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?

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 Selection sort Shell sort …	Heap sort Merge sort Quick sort (avg) …		Bucket sort Radix sort	External sorting

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

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

Second pass: stable bucket sort by ten's digit

0	1	2	3	4	5	6	7	8	9

Third pass: stable bucket sort by hundred's digit

0	1	2	3	4	5	6	7	8	9

Output:

(extra space for scratch work / notes)

Analysis

Input size: *n*
Number of buckets = Radix: *B*
Number of passes = "Digits": *P*

Work per pass is 1 bucket sort:

Total work is

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

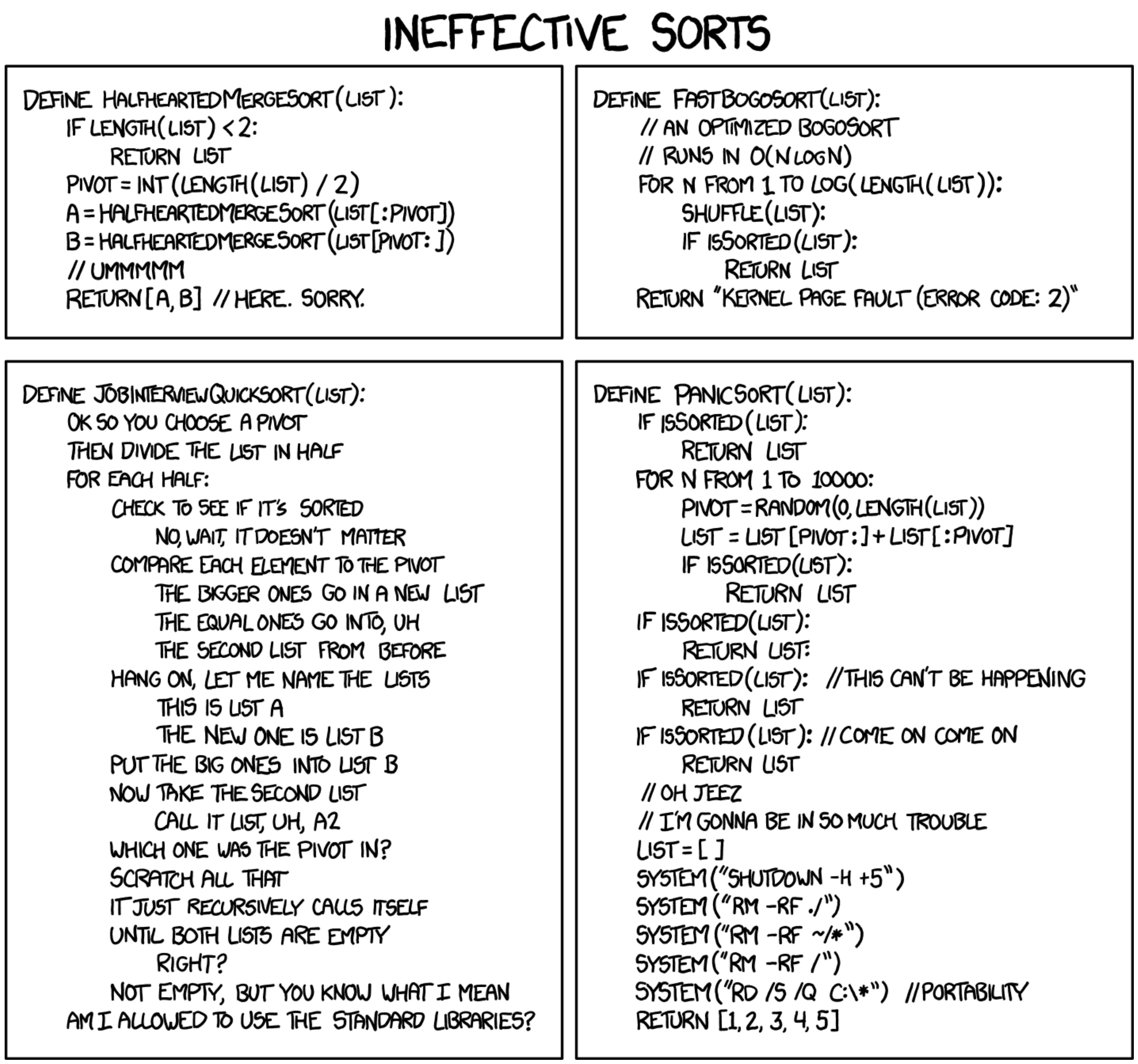
- Need sorting algorithms that minimize disk/tape access time:
  - Quicksort and Heapsort both jump all over the array, leading to random disk accesses
  - Merge sort scans linearly through arrays, leading to (relatively) sequential disk access
- Merge sort is the basis of massive sorting
- Merge sort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



```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST):  //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): //COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*")  //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```

Source: https://xkcd.com/1185/

Complexity Theory: P vs NP

Just a small taste of Complexity Theory

"Easy" Problems for the Computer

Sorting a list of n numbers

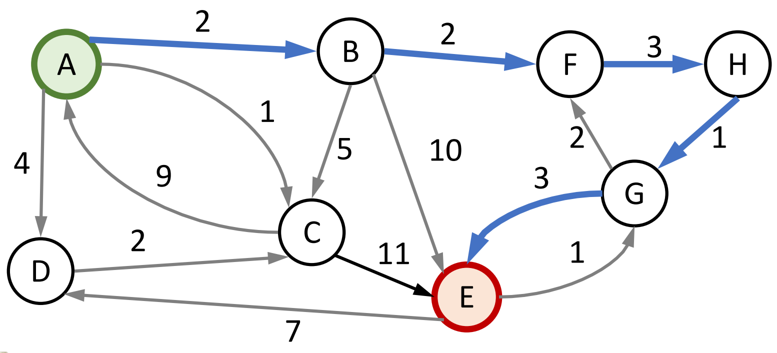
Multiplying two n x n matrices

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

$$\qquad n \qquad\qquad\qquad n \qquad\qquad\qquad n$$

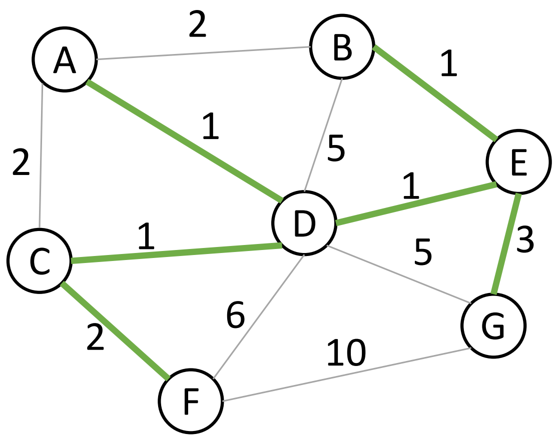
“Easy” Problems for the Computer

Shortest Path Algorithm



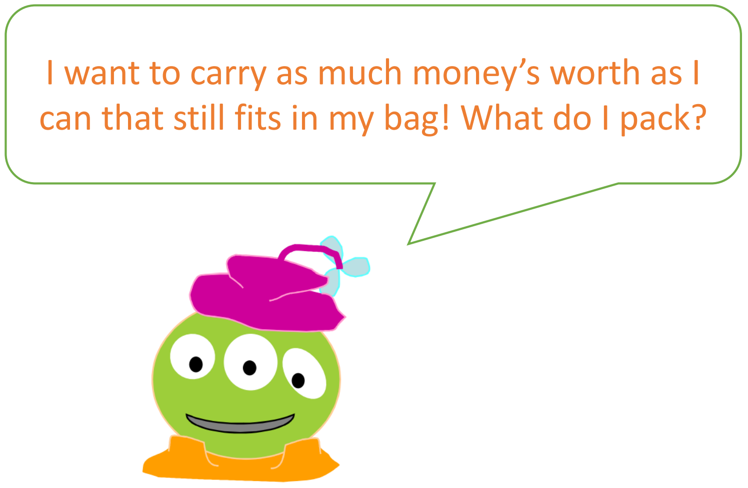
Edsgar Dijkstra

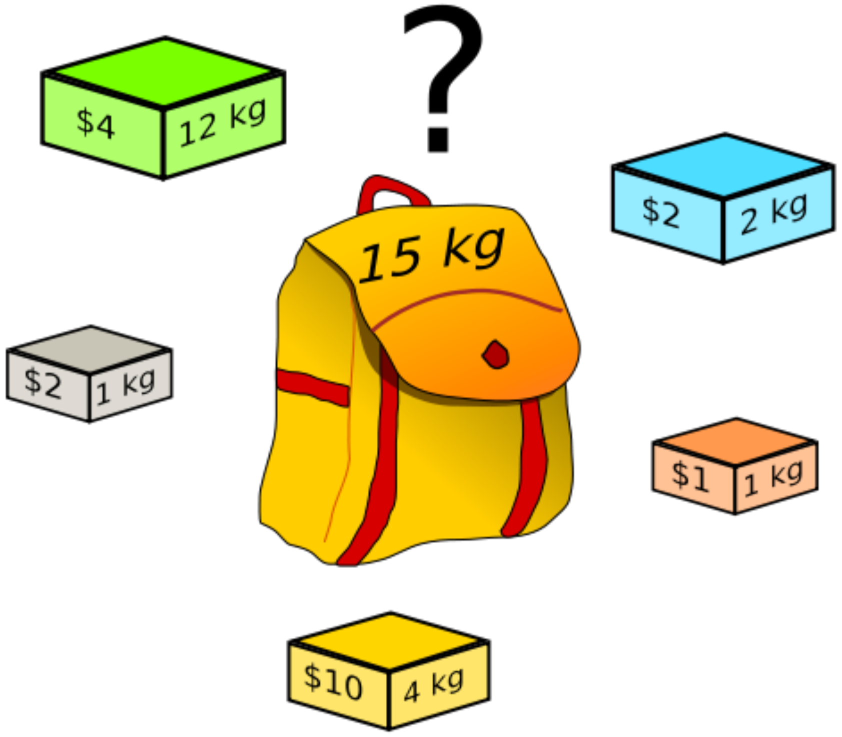
Minimum Spanning Tree Algorithms



“Hard” Problems for the Computer

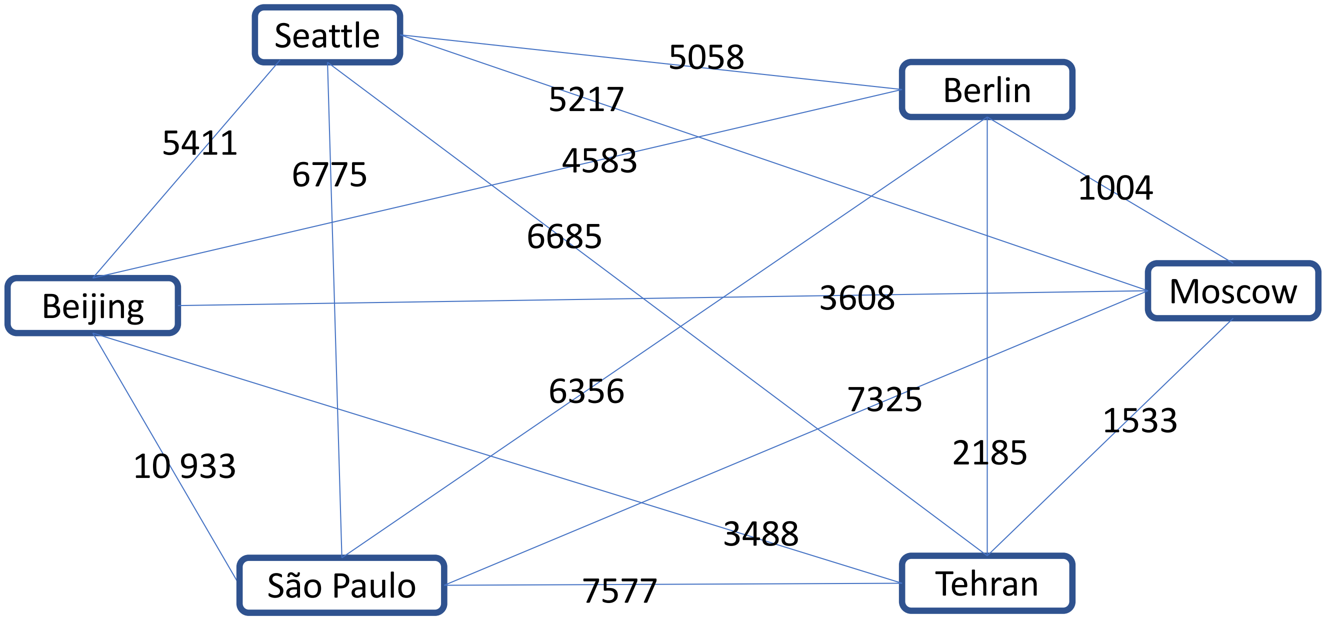
The Knapsack Problem


I want to carry as much money’s worth as I can that still fits in my bag! What do I pack?



“Hard” Problems for the Computer

The Traveling Salesperson Problem

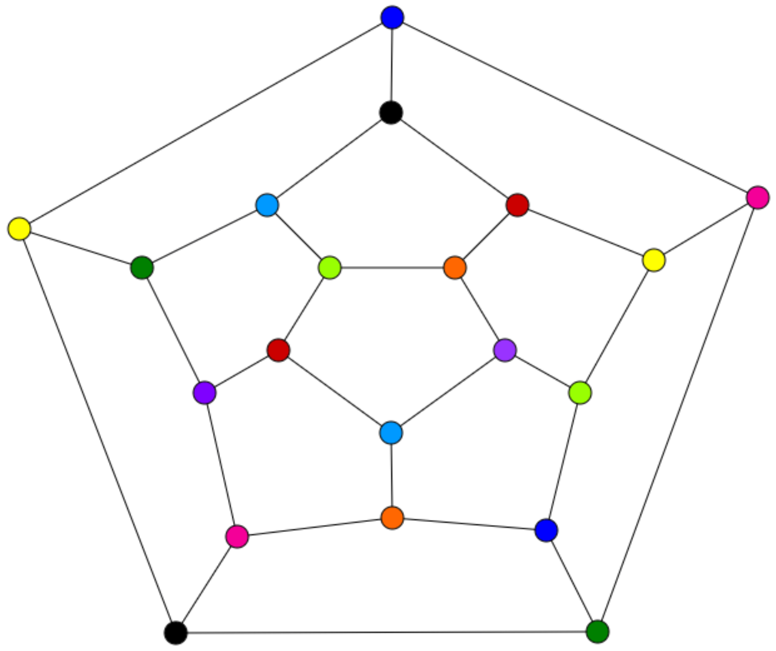


I’ll leave Seattle to sell goods, visiting each city only once, and return to Seattle. What’s the shortest route?

“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 < 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”

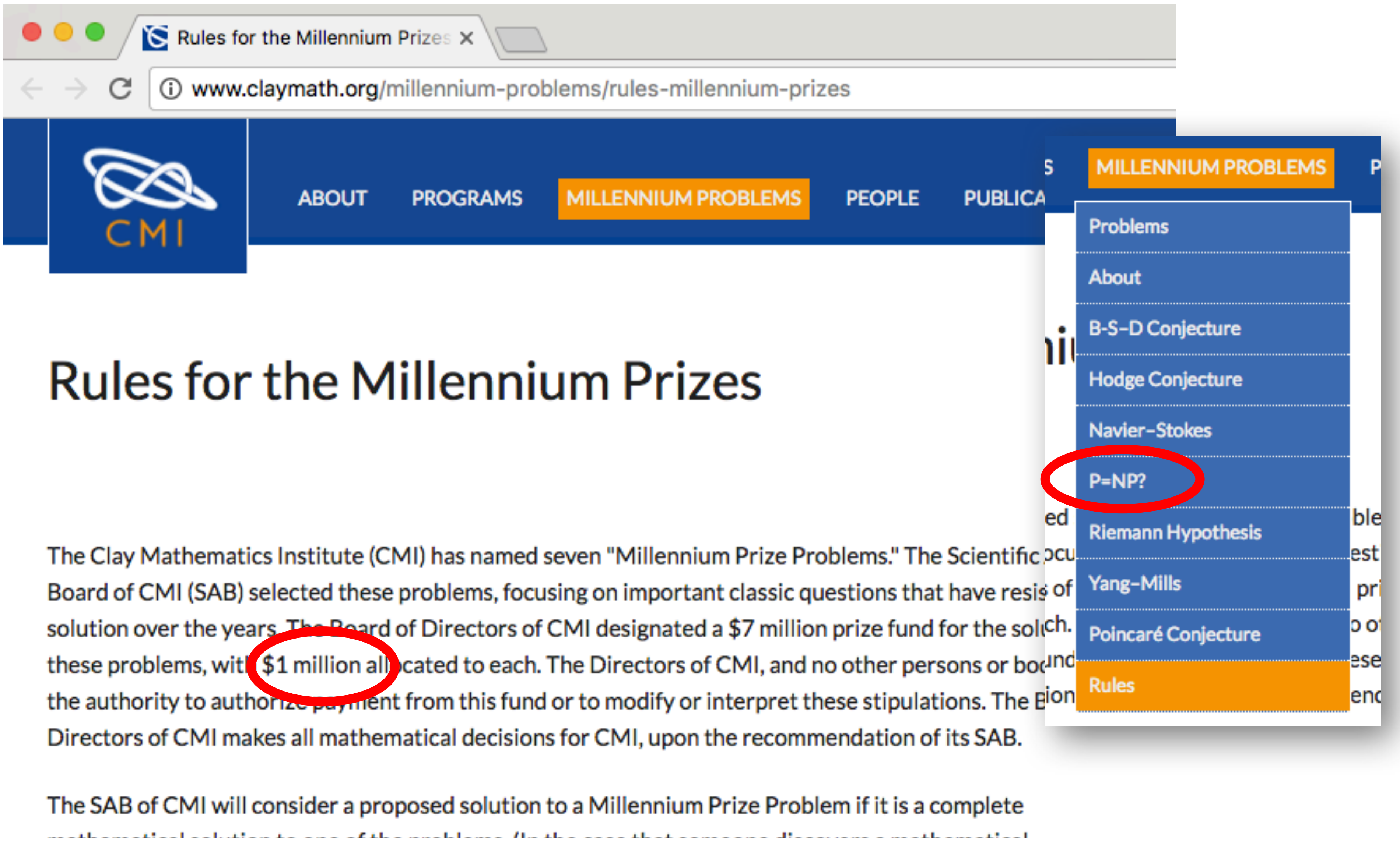
Is an algorithm “efficient” with…

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

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

- 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!

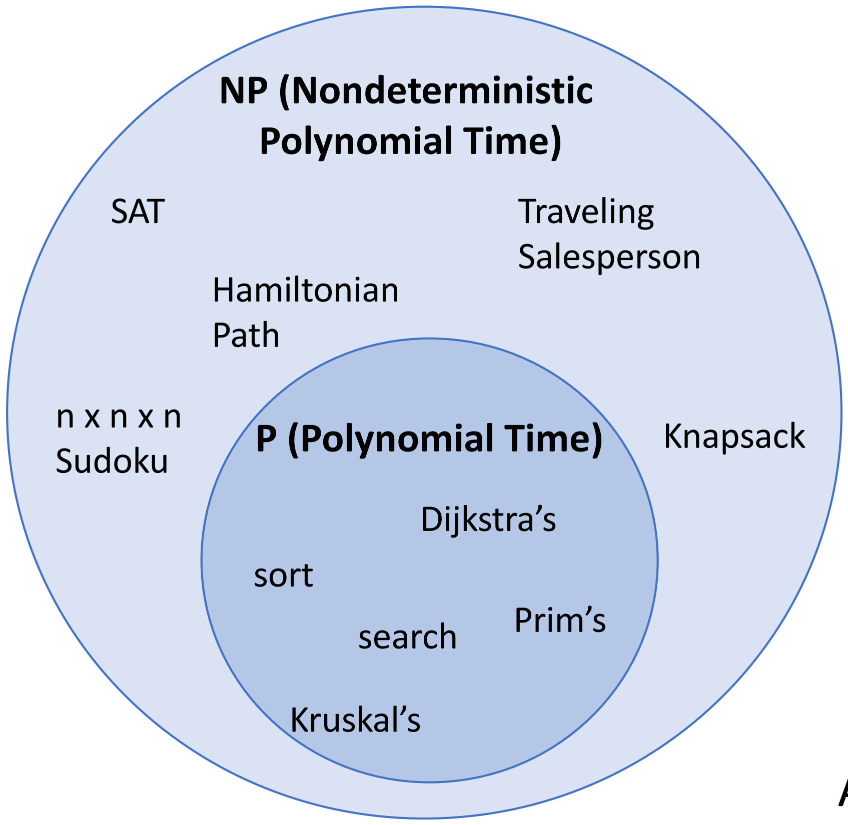


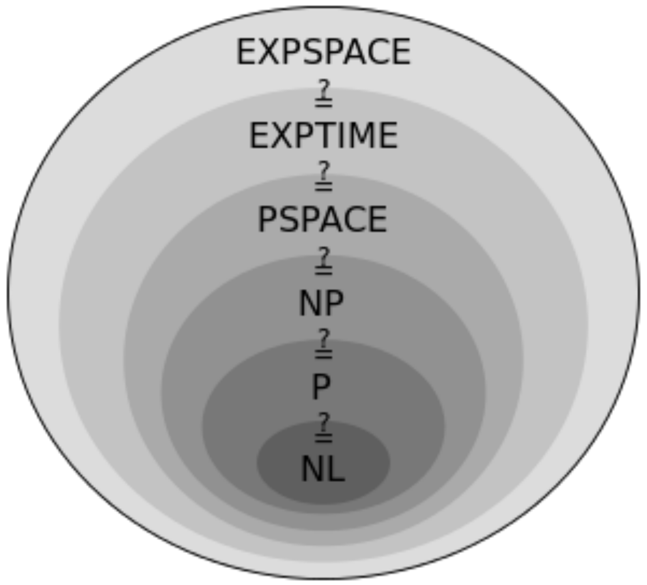
Rules for the Millennium Prizes

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?





And there are problems even harder than NP!

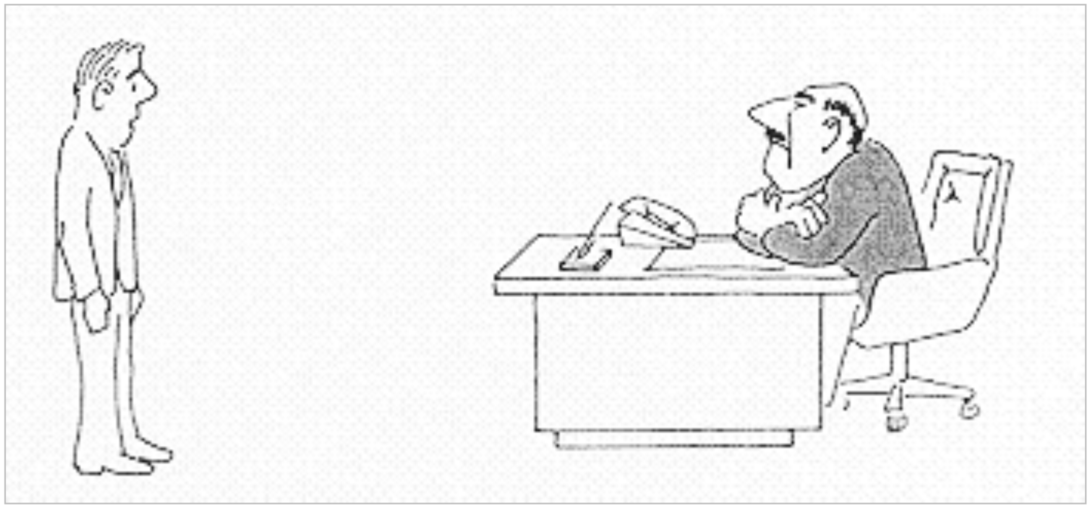
Relevance of P = NP

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*

  ...

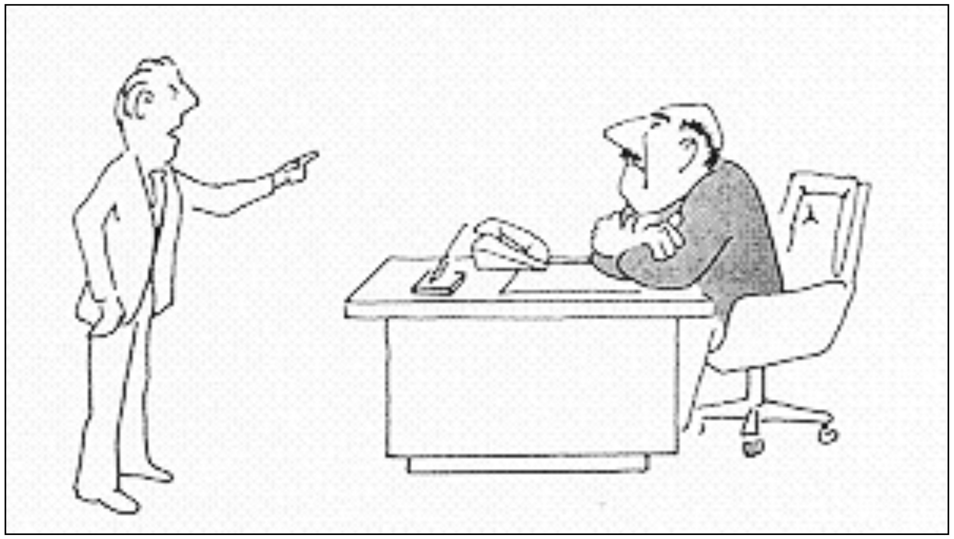
With this knowledge, we can avoid saying…



"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

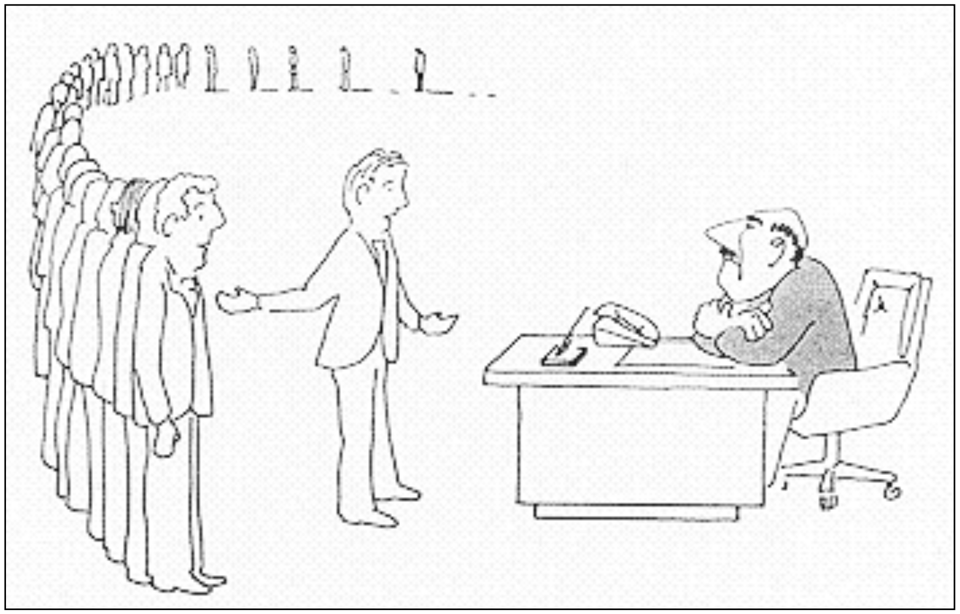
But know it isn’t wise to say…



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

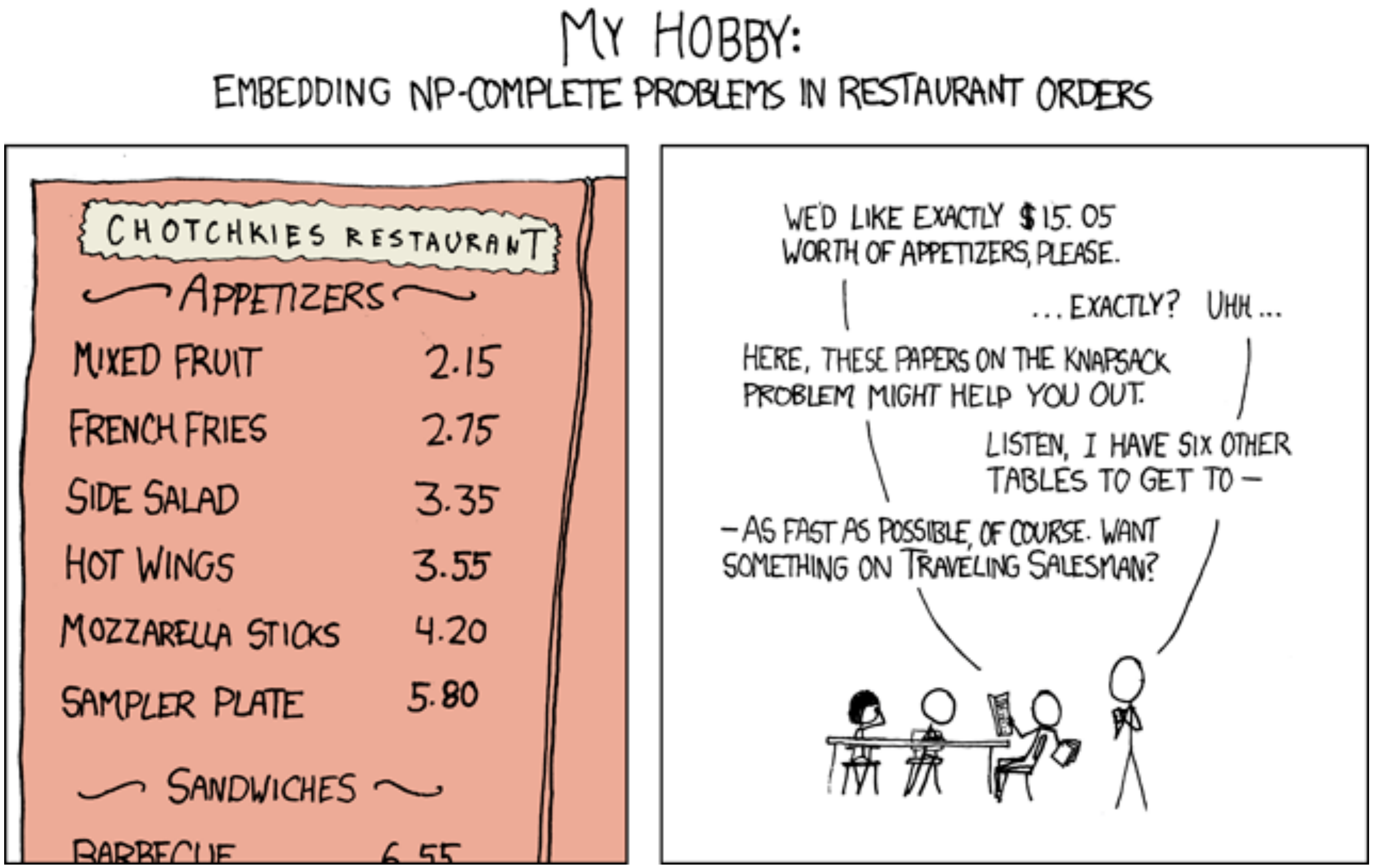
Cartoon courtesy of “Computers and Intractability: A Guide to the Theory of NP-Completeness” by M. Garey and D. Johnson

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


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Source: https://xkcd.com/287/

Preparing for Final Exam

Final Exam Study Strategies

Practice Problem

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