Lecture 2: Divide-and-Conquer!

Today:
- Quick review of Divide-and-Conquer ideas from 6.006
- Two new applications
  - Finding the closest pair of points
  - Median finding

Recitation this Friday:
- Strassen’s Fast Matrix Multiplication Algorithm

What do MIT and Harvey Mudd have in common?

What do MIT and Harvey Mudd have in common?

Last Time...
- Small variations in a problem statement can have a large impact on the solution method...
  - Simple interval scheduling: Greed
  - Weighted interval scheduling: DP
  - Flexible interval scheduling: ILP

Use the “simplest” tool possible...

From the Course Information Sheet: “Simple and elegant solutions are better than complicated [ones]”

Greed
DP
Linear Programming
Integer Linear Programming

Closest Pair of Points

We have n very simple planes flying at the same altitude. Wanna find the closest pair!
Closest Pair of Points

Finding the \(i\)th smallest element

Given: \(n\) points in Euclidean 2-space
Find: The closest pair of points

Given: An unsorted array of length \(n\) and a number \(i\)
between 1 and \(n\)
Find: The \(i\)th smallest element in the array

Mergesort

Given: An unsorted array of length \(n\)
Find: The \(i\)th smallest element in the array

Mergesort

Unsorted array of length \(n\)
Mergesort

Unsorted array of length $n$

Unsorted array of length $n/2$  Unsorted array of length $n/2$

Mergesort

Unsorted array of length $n$

Unsorted array of length $n/2$  Unsorted array of length $n/2$

Mergesort

Unsorted array of length $n$

Unsorted array of length $n/2$  Unsorted array of length $n/2$

Mergesort

Sorted array of length $n/2$  Sorted array of length $n/2$

Mergesort

Sorted array of length $n/2$  Sorted array of length $n/2$

Mergesort

Sorted array of length $n/2$  Sorted array of length $n/2$
Mergesort

Recurrences and Running Times

\[ T(n) = \text{worst case running time of mergesort on input of length } n \]

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

Having two different constants here is annoying!
Recurrences and Running Times

T(n) = worst case running time of mergesort on input of length n

T(n) = 2 T(n/2) + cn  \quad \text{Assume } n = 2^k
T(1) = d

"Recursion Tree" Analysis

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Problem size</th>
<th>Work/node</th>
<th>Total work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^k$</td>
<td>c $2^k$</td>
<td>c $2^k$</td>
</tr>
<tr>
<td>2</td>
<td>$2^{k-1}$</td>
<td>c $2^{k-1}$</td>
<td>2 c $2^{k-1}$</td>
</tr>
<tr>
<td>$2^2$</td>
<td>$2^{k-2}$</td>
<td>c $2^{k-2}$</td>
<td>$2^2$ c $2^{k-2}$</td>
</tr>
<tr>
<td>$2^3$</td>
<td>$2^{k-3}$</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$2^k$</td>
<td>$2^{k-k}$</td>
<td>d</td>
<td>$2^k$ d $2^{k-k}$</td>
</tr>
</tbody>
</table>

T(n) = $2^T(n/2) + cn$  \quad \text{Assume } n = 2^k
T(1) = d

"Recursion Tree" Analysis

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Problem size</th>
<th>Work/node</th>
<th>Total work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2^k$</td>
<td>c $2^k$</td>
<td>c $2^k$</td>
</tr>
<tr>
<td>2</td>
<td>$2^{k-1}$</td>
<td>c $2^{k-1}$</td>
<td>2 c $2^{k-1}$</td>
</tr>
<tr>
<td>$2^2$</td>
<td>$2^{k-2}$</td>
<td>c $2^{k-2}$</td>
<td>$2^2$ c $2^{k-2}$</td>
</tr>
<tr>
<td>$2^3$</td>
<td>$2^{k-3}$</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$2^k$</td>
<td>$2^{k-k}$</td>
<td>d</td>
<td>$2^k$ d $2^{k-k}$</td>
</tr>
</tbody>
</table>

T(n) = c ($2^k + 2^k + ... + 2^k$) + d$2^k$

How many terms here?
The “Mergesort Template”

\[ T(n) = 2T(n/2) + O(n) \]

- Divide the problem into two halves
- Recursively solve each half
- Spend \( O(n) \) time completing the solution

Examples:
- Finding the convex hull of \( n \) points in the plane
- Fast Fourier Transform
- Many others...

Recursion Trees or Master Theorem?

Recursion Trees or Master Theorem?

```
The Master Theorem
Let \( a \) be an integer greater than or equal to 1 and \( b \) be a real number greater than 1. Let \( c \) be a positive real number and \( d \) a nonnegative real number. Given a recurrence of the form

\[ T(n) = \begin{cases} 
   aT(n/b) + f(n) & \text{if } n > 1 \\
   \Theta(n^d) & \text{if } n = 1
\end{cases} \]

then for \( n \) a power of \( b \),
1. if \( \log_b a < c \), \( T(n) = \Theta(n^c) \)
2. if \( \log_b a = c \), \( T(n) = \Theta(n^c \log n) \)
3. if \( \log_b a > c \), \( T(n) = \Theta(n^{\log_b a}) \)

This theorem isn’t enough if you want to be the Master of your own (algorithmic) destiny!
```

Finding the Closest Pair of Points

**Input:** A list of \( n \) points in \( \mathbb{R}^2 \): \( \{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\} \)

**Output:** The closest pair of points

**Assumptions:** No shared \( x \)- or \( y \)-coordinates

**Goal:** \( O(n \log n) \) algorithm

If we’re willing to spend \( O(n \log n) \) time, we may as well sort these points (just once)!

Divide-and-Conquer!

- Build two copies of the input list: One sorted by \( x \)-coordinate and one sorted by \( y \)-coordinate

```
\[ d = \min(d_{\text{left}}, d_{\text{right}}) \]
```

Are we done?
Good news! (?)

How many points could be in this red $d \times 2d$ rectangle?
Total run time

- Sorting once by x-coordinate: $O(n \log n)$
- Sorting once by y-coordinate: $O(n \log n)$
- Recursive algorithm: $T(n) = 2T(n/2) + O(n)$

Order Statistics

Given: An unsorted array of length $n$ and a number $i$ between 1 and $n$
Find: The $i^{th}$ smallest element in the array
Assumption: All numbers are distinct (this isn’t necessary!)

### Example

$A = \begin{bmatrix} 6 & 42 & 13 & 5 & 7 \end{bmatrix}$

>>> select(A, 1) 5
>>> select(A, 3) 7

Goal: An algorithm that is asymptotically faster than $n \log n$.

**Little O of $n \log n$**

Attempt Number 1

$A = \begin{bmatrix} 6 & 42 & 13 & 5 & 7 \end{bmatrix}$

>>> select(A, 4)

- Find the median: It's 7
- Partition the array w.r.t the median...

Attempt Number 1

$A = \begin{bmatrix} 6 & 42 & 13 & 5 & 7 \end{bmatrix}$

>>> select(A, 4)

- Find the median: It's 7
- Partition the array w.r.t the median...
Find the median: It's 7

Parition the array w.r.t the median...

4 > 3, so recursively call...

select([13, 42], 4-3)
Order Statistics Revisited

- If we could find the median in $O(n)$ time, we’d be happy!
- Unfortunately: Find the median in $O(n)$ time?
- Fortunately, we can find something close enough to the median in $O(n)$ time!

\[ T(n) = T(n/2) + cn \]
\[ T(1) = d \]

What’s wrong here?

We’ll try for a running time of $O(n)$ regardless of the value of $i$.

select($A, i$)

An unsorted array of $n$ (distinct) elements

find the $i^{th}$ smallest element

Let $T(n)$ denote the worst case running time for array of length $n$.

1. Partition the $n$ elements of the array $A$ into groups of size 3 and

... sort each group of 3 in increasing order

What sorting algorithm should we use here?
Let $T(n)$ denote the worst case running time for an array of length $n$.

**select(A, i)**

An unsorted array of $n$ (distinct) elements

1. Partition the $n$ elements of the array $A$ into groups of size 3 and...

... sort each group of 3 in increasing order

$O(n)$

What sorting algorithm should we use here?

2. Collect the middle elements of those groups into an array $B$ of size $n/3$ and find the median element of $B$.

$B = \begin{array}{cccccccc}
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\end{array}$

$B_{median} = select(B, \, ?)$

How long does this take?

Let $T(n)$ denote the worst case running time for an array of length $n$.

**select(A, i)**

An unsorted array of $n$ (distinct) elements

2. Collect the middle elements of those groups into an array $B$ of size $n/3$ and find the median element of $B$.

$B = \begin{array}{cccccccc}
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\end{array}$

$B_{median} = select(B, n/6)$

How long does this take?

Let $T(n)$ denote the worst case running time for an array of length $n$.

**select(A, i)**

An unsorted array of $n$ (distinct) elements

2. Collect the middle elements of those groups into an array $B$ of size $n/3$ and find the median element of $B$.

$B = \begin{array}{cccccccc}
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\end{array}$

$B_{median} = select(B, n/6)$

T($n/3$) time!

How long does this take?

Let $T(n)$ denote the worst case running time for an array of length $n$.

**select(A, i)**

An unsorted array of $n$ (distinct) elements

3. Partition the $n$ elements of $A$ w.r.t. $B_{median}$.

$B = \begin{array}{cccccccc}
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} & \hat{v} \\
\end{array}$

$1 \rightarrow \cdots \rightarrow n$

$< B_{median} \rightarrow \rightarrow > B_{median}$

How long does this take?
Let $T(n)$ denote the worst case running time for array of length $n$.

3. Partition the $n$ elements of $A$ w.r.t. $B_{median}$.

$$B = \begin{array}{cccccccc}
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
1 & & j & & n \\
\end{array}$$

- If (we’re really lucky) and $i = j$... return $B_{median}$.
- If $i < j$... call $select(A[1, \ldots, j-1], i)$
- If $i > j$... call $select(A[j+1, \ldots, n], i-j)$
select(A, i)

An unsorted array of n (distinct) elements

3. Partition the n elements of A w.r.t. Bmedian

- If (we're really lucky) and i = j, return Bmedian.
- If i < j, call select(A[1, ..., j-1], i)
- If i > j, call select(A[j+1, ..., n], i-j)

T(n) = O(n) + T(n/3) + O(n) + ???

Sort groups of 3
Recurse to find Bmedian
Partition A w.r.t. Bmedian
Recurse
\[ T(n) = O(n) + T(n/3) + O(n) + ??? \]

\[ T(n) \leq O(n) + T(n/3) + O(n) + T(2n/3) \]

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

\[ T(1) \leq c \]

\[ \text{Problem size: } n \]
\[ \text{Work: } cn \]

\[ \text{Problem size: } n/3 \]
\[ \text{Work: } c \cdot n/3 \]

\[ \text{Problem size: } 2n/3 \]
\[ \text{Work: } c \cdot 2n/3 \]
$T(n) \leq T(n/3) + T(2n/3) + cn$

Problem size: $n$

Work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$

Total work: $cn$
T(n) ≤ O(n) + T(n/5) + O(n) + T(7n/10)

Sort groups of 5
Recurse to find Bmedian
Partition A w.r.t. Bmedian
Recurse

3n/10

Why is this likely to be any better than the
T(n) = T(n/3) + T(2n/3) + cn
that we had before?

Problem size: n
Work: cn

T(n) ≤ T(2n/10) + T(7n/10) + cn
T(1) ≤ c

Problem size: 2n/10
Work: c 2n/10

Problem size: 7n/10
Work: c 7n/10
\[ T(n) \leq T(2n/10) + T(7n/10) + cn \]
\( T(1) \leq c \)

Problem size: \( n \)
Work: \( cn \)
Total work: \( c9n/10 \)
Problem size: \( 2n/10 \)
Work: \( c2n/10 \)
Problem size: \( 7n/10 \)
Work: \( c7n/10 \)

\[ T(n) \leq T(2n/10) + T(7n/10) + cn \]
\( T(1) \leq c \)

Problem size: \( n \)
Work: \( cn \)
Total work: \( c9n/10 \)
Problem size: \( 2n/10 \)
Work: \( c2n/10 \)
Problem size: \( 7n/10 \)
Work: \( c7n/10 \)

\[ T(n) = c \left( 1 + 9/10 + (9/10)^2 + \ldots (9/10)^k \right) \]
\[ < \lim_{k \to \infty} c \left( 1 + 9/10 + (9/10)^2 + \ldots (9/10)^k \right) \]
\[ = \lim_{k \to \infty} \frac{1 - (9/10)^k}{1 - 9/10} \]
\[ = \frac{10}{9} \]

This term goes to 1

\[ O(n) \]
Multiplying Big Numbers

```python
>>> x = 123456789101112131415161718192021222324
>>> y = 6046046046046046046046046046046046046046
>>> x*y
What the "L"?!?
```

$x =$ n digits long

$y =$ Is green times red = blue?

All that took $O(n^2)$ time
All that took $O(n^2)$ time... and now we add up the columns...

$O(n^2)$ total time!

Karatsuba!

$x = \overline{x_{\text{high}} \, x_{\text{low}}}$
$y = \overline{y_{\text{high}} \, y_{\text{low}}}$

$x y = (x_{\text{high}} 10^{n/2} + x_{\text{low}})(y_{\text{high}} 10^{n/2} + y_{\text{low}}) = x_{\text{high}} y_{\text{high}} 10^n + (x_{\text{high}} y_{\text{low}} + x_{\text{low}} y_{\text{high}}) 10^{n/2} + x_{\text{low}} y_{\text{low}}$

Recursive calls on problems of length $n/2$
Karatsuba!

$$x = \begin{array}{c} x_{\text{high}} \\ x_{\text{low}} \end{array} \quad n \text{ digits long}$$

$$y = \begin{array}{c} y_{\text{high}} \\ y_{\text{low}} \end{array}$$

$$xy = (x_{\text{high}} \cdot 10^{n/2} + x_{\text{low}})(y_{\text{high}} \cdot 10^{n/2} + y_{\text{low}}) = x_{\text{high}}y_{\text{high}} \cdot 10^n + (x_{\text{high}}y_{\text{low}} + x_{\text{low}}y_{\text{high}}) \cdot 10^{n/2} + x_{\text{low}}y_{\text{low}}$$

What about these multiplications?

Assume $$n = 2^k$$

# Nodes | Problem size | Work/node | Total work
---|---|---|---
1 | $$2^k$$ | $$c \cdot 2^k$$ | $$c \cdot 2^k$$
4 | $$2^{k-1}$$ | $$c \cdot 2^{k-1}$$ | $$4 \cdot 2^{k-1}$$
$$4^2$$ | $$2^{k-2}$$ | $$c \cdot 2^{k-2}$$ | $$4^2 \cdot 2^{k-2}$$

$$T(n) = 4 \cdot T(n/2) + cn$$

$$T(1) = c$$

Assume $$n = 2^k$$

# Nodes | Problem size | Work/node | Total work
---|---|---|---
1 | $$2^k$$ | $$c \cdot 2^k$$ | $$c \cdot 2^k$$
4 | $$2^{k-1}$$ | $$c \cdot 2^{k-1}$$ | $$4 \cdot 2^{k-1}$$
$$4^2$$ | $$2^{k-2}$$ | $$c \cdot 2^{k-2}$$ | $$4^2 \cdot 2^{k-2}$$
$$\vdots$$ | $$\vdots$$ | $$\vdots$$ | $$\vdots$$
$$4^k$$ | $$2^{k-k}$$ | $$c \cdot 2^{k-k}$$ | $$4^k \cdot 2^{k-k}$$

$$T(n) = 4 \cdot T(n/2) + cn$$

$$T(1) = c$$

Assume $$n = 2^k$$

# Nodes | Problem size | Work/node | Total work
---|---|---|---
1 | $$2^k$$ | $$c \cdot 2^k$$ | $$c \cdot 2^k$$
4 | $$2^{k-1}$$ | $$c \cdot 2^{k-1}$$ | $$4 \cdot 2^{k-1}$$
$$4^2$$ | $$2^{k-2}$$ | $$c \cdot 2^{k-2}$$ | $$4^2 \cdot 2^{k-2}$$
$$\vdots$$ | $$\vdots$$ | $$\vdots$$ | $$\vdots$$
$$4^k$$ | $$2^{k-k}$$ | $$c \cdot 2^{k-k}$$ | $$4^k \cdot 2^{k-k}$$
T(n) = 4 T(n/2) + cn
T(1) = c

Assume n = 2^k

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Problem size</th>
<th>Work/node</th>
<th>Total work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2^k</td>
<td>c 2^k</td>
<td>c 2^k</td>
</tr>
<tr>
<td>4</td>
<td>2^k-1</td>
<td>c 2^k-1</td>
<td>4 c 2^k-1</td>
</tr>
<tr>
<td>4^2</td>
<td>2^k-2</td>
<td>c 2^k-2</td>
<td>4^2 c 2^k-2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4^k</td>
<td>2^k-k</td>
<td>c 2^k-k</td>
<td>4^k c 2^k-k</td>
</tr>
</tbody>
</table>

\[ c2^1(1 + 4/2 + (4/2)^2 + \ldots + (4/2)^k) = cn(1 + 2 + 2^2 + 2^k) = cn(2^{k+1} - 1)/(2-1) \in O(n^3) \]

Introduce Karatsuba, Part Deux

\[ x = \begin{array}{c}
\text{x}_\text{high} \\
\text{n digits long}
\end{array} \]

\[ y = \begin{array}{c}
\text{y}_\text{high} \\
\text{y}_\text{low}
\end{array} \]

\[ xy = (x_{\text{high}} 10^{n/2} + x_{\text{low}} y_{\text{high}} + 10^{n/2} y_{\text{low}}) = x_{\text{high}} y_{\text{high}} 10^n + (x_{\text{high}} y_{\text{low}} + x_{\text{low}} y_{\text{high}}) 10^{n/2} + x_{\text{low}} y_{\text{low}} \]

Alternatively...

- Let \( P_1 = x_{\text{high}} y_{\text{high}} \)
- Let \( P_2 = x_{\text{low}} y_{\text{low}} \)
- Let \( P_3 = (x_{\text{high}} + x_{\text{low}}) (y_{\text{high}} + y_{\text{low}}) \)
- Now, \( x_{\text{high}} y_{\text{low}} + x_{\text{low}} y_{\text{high}} = P_3 - P_1 - P_2 \)

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

Assume n = 2^k

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Problem size</th>
<th>Work/node</th>
<th>Total work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2^k</td>
<td>c 2^k</td>
<td>c 2^k</td>
</tr>
<tr>
<td>3</td>
<td>2^k-1</td>
<td>c 2^k-1</td>
<td>3 c 2^k-1</td>
</tr>
<tr>
<td>3^2</td>
<td>2^k-2</td>
<td>c 2^k-2</td>
<td>3^2 c 2^k-2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3^k</td>
<td>2^k-k</td>
<td>c 2^k-k</td>
<td>3^k c 2^k-k</td>
</tr>
</tbody>
</table>

\[ c2^1(1 + (3/2) + (3/2)^2 + \ldots + (3/2)^k) = c2^1(1 + (3/2) + (3/2)^2 + \ldots + (3/2)^k) \]

\[ = c2^k \left( \left( \frac{3}{2} \right)^{k+1} - 1 \right) \]

Coming to a Recitation Near You...

Matrix Multiplication!

Volker Strassen  
Don Coppersmith  
Shmuel Winograd

<table>
<thead>
<tr>
<th>Name</th>
<th>Team</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reid W. Barton</td>
<td>MIT</td>
<td>2001 2002 2003 2004</td>
</tr>
<tr>
<td>Daniel Kane</td>
<td>MIT</td>
<td>2003 2004 2005 2006</td>
</tr>
<tr>
<td>Brian R. Lawrence</td>
<td>Caltech</td>
<td>2007 2008 2010 2011</td>
</tr>
<tr>
<td>Edward L. Kaplan</td>
<td>Carnegie Tech</td>
<td>1939 1940 1941</td>
</tr>
</tbody>
</table>
A Breakthrough On Matrix Product

Finding the Convex Hull

Input: A set of n points in the plane
Output: The points on the hull in CCW order

Sort points by x-coordinate just once!
Does it really work to just connect these?

Connect uppermost points?

Connect bottommost points?

Finding the Convex Hull
Finding the Convex Hull

Can the Convex Hull be found faster?

Input: A set of $n$ points in the plane
Output: The points on the hull in CCW order
$A = [5, 1, 2, 3, ...]$

Can the Convex Hull be found faster?

Input: A set of $n$ points in the plane
Output: The points on the hull in CCW order

The End!

Adios, ciao!