Divide and Conquer
Reading: 5.1, 5.4-5.5, 13.5

Some of the slides were Adapted from Paul Beame
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 linear time.

Consequence.

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

Divide et impera.
Veni, vidi, vici.
- *Julius Caesar*
Binary search for roots (bisection method)

Given:
- continuous function $f$ and two points $a < b$ with $f(a) \leq 0$ and $f(b) > 0$

Find:
- approximation to $c$ s.t. $f(c) = 0$ and $a \leq c < b$
Bisection method

Bisection(a, b, ε)
if (a-b) < ε then
    return(a)
else
    c ← (a+b)/2
    if f(c) ≤ 0 then
        return(Bisection(c, b, ε))
    else
        return(Bisection(a, c, ε))

Time Analysis:
At each step we halved the size of the interval
It started at size b-a
It ended at size ε

# of calls to f is log₂((b-a)/ε)
Old favorites

Binary search

- One subproblem of half size plus one comparison

  Recurrence \( T(n) = T(\lceil n/2 \rceil) + 1 \) for \( n \geq 2 \)
  
  \[ T(1) = 0 \]

  So \( T(n) \) is \( \lceil \log_2 n \rceil + 1 \)

Mergesort

- Two subproblems of half size plus merge cost of \( n-1 \) comparisons

  Recurrence \( T(n) \leq 2T(\lceil n/2 \rceil) + n-1 \) 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

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

- \( T(n/2) \)
- \( T(n/4) \)
- \( T(n/4) \)
- \( T(n/4) \)
- \( T(n/2) \)
- \( T(n/2) \)
- \( T(2) \)
- \( T(2) \)
- \( T(2) \)
- \( T(2) \)
- \( T(2) \)
- \( T(2) \)

\( n \)
\( 2(n/2) \)
\( 4(n/4) \)
\( \ldots \)
\( 2^k (n / 2^k) \)
\( \ldots \)
\( n/2 \)
\( n \log_2 n \)
Proof by Telescoping

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

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

assumes $n$ is a power of 2

Pf. For $n > 1$:

$$
\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1
$$

$$
= \frac{T(n/2)}{n/2} + 1 + 1
$$

$$
= \frac{T(n/4)}{n/4} + 1 + \ldots + 1
$$

$$
= \log_2 n
$$
Proof by Induction

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

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

assumes \( n \) is a power of 2

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

- **Base case:** \( n = 1 \).
- **Inductive hypothesis:** \( T(n) = n \log_2 n \).
- **Goal:** show that \( T(2n) = 2n \log_2 (2n) \).

\[
egin{align*}
T(2n) &= 2T(n) + 2n \\
&= 2n \log_2 n + 2n \\
&= 2n \log_2 (2n) - 1 + 2n \\
&= 2n \log_2 (2n)
\end{align*}
\]
Analysis of Mergesort Recurrence

Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \log_2 n \rceil \).

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T(\lceil n/2 \rceil) & \text{solve left half} \\
T(\lfloor n/2 \rfloor) & \text{solve right half} \\
n & \text{merging}
\end{cases}
\]

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

\[
T(n) \leq T(n_1) + T(n_2) + n
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lfloor \log_2 n_2 \rfloor + n
\leq n \lceil \log_2 n \rceil + n
\leq n(\lceil \log_2 n \rceil - 1) + n
= n \lceil \log_2 n \rceil
\]

\[
n_2 = \lfloor n/2 \rfloor \\
\leq 2^{\lceil \log_2 n \rceil} / 2\\n\Rightarrow \log_2 n_2 \leq \lceil \log_2 n \rceil - 1
\]
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 $\Theta(n^{\log_b a})$
- 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 $n/b$. 

Master Divide and Conquer Recurrence
Proving Master recurrence

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

\( T(1) = c \)
Proving Master recurrence

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

Problem size

\[ T(1) = c \]
Problem size | \( T(n) = a \cdot T(n/b) + c \cdot n^k \) | # probs | cost
---|---|---|---
\( n \) | 1 | \( c \cdot n^k \)
\( n/b \) | \( a \) | \( c \cdot a \cdot n^{k/b^k} \)
\( n/b^2 \) | \( a^2 \) | \( c \cdot a^2 \cdot n^{k/b^{2k}} \)
\( b \) | \( a^d \) | \( c \cdot n^k (a/b^k)^d \)
\( 1 \) | \( c \cdot a^d \)
Geometric Series

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

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

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

so \[ S = \frac{t(r^n -1)}{(r-1)} \text{ if } r \neq 1. \]

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

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

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

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

If $a/b^k>1$
- last term is largest $T(n)$ is $\Theta(a^d) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$
  (To see this take $\log_b$ of both sides)
13.5 Median Finding and Quicksort
Order problems: Find the $k^{th}$ largest

Runtime models
- Machine Instructions
- Comparisons

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

$2^{nd}$ Largest
- $O(n)$ time
- ? Comparisons

$k^{th}$ 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 $k$-th largest from a list of length $n$
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
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 $k$-th largest of $n$ to the $j$-th largest of $\alpha n$, for $\alpha < 1$

**QSelect($k$, $S$)**

Choose element $x$ from $S$

$S_L = \{y \in S \mid y < x \}$

$S_E = \{y \in S \mid y = x \}$

$S_G = \{y \in S \mid y > x \}$

if $|S_L| \geq k$

    return QSelect($k$, $S_L$)

else if $|S_L| + |S_E| \geq k$

    return $y$ in $S_E$

else

    return QSelect($k - |S_L| - |S_E|$, $S_G$)
“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 $\text{QSelect}(k, S)$, and let $S’$ be the elements passed to the recursive call.

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

$\Rightarrow$ On average only 2 recursive calls before the size of $S’$ is at most $3n/4$

elements of $S$ listed in sorted order
Given $x$, one pass over $S$ to determine $S_L$, $S_E$, and $S_G$ 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 $3/4 \times n$

- BFPRT (Blum-Floyd-Pratt-Rivest-Tarjan) Algorithm
Quicksort

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

```c
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.

$\uparrow$

$O(\log n)$ extra space
Quicksort

Running time.

- **[Best case.]** Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons.
- **[Worst case.]** Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons.

**Randomize.** Protect against worst case by choosing splitter at random.

**Intuition.** If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $\Theta(n \log n)$ comparisons.

**Notation.** Label elements so that $x_1 < x_2 < \ldots < x_n$. 
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: \(\sum_{i<j} p_{ij}\)

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 $O(n \log n)$.

Pf.

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

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.** $\Pr[|X - \mu| \geq k\delta] \leq 1 / k^2$. 
5.4 Closest Pair of Points
Closest Pair of Points

Closest pair. 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.

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

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

Assumption. No two points have same x coordinate.

fast closest pair inspired fast algorithms for these problems
Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.
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.
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.
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. $\leftarrow$ seems like $\Theta(n^2)$
- Return best of 3 solutions.
Closest Pair of Points

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

$\delta = \min(12, 21)$
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 $L$. 

\[ \delta = \min(12, 21) \]
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.
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!

\[ \delta = \min(12, 21) \]
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 precedes it in the $y$-coordinate order.

**Fact.** Still true if we replace 11 with 6.
Closest Pair Algorithm

\[ \text{Closest-Pair}(p_1, \ldots, p_n) \] 

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 \( \delta \).
Closest Pair of Points: Analysis

Running time.

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

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.

\[ T(n) \leq 2T(n/2) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n) \]
5.5 Integer Multiplication
**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.

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

```
  1 1 0 1 0 1 0 1
*  0 1 1 1 1 1 0 1
  ---------------
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  0 1 1 0 1 0 0 0 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
  1 1 0 1 0 1 0 1 0
```
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

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

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

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

Obvious Algorithm:
- Compute all \( a_i b_j \) and collect terms
- \( \Theta(n^2) \) time
Naive Divide and Conquer

Assume $n=2^k$

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

  \[ P = P_0 + P_1 x^k \]  
  where $P_0$ and $P_1$ are degree $k-1$ polynomials

- Similarly $Q = Q_0 + Q_1 x^k$

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

- 4 sub-problems of size $k=n/2$ plus linear combining

  \[ T(n)=4 \cdot T(n/2)+cn \]  
  Solution \[ T(n) = \Theta(n^2) \]
Karatsuba’s Algorithm

A better way to compute the terms

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

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

- 3 sub-problems of size $n/2$ plus $O(n)$ work
  - $T(n) = 3 \ T(n/2) + cn$
  - $T(n) = O(n^\alpha)$ where $\alpha = \log_2 3 = 1.59...$
Karatsuba’s algorithm and evaluation and interpolation

Karatsuba’s algorithm can be thought of as a way of multiplying degree 1 polynomials (which have 2 coefficients) using fewer multiplications

- $PQ = (P_0+P_1z)(Q_0+Q_1z)$
  $= P_0Q_0 + (P_1Q_0+P_0Q_1)z + P_1Q_1z^2$

- Evaluate at 0,1,-1 (Could also use other points)
  - $A = P(0)Q(0) = P_0Q_0$
  - $C = P(1)Q(1) = (P_0+P_1)(Q_0+Q_1)$
  - $D = P(-1)Q(-1) = (P_0-P_1)(Q_0-Q_1)$
Multiplication

Polynomials

- Naïve: $\Theta(n^2)$
- Karatsuba: $\Theta(n^{1.59...})$
- 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

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

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

for $i=1$ to $n$
    for $j=1$ to $n$
        $C[i,j] \leftarrow 0$
        for $k=1$ to $n$
            $C[i,j] = C[i,j] + A[i,k] \cdot B[k,j]$
        endfor
    endfor
endfor

$n^3$ multiplications, $n^3-n^2$ additions
Multiplying Matrices

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

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

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

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

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

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

Let's consider multiplying two matrices $A$ and $B$. The result of multiplying two matrices $A$ and $B$ is given by the formula:

$$
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\cdot
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
$$

The result of multiplying these matrices is another matrix:

$$
\begin{bmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}
$$

This formula shows how each element of the resulting matrix is obtained by taking the dot product of the corresponding row of the first matrix and the corresponding column of the second matrix.

The diagram illustrates this process with the matrices:

- $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$
- $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$

The result of the multiplication is:

$$
\begin{bmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}
$$

The elements of the resulting matrix are:

- $(1,1): a_{11}b_{11} + a_{12}b_{21}$
- $(1,2): a_{11}b_{12} + a_{12}b_{22}$
- $(2,1): a_{21}b_{11} + a_{22}b_{21}$
- $(2,2): a_{21}b_{12} + a_{22}b_{22}$

This illustrates how each element of the resulting matrix is computed by multiplying the corresponding row of the first matrix with the corresponding column of the second matrix and summing the products.
Simple Divide and Conquer

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
B_{11} \\
B_{21}
\end{pmatrix}
=
\begin{pmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{pmatrix}
\]

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

- \(8 > 2^2\) so \(T(n)\) is \(\Theta(n^{\log_2 8}) = \Theta(n^3)\)
Strassen’s Divide and Conquer Algorithm

Strassen’s algorithm

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

\[
T(n) = 7 \cdot T(n/2) + cn^2
\]

- \(7 > 2^2\) so \(T(n)\) is \(\Theta(n^{\log_2 7})\) which is \(O(n^{2.81\ldots})\)

- Fastest algorithms theoretically use \(O(n^{2.373})\) 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

\( P_1 \leftarrow A_{12}(B_{11} + B_{21}) \); \hspace{1cm} \( P_2 \leftarrow A_{21}(B_{12} + B_{22}) \)

\( P_3 \leftarrow (A_{11} - A_{12})B_{11} \); \hspace{1cm} \( P_4 \leftarrow (A_{22} - A_{21})B_{22} \)

\( P_5 \leftarrow (A_{22} - A_{12})(B_{21} - B_{22}) \)

\( P_6 \leftarrow (A_{11} - A_{21})(B_{12} - B_{11}) \)

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

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

\( C_{11} \leftarrow P_1 + P_3 \); \hspace{1cm} \( C_{12} \leftarrow P_2 + P_3 + P_6 - P_7 \)

\( C_{21} \leftarrow P_1 + P_4 + P_5 + P_7 \); \hspace{1cm} \( C_{22} \leftarrow P_2 + P_4 \)
Fast Matrix Multiplication in Practice

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

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

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \( \Theta(n^{\log_3 21}) = O(n^{2.77}) \)

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)