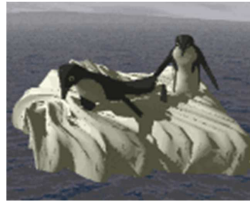


## The CS 5 Times

### CS 5 and Physics Penguins Stranded in Spaceship Crash



Wellington (AP): Two HMC penguins were missing after their spaceship lost power and crashed into the southern ocean. "The CS penguin had kindly offered a ride to her friend," blubbered a distraught professor, "and apparently he was fiddling with the flight computer just before takeoff. I don't know what I'll do for classroom examples now."

With weather worsening, there is little hope for rescue. A memorial service will be held Sunday in the Hoch-Shanahan freezer.

## This Week

### Homework 3:

- Reading
- Black lab (Problem 1) is **same** as **Gold** this week



You're ready  
for this today!

- Problem 2: Spel Chekking
- Problem 3: Word Break

You're ready  
for this today!

Thursday!

## Beyond LCS: Edit Distance

```
>>> ED("ATTATCG", "ACATTC")
```

4

ATTAT-CG  
A-CATTC-



The lower the edit  
distance the better!

## Beyond LCS: Edit Distance

```
>>> ED("ATTATCG", "ACATTC")
```

4

ATTAT-CG  
A-CATTC-

```
>>> ED("spam", "scramble")
```

5

sp\_am\_\_  
scramble

```
spam ->  
scam ->  
scram ->  
scramb -> scrambl -> scramble
```



## Beyond LCS: Edit Distance

```
>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
scramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '':
        return ???
    elif S2 == '':
        return ???
    elif S1[0] == S2[0]:
        return ???
    else: # substitute, insert, or delete!
```

[Worksheet!](#)

## Problem 2 This Week...

New! Millisoft Office, featuring

Millisoft Sentence (word processor)  
Energy Dot (presentation software)  
Succeed (spreadsheet software)



Gill Bates

Spelling A La Millisoft (SPAM)



## Aside: Another Way to map

```
def doubleList1(L):
    return list(map(lambda x: 2*x, L))

def doubleList2(L):
    return [2*x for x in L]

def doubleListFiltered1(L):
    return list(map(lambda x: 2*x, filter(lambda x: x != 42, L)))

def doubleListFiltered2(L):
    return [2*x for x in L if x != 42]
```

These are called *list comprehensions*!



## Conditionalizing lambda

```
>>> list(map(lambda x: "nice" if x == 42 else "blech!",
             [42, 7, 6, 42, 3]))

["nice", "blech!", "blech!", "nice", "blech!"]

>>> list(map(lambda x: "HM" if x == 42
             else "PO" if x == 47
             else x,
             [42, 7, 6, 47, 3]))
['HM', 7, 6, 'PO', 3]

>>> ["HM" if x == 42 else "PO" if x == 47 else x
     for x in [42, 7, 6, 47, 3]]
['HM', 7, 6, 'PO', 3]
```

## “Easy” Problems

Sorting a list of  $n$  numbers: [42, 3, 17, 26, ..., 100]

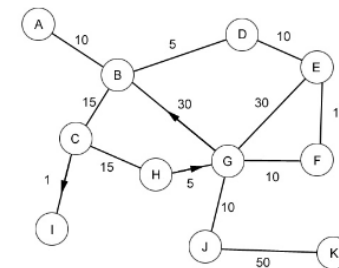
$$n \log_2 n$$

Multiplying two  $n \times n$  matrices:

$$n \begin{pmatrix} 3 & 5 & 2 & 7 \\ 1 & 6 & 8 & 9 \\ 2 & 4 & 6 & 10 \\ 9 & 3 & 2 & 12 \end{pmatrix} \begin{pmatrix} 1 & 5 & 5 & 4 \\ 5 & 12 & 8 & 6 \\ 7 & 6 & 1 & 5 \\ 9 & 23 & 5 & 8 \end{pmatrix} = \begin{pmatrix} \phantom{0} \\ \phantom{0} \\ \phantom{0} \\ \phantom{0} \end{pmatrix} n$$

## “Easy” Problems

The Shortest Path Problem (i.e. “Google Maps”)



Edsger Dijkstra

## “Easy” Problems

“Polynomial Time” = “Efficient”

$$n, n^2, n^3, n^4, n^5, \dots$$



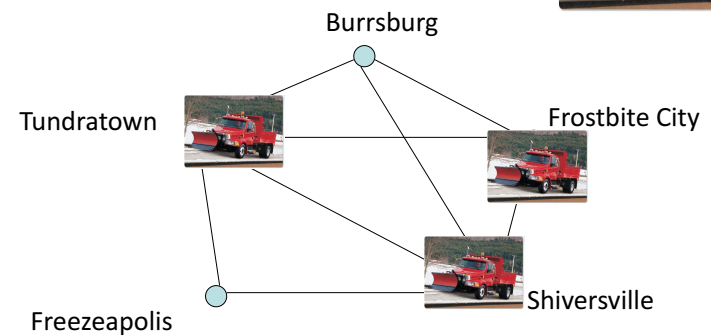
How about something like  $n \log_2 n$ ?

sorting  
matrix  
multiplication  
shortest paths

The “class” P

## “Hard” Problems

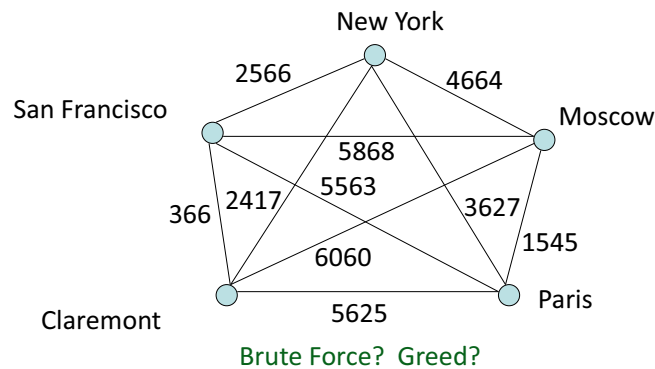
Snowplows of Northern Minnesota



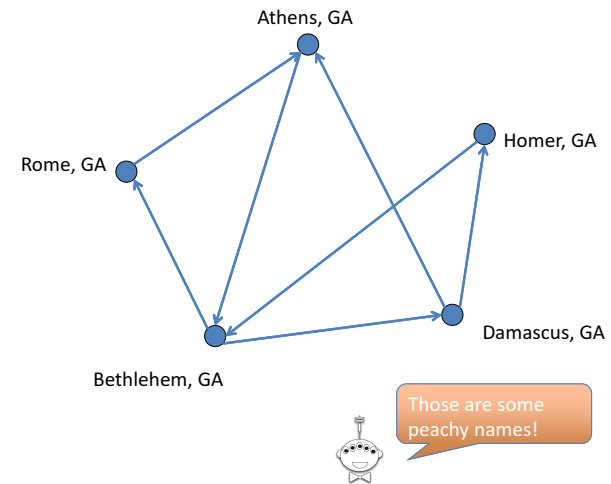
Brute-force? Greed?

## “Hard” Problems

### The Traveling Salesperson Problem



## The Hamiltonian Path Problem



## $n^2$ Versus $2^n$



The Geoff-O-Matic 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
$n!$	< 1 sec	$10^{16}$ years	$10^{57}$ years	$10^{93}$ years

## Here's an Idea!

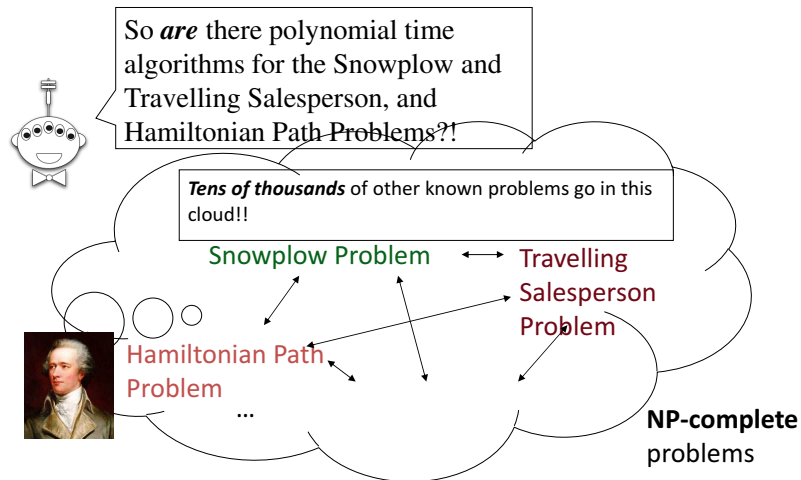
Let's just buy a computer that's twice as fast!



Size of largest problem solvable with “old” computer in one hour =  $S$

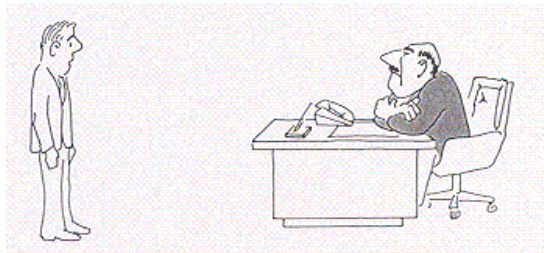
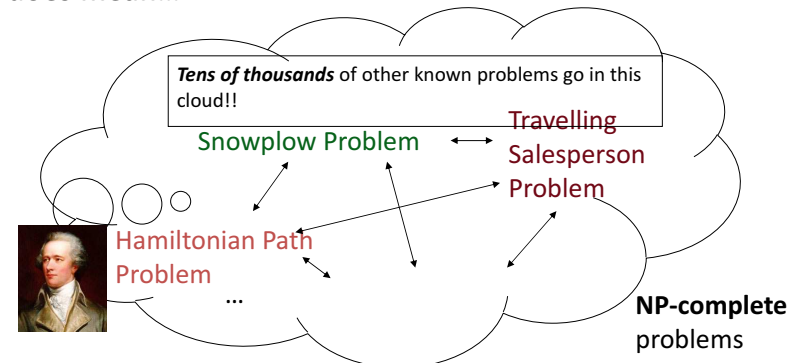
	$n$	$n^2$	$n^3$	$n^5$	$2^n$	$n!$
Size of largest problem solvable with “new” twice-as-fast computer in one hour	<b>2S</b>	<b>1.41S</b>	<b>1.26S</b>	<b>1.15S</b>	<b>S+1</b>	<b>S</b>

## Snowplows and Traveling Salesperson Revisited!



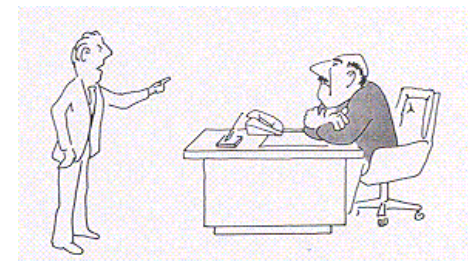
## Snowplows and Traveling Salesperson Revisited!

If a problem is NP-complete, it doesn't necessarily mean that it *can't* be solved in polynomial time. It does mean...



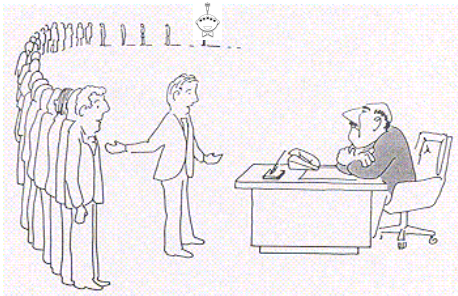
"I can't find an efficient algorithm. I guess I'm too dumb."

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



"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

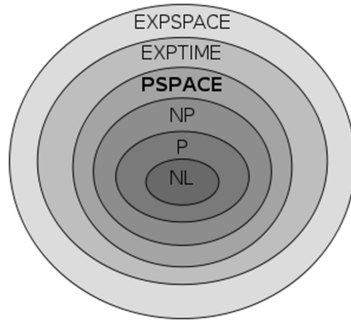


"I can't find an efficient algorithm, but neither can all these famous people."

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

## Are There Problems That Are Even Harder Than NP-Complete?

Kryptonite problems?

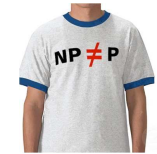


### Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each. During the *Millennium Meeting* held on May 24, 2000 at the Collège de France, Timothy

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- **P vs NP**
- Riemann Hypothesis
- Yang-Mills Theory

\$1 million



Vinay Deolalikar

## Is LCS NP-Complete?

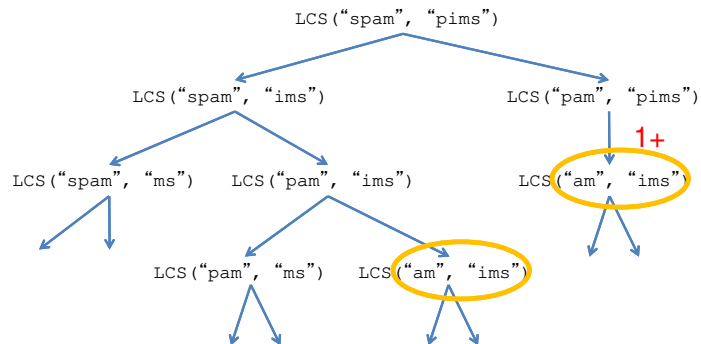
```
def LCS(S1, S2):
    if S1 == "" or S2 == "":
        return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
```

### Demo LCS

Two strings of length 100 nucleotides each...

```
>>> steps = 2**100
>>> speed = 3 * 10**9
>>> seconds = steps / speed
>>> years = seconds / (60*60*24*365.25)
>>> years
13389807845846.213 13 trillion years!
>>>
```

60 seconds per minute  
60 minutes per hour  
24 hours per day  
365.25 days per year



## Dictionary Revisited

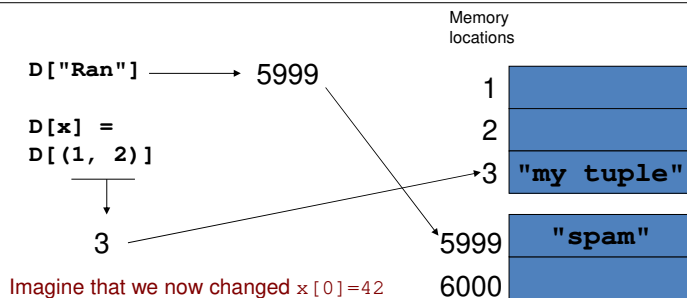
```

>>> D = {"spam": "yummy!", (42, 42): "an important point"}
>>> D = {}
>>> D["spam"] = "yummy!"
>>> D[(42, 42)] = "an important point"
>>> D[[1, 2]] = "but this is bad"
BARF!
>>> "spam" in D
True
>>> 42 in D
False
>>> (42, 42) in D
True
>>> D[(42, 42)]
"an important point"
  
```

## How Dictionaries Work: Hashing

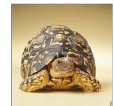
```

>>> D
{"Ran": "spam", ...}
>>> x = (1, 2)
>>> D[x] = "my tuple"
>>> D
{"Ran": "spam", (1, 2): "my tuple"}
  
```



```

def LCS(S1, S2):
    if S1 == "" or S2 == "":
        return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
  
```



Old slow version

```

memo = {} # global empty dictionary
def fastLCS(S1, S2):
    if (S1, S2) in memo:
        return memo[(S1, S2)]
    elif S1 == "" or S2 == "":
        answer = 0
    elif S1[0] == S2[0]:
        answer = 1 + LCS(S1[1:], S2[1:])
    else:
        answer = max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
    memo[(S1, S2)] = answer
    return answer
  
```



New fast "memoized" version

## Changing change

---

```
def change(value, coins):
    if value <= 0:
        return 0
    elif coins == []:
        return float("inf")

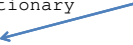
    loseIt = change(value, coins[1:])
    if value < coins[0]:
        return loseIt
    else:
        useIt = 1 + change(value - coins[0], coins)
        return min(useIt, loseIt)
```

## Changing change

---

```
memo = {}          # Empty dictionary
def fastChange(value, coins):
    if (value, coins) in memo:
        return memo[(value, coins)]
    elif value <= 0:
        return 0
    elif coins == []:
        return float("inf")

    # Finish writing this!
```

 coins must be a tuple rather than a list!

Geoff's solution coming up...

[Worksheet!](#)