A Divide & Conquer Example: Closest Pair of Points
Given \( n \) points on the real line, find the closest pair

Closest pair is *adjacent* in ordered list

Time \( O(n \log n) \) to sort, if needed

Plus \( O(n) \) to scan adjacent pairs
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 \( \Theta(n^2) \) time.

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

Assumption. No two points have same \( x \) coordinate.

\[ \uparrow \]

Just to simplify presentation
Algorithm.
Divide: draw vertical line $L$ with $\approx n/2$ points on each side.
Algorithm.

Divide: draw vertical line $L$ with $\approx \frac{n}{2}$ points on each side.

Conquer: find closest pair on each side, recursively.
Algorithm.

Divide: draw vertical line \( L \) with \( \approx \frac{n}{2} \) points on each side.

Conquer: find closest pair on each side, recursively.

Combine to find closest pair overall

Return
Find closest pair with one point in each side, \textit{assuming distance} < \delta.
Find closest pair with one point in each side, assuming distance $< \delta$.

Observation: suffices to consider points within $\delta$ of line $L$.  

$$\delta = \min(12, 21)$$
Find closest pair with one point in each side, assuming distance < $\delta$.

Observation: suffices to consider points within $\delta$ of line $L$.
Almost the one-D problem again: Sort points in $2\delta$-strip by their $y$ coordinate.
Find closest pair with one point in each side, *assuming* distance $< \delta$.

Observation: suffices to consider points within $d$ of line $L$. Almost the one-D problem again: Sort points in 2d-strip by their $y$ coordinate. Only check pts within $11$ in sorted list!
Claim: No two points lie in the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
**Claim**: No two points lie in the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.

**Pf**: Such points would be within

$$\delta \sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \delta \sqrt{\frac{1}{2}} = \delta \frac{\sqrt{2}}{2} \approx 0.7\delta < \delta$$
**Claim:** No two points lie in the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.

**Pf:** Such points would be within

$$
\delta \sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \delta \sqrt{\frac{1}{2}} = \delta \frac{\sqrt{2}}{2} \approx 0.7\delta < \delta
$$

**Def.** Let $s_i$ have the $i^{th}$ smallest $y$-coordinate among points in the $2\delta$-width-strip.

**Claim:** If $|i - j| > 11$, then the distance between $s_i$ and $s_j$ is $> \delta$. 
Claim: No two points lie in the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.

Pf: Such points would be within

$$\delta \sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \delta \sqrt{\frac{1}{2}} = \delta \frac{\sqrt{2}}{2} \approx 0.7\delta < \delta$$

Def. Let $s_i$ have the $i^{th}$ smallest $y$-coordinate among points in the $2\delta$-width-strip.

Claim: If $|i - j| > 11$, then the distance between $s_i$ and $s_j$ is $> \delta$.

Pf: only 11 boxes within $+\delta$ of $y(s_i)$. 
Closest-Pair(p₁, ..., pₙ) {
    if(n <= ??) return ??

    Compute separation line L such that half the points are on one side and half on the other side.

    δ₁ = Closest-Pair(left half)
    δ₂ = Closest-Pair(right half)
    δ  = min(δ₁, δ₂)

    Delete all points further than δ from separation line L

    Sort remaining points p[1]...p[m] by y-coordinate.

    for i = 1..m
        for k = 1...11
            if i+k <= m
                δ = min(δ, distance(p[i], p[i+k]));

    return δ.
}
Analysis, I: Let $D(n)$ be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on $n > 1$ points

$$
D(n) \leq \begin{cases} 
0 & n = 1 \\
2D(n/2) + 11n & n > 1
\end{cases} \implies D(n) = O(n \log n)
$$

BUT – that’s only the number of distance calculations

What if we counted running time?
Analysis, II: Let $T(n)$ be the running time in the Closest-Pair Algorithm when run on $n > 1$ points

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + O(n \log n) & \text{if } n > 1 \end{cases} \Rightarrow T(n) = O(n \log^2 n).$$

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

A. Yes. Don't sort points from scratch each time.
   Sort by $x$ at top level only.
   Each recursive call returns $\delta$ and list of all points sorted by $y$
   Sort by merging two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$
Recurrences

Applications:
  multiplying numbers
  multiplying matrices
  computing medians
Idea:

“Two halves are better than a whole”

if the base algorithm has super-linear complexity.

“If a little's good, then more's better”

repeat above, recursively

Applications: Many.

Binary Search, Merge Sort, (Quicksort), Closest points, Integer multiply,…
Recurrences

Above: Where they come from, how to find them

Next: how to solve them
\[ T(n) = aT(n/b) + cn^d \text{ then} \]

\[ a > b^d \Rightarrow T(n) = \Theta(n^{\log_b a}) \quad \text{[many subprobs \(
Rightarrow\) leaves dominate]} \]

\[ a < b^d \Rightarrow T(n) = \Theta(n^d) \quad \text{[few subprobs \(
Rightarrow\) top level dominates]} \]

\[ a = b^d \Rightarrow T(n) = \Theta(n^d \log n) \quad \text{[balanced \(
Rightarrow\) all log n levels contribute]} \]

Fine print:
\[ a \geq 1; \ b > 1; \ c, \ d \geq 0; \ T(1) = c; \]
\[ a, \ b, \ k, \ t \text{ integers.} \]
Solve: \[ T(n) = a \cdot T(n/b) + cn^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Num</th>
<th>Size</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( l = a^0 )</td>
<td>( n )</td>
<td>( cn^d )</td>
</tr>
<tr>
<td>1</td>
<td>( a^1 )</td>
<td>( n/b )</td>
<td>( ac(n/b)^d )</td>
</tr>
<tr>
<td>2</td>
<td>( a^2 )</td>
<td>( n/b^2 )</td>
<td>( a^2c(n/b^2)^d )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( i )</td>
<td>( a^i )</td>
<td>( n/b^i )</td>
<td>( a^i \cdot c(n/b^i)^d )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( k-1 )</td>
<td>( a^{k-1} )</td>
<td>( n/b^{k-1} )</td>
<td>( a^{k-1}c(n/b^{k-1})^d )</td>
</tr>
<tr>
<td>( k )</td>
<td>( a^k )</td>
<td>( n/b^k = 1 )</td>
<td>( a^k \cdot T(1) )</td>
</tr>
</tbody>
</table>

\( n = b^k \); \( k = \log_b n \)

Total Work: \[ \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d \] (add last col)
Theorem:

\[ 1 + x + x^2 + x^3 + \ldots + x^k = \frac{x^{k+1} - 1}{x - 1} \]

proof:

\[ S = 1 + x + x^2 + x^3 + \ldots + x^k \]
\[ xS = x + x^2 + x^3 + \ldots + x^k + x^{k+1} \]
\[ xS - S = x^{k+1} - 1 \]
\[ S(x - 1) = x^{k+1} - 1 \]
\[ S = \frac{x^{k+1} - 1}{x - 1} \]
\[ T(1) = d \]
\[ T(n) = a \cdot T(n/b) + cn^d, \quad a > b^d \]

\[
T(n) = \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d
\]

\[
= cn^d \sum_{i=0}^{\log_b n} (a/b^d)^i
\]

\[
= cn^d \left( \frac{a}{b^d} \right)^{\log_b n+1} - 1
\]

\[
\sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1} \quad (x \neq 1)
\]
Solve: \[ T(1) = d \]
\[ T(n) = a \ T(n/b) + c \ n^d \quad , \ a > b^d \]

\[
\begin{align*}
    c n^d \left( \frac{a}{b^d} \right)^{\log_b n + 1} & - 1 < c n^d \left( \frac{a}{b^d} \right)^{\log_b n + 1} \\
    &= c \left( \frac{n^d}{b^{d \log_b n}} \right) \left( \frac{a}{b^d} \right) a^{\log_b n} \\
    &= c \left( \frac{a}{b^d} \right) a^{\log_b n} \\
    &= O(n^{\log_b a})
\end{align*}
\]
Solve:

\[ T(1) = d \]
\[ T(n) = a \ T(n/b) + cn^d \quad , \quad a < b^d \]

\[
T(n) = \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d \\
= cn^d \sum_{i=0}^{\log_b n} a^i / b^{id} \\
= \frac{cn^d}{1 - \left(\frac{a}{b^d}\right)^{\log_b n+1}} \\
< cn^d \frac{1}{1 - \left(\frac{a}{b^d}\right)} \\
= O(n^d)
\]
Solve:

\[ T(1) = d \]

\[ T(n) = a \cdot T(n/b) + cn^d, \quad a = b^d \]

\[
T(n) = \sum_{i=0}^{\log_b{n}} a^i c(n / b^i)^d \\
= cn^d \sum_{i=0}^{\log_b{n}} a^i / b^{id} \\
= O(n^d \log_b{n})
\]
divide and conquer – master recurrence

\[ T(n) = aT(n/b) + cn^d \text{ for } n > b \text{ then} \]

- \( a > b^d \implies T(n) = \Theta(n^{\log_b a}) \) [many subprobs \( \rightarrow \) leaves dominate]
- \( a < b^d \implies T(n) = \Theta(n^d) \) [few subprobs \( \rightarrow \) top level dominates]
- \( a = b^d \implies T(n) = \Theta(n^d \log n) \) [balanced \( \rightarrow \) all \( \log n \) levels contribute]

Fine print:
- \( a \geq 1; b > 1; c, d \geq 0; T(1) = c; \)
- \( a, b, k, t \) integers.
Integer Multiplication
Add. Given two n-bit integers $a$ and $b$, compute $a + b$. 

$O(n)$ bit operations.
integer arithmetic

Add. Given two n-bit integers a and b, compute a + b.

O(n) bit operations.

Multiply. Given two n-bit integers a and b, compute a \times b.
The “grade school” method:
Add. Given two n-bit integers $a$ and $b$, compute $a + b$.

$O(n)$ bit operations.

Multiply. Given two n-bit integers $a$ and $b$, compute $a \times b$.

The “grade school” method:

$\Theta(n^2)$ bit operations.
To multiply two 2-digit integers:

Multiply four 1-digit integers.

Add, shift some 2-digit integers to obtain result.

\[
x = 10 \cdot x_1 + x_0 \\
y = 10 \cdot y_1 + y_0 \\
xy = (10 \cdot x_1 + x_0)(10 \cdot y_1 + y_0) \\
= 100 \cdot x_1 y_1 + 10 \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
\]

Same idea works for long integers – can split them into 4 half-sized ints
To multiply two n-bit integers:

Multiply four $\frac{1}{2}n$-bit integers.

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

$$x = 2^{n/2} \cdot x_1 + x_0$$

$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = \left(2^{n/2} \cdot x_1 + x_0\right) \left(2^{n/2} \cdot y_1 + y_0\right)$$

$$= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$
To multiply two n-bit integers:

Multiply four \( \frac{1}{2}n \)-bit integers.

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

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

\[T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)\]
key trick: 2 multiplies for the price of 1:

\[
\begin{align*}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
\end{align*}
\]

Well, ok, 4 for 3 is more accurate...

\[
\begin{align*}
\alpha &= x_1 + x_0 \\
\beta &= y_1 + y_0 \\
\alpha\beta &= (x_1 + x_0) (y_1 + y_0) \\
&= x_1 y_1 + (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
(x_1 y_0 + x_0 y_1) &= \alpha\beta - x_1 y_1 - x_0 y_0
\end{align*}
\]
To multiply two n-bit integers:

Add two $\frac{1}{2}n$ bit integers.

Multiply three $\frac{1}{2}n$-bit integers.

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

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

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

\[
T(n) \leq 3T(n/2) + O(n)
\]
\[
\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})
\]
Naïve: $\Theta(n^2)$
Karatsuba: $\Theta(n^{1.59\ldots})$

Amusing exercise: generalize Karatsuba to do 5 size $n/3$ subproblems $\rightarrow \Theta(n^{1.46\ldots})$

Best known: $\Theta(n \log n \log \log n)$

"Fast Fourier Transform"
Another Example:

Matrix Multiplication –

Strassen’s Method
Multiplying Matrices

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

\[=\]

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

\[n^3 \text{ multiplications, } n^3-n^2 \text{ additions}\]
for i = 1 to n
    for j = 1 to n
        C[i,j] = 0
        for k = 1 to n

\( n^3 \) multiplications, \( n^3 - n^2 \) additions
Multiplying Matrices

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

= 

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

= 

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

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

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

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

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

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

Counting arithmetic operations:

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

$$T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
8T(n/2) + n^2 & \text{if } n > 1 
\end{cases}$$

By Master Recurrence, if

$$T(n) = aT(n/b)+cn^d \& a > b^d \text{ then}$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$
The algorithm

\[ P_1 = A_{12}(B_{11} + B_{21}) \]
\[ P_3 = (A_{11} - A_{12})B_{11} \]
\[ P_5 = (A_{22} - A_{12})(B_{21} - B_{22}) \]
\[ P_6 = (A_{11} - A_{21})(B_{12} - B_{11}) \]
\[ P_7 = (A_{21} - A_{12})(B_{11} + B_{22}) \]
\[ C_{11} = P_1 + P_3 \]
\[ C_{21} = P_1 + P_4 + P_5 + P_7 \]
\[ P_2 = A_{21}(B_{12} + B_{22}) \]
\[ P_4 = (A_{22} - A_{21})B_{22} \]
\[ C_{12} = P_2 + P_3 + P_6 - P_7 \]
\[ C_{22} = P_2 + P_4 \]
Strassen’s algorithm

Multiply 2x2 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 \( O(n^{2.81}) \)

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