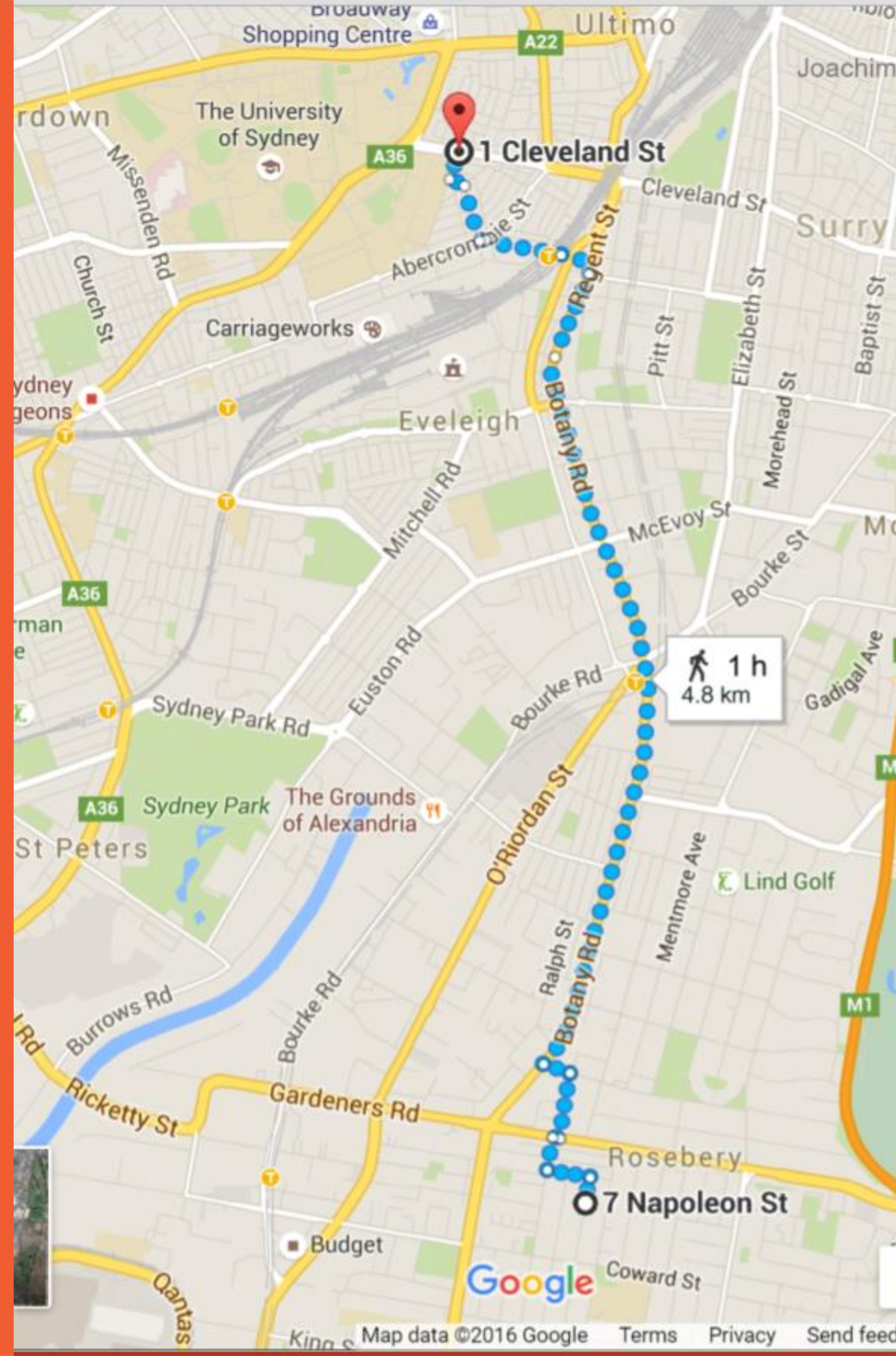
Lecture 3:
Greedy algorithms (Adv.) cont'd from last week



THE UNIVERSITY OF SYDNEY

SET-COVER

[Slides by D Moshkovitz]

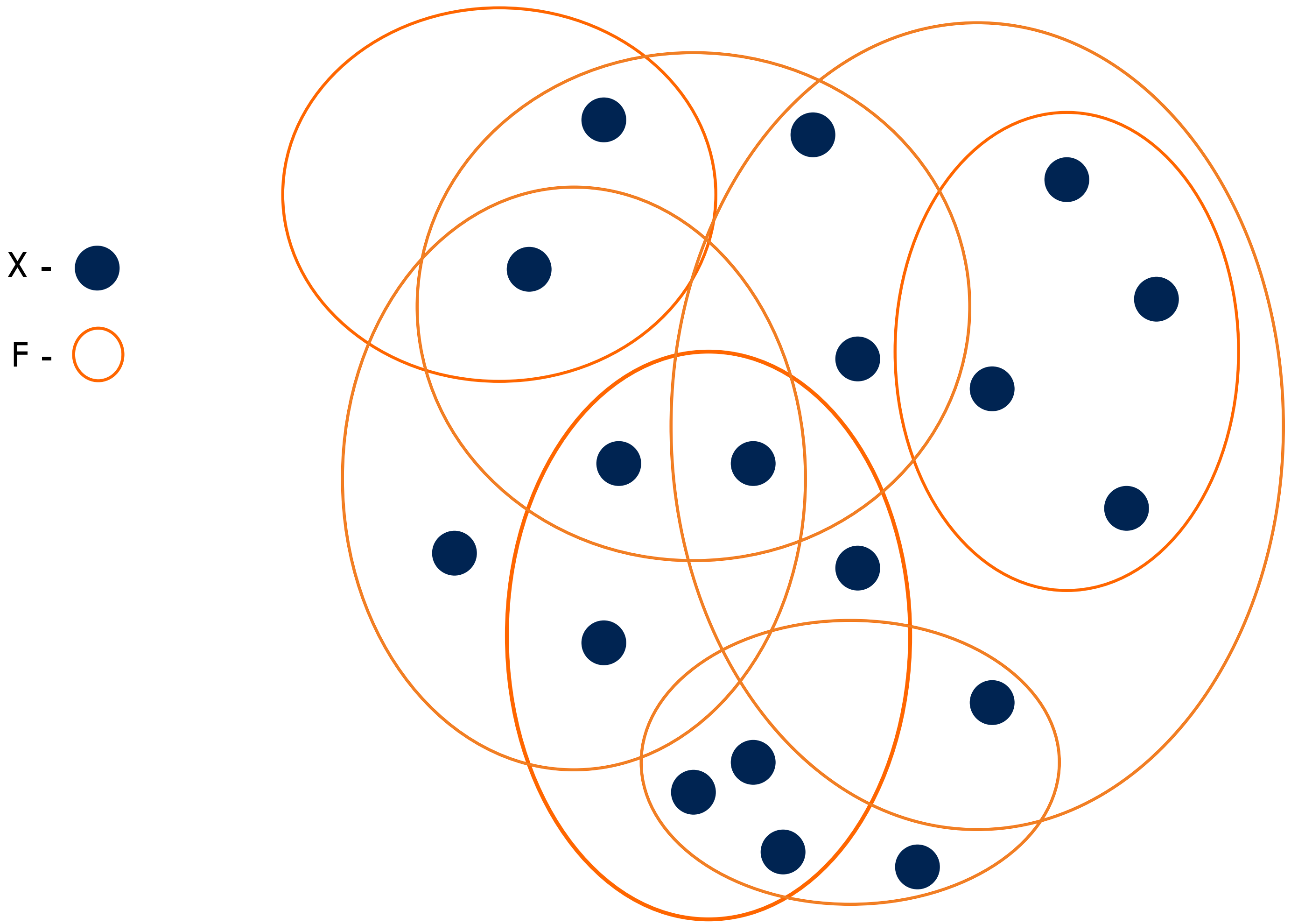
Instance: a finite set X and a family F of subsets of X, such that

$$X = \bigcup_{S \in F} S$$

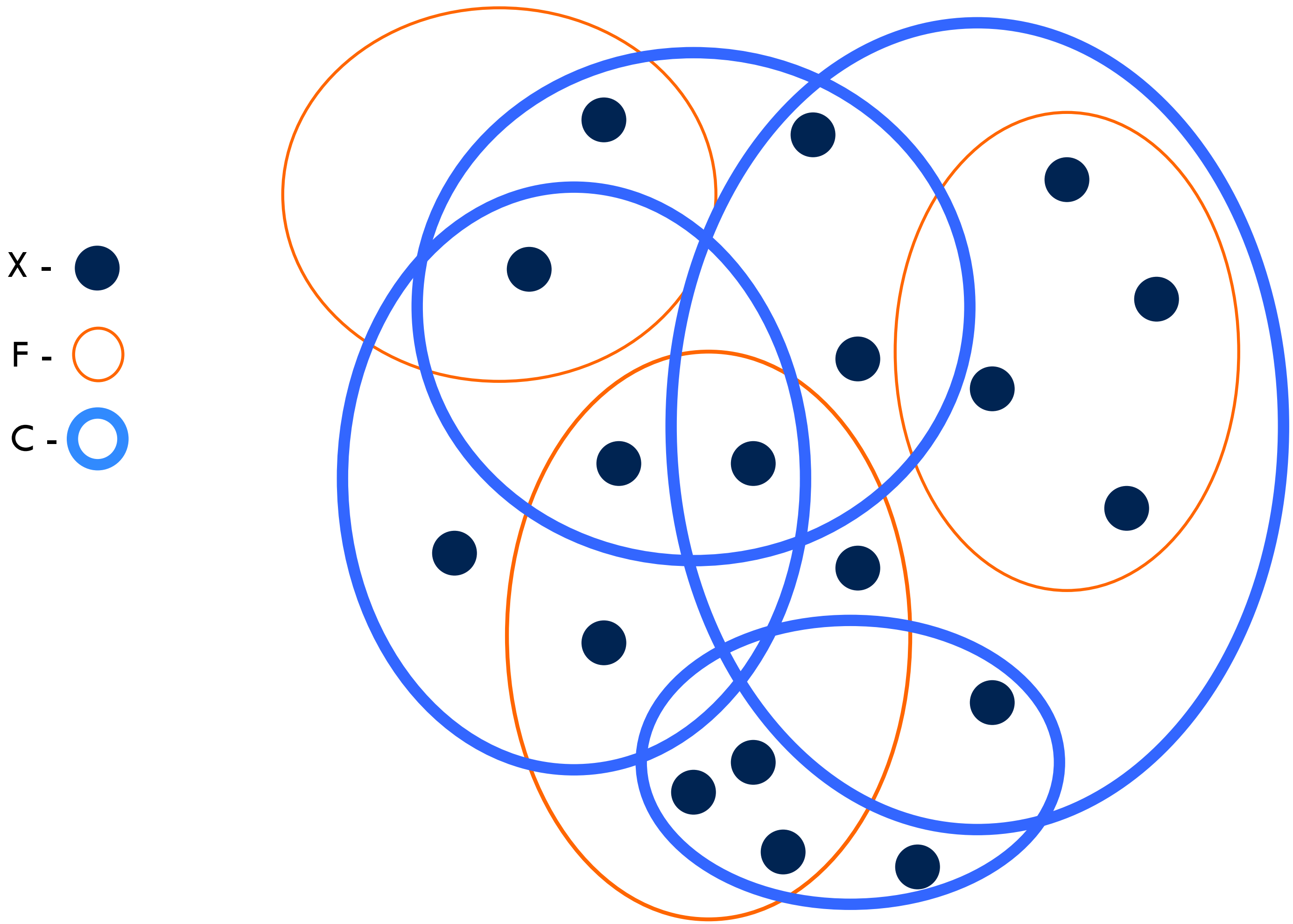
Problem: find a set $C \subseteq F$ of minimal size which *covers* X, i.e.

$$X = \bigcup_{S \in C} S$$

SET-COVER: Example



SET-COVER: Example



The Greedy Algorithm

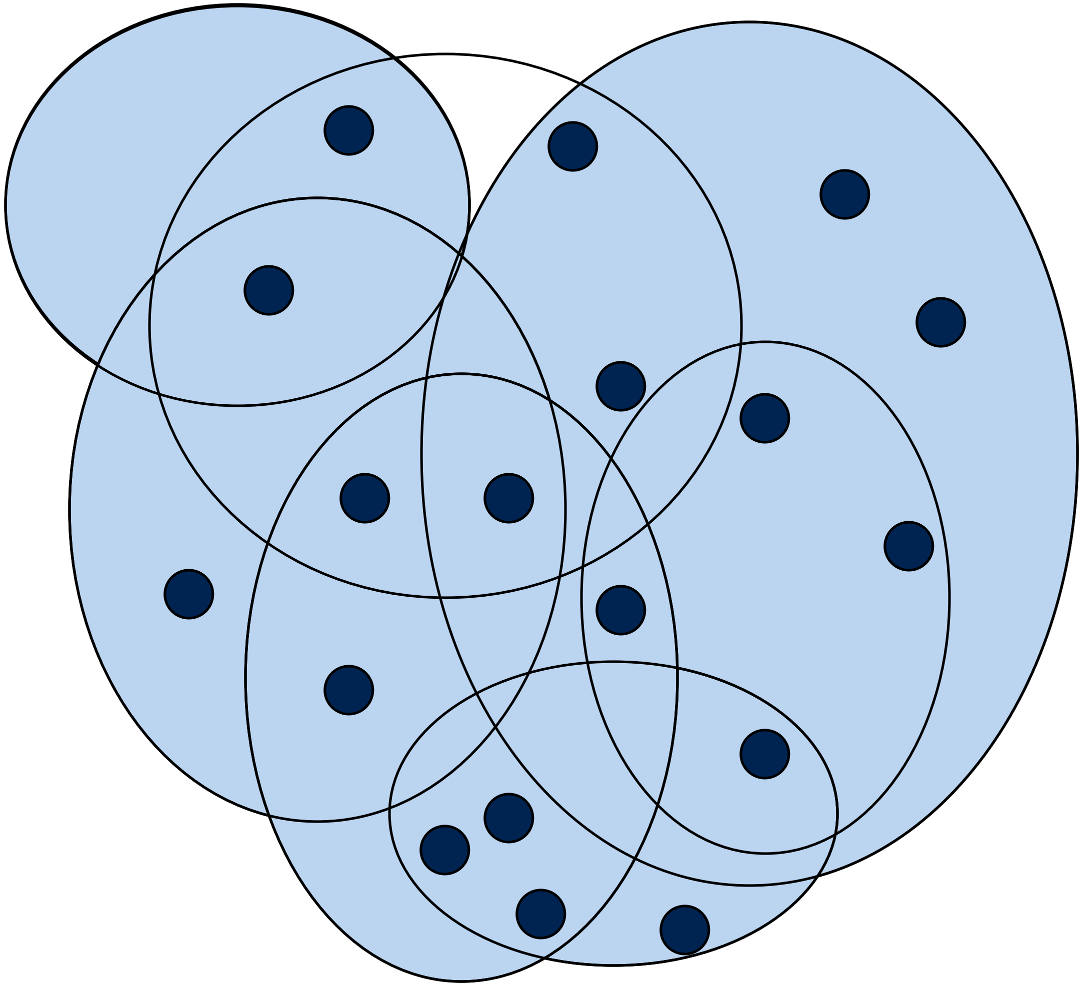
- $C \leftarrow \varnothing$
- $U \leftarrow X$
- **while** $U \neq \varnothing$ **do**
  - select $S \in F$ that maximizes $|S \cap U|$
  - $C \leftarrow C \cup \{S\}$
  - $U \leftarrow U \setminus S$
- return C

$O(|F| \cdot |X|)$

$\min\{|X|, |F|\}$

Example

5



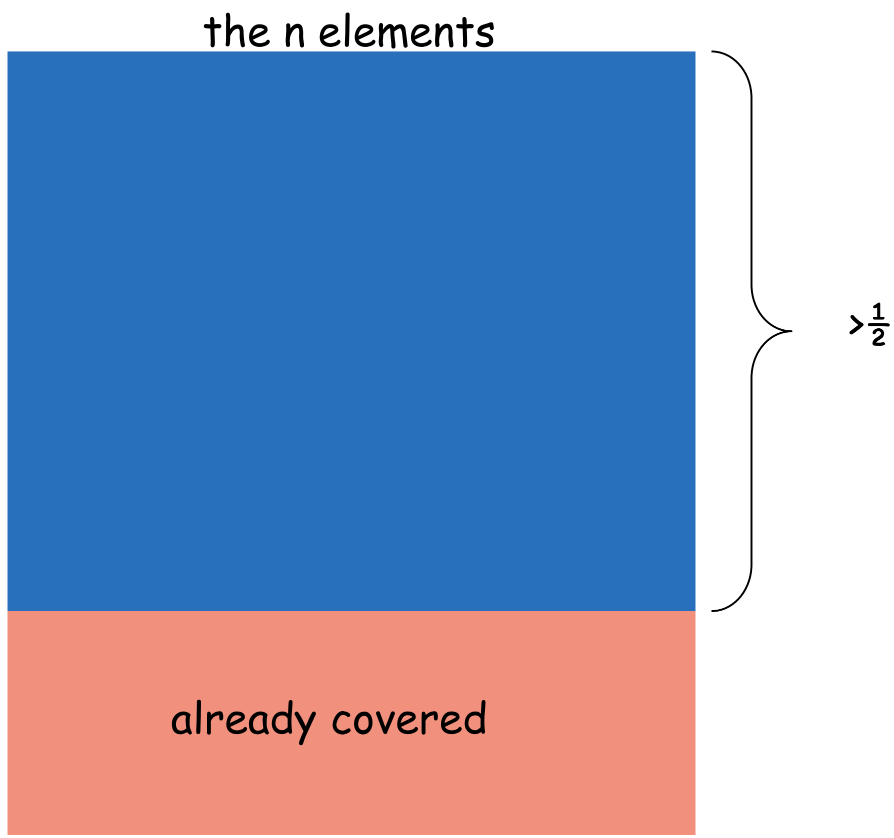
The Trick

- We'd like to compare the number of subsets returned by the greedy algorithm to the optimal
- The optimal is unknown, however, if it consists of k subsets, then any part of the universe can be covered by k subsets!

Loose Ratio-Bound

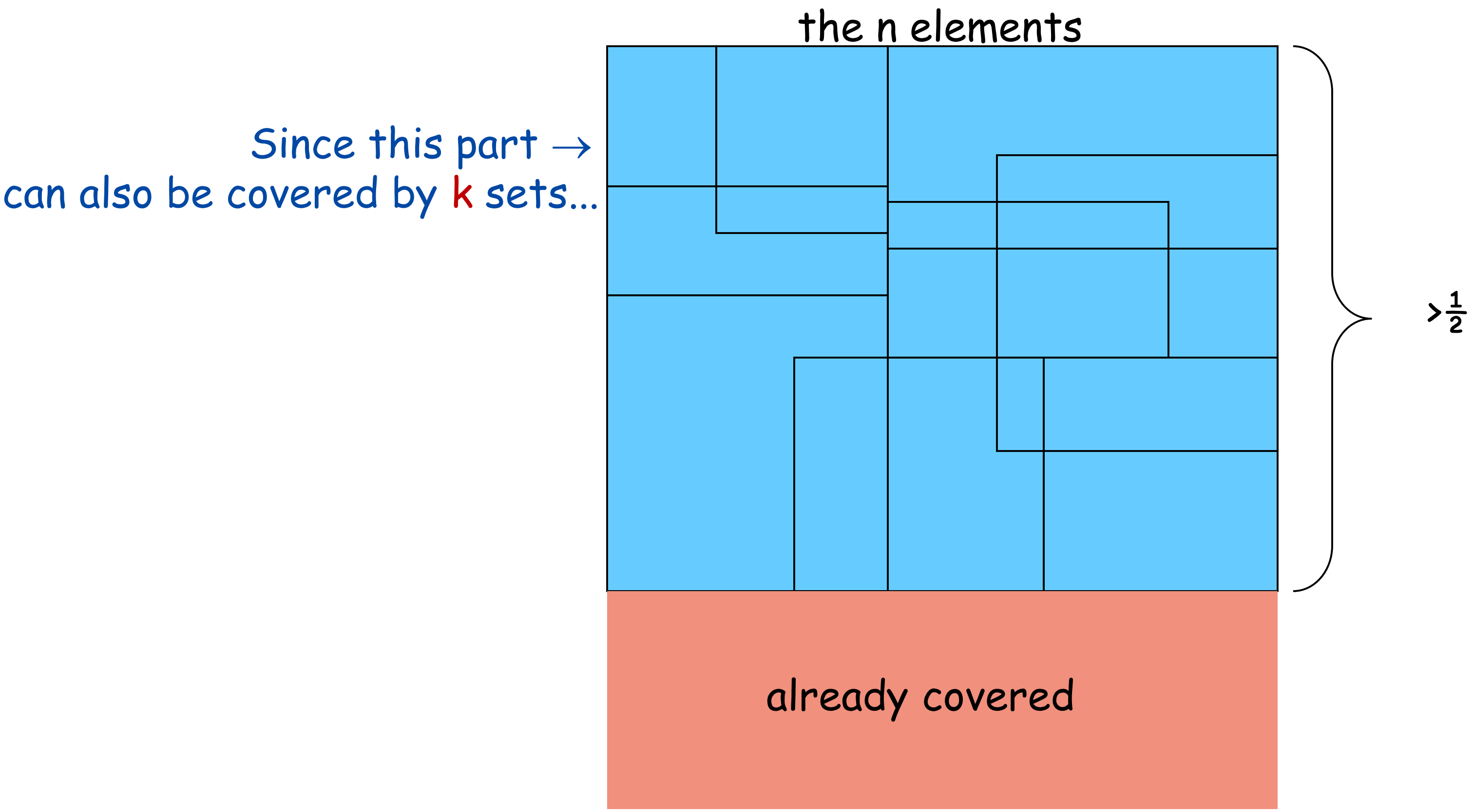
Claim: If $\exists$ cover of size k, then after k iterations the algorithm covered at least ½ of the elements

Assume the opposite and observe the situation after k iterations:



Loose Ratio-Bound

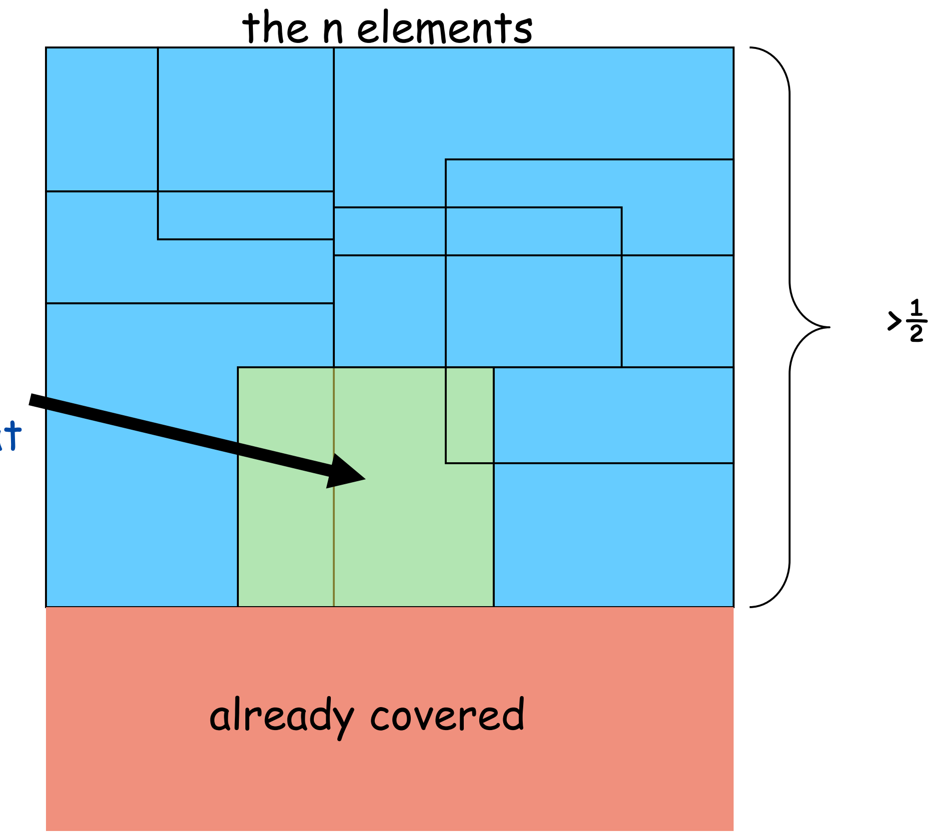
Claim: If $\exists$ cover of size k, then after k iterations the algorithm covered at least ½ of the elements



Loose Ratio-Bound

Claim: If ∃ cover of size k, then after k iterations the algorithm covered at least ½ of the elements

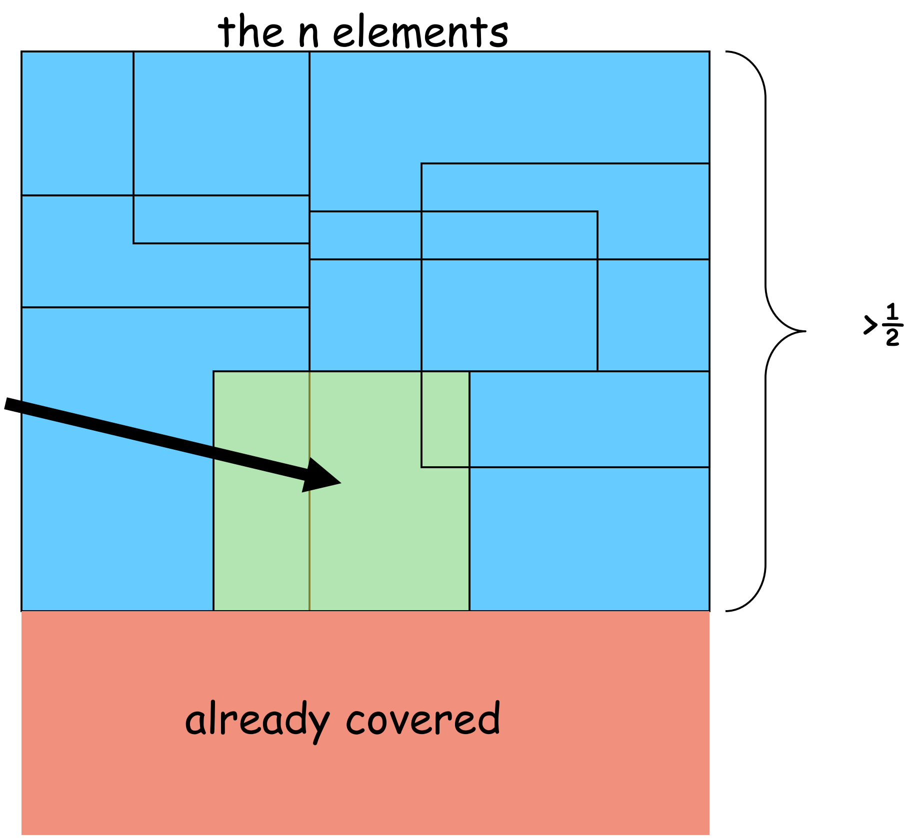
there must be a set not chosen yet, whose size is at least (n/2)/k



Loose Ratio-Bound

Claim: If $\exists$ cover of size k, then after k iterations the algorithm covered at least ½ of the elements

and the claim is proven!



there must be a set not chosen yet, whose size is at least (n/2)/k

Thus in each of the k iterations we've covered at least (n/2)/k new elements

Loose Ratio-Bound

Claim: If $\exists$ cover of size k, then after k iterations the algorithm covered at least ½ of the elements.

How many times can we half the set? O(log n) times!

Each time we perform at most k iterations.

$\Rightarrow$ total number of iterations $\leq k \log n$

Therefore after k log n iterations (i.e - after choosing k log n sets) all the n elements must be covered, and the bound is proved.

■

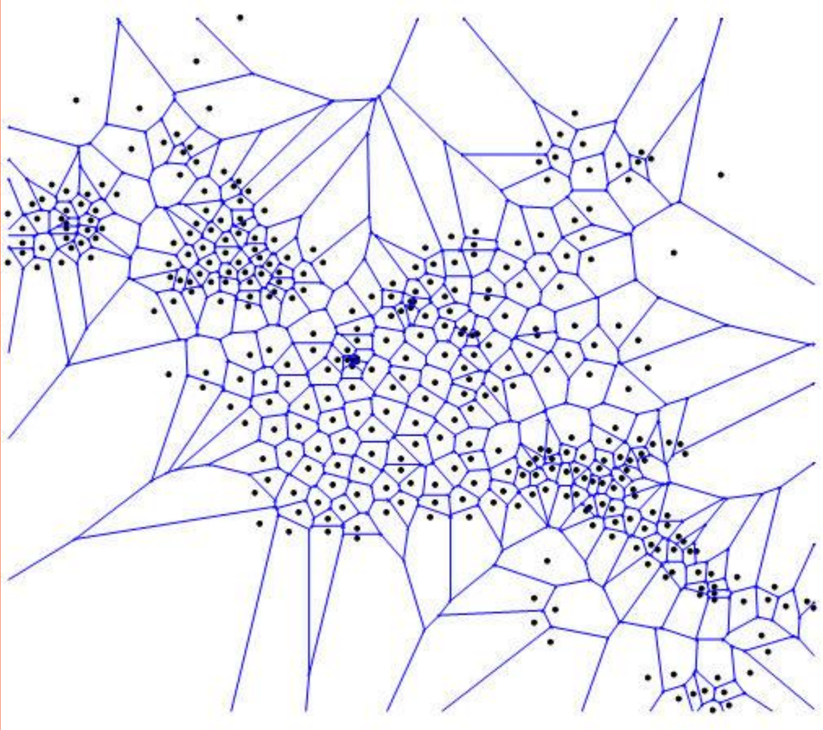
Summary: Greedy algorithms

A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

Problems

- Interval scheduling
- Scheduling: minimize lateness
- Shortest path in graphs (Dijkstra's algorithms)
- Minimum spanning tree (Prim's algorithm)
- Clustering
- …

Lecture 4:
Divide and Conquer (Adv.)



THE UNIVERSITY OF SYDNEY

The median problem

- The median is the "half-way" point of a set

- Given a sequence of n numbers, the median can be found as follows:
  - sort the numbers $\Omega(n \log n)$
  - the median is the middle element (element at position "n/2")

- Can we find the median in linear time?

The selection problem

- Given an unsorted array A with n number and a number k, find k-th smallest number in A
- Trivial solution: Sort the elements and return kth element.
- Can we do better than $O(n \log n)$?
- How could we solve this problem with divide and conquer?

First attempt

- Suppose we could compute the median element of A in $O(n)$ time
  - If $k < n/2$ then find k-th among elements smaller than the median
  - If $k > n/2$ then find $(k-n/2)$-th among elements larger than the median

- This leads to the recurrence $T(n) = T(n/2) + O(n)$, which solves to $T(n) = O(n)$

- But how can we compute the median in O(n) time?

Approximating the median

- We don't need the exact median. Suppose we could find in $O(n)$ time an element x in A such that
$$|A|/3 < \text{rank}(A, x) < 2|A|/3$$

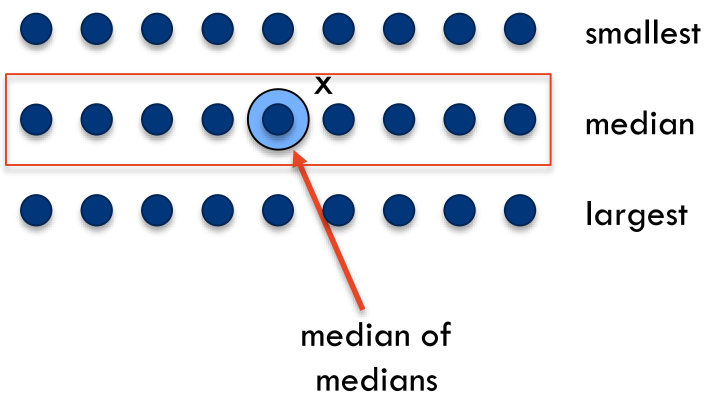
- Then we get the recurrence
$$T(n) = T(2n/3) + O(n)$$

- Which again solves to $T(n) = O(n)$

- To approximate the median we can use a recursive call!

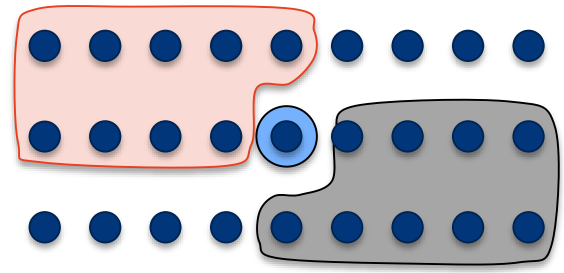
Median of 3-medians

- Consider the following procedure
  - Partition A into |A|/3 groups of 3
  - Sort each group
  - For each group find the median
  - Let x be the median of the medians



Median of 3-medians

– Consider the following procedure
  – Partition A into |A|/3 groups of 3
  – For each group find the median
  – Let x be the median of the medians



– We claim that x has the desired property

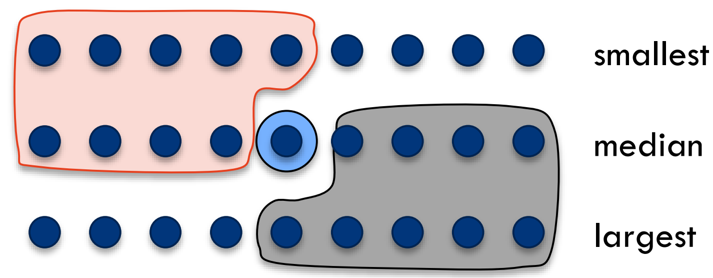
$$|A|/3 < \text{rank}(A, x) < 2|A|/3$$

– Half of the groups have a median that is smaller than x, and each group has two elements smaller than x, thus

$$\#\text{ elements smaller than } x > 2\,(|A|/6) = |A|/3$$
$$\#\text{ elements greater than } x > 2\,(|A|/6) = |A|/3$$

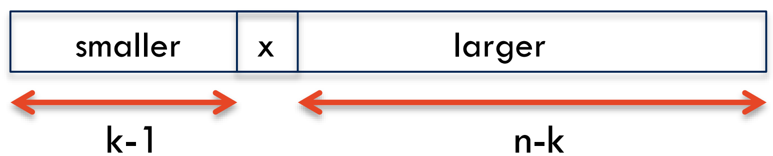
Median of 3-medians

- Consider the following procedure
  - Partition A into |A|/3 groups of 3
  - Sort each group
  - For each group find the median
  - Let x be the median of the medians



- Claim: $|A|/3 < \text{rank}(A, x) < 2|A|/3$

- Partition the initial array of elements quicksort-style around x.
  - if k=n/2 we are done, return x
  - otherwise, if n/2<k recursively find the n/2-th largest element in the low partition, or (n/2-k+1)-th in the high partition otherwise



Median of 3-medians

– We don’t need the exact median. With a recursive call on $n/3$ elements, we can find x in A such that

$$|A|/3 < \text{rank}(A, x) < 2|A|/3$$

– We get the recurrence

the median of medians

$$T(n) = T(2n/3) + T(n/3) + O(n)$$

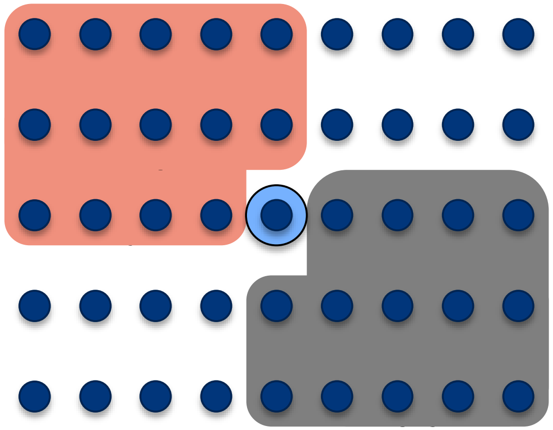
size of new subproblem

partition the array

Which solves to $T(n) = O(n \log n)$

No better than sorting!

Median of 5-medians



- What if we try dividing the set into groups of 5?

- We get:

  $3|A|/10 < \text{rank}(A, x) < 7|A|/10$

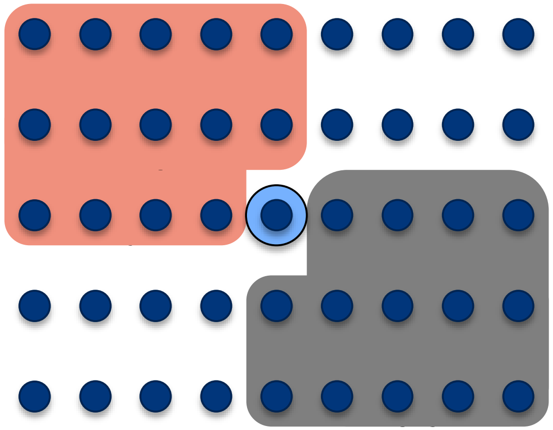
Then we get the recurrence

$$T(n) = T(7n/10) + T(n/5) + O(n)$$

Which solves to $T(n) = O(n)$

Asymptotically faster than sorting!

Median and Selection



Theorem:

Median and Selection can be solved in O(n) time.

Matrix Multiplication

Matrix Multiplication

– Matrix multiplication. Given two n-by-n matrices A and B, compute C = AB.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

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

– Brute force. $\Theta(n^3)$ arithmetic operations.

– Fundamental question. Can we improve upon brute force?

Matrix Multiplication: Warmup

- Divide-and-conquer.
  - Divide: partition A and B into ½n-by-½n blocks.
  - Conquer: multiply 8 ½n-by-½n recursively.
  - Combine: add appropriate products using 4 matrix additions.

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

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)$$

Matrix Multiplication: Warmup

MMult(A, B, n)

A. If n = 1 then Output A $\times$ B

B. else

1) Compute A11, B11, . . ., A22, B22
2) X1 $\leftarrow$ MMult(A11, B11, n/2)
3) X2 $\leftarrow$ MMult(A12, B21, n/2)
4) X3 $\leftarrow$ MMult(A11, B12, n/2)
5) X4 $\leftarrow$ MMult(A12, B22, n/2)
6) X5 $\leftarrow$ MMult(A21, B11, n/2)
7) X6 $\leftarrow$ MMult(A22, B21, n/2)
8) X7 $\leftarrow$ MMult(A21, B12, n/2)
9) X8 $\leftarrow$ MMult(A22, B22, n/2)

$8T(n/2)$

10) C 11 $\leftarrow$ X1 + X2
11) C 12 $\leftarrow$ X3 + X4
12) C 21 $\leftarrow$ X5 + X6
13) C 22 $\leftarrow$ X7 + X8

$O(n^2)$

14) Output C

C. End If

Matrix Multiplication: Key Idea

– Key idea. multiply 2-by-2 block matrices with only 7 multiplications.

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

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

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

Fast Matrix Multiplication

- Fast matrix multiplication. (Strassen, 1969)
  - Divide: partition A and B into ½n-by-½n blocks.
  - Compute: 14 ½n-by-½n matrices via 10 matrix additions.
  - Conquer: multiply 7 ½n-by-½n matrices recursively.
  - Combine: 7 products into 4 terms using 8 matrix additions.

- Analysis.
  - Assume n is a power of 2.
  - T(n) = # arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication

Strassen(A, B)

A. If $n = 1$ Output $A \times B$

B. Else

C. Compute A11, B11, . . ., A22, B22

1. $P1 \leftarrow$ Strassen(A11, B12 − B22)
2. $P2 \leftarrow$ Strassen(A11 + A12, B22)
3. $P3 \leftarrow$ Strassen(A21 + A22, B11)
4. $P4 \leftarrow$ Strassen(A22, B21 − B11)
5. $P5 \leftarrow$ Strassen(A11 + A22, B11 + B22)
6. $P6 \leftarrow$ Strassen(A12 − A22, B21 + B22)
7. $P7 \leftarrow$ Strassen(A11 − A21, B11 + B12)

$7T(n/2)$

8. $C11 \leftarrow P5 + P4 - P2 + P6$
9. $C12 \leftarrow P1 + P2$
10. $C\,21 \leftarrow P3 + P4$
11. $C\,22 \leftarrow P1 + P5 - P3 - P7$

$O(n^2)$

12. Output C

D. End If

Fast Matrix Multiplication in Practice

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

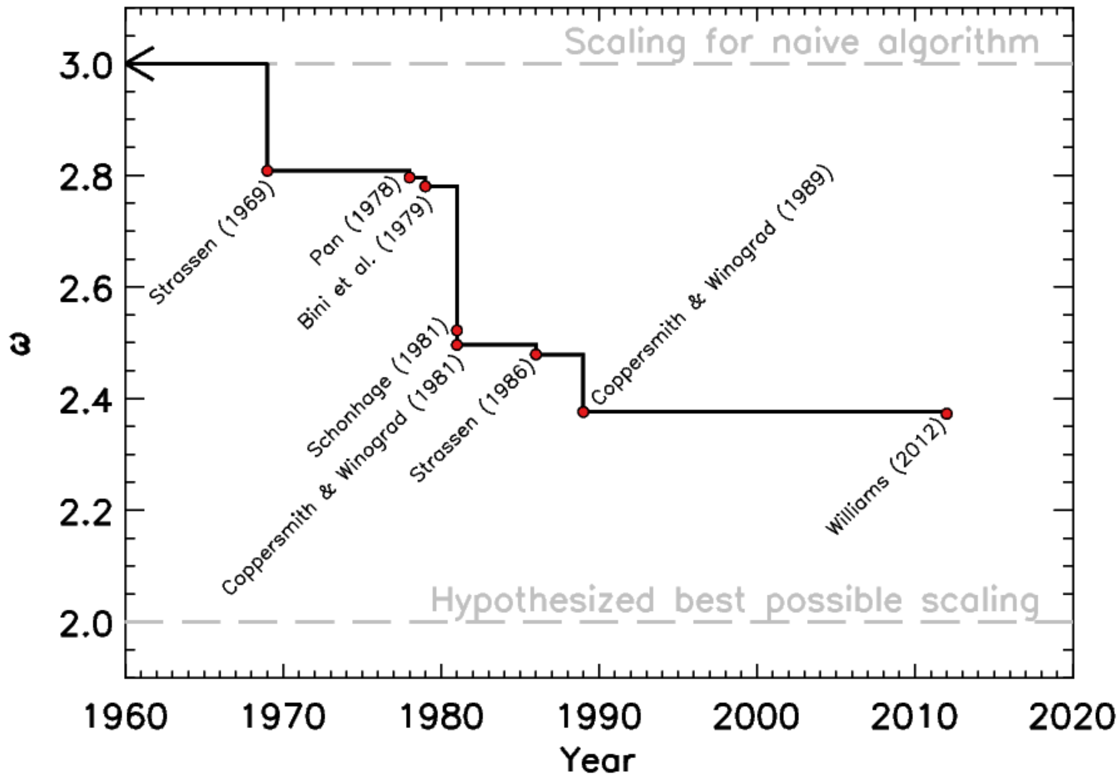
- Common misperception: "Strassen is only a theoretical curiosity."
  - Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when n ~ 2,500.
  - Range of instances where it's useful is a subject of controversy.

- 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] $O(n^{2.81})$

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



Fast Matrix Multiplication in Theory

– Best known. $O(n^{2.373})$ [Williams, 2012]

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

– Caveat. Theoretical improvements to Strassen are progressively less practical.

Summary: Divide-and-Conquer

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