Divide and Conquer

Reading: 5.1, 5.4-5.5, 13.5

Divide-and-Conquer

Divide-and-conquer

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage

- Break up problem of size n into two equal parts of size \( \frac{1}{2}n \).
- Solve two parts recursively.
- Combine two solutions into overall solution in \textit{linear time}.

Consequence

- Brute force: \( n^2 \).
- Divide-and-conquer: \( n \log n \).

\textit{Divide et impera. Veni, vidi, vici.}

- \textit{Julius Caesar}

Binary search for roots (bisection method)

\[ \text{Bisection}(a, b, \epsilon) \]

- if \((a-b) < \epsilon\) then
  - return(a)
- else
  - \[ c \leftarrow \frac{(a+b)}{2} \]
  - if \( f(c) \leq 0 \) then
    - return(Bisection(c, b, \epsilon))
  - else
    - return(Bisection(a, c, \epsilon))

Time Analysis:

- At each step we halved the size of the interval.
- It started at size \( b-a \)
- It ended at size \( \epsilon \)

\# of calls to \( f \) is \( \log_2 \left( \frac{b-a}{\epsilon} \right) \)
Old favorites

Binary search
- One subproblem of half size plus one comparison
- Recurrence \( T(n) = T\left(\frac{n}{2}\right) + 1 \) for \( n \geq 2 \)
  \( T(1) = 0 \)
  So \( T(n) \) is \( \log_2 n + 1 \)

Mergesort
- Two subproblems of half size plus merge cost of \( n - 1 \) comparisons
- Recurrence \( T(n) \leq 2T\left(\frac{n}{2}\right) + n - 1 \) for \( n \geq 2 \)
  \( T(1) = 0 \)
  Roughly \( n \) comparisons at each of \( \log_2 n \) levels of recursion
  So \( T(n) \) is roughly \( 2n \log_2 n \)

Proof by Recursion Tree

\[
\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 1 \\
&\leq 2T\left(\frac{n}{2}\right) + n - 1 \\
&\leq 2T\left(\frac{n}{2}\right) + n - \log_2 n \\
&\leq 2T\left(\frac{n}{2}\right) + n - 2 \\
&\leq 2T\left(\frac{n}{2}\right) + n + 2 \\
&\leq 2^n + 2n + 2 \\
&\leq 2n \log_2 n + 2n + 2 \\
&\leq 2n \log_2 n + 2n \\
&\leq 2n \log_2 n \\
\end{aligned}
\]

Proof by Telescoping

Claim. If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

Pf. For \( n \geq 1 \):

\[
\begin{aligned}
T(n) &= \frac{2T(n/2)}{n/2} + 1 \\
&= \frac{T(n/2)}{n/2} + 1 + 1 \\
&= \frac{T(n/4)}{n/4} + 1 + 1 + 1 \\
&\vdots \\
&= \frac{\log_2 n}{n/2} + 1 + \ldots + 1 \\
&= \log_2 n
\end{aligned}
\]

Proof by Induction

Claim. If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

Pf. (by induction on \( n \))
- Base case: \( n = 1 \)
- Inductive hypothesis: \( T(n) = n \log_2 n \)
- Goal: show that \( T(2n) = 2n \log_2 (2n) \)

\[
\begin{aligned}
T(2n) &= 2T(n) + 2n \\
&= 2n \log_2 n + 2n \\
&= 2n(\log_2 (2n) + 1) + 2n \\
&= 2n \log_2 (2n)
\end{aligned}
\]
Analysis of Mergesort Recurrence

Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \log n \rceil \).

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{n}{2} T(n/2) + \frac{n}{2} & \text{otherwise}
\end{cases}
\]

Pf. (by induction on \( n \))

- **Base case**: \( n = 1 \).
- **Define** \( n_1 = \lceil n/2 \rceil \), \( n_2 = \lfloor n/2 \rfloor \).
- **Induction step**: assume true for \( 1, 2, \ldots, n-1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n
\]

\[
\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_2 \rceil + n
\]

\[
\leq n_1 \lfloor \log n_1 \rfloor + n_2 \lfloor \log n_2 \rfloor + n
\]

\[
= n_1 \lfloor \log n \rfloor + n_2 \lfloor \log n \rfloor + n
\]

\[
= n \lfloor \log n \rfloor + n
\]

\[
\leq n \lfloor \log n \rfloor + n
\]

\[
\leq n \lceil \log n \rceil + n
\]

\[
\leq n \lfloor \log n \rfloor + n
\]

\[
\Rightarrow \lfloor \log n \rfloor \leq \lceil \log n \rceil - 1
\]

\[
T(n) \leq n \lceil \log n \rceil
\]

\[
\Rightarrow \text{if } n = 1 \text{ then } T(1) = c
\]

\[
\Rightarrow \text{otherwise}
\]

Proving Master recurrence

\[
T(n) = a \cdot T(n/b) + c \cdot n^k
\]

Problem size

- \( n \)
- \( n/b \)
- \( n/b^2 \)
- \( b \)
- \( 1 \)

Proving Master recurrence

\[
T(n) = a \cdot T(n/b) + c \cdot n^k
\]

Problem size

- \( n \)
- \( n/b \)
- \( n/b^2 \)
- \( b \)
- \( 1 \)

Master Divide and Conquer Recurrence

Let \( a \) and \( b \) be positive constants.

If \( T(n) \leq a \cdot T(n/b) + c \cdot n^k \) for \( n > b \) then

- if \( a > b^k \) then \( T(n) \) is \( \Theta(n^{k \cdot \log_b a}) \)
- if \( a < b^k \) then \( T(n) \) is \( \Theta(n^k) \)
- if \( a = b^k \) then \( T(n) \) is \( \Theta(n^k \log n) \)

Works even if it is \( \lceil n/b \rceil \) instead of \( n/b \).
Proving Master Recurrence

Problem size: \( T(n) = a \cdot T(n/b) + c \cdot n^k \)

<table>
<thead>
<tr>
<th>Problem size</th>
<th># probs</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td></td>
<td>( cn^k )</td>
</tr>
<tr>
<td>( n/b )</td>
<td></td>
<td>( c \cdot a \cdot n^k / b^k )</td>
</tr>
<tr>
<td>( n/b^2 )</td>
<td></td>
<td>( c \cdot a^2 \cdot n^k / b^{2k} )</td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td>( c \cdot n^k (a/b^k)^d )</td>
</tr>
<tr>
<td>( 1 )</td>
<td></td>
<td>( c \cdot a^d )</td>
</tr>
</tbody>
</table>

\( T(1) = c \)

Geometric Series

\[
S = t + tr + tr^2 + ... + tr^{n-1}
\]

\[
r \cdot S = tr + tr^2 + ... + tr^{n-1} + tr^n
\]

\[
(r-1)S = tr^n - t
\]

So \( S = (r^n - 1)/(r - 1) \) if \( r \neq 1 \).

Simple rule:
- If \( r = 1 \) then \( S \) is a constant times the largest term in series

Total Cost

Geometric series
- ratio: \( a/b^k \)
- \( d+1 = \log_b n \) terms
- first term: \( cn^k \), last term: \( ca^d \)

If \( a/b^k = 1 \)
- all terms are equal \( T(n) = \Theta(n^k \log n) \)

If \( a/b^k < 1 \)
- first term is largest \( T(n) = \Theta(n^k) \)

If \( a/b^k > 1 \)
- last term is largest \( T(n) = \Theta(a^d) = \Theta(\alpha^{\log_b n}) = \Theta(n^{\log_b a}) \)
  (To see this take \( \log_b \) of both sides)

13.5 Median Finding and Quicksort
Order problems: Find the k\textsuperscript{th} largest

- **Runtime models**
  - Machine Instructions
  - Comparisons

- **Maximum**
  - $O(n)$ time
  - $n-1$ comparisons

- **2\textsuperscript{nd} Largest**
  - $O(n)$ time
  - $? \text{ Comparisons}$

- **$k\textsuperscript{th}$ largest for $k = n/2$**
  - Easily done in $O(n \log n)$ time with sorting
  - How can the problem be solved in $O(n)$ time?

QuickSelect($k, n$) - find the $k\text{-th}$ largest from a list of length $n$

---

Divide and Conquer

Linear time solution: $T(n) = n + T(\alpha n)$ for $\alpha < 1$

QuickSelect algorithm - in linear time, reduce the problem from selecting the $k\text{-th}$ largest of $n$ to the $j\text{-th}$ largest of $\alpha n$, for $\alpha < 1$

QuickSelect($k, S$)

- Choose element $x$ from $S$
- $S_1 = \{y \in S \mid y < x\}$
- $S_2 = \{y \in S \mid y = x\}$
- $S_3 = \{y \in S \mid y > x\}$
- if $|S_1| \geq k$
  - return QuickSelect($k, S_1$)
- else if $|S_1| + |S_2| \geq k$
  - return $y$ in $S_2$
- else
  - return QuickSelect($k - |S_1| - |S_2|, S_3$)

---

"Choose an element $x"$: Random Selection

Ideally, we would choose an $x$ in the middle, to reduce both sets in half and guarantee progress. But it’s enough to choose $x$ at random

Consider a call to QuickSelect($k, S$), and let $S'$ be the elements passed to the recursive call.

With probability at least $\frac{1}{2}$, $|S'| < \frac{3}{4}|S|$

⇒ On average only 2 recursive calls before the size of $S'$ is at most $3n/4$

```
bad x good x good x bad x
```

elements of $S$ listed in sorted order

---

Announcements

- Homework 4 will be out later today, due date in 2 weeks on Wednesday 2/15
- The midterm is next Wednesday 2/8/2012
- Divide and conquer is not included in the midterm but recurrences are included.
- We will post sample exercises for recurrences on the webpage along with their solutions for practice.
- Remember NO outside sources (Google, other textbooks, people not in the class, etc.) may not be consulted on the homework.

Expected runtime is $O(n)$

Given $x$, one pass over $S$ to determine $S_L$, $S_E$, and $S_R$ and their sizes: $cn$ time.
- Expect $2cn$ cost before size of $S'$ drops to at most $3|S|/4$

Let $T(n)$ be the expected running time: $T(n) \leq T(3n/4) + 2cn$

By Master's Theorem, $T(n) = O(n)$

Making the algorithm deterministic
- In $O(n)$ time, find an element that guarantees that the larger set in the split has size at most $\frac{1}{4}n$
- BFPRT (Blum-Floyd-Pratt-Rivest-Tarjan) Algorithm

Quicksort

Sorting. Given a set of $n$ distinct elements $S$, rearrange them in ascending order.

```java
RandomizedQuicksort(S) { if |S| = 0 return choose a splitter $a_i \in S$ uniformly at random foreach $(a \in S)$ { if ($a < a_i$) put $a$ in $S_-$ else if ($a > a_i$) put $a$ in $S_+$ RandomizedQuicksort(S_-) output $a_i$ RandomizedQuicksort(S_+) }
}
```

Remark. Can implement in-place.
$O(\log n)$ extra space

Expected run time for QuickSort: “Global analysis”

Count comparisons
- $a_i, a_j$ - elements in positions $i$ and $j$ in the final sorted list. $p_{ij}$ the probability that $a_i$ and $a_j$ are compared

Expected number of comparisons: $\sum p_{ij}$

Prob $a_i$ and $a_j$ are compared:
- If $a_i$ and $a_j$ are compared then it must be during the call when they end up in different subproblems
  - Before that, they aren’t compared to each other
  - After they aren’t compared to each other
- During this step they are only compared if one of them is the pivot
- Since all elements between $a_i$ and $a_j$ are also in the subproblem this is $2$ out of at least $j-i+1$ choices

Lemma: $P_{ij} \leq 2/(j-i+1)$
Theorem. Expected # of comparisons is $O(n \log n)$.

\[ \sum_{1 \leq i < j \leq n} \frac{2}{j+1} = 2 \sum_{j=2}^{n} \frac{1}{j} \leq 2n \sum_{j=2}^{n} \frac{1}{j} \approx 2n \int_{x=1}^{n} \frac{1}{x} \, dx = 2n \ln n \]

Probability that $i$ and $j$ are compared

Theorem. [Knuth 1973] Stddev of number of comparisons is $\sim 0.65n$.

Ex. If $n = 1$ million, the probability that randomized quicksort takes less than $4n \ln n$ comparisons is at least 99.94%.

Chebyshev's inequality.

\[ \Pr[|X - \mu| \geq k\delta] \leq \frac{1}{k^2} \]

Closest Pair of Points

Given $n$ points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.
- Graphics, computer vision, geographic information systems
- Molecular modeling, air traffic control
- Special case of nearest neighbor, Euclidean MST, Voronoi
- Fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points $p$ and $q$ with $O(n^2)$ comparisons.

1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same $x$ coordinate.

to make presentation cleaner
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.

**Obstacle.** Impossible to ensure n/4 points in each piece.

Closest Pair of Points

**Algorithm.**
- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < \( \delta \).

Observation: only need to consider points within \( \delta \) of line L.

Sort points in \( 2\delta \)-strip by their y coordinate.

Only check distances of those within 11 positions in sorted list!
Closest Pair of Points

Def. Let \( s_i \) be the point in the \( 2\delta \)-strip, with the \( i \)th smallest y-coordinate.

Claim. If \(|i - j| \geq 12\), then the distance between \( s_i \) and \( s_j \) is at least \( \delta \).

Pf.
- No two points lie in same \( \frac{1}{2}\delta \)-by-\( \frac{1}{2}\delta \) box.
- Two points at least 2 rows apart have distance \( \geq 2(\frac{1}{2}\delta) \).

Corollary. For each point \( s_i \), we only need to check its distance to the 11 points that precede it in the y-coordinate order.

Fact. Still true if we replace 11 with 6.

Closest Pair Algorithm

```
Closest-Pair(p_1, ..., p_n) {
    Compute separation line \( L \) such that half the points are on one side and half on the other side.
    \( \delta_1 = \text{Closest-Pair(left half)} \)
    \( \delta_2 = \text{Closest-Pair(right half)} \)
    \( \delta = \min(\delta_1, \delta_2) \)
    Delete all points further than \( \delta \) from separation line \( L \)
    Sort remaining points by y-coordinate.
    Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
    return \( \delta \).
}
```

5.5 Integer Multiplication

Running time.

\[
T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)
\]

Q. Can we achieve \( O(n \log n) \)?

A. Yes. Don’t sort points in strip from scratch each time.
- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

\[
T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)
\]
**Integer Arithmetic**

Add. Given two n-digit integers a and b, compute a + b.
- \(O(n)\) bit operations.

Multiply. Given two n-digit integers a and b, compute a \(\times\) b.
- Brute force solution: \(\Theta(n^2)\) bit operations.

**Multiplying Faster**

If you analyze our usual grade school algorithm for multiplying numbers
- \(\Theta(n^2)\) time
- On real machines each "digit" is, e.g., 32 bits long but still get \(\Theta(n^2)\) running time with this algorithm when run on n-bit multiplication

We can do better!
- We’ll describe the basic ideas by multiplying polynomials rather than integers
- Advantage is we don’t get confused by worrying about carries at first

**Notes on Polynomials**

These are just formal sequences of coefficients
- when we show something multiplied by \(x^k\) it just means shifted \(k\) places to the left - basically no work

Usual polynomial multiplication

\[
\begin{array}{l}
4x^2 + 2x + 2 \\
x^2 - 3x + 1 \\
\hline
4x^2 + 2x + 2 \\
-12x^3 - 6x^2 - 6x \\
\hline
4x^4 + 2x^3 + 2x^2 \\
4x^4 - 10x^3 + 0x^2 - 4x + 2
\end{array}
\]

**Polynomial Multiplication**

Given:
- Degree \(n-1\) polynomials P and Q
  - \(P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}\)
  - \(Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-2} x^{n-2} + b_{n-1} x^{n-1}\)

Compute:
- Degree \(2n-2\) Polynomial \(P \cdot Q\)
  - \(P \cdot Q = a_0 b_0 \cdot (a_0 b_1 + a_1 b_0) \cdot x \cdot (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2 \cdot \ldots \cdot (a_{n-2} b_{n-2} + a_{n-1} b_{n-2}) x^{2n-3} + a_{n-1} b_{n-1} x^{2n-2}\)

Obvious Algorithm:
- Compute all \(a_0 b_j\) and collect terms
- \(\Theta(n^3)\) time
Naive Divide and Conquer

Assume \( n=2^k \)

- \( P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-2} x^{k-2} + a_{k-1} x^{k-1}) + (a_k + a_{k+1} x + \ldots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}) x^k \)

  \( = P_0 + P_1 x^k \) where \( P_0 \) and \( P_1 \) are degree \( k-1 \) polynomials

- Similarly \( Q = Q_0 + Q_1 x^k \)

- \( PQ = (P_0 \cdot P_1 x^k)(Q_0 + Q_1 x^k) = P_0 Q_0 + (P_1 Q_0 + P_0 Q_1) x^k + P_1 Q_1 x^{2k} \)

- 4 sub-problems of size \( k=n/2 \) plus linear combining

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

Solution \( T(n) = \Theta(n^2) \)

---

Karatsuba's Algorithm

A better way to compute the terms

- Compute
  - \( A \leftarrow P_0 Q_0 \)
  - \( B \leftarrow P_1 Q_1 \)
  - \( C \leftarrow (P_0 + P_1)(Q_0 + Q_1) = P_0 Q_0 + P_0 Q_1 + P_1 Q_0 + P_1 Q_1 \)

- Then
  - \( P_0 Q_1 + P_1 Q_0 \equiv C - A - B \)
  - So \( PQ = A + (C - A - B)x^k + Bx^{2k} \)

- 3 sub-problems of size \( n/2 \) plus \( O(n) \) work

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

\[ T(n) = \Theta(n^{\alpha}) \text{ where } \alpha = \log_2 3 = 1.59... \]

---

Multiplication

Polynomials

- Naive: \( \Theta(n^2) \)
- Karatsuba: \( \Theta(n^{\log_2 3}) \)
- Best known: \( \Theta(n \log n) \)
  - "Fast Fourier Transform"
  - FFT widely used for signal processing

Integers

- Similar, but some ugly details re: carries, etc. gives \( \Theta(n \log n \log \log n) \),
  - mostly unused in practice except for symbolic manipulation systems like Maple

---

Matrix Multiplication

---
Multiplying Matrices

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} & a_{14} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & a_{34} \\
    a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix}
\begin{bmatrix}
    b_{11} & b_{12} & b_{13} & b_{14} \\
    b_{21} & b_{22} & b_{23} & b_{24} \\
    b_{31} & b_{32} & b_{33} & b_{34} \\
    b_{41} & b_{42} & b_{43} & b_{44}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
    a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} \\
    a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \\
    a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} + a_{34}b_{43} \\
    a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44}
\end{bmatrix}
\]

for \(i=1\) to \(n\)

\[
\text{endfor}
\]

for \(j=1\) to \(n\)

\[
C[i,j] \leftarrow 0
\]

\[
\text{endfor}
\]

for \(k=1\) to \(n\)

\[
C[i,j] = C[i,j] + A[i,k] \cdot B[k,j]
\]

\[
\text{endfor}
\]

\[
\text{endfor}
\]

\(n^3\) multiplications, \(n^3\) additions
Multiplying Matrices

\[
\begin{bmatrix}
  a_{11} & a_{12} \\
  a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
  b_{11} & b_{12} \\
  b_{21} & b_{22}
\end{bmatrix} =
\begin{bmatrix}
  a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\
  a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22}
\end{bmatrix}
\]

Simple Divide and Conquer

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

\[
\frac{5}{2} > 2 \quad \text{so} \quad T(n) \text{ is } \Theta(n^{\log_2 8}) = \Theta(n^2)
\]

Strassen's Divide and Conquer Algorithm

Strassen's algorithm

- Multiply 2x2 matrices using 7 instead of 8 multiplications (and lots more than 4 additions)

- \( T(n) = 7T(n/2) + cn^2 - 7 \cdot 2^2 \) so \( T(n) \) is \( O(n^{\log_2 7}) \) which is \( O(n^{2.81}) \)

- Fastest algorithms theoretically use \( O(n^{2.376}) \) time

- Not practical but Strassen's is practical provided calculations are exact and we stop recursion when matrix has size about 100 (maybe 10)

The algorithm

\[
P_1 \leftarrow A_{12}(B_{11} + B_{21}) ; \quad P_2 \leftarrow A_{21}(B_{12} + B_{22})
\]

\[
P_3 \leftarrow (A_{11} - A_{12})B_{11} ; \quad P_4 \leftarrow (A_{22} - A_{21})B_{22}
\]

\[
P_5 \leftarrow (A_{22} - A_{12})(B_{21} - B_{22}) ; \quad P_6 \leftarrow (A_{11} - A_{21})(B_{12} - B_{11})
\]

\[
P_7 \leftarrow (A_{21} - A_{12})(B_{11} + B_{22})
\]

7 multiplications.
18 = 10 + 8 additions (or subtractions).

\[
C_{11} \leftarrow P_1 + P_3 ; \quad C_{12} \leftarrow P_2 + P_3 + P_6 + P_7
\]

\[
C_{21} \leftarrow P_1 + P_4 + P_5 + P_7 ; \quad C_{22} \leftarrow P_2 + P_4
\]
Fast Matrix Multiplication in Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n \approx 128$.

Common misperception: "Strassen is only a theoretical curiosity."
- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \approx 2,500$.
- Range of instances where it’s useful is a subject of controversy.

Remark. Can “Strassenize” $Ax=b$, determinant, eigenvalues, and other matrix ops.

Fast Matrix Multiplication in Theory

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

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] $\Theta(n^2) = O(n^{2.376})$

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. $\Theta(n^6) = O(n^{2.373})$

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] $\Theta(n^{3.62} \log n) = O(n^{2.80})$

Decimal wars.
- December, 1979: $O(n^{2.521813})$.
- January, 1980: $O(n^{2.521801})$.

Fast Matrix Multiplication in Theory


Best known. $O(n^{2.373})$ [V. Williams, Nov 2011]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. not practical but Strassen’s is practical provided calculations are exact and we stop recursion when matrix has size about 100 (maybe 10).