Announcement: Homework 2 due tonight at 11:59PM
Recap: Divide and Conquer

Recursive Approach:
1. Divide input into smaller parts (Divide)
2. Solve each smaller instance (Conquer)
3. Combine solutions from each smaller instance (Merge)

Example: Merge Sort (Sort list of $n$ items in $O(n \log n)$ time)
1. Divide array into two equal size parts ($n/2$)
2. Sort each sub-array (Conquer)
3. Merge the each sub-array to obtain the sorted list

Recurrence Relationships
- Useful to express the running time of recursive algorithm
- Analyzing a Recurrence: Unrolling, Telescoping, Induction,…
- Master's Theorem ($T(n) = a \cdot T(n/b) + n^c$)
5.3 Counting Inversions
Counting Inversions

Music site tries to match your song preferences with others.
- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.
- My rank: 1, 2, ..., n.
- Your rank: a_1, a_2, ..., a_n.
- Songs i and j inverted if i < j, but a_i > a_j.

<table>
<thead>
<tr>
<th>Songs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Brute force: check all $\Theta(n^2)$ pairs i and j.

Inversions
3-2, 4-2
Applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

  
  ![List divided into two pieces](image)

  Divide: \(O(1)\).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** separate list into two pieces.
- **Conquer:** recursively count inversions in each half.

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 \\
\end{array}
\begin{array}{cccccccc}
12 & 11 & 3 & 7 & & & & \\
\end{array}
\]

- **Divide:** $O(1)$.

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 \\
\end{array}
\begin{array}{cccccccc}
12 & 11 & 3 & 7 & & & & \\
\end{array}
\]

5 blue-blue inversions 8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2 6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

- **Conquer:** $2T(n/2)$.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where \( a_i \) and \( a_j \) are in different halves, and return sum of three quantities.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

5 blue-blue inversions
8 green-green inversions
9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Divide: \( O(1) \).
Conquer: \( 2T(n / 2) \)
combine: ???

Total = 5 + 8 + 9 = 22.
Counting Inversions: Combine

**Combine**: count blue-green inversions
- Assume each half is **sorted**.
- Count inversions where \( a_i \) and \( a_j \) are in different halves.
- **Merge** two sorted halves into sorted whole.

Count: \( O(n) \)

Merge: \( O(n) \)

\[
T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    (r_A, A) ← Sort-and-Count(A)
    (r_B, B) ← Sort-and-Count(B)
    (r, L) ← Merge-and-Count(A, B)

    return r = r_A + r_B + r and the sorted list L
}
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.

↑
to make presentation cleaner
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. ✗ 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 < $\delta$.
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!

$\delta = \min(12, 21)$
**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 the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. □

**Fact.** Still true if we replace 12 with 7.
Closest-Pair Algorithm

Closest-Pair(p₁, ..., pₙ) {
    \textbf{Compute} separation line L such that half the points are on one side and half on the other side.

    \[ \delta_1 = \text{Closest-Pair(left half)} \]
    \[ \delta_2 = \text{Closest-Pair(right half)} \]
    \[ \delta = \min(\delta_1, \delta_2) \]

    \textbf{Delete} all points further than \( \delta \) from separation line L

    \textbf{Sort} remaining points by y-coordinate.

    \textbf{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 \( \delta \).

    \textbf{return} \( \delta \).
}
Closest Pair of Points: Analysis

Running time.

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

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.

\[ T(n) \leq 2T(n/2) + O(n) \implies T(n) = O(n \log n) \]
integers, matrices, and polynomials
5.5 Integer Multiplication
Motivation: Complex Multiplication

**Complex multiplication.** \((a + bi) (c + di) = x + yi.\)

**Grade-school.** \(x = ac - bd, \ y = bc + ad.\)

4 multiplications, 2 additions

**Q.** Is it possible to do with fewer multiplications?

Our Prices Are Fantastic!

Multiplication: $100 (reals only \(\mathbb{R}\))
Addition: $1 (reals only \(\mathbb{R}\))

$402 for Grade-School Approach: 4 multiplications, 2 additions
Complex Multiplication

**Complex multiplication.** \((a + bi) (c + di) = x + yi.\)

**Grade-school.** \(x = ac - bd, \ y = bc + ad.\)

\[ \text{4 multiplications, 2 additions} \]

\[ \text{3 multiplications, 5 additions ($305$)} \]

**Q.** Is it possible to do with fewer multiplications?  
**A.** Yes. [Gauss] \(x = ac - bd, \ y = (a + b) (c + d) - ac - bd.\)

**Remark.** Improvement if no hardware multiply.
Addition. Given two $n$-bit integers $x$ and $y$, compute $x + y$.

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

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Remark. Grade-school addition algorithm is optimal.
**Integer Multiplication**

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

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

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\times & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

**Q.** Is grade-school multiplication algorithm optimal?
Divide-and-Conquer Multiplication: Warmup

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

- Multiply four \( \frac{1}{2}n \)-bit integers, recursively.
- Add and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0 \\
y = 2^{n/2} \cdot y_1 + y_0 \\
x y = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) \\
= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_0 y_1 + x_1 y_0) + x_0 y_0
\]

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

\[
\begin{array}{c c c c}
1 & 2 & 3 & 4 \\
\hline
x & y \\
x_1 & x_0 & y_1 & y_0 \\
\end{array}
\]

\[
T(n) = 4T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^2)
\]
Divide-and-Conquer Multiplication: Warmup

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

- Multiply four \( \frac{1}{2}n \)-bit integers, recursively.
- Add and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0
\]
\[
y = 2^{n/2} \cdot y_1 + y_0
\]
\[
xy = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0)
\]
\[
= 2^n \cdot x_1y_1 + 2^{\frac{n}{2}} \cdot (x_0y_1 + x_1y_0) + x_0y_0
\]

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

\( x_1 \quad x_0 \quad y_1 \quad y_0 \)

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

Master’s Theorem: \( a = 4, b=2, c=1 \) \( \left( \frac{a}{b^c} \right) > 1, \ O\left(n^{\log_b a}\right) = O(n^2) \)
Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 0 \\
4T(n/2) + n & \text{otherwise}
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\lfloor \log n \rfloor} n \cdot 2^k = n \left( \frac{2^{1+\lfloor \log n \rfloor} - 1}{2 - 1} \right) = 2n^2 - n \]
Karatsuba Multiplication

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

- Add two \( \frac{1}{2}n \) bit integers.
- Multiply three \( \frac{1}{2}n \)-bit integers, recursively.
- Add, subtract, and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0
\]
\[
y = 2^{n/2} \cdot y_1 + y_0
\]

\[
xy = 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot (x_0 y_1 + x_1 y_0) + x_0 y_0
\]
\[
= 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot ((x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1) + x_0 y_0
\]
Karatsuba Multiplication

To multiply two $n$-bit integers $x$ and $y$:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0
\]
\[
y = 2^{n/2} \cdot y_1 + y_0
\]

\[
xy = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_0 y_1 + x_1 y_0) + x_0 y_0
\]
\[
= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot ((x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1) + x_0 y_0
\]
\[1 \quad 2 \quad 3 \quad 1 \quad 3\]

**Theorem.** [Karatsuba-Ofman 1962] Can multiply two $n$-bit integers in $O(n^{1.585})$ bit operations.

\[
T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + T\left(1+\left\lceil \frac{n}{2} \right\rceil \right) + \Theta(n) \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})
\]

**Master's Theorem:** $a = 3$, $b=2$, $c=1$ \( \left(\frac{a}{b^c}\right) > 1 \)
Karatsuba: Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 0 \\
3T(n/2) + n & \text{otherwise} 
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\log n} n \left(\frac{3}{2}\right)^k = n \left(\frac{\left(\frac{3}{2}\right)^{\log n} - 1}{\frac{3}{2} - 1}\right) = 3n^{\log 3} - 2n \]
**Integer division.** Given two $n$-bit (or less) integers $s$ and $t$, compute quotient $q = \lfloor s / t \rfloor$ and remainder $r = s \mod t$ (such that $s=qt+r$).

**Fact.** Complexity of integer division is (almost) same as integer multiplication.

To compute quotient $q$: $x_{i+1} = 2x_i - tx_i^2 \quad$ using fast multiplication

- Approximate $x = 1 / t$ using Newton's method:
- After $i=\log n$ iterations, either $q = \lfloor s x_i \rfloor$ or $q = \lceil s x_i \rceil$.
  - If $\lfloor s x \rfloor t > s$ then $q = \lceil s x \rceil$ (1 multiplication)
  - Otherwise $q = \lfloor s x \rfloor$
  - $r=s-qt$ (1 multiplication)

- **Total:** $O(\log n)$ multiplications and subtractions
Toom-3 Generalization

Split into 3 parts

\[
\begin{align*}
a &= 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0 \\
b &= 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0
\end{align*}
\]

Requires: 5 multiplications of n/3 bit numbers and O(1) additions, shifts

\[
T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_3 5}\right)
\]

\[
\approx 1.465
\]

Toom-Cook Generalization (split into k parts):

\[
\begin{align*}
a &= 2^{\frac{n(k-1)}{k}} \cdot a_{k-1} + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0 \\
b &= 2^{\frac{n(k-1)}{k}} \cdot a_k + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0
\end{align*}
\]

\[
T(n) = (2k - 1) \cdot T\left(\frac{n}{k}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_k (2k-1)}\right)
\]

\[
\lim_{k \to \infty} (\log_k (2k - 1)) = 1
\]
Toom-3 Generalization

Split into 3 parts

\[ a = 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0 \]
\[ b = 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0 \]

**Requires:** 5 multiplications of n/3 bit numbers and \( O(1) \) additions, shifts

\[ T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O(n^{\log_3 5}) \]
\[ \approx 1.465 \]

**Schönhage-Strassen algorithm**

\[ T(n) \in O(n \log n \log \log n) \]

**Only used for really big numbers:** \( a > 2^{2^{15}} \)

**State of the Art:** \( O(n \log n \ g(n)) \) for increasing small

\[ g(n) \ll \log \log n \]
Matrix Multiplication
Dot Product

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

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

\[
a \cdot b = \sum_{i=1}^{n} a_i b_i
\]

\[
\begin{align*}
a & = [.70 \  .20 \  .10 ] \\
b & = [.30 \  .40 \  .30 ] \\
a \cdot b & = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32
\end{align*}
\]

Remark. Grade-school dot product algorithm is optimal.
Matrix multiplication. Given two \( n \)-by-\( n \) matrices \( A \) and \( B \), compute \( C = AB \).

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

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

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

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

Q. Is grade-school matrix multiplication algorithm optimal?
Block Matrix Multiplication

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

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

\[
= \begin{bmatrix}
0 & 1 \\
4 & 5
\end{bmatrix}
\times
\begin{bmatrix}
16 & 17 \\
20 & 21
\end{bmatrix}
+ \begin{bmatrix}
2 & 3 \\
6 & 7
\end{bmatrix}
\times
\begin{bmatrix}
24 & 25 \\
28 & 29
\end{bmatrix}
\]

\[
= \begin{bmatrix}
152 & 158 \\
504 & 526
\end{bmatrix}
\]
Matrix Multiplication: Warmup

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

- **Divide:** partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- **Conquer:** multiply 8 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- **Combine:** add appropriate products using 4 matrix additions.

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

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

\[
T(n) = 8T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^3)
\]
Fast Matrix Multiplication

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

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

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

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

- 7 multiplications.
- 18 = 8 + 10 additions and subtractions.
Fast Matrix Multiplication

To multiply two \( n \)-by-\( n \) matrices \( A \) and \( B \): [Strassen 1969]
- Divide: partition \( A \) and \( B \) into \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) blocks.
- Compute: 14 \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.
- \( T(n) = \# \) arithmetic operations.

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

- Apply Master Theorem (\( a=7, b=2, c=2 \))
  \[
  - \left(\frac{a}{bc}\right) = \frac{7}{4} > 1 \quad \Rightarrow \quad T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.81})
  \]
Fast Matrix Multiplication: 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.”
- Apple 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, SVD, ....
Fast Matrix Multiplication: Theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969] \[ \Theta(n^\log_2 7) = O(n^{2.807}) \]

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

Q. Two 3-by-3 matrices with 21 scalar multiplications?
A. Also impossible. \[ \Theta(n^\log_2 21) = O(n^{2.77}) \]

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. \[ O(n^{2.7799}) \]
- December, 1979. \[ O(n^{2.521813}) \]
- January, 1980. \[ O(n^{2.521801}) \]
Fast Matrix Multiplication: Theory

Best known. \( O(n^{2.376}) \) [Coppersmith-Winograd, 1987]

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

Caveat. Theoretical improvements to Strassen are progressively less practical.
Fast Matrix Multiplication: Theory

**Best known.** $O(n^{2.373})$ [Williams, 2014]

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

**Caveat.** Theoretical improvements to Strassen are progressively less practical.
Fast Matrix Multiplication: Theory

**Best known.** $O(n^{2.3729})$ [Le Gall, 2014]

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

**Caveat.** Theoretical improvements to Strassen are progressively less practical.
Extra Slides