.

Integer multiplication

Integer addition and subtraction

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.

=	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**
		1	1	0	1	0	1	0	1
+		0	1	1	1	1	1	0	1
	1	1	1	1	1	1	0	1	

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

Integer Multiplication Problem

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

```
   12
 × 13
 ----
   36
  12
 ----
  156
```

```
     1100
   × 1101
   ------
     1100
    0000
   1100
  1100
 --------
 10011100
```

Conjecture. [Kolmogorov 1956] Grade-school algorithm is optimal.
Theorem. [Karatsuba 1960] Conjecture is false.

DnC multiplication

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
  - $m = \lceil n/2 \rceil$
  - $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$
  - $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$
- Multiply four $(n/2)$-bit integers, recursively.
- Add and shift to obtain result.

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

Ex.

	a				b			
x =	1	0	0	0	1	1	0	1
y =	1	1	1	0	0	0	0	1

DnC multiplication: algorithm

1. IF $(n = 1)$: RETURN $x \times y$;
2. ELSE:
    1. $m = \lceil n/2 \rceil$;
    2. $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$;
    3. $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$;
    4. e = `MULTIPLY`(a, c, m);
    5. f = `MULTIPLY`(b, d, m);
    6. g = `MULTIPLY`(b, c, m);
    7. h = `MULTIPLY`(a, d, m);
3. RETURN $2^{2m}e + 2^m(g + h) + f$;

DnC multiplication: algorithm

1. IF ($n = 1$): RETURN $x \times y$;
2. ELSE:
   1. $m = \lceil n/2 \rceil$;
   2. $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$;
   3. $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$;
   4. e = `MULTIPLY`(a, c, m);
   5. f = `MULTIPLY`(b, d, m);
   6. g = `MULTIPLY`(b, c, m);
   7. h = `MULTIPLY`(a, d, m);
3. RETURN $2^{2m}e + 2^m(g + h) + f$;

Time.

- `2.1-2.3`: $\Theta(n)$
- `2.4-2.7`: $4T(\lceil n/2 \rceil)$
- `3`: $\Theta(n)$

Quiz: DnC multiplication

How many bit operations to multiply two n-bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

A. $T(n) = \Theta(n^{1/2})$.
B. $T(n) = \Theta(n \log n)$.
C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.
D. $T(n) = \Theta(n^2)$.

Quiz: DnC multiplication

How many bit operations to multiply two n-bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

A. $T(n) = \Theta(n^{1/2})$.
B. $T(n) = \Theta(n \log n)$.
C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.
D. $T(n) = \Theta(n^2)$.

D: $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

Karatsuba trick

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
  - $m = \lceil n/2 \rceil$
  - $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$
  - $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$
- To compute middle term $bc + ad$, use identity:
  - $bc + ad = ac + bd - (a - b)(c - d)$
- Multiply only *three* $(n/2)$-bit integers, recursively.
  - reuse: ac and bd.

$$\begin{aligned} xy = (2^m a + b)(2^m c + d) &= 2^{2m} ac + 2^m (bc + ad) + bd \\ &= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd \end{aligned}$$

Karatsuba multiplication

1. IF $(n = 1)$: RETURN $x \times y$;
2. ELSE:
   1. $m = \lceil n/2 \rceil$;
   2. $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$;
   3. $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$;
   4. e = `KARATSUBA-MULTIPLY`(a, c, m);
   5. f = `KARATSUBA-MULTIPLY`(b, d, m);
   6. g = `KARATSUBA-MULTIPLY`$(|a - b|, |c - d|, m)$;
   7. Flip sign of g if needed.
3. RETURN $2^{2m}e + 2^m(e + f - g) + f$;

Karatsuba multiplication

1. IF $(n = 1)$: RETURN $x \times y$;
2. ELSE:
   1. $m = \lceil n/2 \rceil$;
   2. $a = \lfloor x/2^m \rfloor, b = x \mod 2^m$;
   3. $c = \lfloor y/2^m \rfloor, d = y \mod 2^m$;
   4. e = `KARATSUBA-MULTIPLY`(a, c, m);
   5. f = `KARATSUBA-MULTIPLY`(b, d, m);
   6. g = `KARATSUBA-MULTIPLY`$(|a - b|, |c - d|, m)$;
   7. Flip sign of g if needed.
3. RETURN $2^{2m}e + 2^m(e + f - g) + f$;

Time.

- `2.1-2.3`: $\Theta(n)$
- `2.4-2.6`: $3T(\lceil n/2 \rceil)$
- `3`: $\Theta(n)$

Karatsuba analysis

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

Pf. Apply Case 3 of the master theorem to the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

≡

Karatsuba analysis

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

Pf. Apply Case 3 of the master theorem to the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

Practice.

- Use base-32 or -64 (instead of base-2).
- Faster than grade-school algorithm for about 320 – 640 bits.

Integer arithmetic reductions

arithmetic problem	formula	bit complexity
integer multiplication	$a \times b$	$M(n)$
integer square	a^2	$\Theta(M(n))$
integer division	$\lfloor a/b \rfloor, a \mod b$	$\Theta(M(n))$
integer square root	$\lfloor \sqrt{a} \rfloor$	$\Theta(M(n))$

Integer arithmetic problems have the same bit complexity $M(n)$ as integer multiplication.

Matrix multiplication

Dot product

Dot product. Given two length-n vectors a and b, compute $c = a \cdot b = \sum_{i=1}^{n} a_i b_i$.
Grade-school. $\Theta(n)$ arithmetic operations.

Ex. $a = [.70 .20 .10], b = [.30 .40 .30]$:

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

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

Matrix Multiplication Problem

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

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

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

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

Block matrix multiplication

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

$$\begin{aligned} C_{11} &= A_{11} \times B_{11} + A_{12} \times B_{21} \\ &= \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} \\ &= \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix} \end{aligned}$$

Block matrix multiplication: warmup

To multiply two n-by-n matrices A and B:

- Divide: partition A and B into $n/2$-by-$n/2$ blocks.
- Conquer: multiply 8 pairs of $n/2$-by-$n/2$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

$$C = A \times B$$

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

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$
$$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$
$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$
$$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

Running time. Apply Case 3 of the master theorem.

$$T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$$

Strassen's trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

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

$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_1 + P_5 - P_3 - P_7$$

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

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

Strassen's algorithm

1. IF ($n = 1$): RETURN $A \times B$;
2. Partition A and B into $n/2$-by-$n/2$ blocks;
3. P_1 = `STRASSEN`$(n/2, A_{11}, (B_{12} - B_{22}))$;
4. P_2 = `STRASSEN`$(n/2, (A_{11} + A_{12}), B_{22})$;
5. P_3 = `STRASSEN`$(n/2, (A_{21} + A_{22}), B_{11})$;
6. P_4 = `STRASSEN`$(n/2, A_{22}, (B_{21} - B_{11}))$;
7. P_5 = `STRASSEN`$(n/2, (A_{11} + A_{22}), (B_{11} + B_{22}))$;
8. P_6 = `STRASSEN`$(n/2, (A_{12} - A_{22}), (B_{21} + B_{22}))$;
9. P_7 = `STRASSEN`$(n/2, (A_{11} - A_{21}), (B_{11} + B_{12}))$;
10. $C_{11} = P_5 + P_4 - P_2 + P_6$;
11. $C_{12} = P_1 + P_2$;
12. $C_{21} = P_3 + P_4$;
13. $C_{22} = P_1 + P_5 - P_3 - P_7$;
14. RETURN C;

Time.

- `3-9`: $7T(n/2) + \Theta(n^2)$
- `10-13`: $\Theta(n^2)$

Strassen's algorithm: analysis

Theorem. [Strassen 1968] Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:
  - $T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81})$
- When n is not a power of 2, pad matrices with zeros to be n'-by-n', where $n \leq n' < 2n$ and n' is a power of 2.

Strassen's algorithm: practice

Implementation issues.

- Sparsity.
- Caching.
- n not a power of 2.
- Numerical stability.
- Non-square matrices.
- Storage for intermediate sub-matrices.
- Crossover to classical algorithm when n is "small."
- Parallelism for multi-core and many-core architectures.

Common misperception. "Strassen's algorithm is only a theoretical curiosity."

- Apple reports 8x speedup when $n \approx 2,048$.
- Range of instances where it's useful is a subject of controversy.

Quiz: matrix multiplication

Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications.
How fast could you multiply two n-by-n matrices?

A. $\Theta(n^{\log_3 21})$
B. $\Theta(n^{\log_2 21})$
C. $\Theta(n^{\log_9 21})$
D. $\Theta(n^2)$

Quiz: matrix multiplication

Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n-by-n matrices?

A. $\Theta(n^{\log_3 21})$
B. $\Theta(n^{\log_2 21})$
C. $\Theta(n^{\log_9 21})$
D. $\Theta(n^2)$

A

Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?
A. Impossible. [Hopcroft–Kerr, Winograd 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Numeric linear algebra reductions

linear algebra problem	expression	arithmetic complexity
matrix multiplication	A × B	$MM(n)$
matrix squaring	A^2	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$\|A\|$	$\Theta(MM(n))$
rank	$rank(A)$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = LU$	$\Theta(MM(n))$
least squares	$\min \|Ax - b\|_2$	$\Theta(MM(n))$

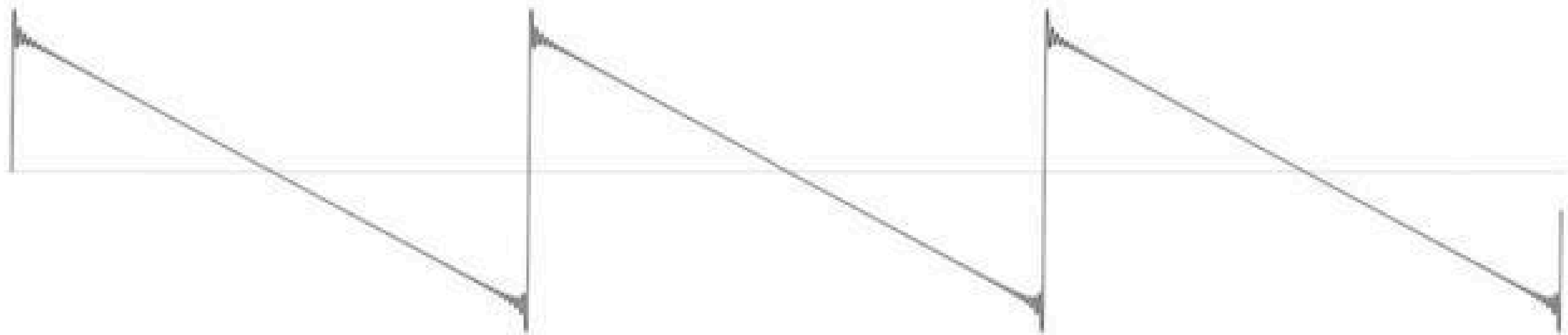
Numerical linear algebra problems have the same arithmetic complexity $MM(n)$ as matrix multiplication

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.

- transform: *time* domain -> *frequency* domain.



$$y(t) = \frac{2}{\pi} \sum_{k=1}^{n} \frac{\sin kt}{k} \qquad n = 100$$

Euler's identity

Euler's identity. $e^{ix} = \cos x + i \sin x$.

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

- $s_N(x) = \frac{a_0}{2} + \sum_{n=1}^{N} (a_n \cos(2\pi n x) + b_n \sin(2\pi n x))$
- $s_N(x) = \sum_{n=-N}^{N} c_n \cdot e^{i 2\pi n x}$

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*

≡

Polynomials: coefficient representation

Univariate polynomial. [coefficient representation]

- $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$: $a_0, a_1, \ldots + a_{n-1}$
- $B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$: $b_0, b_1, \ldots + b_{n-1}$

Addition. $O(n)$ arithmetic operations.

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

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

- $A(x) = a_0 + (x(a_1 + x(a_2 + \ldots + x(a_{n-2} + x(a_{n-1}))\ldots))$
  - `for` $j = n - 1..0$: $v = a[j] + (x * v)$;

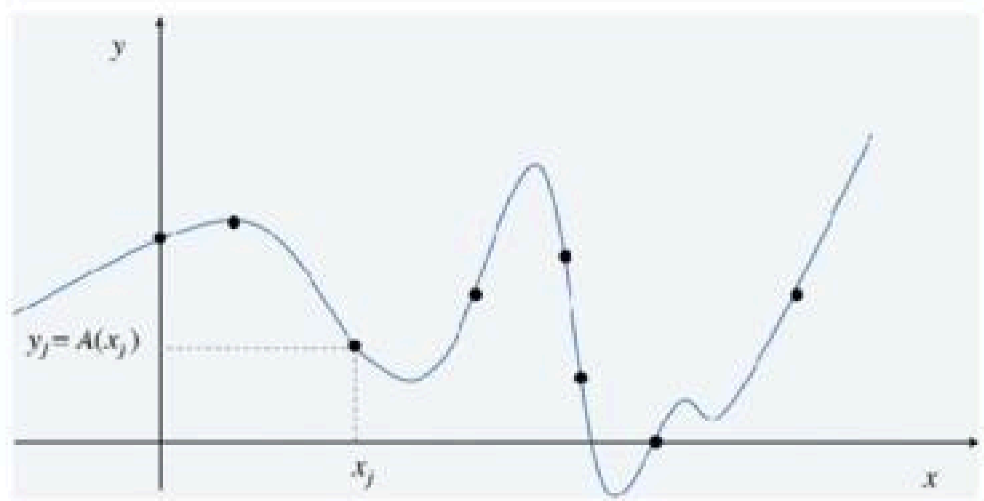
Multiplication (linear convolution). $O(n^2)$ using brute-force.

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

Polynomials: point-value representation

Fundamental theorem of algebra. A degree n univariate polynomial with complex coefficients has exactly n complex roots.

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



Polynomials: point-value (cont.)

Univariate polynomial. [point-value representation]

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

Addition. $O(n)$ arithmetic operations.

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

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

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

Evaluation. $O(n^2)$ using Lagrange's formula.

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

Representations: tradeoff

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

representation	**multiply**	**evaluate**
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

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

Converting between two representations

Application. Polynomial multiplication (coefficient representation).

- exactly the reason to do Fourier transform

coefficient	**transform**	**point-value**
coefficient	FFT: $O(n \log n)$	point-value
		multiplication: $O(n)$
coefficient	inv. FFT: $O(n \log n)$	point-value

Converting representation: brute-force

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

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

Running time. $O(n^2)$ via matrix-vector multiply (or $O(n)$ Horner's).

Converting: brute-force (cont.)

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

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

- Vandermonde matrix is invertible iff x_i distinct.

Running time. $O(n^3)$ via Gaussian elimination.

- or $O(n^{2.38})$ via fast matrix multiplication

Divide-and-conquer

Decimation in time. Divide into even- and odd- degree terms.

- $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$.
- $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3$.
- $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3$.
- $A(x) = A_{even}(x^2) + xA_{odd}(x^2)$.

Decimation in frequency. Divide into low- and high-degree terms.

- $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$.
- $A_{low}(x) = a_0 + a_1x + a_2x^2 + a_3x^3$.
- $A_{high}(x) = a_4 + a_5x + a_6x^2 + a_7x^3$.
- $A(x) = A_{low}(x) + x^4A_{high}(x)$.

Coefficient $\Rightarrow$ Point-value: intuition

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

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
    - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
    - $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
    - $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$.

Need 4 evaluations.

Coefficient $\Rightarrow$ Point-value: intuition (cont.)

Intuition. Choose four *complex* points to be $\pm 1, \pm i$.

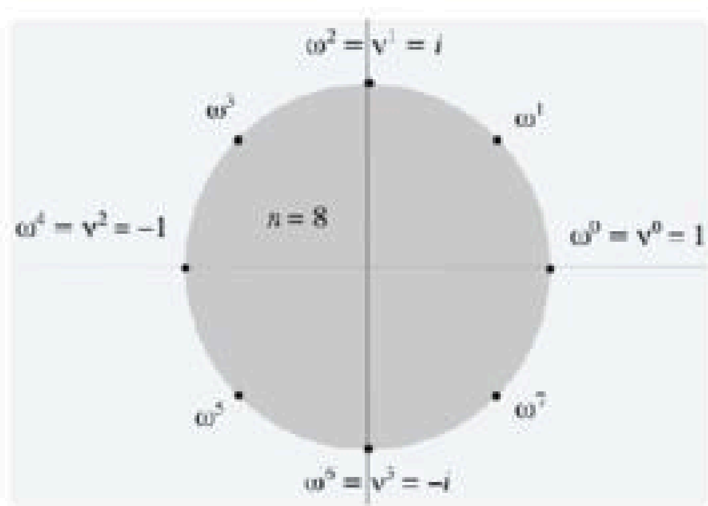
- $A(1) = A_{even}(1) + 1A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1A_{odd}(1)$.
- $A(i) = A_{even}(-1) + iA_{odd}(-1)$.
- $A(-i) = A_{even}(-1) - iA_{odd}(-1)$.

What if $n \geq 8$?

Roots of unity

Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

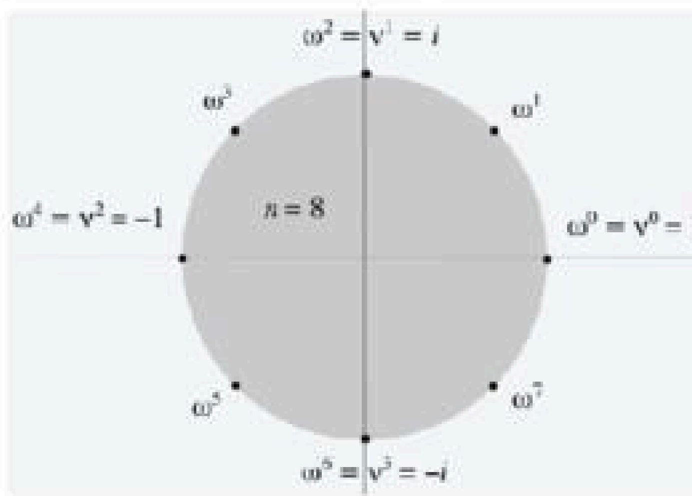
Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$.
Pf. $(\omega^k)^n = (e^{2\pi ik/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.



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



Fact. The $(n/2)^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i/n}$.

Discrete Fourier transform

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

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

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

Fast Fourier transform: steps

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + a_1 x + ... + a_{n-1}x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break up polynomial into even- and odd-degree terms.

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

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

Combine.

- $y_k = A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), 0 \le k < n/2$.
- $y_{k+n/2} = A(\omega^{k+n/2}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), 0 \le k < n/2$.

Fast Fourier transform: algorithm

1. IF $(n = 1)$: RETURN a_0;
2. $(e_0, e_1, \ldots, e_{n/2-1})$ = `FFT`$(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$;
3. $(d_0, d_1, \ldots, d_{n/2-1})$ = `FFT`$(n/2, a_1, a_3, a_5, \ldots, a_{n-1})$;
4. FOR $k = 0..n/2 - 1$:
   1. $\omega^k = e^{2\pi ik/n}$;
   2. $y_k = e_k + \omega^k d_k$;
   3. $y_{k+n/2} = e_k - \omega^k d_k$;
5. RETURN $(y_0, y_1, y_2, \ldots, y_{n-1})$.

Time.

- `2-3`: $2T(n/2)$
- `4.1-4.3`: $\Theta(n)$

FFT: analysis

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

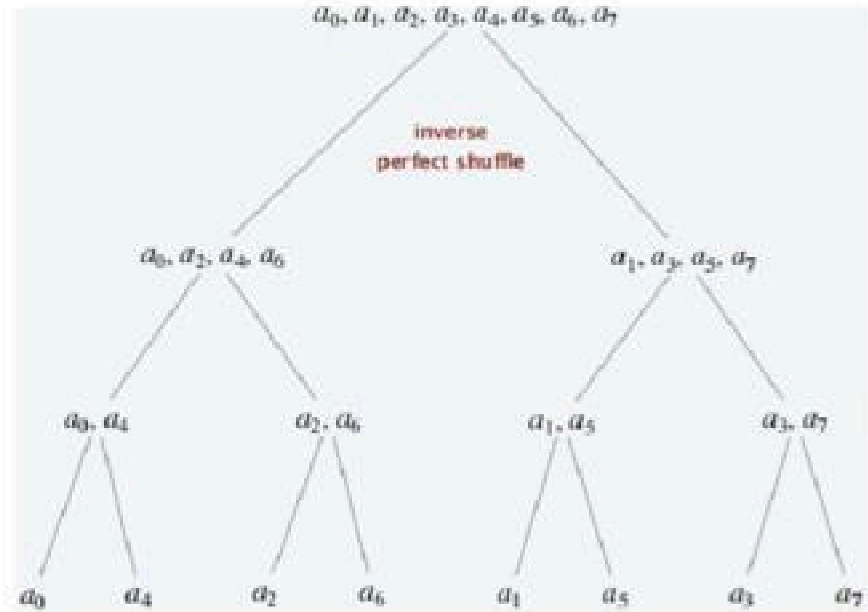
Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 2T(n/2) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

Quiz: FFT tree

When computing the FFT of $(a_0, a_1, a_2, \ldots, a_7)$, which are the first two coefficients involved in an arithmetic operation?

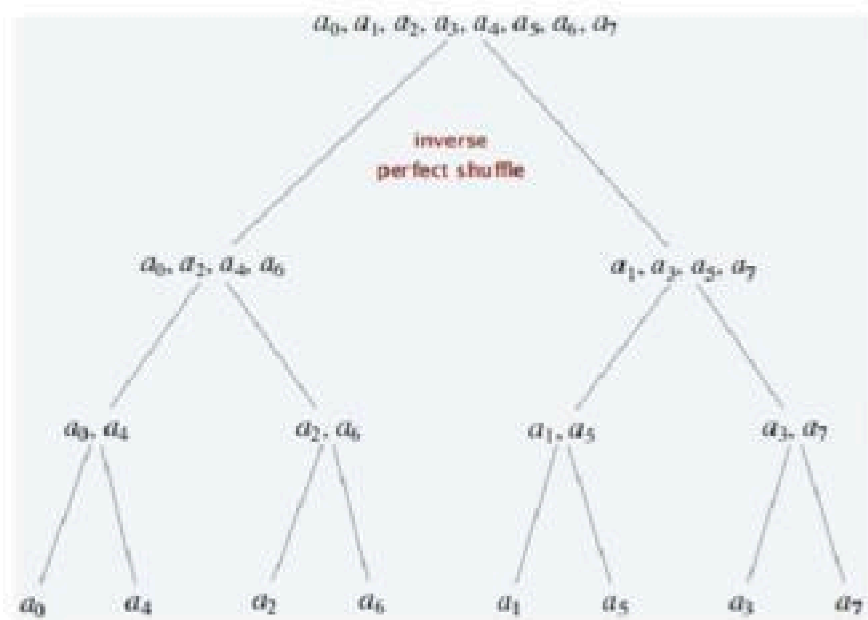
A. a_0 and a_1.
B. a_0 and a_2.
C. a_0 and a_4.
D. a_0 and a_7.
E. None of the above.



Quiz: FFT tree

When computing the FFT of $(a_0, a_1, a_2, \ldots, a_7)$, which are the first two coefficients involved in an arithmetic operation?

A. a_0 and a_1.
B. a_0 and a_2.
C. a_0 and a_4.
D. a_0 and a_7.
E. None of the above.



C: first leaf of the FFT tree.

FFT: Fourier matrix decomposition

Alternative viewpoint. FFT is a recursive decomposition of Fourier matrix.

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{(n-1)} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \quad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \quad D_n = \begin{bmatrix} \omega^0 & 0 & 0 & \cdots & 0 \\ 0 & \omega^1 & 0 & \cdots & 0 \\ 0 & 0 & \omega^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \omega^{n-1} \end{bmatrix}$$

$$y = F_n a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} a_{even} \\ F_{n/2} a_{odd} \end{bmatrix}$$

Inverse discrete Fourier transform

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

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

Inverse FFT

Claim. Inverse of Fourier matrix F_n is given by following formula:

$$G_n = \frac{1}{n} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Consequence. To compute the inverse FFT, apply the 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: correctness

Claim. F_n and G_n are inverses.

Pf.

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

Inverse FFT: correctness (cont.)

Summation lemma. Let ω be a principal n^{th} root of unity. Then

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

Pf.

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

≡

Inverse FFT: algorithm

1. IF $(n = 1)$: RETURN y_0;
2. $(e_0, e_1, \ldots, e_{n/2-1})$ = `INVERSE-FFT(` $n/2, y_0, y_2, y_4, \ldots, y_{n-2})$;
3. $(d_0, d_1, \ldots, d_{n/2-1})$ = `INVERSE-FFT(` $n/2, y_1, y_3, y_5, \ldots, y_{n-1})$;
4. FOR $k = 0..n/2 - 1$:
   1. $\omega^k = e^{-2\pi ik/n}$;
   2. $a_k = e_k + \omega^k d_k$;
   3. $a_{k+n/2} = e_k - \omega^k d_k$;
5. RETURN $(a_0, a_1, a_2, \ldots, a_{n-1})$;

Time.

- `2-3`: $2T(n/2)$
- `4.1-4.3`: $\Theta(n)$

≡

Inverse FFT: analysis

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

- assumes n is a power of 2

Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.

coefficient	transform	point-value
coefficient	FFT: $O(n \log n)$	point-value
		multiplication: $O(n)$
coefficient	inv. FFT: $O(n \log n)$	point-value

Polynomial multiplication

Theorem. Given two polynomials $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ and $B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1}$ of degree $n - 1$, can multiply them in $O(n \log n)$ arithmetic operations.

- pad with 0s to make n a power of 2

Pf.

coefficient	transform	point-value
coefficient	FFT: $O(n \log n)$	point-value
		multiplication: $O(n)$
coefficient	inv. FFT: $O(n \log n)$	point-value

≡

Convolution

Convolution. A vector with $2n-1$ coordinates, where $c_k = \sum_{(i,j):i+j=k} a_i b_j$.

- $a * b = (a_0 b_0, a_0 b_1 + a_1 b_0, a_0 b_2 + a_1 b_1 + a_2 b_0, \ldots, a_{n-2} b_{n-1} + a_{n-1} b_{n-2}, a_{n-1} b_{n-1})$.
  - exactly the coordinates of polynomial multiplication.
- summing along anti-diagonals of the following matrix.

$$\begin{bmatrix} a_0 b_0 & a_0 b_1 & a_0 b_2 & a_0 b_3 & \cdots & a_0 b_{n-1} \\ a_1 b_0 & a_1 b_1 & a_1 b_2 & a_1 b_3 & \cdots & a_1 b_{n-1} \\ a_2 b_0 & a_2 b_1 & a_2 b_2 & a_2 b_3 & \cdots & a_2 b_{n-1} \\ a_3 b_0 & a_3 b_1 & a_3 b_2 & a_3 b_3 & \cdots & a_3 b_{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} b_0 & a_{n-1} b_1 & a_{n-1} b_2 & a_{n-1} b_3 & \cdots & a_{n-1} b_{n-1} \end{bmatrix}$$

Integer multiplication, revisit

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.
  - $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$
  - $B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$
  - Note: $a = A(2), b = B(2)$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ floating-point operations.

Integer multiplication, revisit

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.
  - $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$
  - $B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$
  - Note: $a = A(2), b = B(2)$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ floating-point operations.

Practice. [GNU Multiple Precision Arithmetic Library]

- Switches to FFT-based algorithm when n is large ($\geq 5 - 10$K).

3-Sum: revisit

3-SUM. Given three sets X, Y, and Z of n integers each, determine whether there is a triple $i \in X, j \in Y, k \in Z$ such that $i + j = k$.

Assumption. All integers are between 0 and m.

Goal. $O(m \log m + n \log n)$ time.

Ex.

$m = 19, n = 3$

- $X = \{4, 7, 10\}$
- $Y = \{5, 8, 15\}$
- $Z = \{4, 13, 19\}$

$4 + 15 = 19$

3-Sum: solution

An $O(m \log m + n)$ solution.

- Form polynomial $A(x) = a_0 + a_1 x + \ldots + a_m x^m$ with $a_i = 1$ iff $i \in X$.
- Form polynomial $B(x) = b_0 + b_1 x + \ldots + b_m x^m$ with $b_j = 1$ iff $j \in Y$.
- Compute product/convolution $C(x) = A(x) \times B(x)$.
- The coefficient c_k = number of ways to choose an integer $i \in X$ and an integer $j \in Y$ that sum to exactly k.
- For each $k \in Z$: check whether $c_k > 0$.

Ex.

$m = 19, n = 3$

- $X = \{4, 7, 10\}$
- $Y = \{5, 8, 15\}$
- $Z = \{4, 13, 19\}$

$$A(x) = x^4 + x^7 + x^{10}$$
$$B(x) = x^5 + x^8 + x^{15}$$
$$C(x) = x^9 + 2x^{12} + 2x^{15} + x^{18} + x^{19} + x^{22} + x^{25}$$

$4 + 15 = 19$