CSE 373: Data Structures and Algorithms

Lecture 21: Finish Sorting, P vs NP

Instructor: Lilian de Greef
Quarter: Summer 2017

Today

- Announcements
- Finish up sorting
    - Radix Sort
    - Final comments on sorting
- Complexity Theory: P =? NP

Announcements

- Final Exam:
  - Next week
  - During usual lecture time (10:50am - 11:50am)
  - Cumulative (so all material we’ve covered in class is fair game)
  - ... but with emphasis on material covered after the midterm
  - Date: Friday

Back to Sorting Algorithms

The Big Picture

Surprising amount of juicy computer science: 2-3 lectures…

Simple algorithms: $O(n^2)$	Fancier algorithms: $O(n \log n)$	Comparison lower bound: $\Omega(n \log n)$	Specialized algorithms: $O(n)$	Handling huge data sets
Insertion sort	Heap sort		Bucket sort	External sorting
Selection sort	Merge sort		Radix sort	
Shell sort	Quick sort (avg)			
…	…			

How???

- Change the model – assume more than "compare(a,b)"

Radix sort

- Radix = "the base of a number system"
  - Examples will use 10 because we are used to that
  - In implementations use larger numbers
    - For example, for ASCII strings, might use 128
- Idea:
  - Bucket sort on one digit at a time
    - Number of buckets = radix
    - Starting with *least* significant digit
    - Keeping sort *stable*
  - Do one pass per digit
  - Invariant: After *k* passes (digits), the last *k* digits are sorted
- Aside: Origins go back to the 1890 U.S. census

Radix Sort: Example

\# digits = 3

Input:

478
537
9
721
3
38
143
67

First pass: bucket sort by one's digit

0	1	2	3	4	5	6	7	8	9
	721		3 143				537 67	478 38	9

Second pass: stable bucket sort by ten's digit

0	1	2	3	4	5	6	7	8	9
3 9		721	537 38	143		67	478		

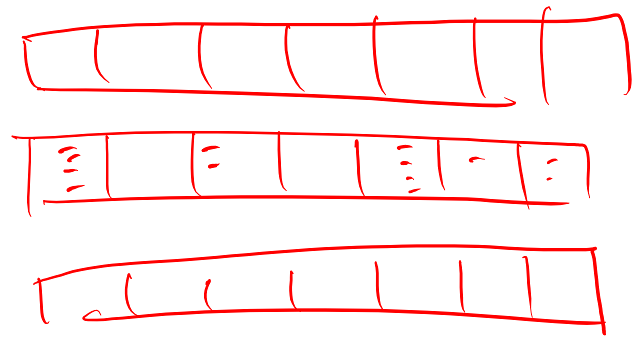
Third pass: stable bucket sort by hundred's digit

0	1	2	3	4	5	6	7	8	9
3 9 38 67	143			478	537		721		

Output:

3
9
38
67
143
478
537
721

Analysis



Input size: n

Number of buckets = Radix: B

Number of passes = "Digits": P

Work per pass is 1 bucket sort: $O(B + n)$

asymptotic $(n \to \infty)$

Total work is $O(P(B+n))$

Compared to comparison sorts, sometimes a win, but often not

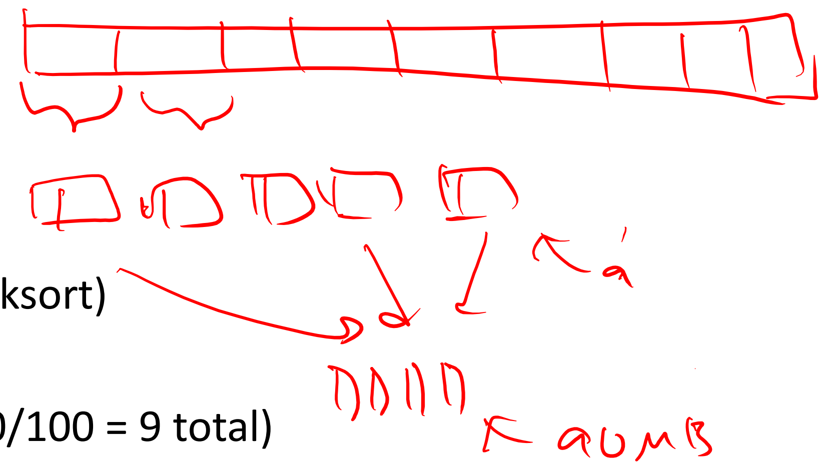
- Example: Strings of English letters up to length 15
  - Run-time proportional to: 15*(52 + n)
  - This is less than n log n only if n > 33,000
  - Of course, cross-over point depends on constant factors of the implementations
    - And radix sort can have poor locality properties

Comments on Sorting Algorithms

Sorting massive data

- Need sorting algorithms that minimize disk/tape access time:
  - Quicksort and Heapsort both jump all over the array, leading to expensive random disk accesses
  - Merge sort scans linearly through arrays, leading to (relatively) efficient sequential disk access
- Merge sort is the basis of massive sorting
- Merge sort can leverage multiple disks

External Merge Sort



- Sort 900 MB using 100 MB RAM
  - Read 100 MB of data into memory
  - Sort using conventional method (e.g. quicksort)
  - Write sorted 100MB to temp file
  - Repeat until all data in sorted chunks (900/100 = 9 total)
- Read first 10 MB of each sorted chuck, merge into remaining 10 MB
  - writing and reading as necessary
  - Single merge pass instead of *log n*
  - Additional pass helpful if data much larger than memory
- Parallelism and better hardware can improve performance
- Distribution sorts (similar to bucket sort) are also used

Wrap-up on Sorting

- Simple $O(n^2)$ sorts can be fastest for small n
  - Insertion sort (latter linear for mostly-sorted)
  - Good "below a cut-off" for divide-and-conquer sorts
- $O(n \log n)$ sorts
  - Heap sort, in-place, not stable, not parallelizable
  - Merge sort, not in place but stable and works as external sort
  - Quick sort, in place, not stable and $O(n^2)$ in worst-case
    - Often fastest, but depends on costs of comparisons/copies
- $\Omega(n \log n)$ is worst-case and average lower-bound for sorting by comparisons
- Non-comparison sorts
  - Bucket sort good for small number of possible key values
  - Radix sort uses fewer buckets and more phases
- Best way to sort? it depends!

design decisions

Complexity Theory: P vs NP

Just a small taste of Complexity Theory

“Easy” Problems for the Computer

Sorting a list of n numbers

$O(n \log n)$

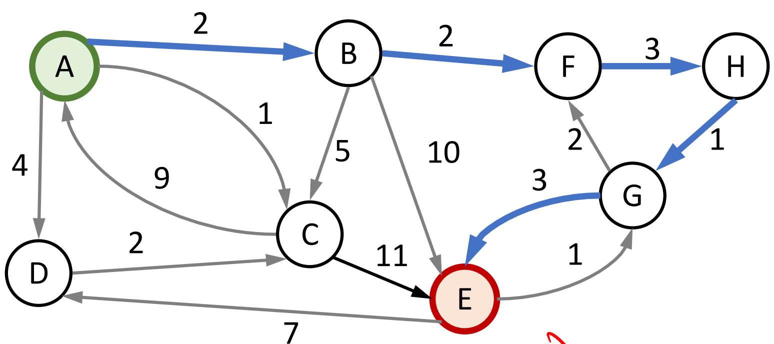
Multiplying two n x n matrices

$O(n^3)$

$$\begin{array}{c c c c c c} n & \begin{bmatrix} 3 & 5 & 2 & 7 \\ 1 & 6 & 8 & 9 \\ 2 & 4 & 6 & 10 \\ 9 & 3 & 2 & 12 \end{bmatrix} & \begin{bmatrix} 1 & 5 & 5 & 4 \\ 5 & 12 & 8 & 6 \\ 7 & 6 & 1 & 5 \\ 9 & 23 & 5 & 8 \end{bmatrix} & = & \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} & n \\ & n & n & & n & \end{array}$$

“Easy” Problems for the Computer

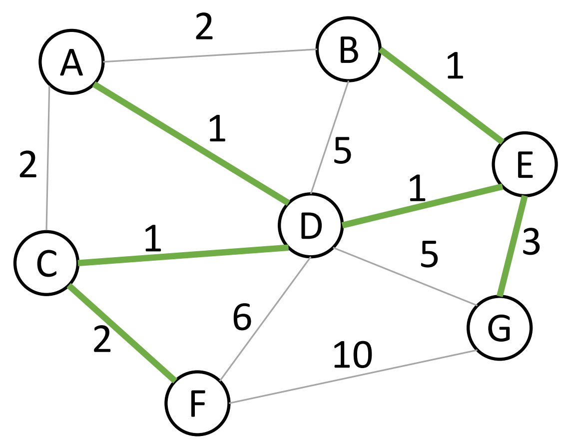
Shortest Path Algorithm



$O(|V|^2)$

Edsgar Dijkstra

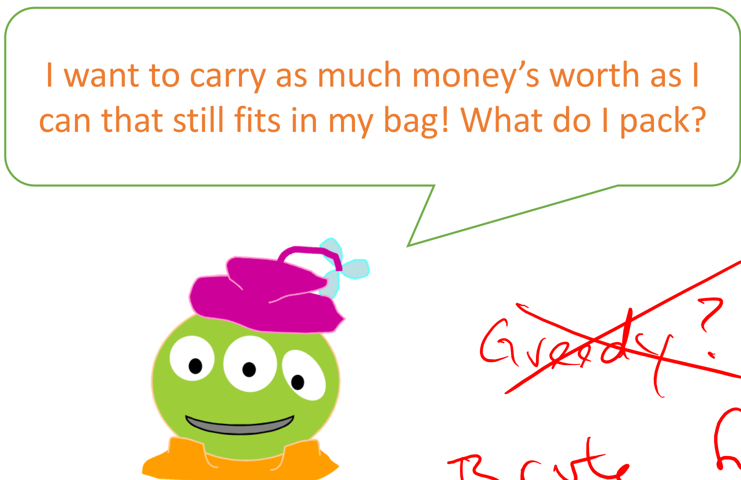
Minimum Spanning Tree Algorithms



$O(|V|\log|V| + |E|\log|V|)$

“Hard” Problems for the Computer

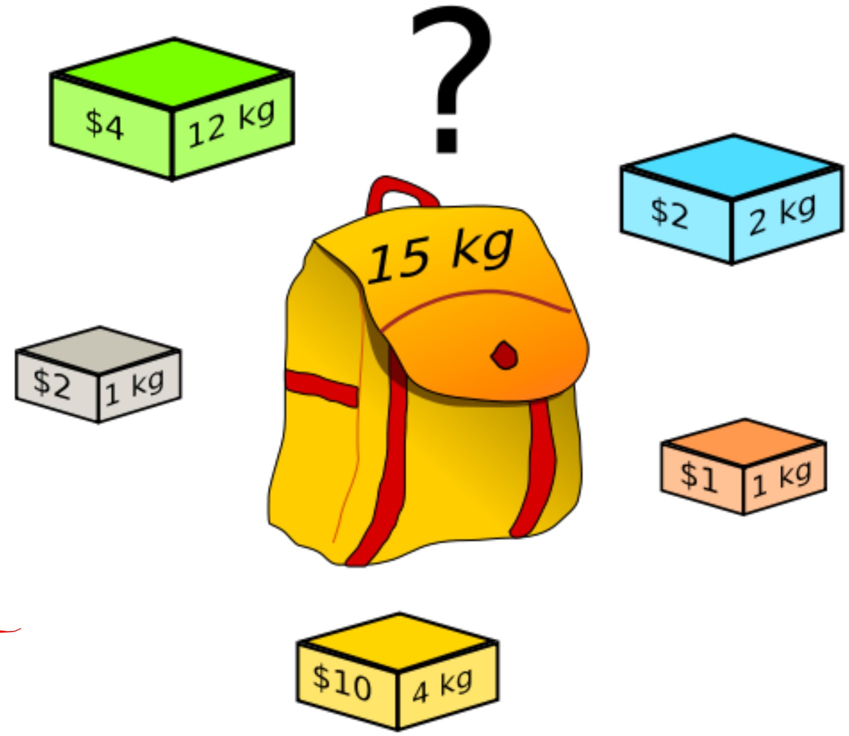
The Knapsack Problem



~~Greedy~~ ?

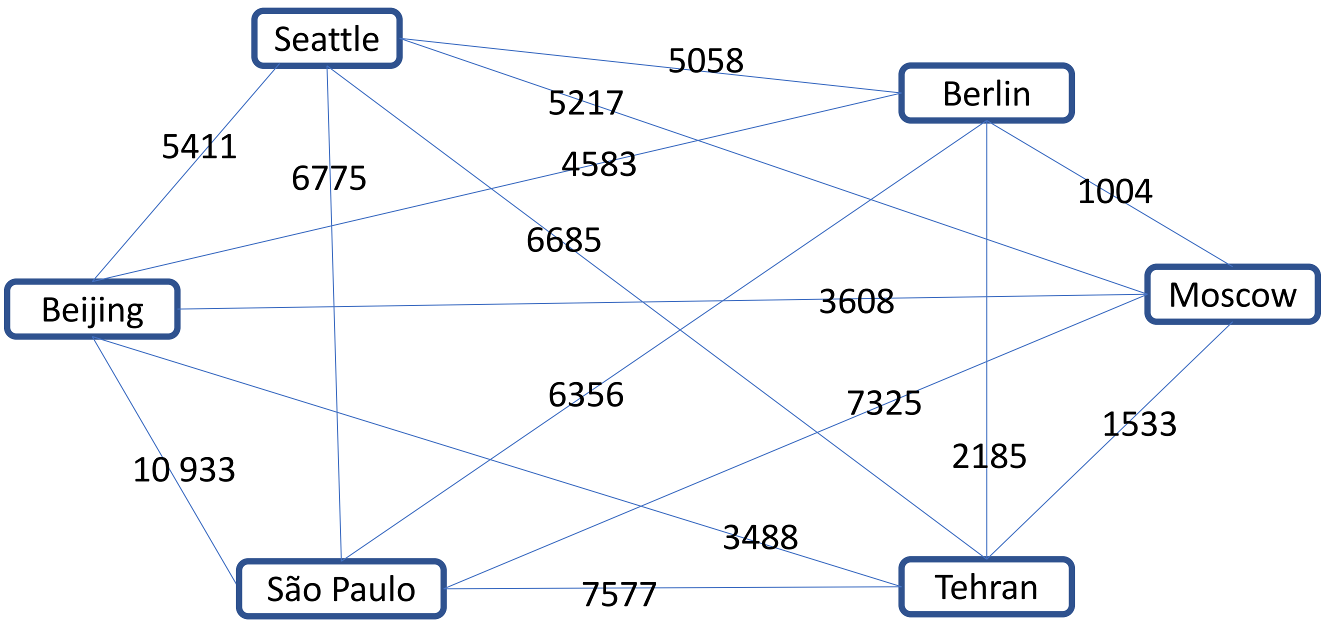
Brute force

↳ $O(n!)$



“Hard” Problems for the Computer

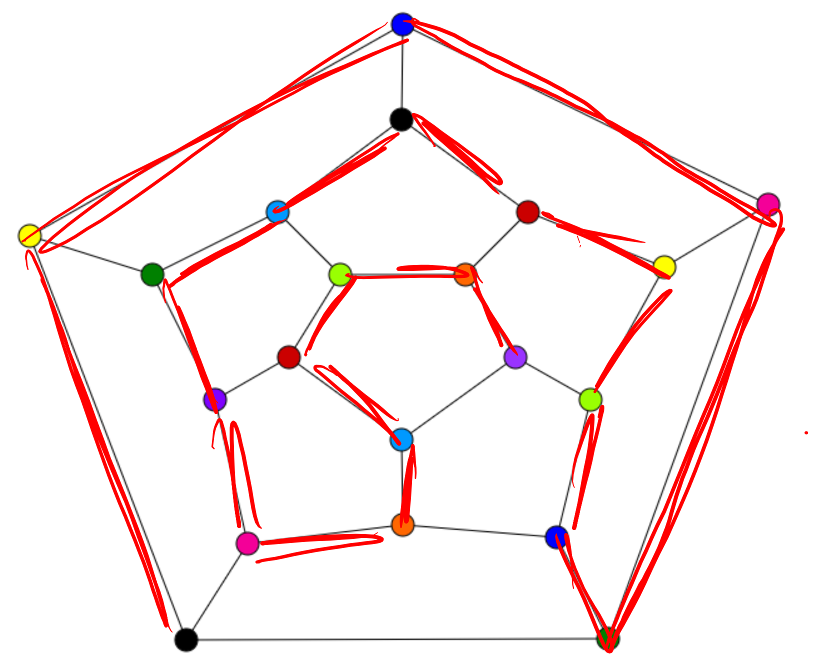
The Traveling Salesperson Problem



“Hard” Problems for the Computer

Find a Hamiltonian path (a path that visits each vertex exactly once)

(never mind weights
or even returning to
our starting point!)



Comparing n^2 vs 2^n

The alien's computer performs 10^9 operations/sec

	n = 10	n = 30	n = 50	n = 70
n^2	100 < 1 sec	900 < 1 sec	2500 < 1 sec	4900 < 1 sec
2^n	1024 < 1 sec	10^9 1 sec	10^{15} 11.6 days	10^{21} 31,688 years
$n!$	3628800 < 1 sec	10^{16} years (10^5 x age of the universe!)	10^{48} years	10^{83} years

"Easy" vs "Hard" Problems for the Computer

"Polynomial Time" = "Efficient"

$O(n^c)$ for some constant c

Is an algorithm "efficient" with…

$O(n)$? $O(n^2)$? $O(n^{10})$? $O(n \log n)$? $O(n^{\log n})$? $O(2^n)$? $O(n!)$?

$n^{(\log n)}$

$$\underbrace{O(n)? \checkmark \quad O(n^2)? \checkmark \quad O(n^{10})? \checkmark \quad O(n \log n)? \checkmark}_{\text{polynomial time}} \qquad \underbrace{O(n^{\log n})? \times \quad O(2^n)? \times \quad O(n!)? \times}_{\text{non-polynomial time}}$$

Polynomial Time?

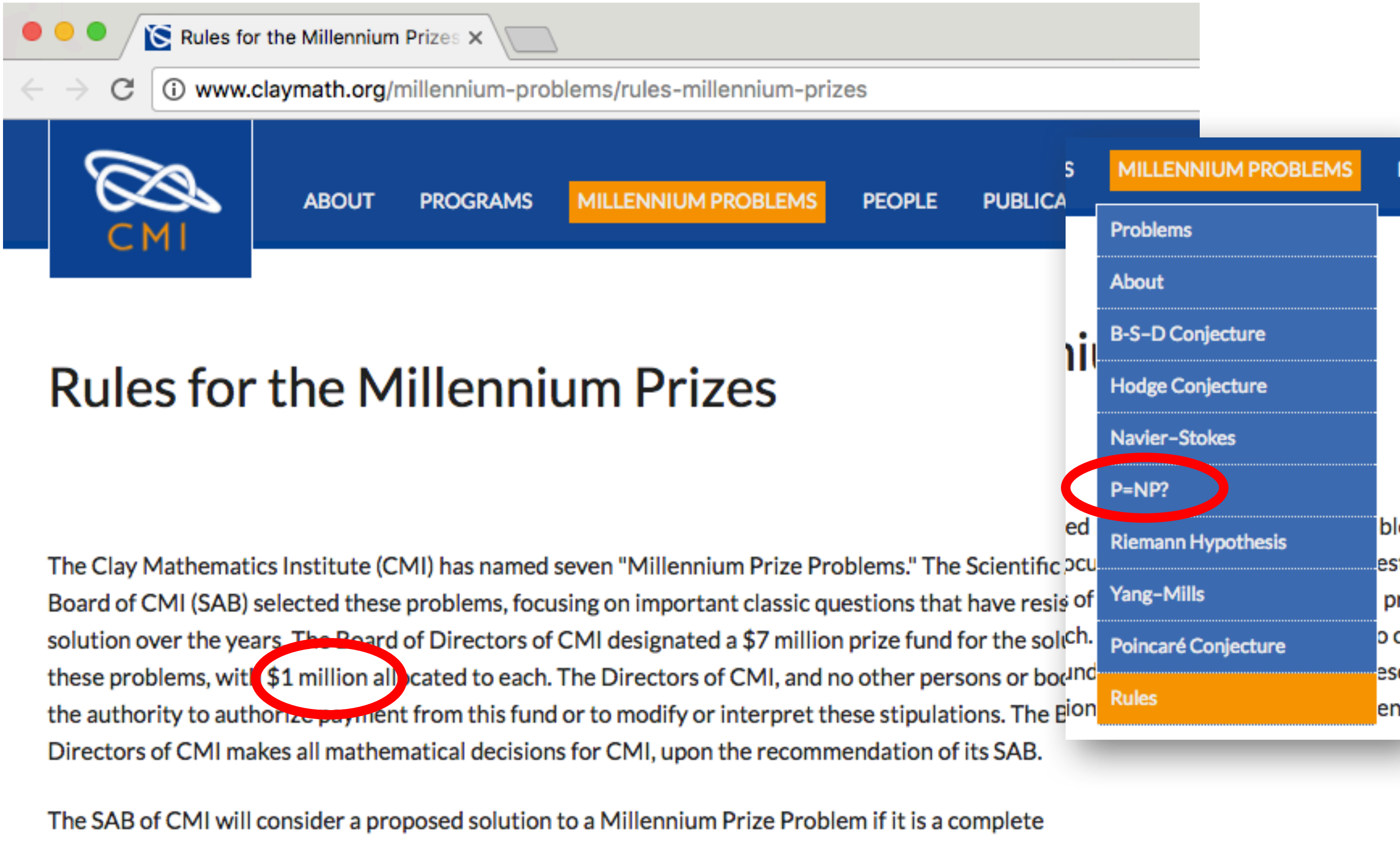
- So we know there are polynomial time algorithms to
  - Sort numbers
  - Multiply n x n matrices
  - Find the shortest path in a graph
  - Find the minimum spanning tree
  - … and more

P

- But the million dollar question is…
  are there polynomial time algorithms to solve
  - The Knapsack Problem?
  - The Traveling Salesperson Problem?
  - Finding Hamiltonian Paths?
  - … and thousands more!

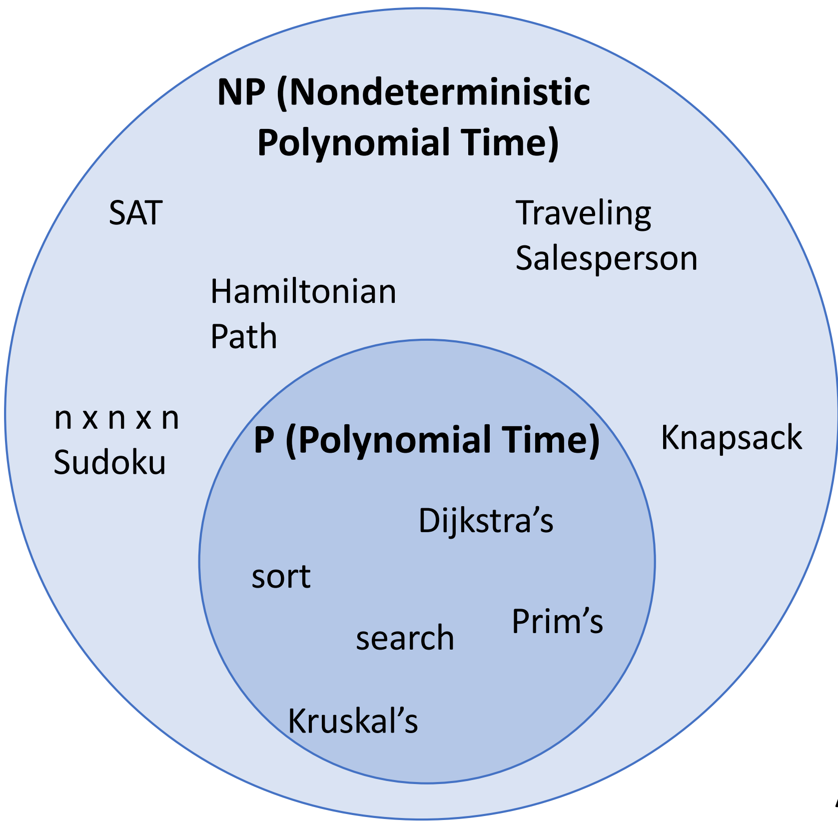
NP

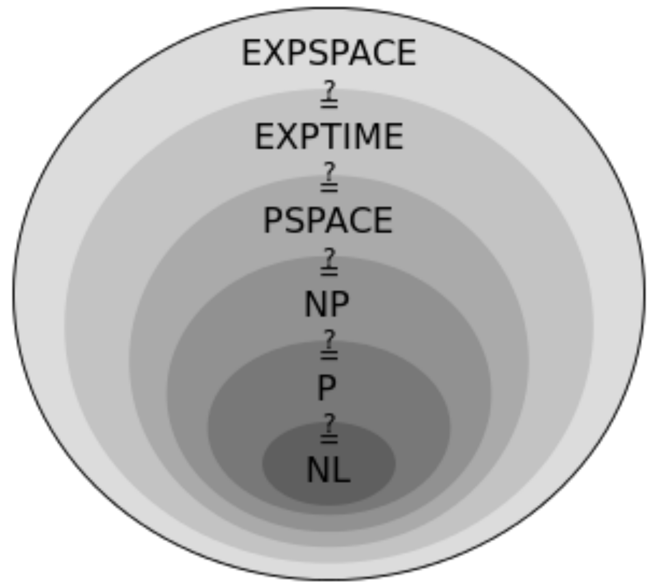
verify in P, but solve??


Rules for the Millennium Prizes
www.claymath.org/millennium-problems/rules-millennium-prizes
CMI
ABOUT
PROGRAMS
MILLENNIUM PROBLEMS
PEOPLE
PUBLICA
MILLENNIUM PROBLEMS
Problems
About
B-S-D Conjecture
Hodge Conjecture
Navier-Stokes
P=NP?
Riemann Hypothesis
Yang-Mills
Poincaré Conjecture
Rules
Rules for the Millennium Prizes
The Clay Mathematics Institute (CMI) has named seven "Millennium Prize Problems." The Scientific
Board of CMI (SAB) selected these problems, focusing on important classic questions that have resis
solution over the years. The Board of Directors of CMI designated a $7 million prize fund for the sol
these problems, with $1 million allocated to each. The Directors of CMI, and no other persons or bod
the authority to authorize payment from this fund or to modify or interpret these stipulations. The B
Directors of CMI makes all mathematical decisions for CMI, upon the recommendation of its SAB.
The SAB of CMI will consider a proposed solution to a Millennium Prize Problem if it is a complete

$P = NP?$

commonly believed
$P \neq NP$





And there are problems even harder than NP!

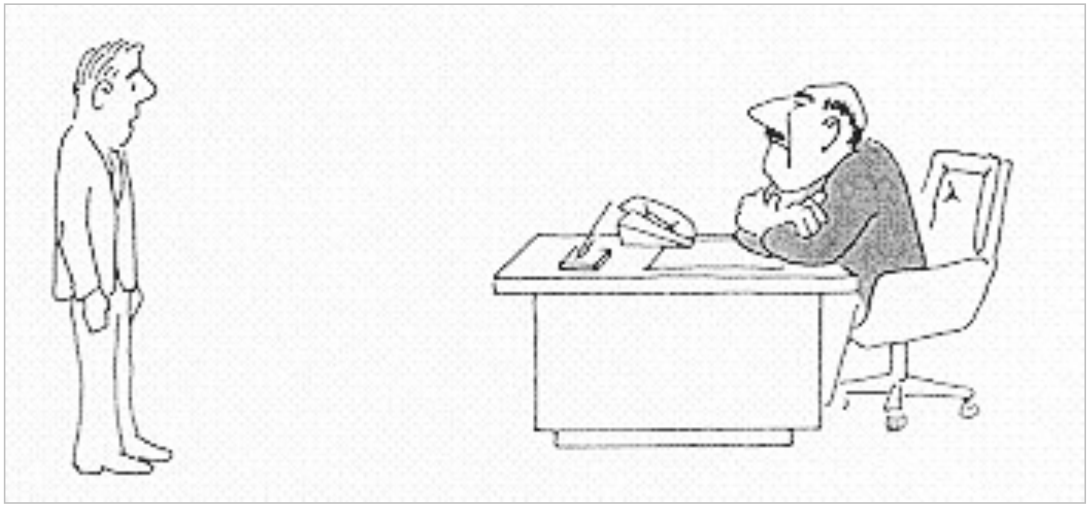
Relevance of P = NP

NP contains lots of problems we don't know to be in P

- Classroom Scheduling
- Packing objects into bins
- Scheduling jobs on machines
- Finding cheap tours visiting a subset of cities
- Finding good packet routings in networks
- *Decryption*

...

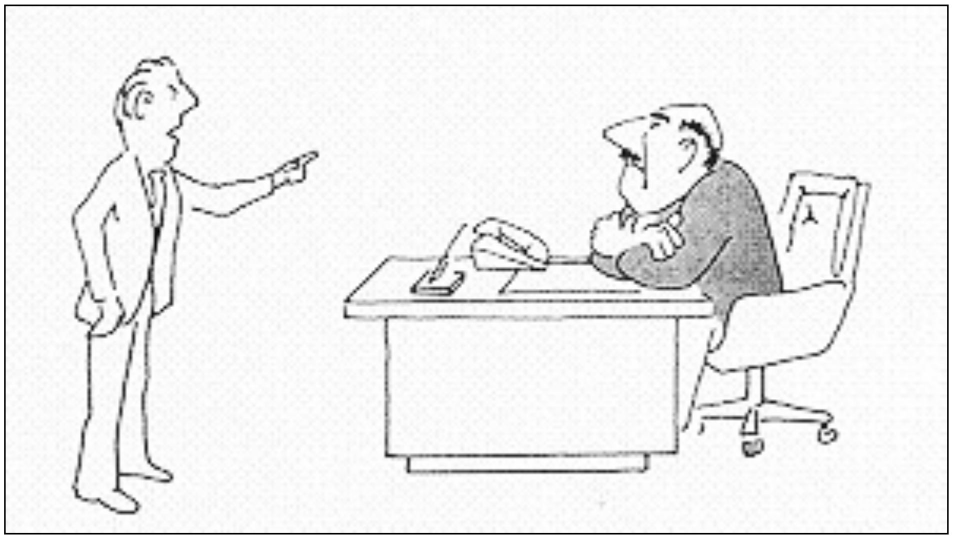
With this knowledge, we can avoid saying…



"I can't find an efficient algorithm.
I guess I'm too dumb."

Cartoon courtesy of "Computers and Intractability: A Guide to the Theory of NP-Completeness" by M. Garey and D. Johnson

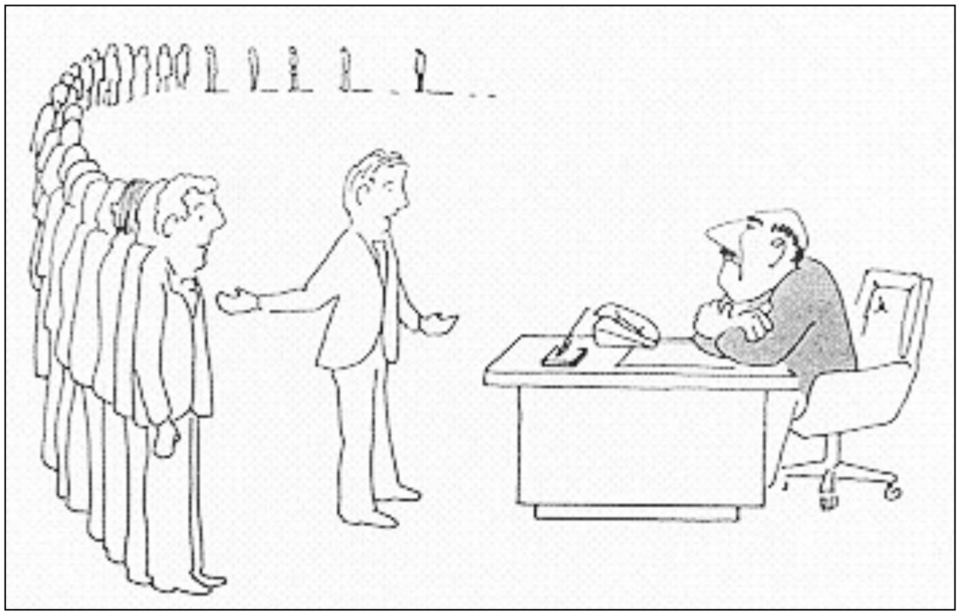
But know it isn't wise to say…



“I can't find an efficient algorithm because no such algorithm is possible!”

Cartoon courtesy of “Computers and Intractability: A Guide to the Theory of NP-Completeness” by M. Garey and D. Johnson

And, instead, prove it's in NP to then say…



"I can't find an efficient algorithm, but neither can all these famous people."

Cartoon courtesy of "Computers and Intractability: A Guide to the Theory of NP-Completeness" by M. Garey and D. Johnson

Preparing for Final Exam

Final Exam Study Strategies

- Problem sets
  - old exams
  - section
- Review slides
  - remind self of concepts
- List pro's & con's of each data structure or alg
- Study buddies
- Sleep
-

Practice Problem

```
Given a value 'x' and an array of integers, determine
whether two of the numbers add up to 'x'
```

Questions you should have asked me:

1) Is the array in any particular order?
2) Should I consider the case where adding two large numbers could cause an overflow?
3) Is space a factor, in other words, can I use an additional structure(s)?
4) Is this method going to be called frequently with different/the same value of 'x'?
5) About how many values should I expect to see in the array, or is that unspecified?
6) Will 'x' always be a positive value?
7) Can I assume the array won't always be empty, what if its null?

Practice Problem

Given a value ‘x’ and an array of integers, determine whether two of the numbers add up to ‘x’

- sorted? a particular order? O(n) if sorted
- how big are values?
- time vs space efficiency / data structures
  ↳ additional data structures okay?
- how many values? Do we know?
- can "x" be negative? & values in array?
- can the array be empty? Null values?