

“It ain’t no good if it ain’t
snappy enough.”
(Efficient Computations)

COS 116: 2/20/2007

Adam Finkelstein



Administrative stuff

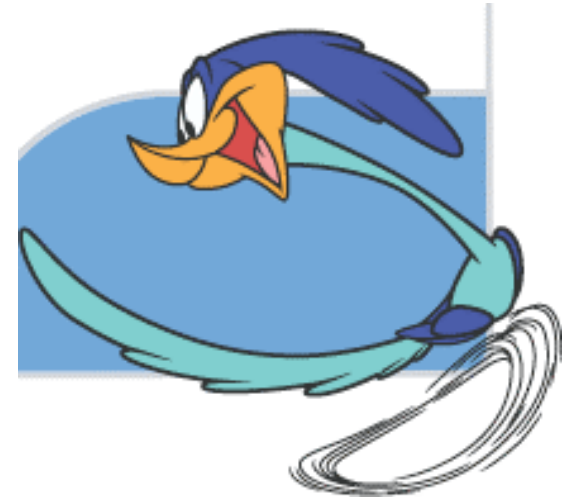
- Readings from course web page
- Feedback form on course web page
- New blogging assignment for this week. (See handout.)
- Reminder for this week's lab:
 Come with robots, cables.
- Make sure you understand pseudocode.
 Come to lab with questions.
- Preview of upcoming labs:
 - This week: controlling Scribbler (maze, ...)
 - Audio
 - Scribbler art/music/dance
 - Computer graphics
 - ...



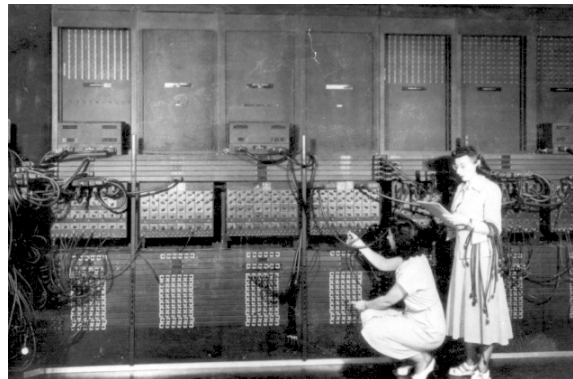
Discussion

1. In what ways (according to Brian Hayes) is the universe like a cellular automaton?
2. What different ways does Brooks describe for a robot to “orient” itself? Did your experiments with Scribbler give you insight into any of them?

Question:
How do we measure the
“speed” of an algorithm?

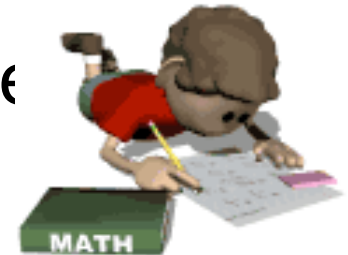


- Ideally, should be independent of:
 - machine
 - technology



“Running time” of an algorithm

- Definition: the number of “elementary operations” performed by the algorithm
- Elementary operations: $+$, $-$, $*$, $/$, assignment, evaluation of conditionals
- “Speed” of computer: number of elementary steps it can perform per second
 - Simplified definition
 - Do *not* consider this in “running time” of algorithm



Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- for $i = 2$ to n do

{

if ($A[i] < A[best]$) then

{ $best \leftarrow i$ }

}

1 assignment & 1 comparison
= 2 operations per loop iteration

Uses at most $2(n - 1) + 1$ operations (roughly = $2n$)

Number of iterations

Initialization

Selection Sort

Do for $i = 1$ to $n - 1$

{

Find cheapest bottle among those numbered i to n

← About $2(n - i)$ steps

Swap that bottle and the i 'th bottle.

← 3 steps

}

- For the i 'th round, takes at most $2(n - i) + 3$
- To figure out running time, need to figure out how to sum $(n - i)$ for $i = 1$ to $n - 1$
...and then double the result.

Gauss's trick : Sum of $(n - i)$ for $i = 1$ to $n - 1$

$$\begin{aligned} S &= 1 + 2 + \dots + (n - 2) + (n - 1) \\ + S &= (n - 1) + (n - 2) + \dots + 2 + 1 \\ \hline 2S &= n + n + \dots + n + n \end{aligned}$$

$n - 1$ times

$$2S = n(n - 1)$$

- So total time for selection sort is
 $\leq n(n - 1) + 3n$
(for large n , roughly $= n^2$)



“20 Questions”:

Guess the number I have in mind.

- My number is an integer from 1 to n .
- You can only ask yes/no questions. What’s the first Q?
- Binary Search Algorithm:
First Question: “Is the number $< n / 2$?”
- Answer halves the range of possible numbers!

$$\underbrace{1, 2, \dots, \frac{n}{2} - 1, \frac{n}{2}}_{\text{Repeat}} \underbrace{\frac{n}{2}, \frac{n}{2} + 1, \dots, n}_{\text{Repeat}}$$

Exercise: Express as pseudocode.

How many times...?

Brief detour: Logarithms (CS view)

- $\log_2 n = K$ means $2^{K-1} \leq n < 2^K$
- In words: K is the number of times you need to divide n by 2 in order to get a number ≤ 1

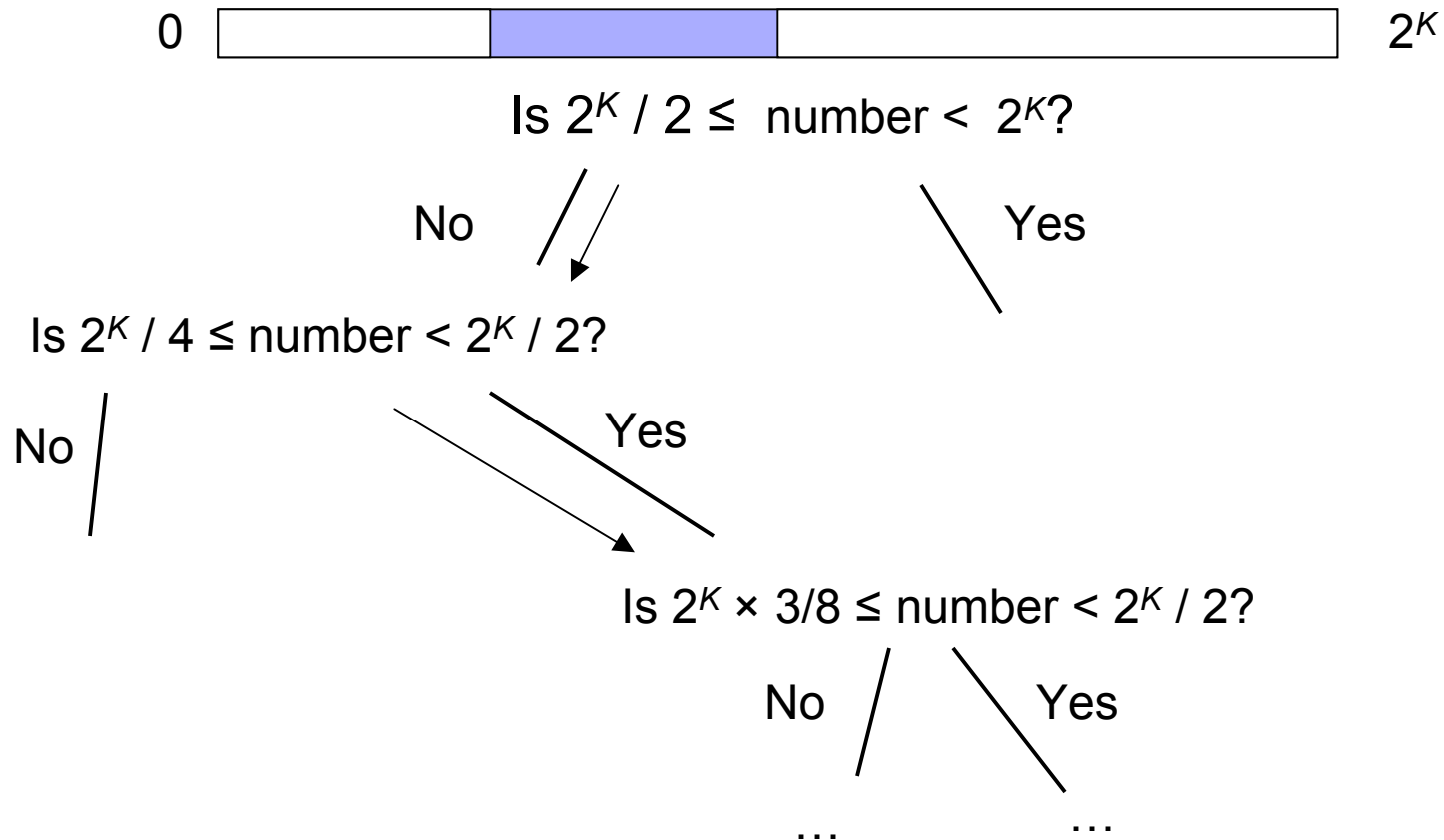


John Napier

| | n= 8 | n= 1024 | n= 1048576 | n=8388608 |
|------------|------|---------|---------------|----------------|
| $\log_2 n$ | 3 | 10 | 20 | 23 |
| n | 8 | 1024 | 1048576 | 8388608 |
| n^2 | 64 | 1048576 | 1099511627776 | 70368744177664 |

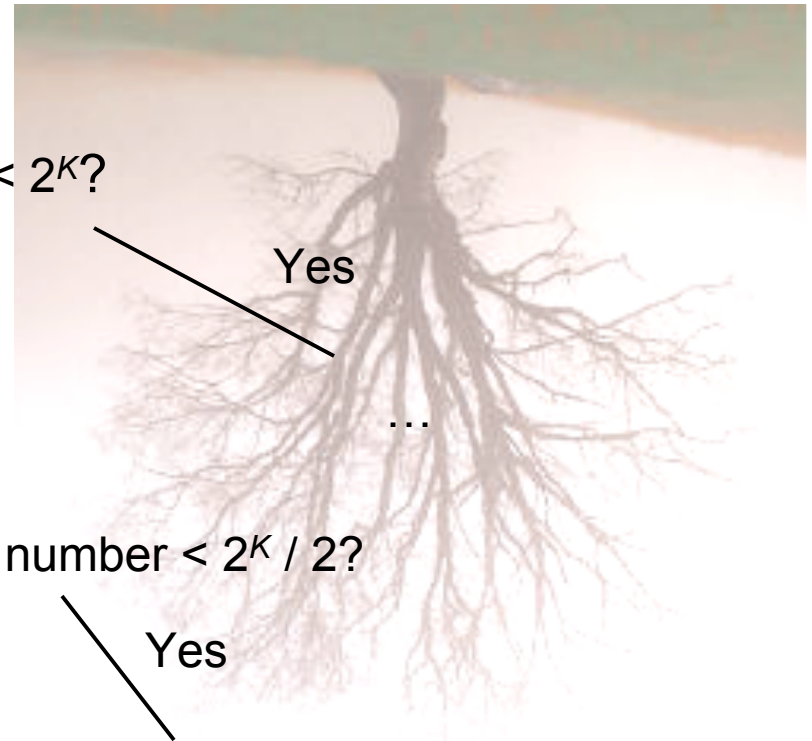
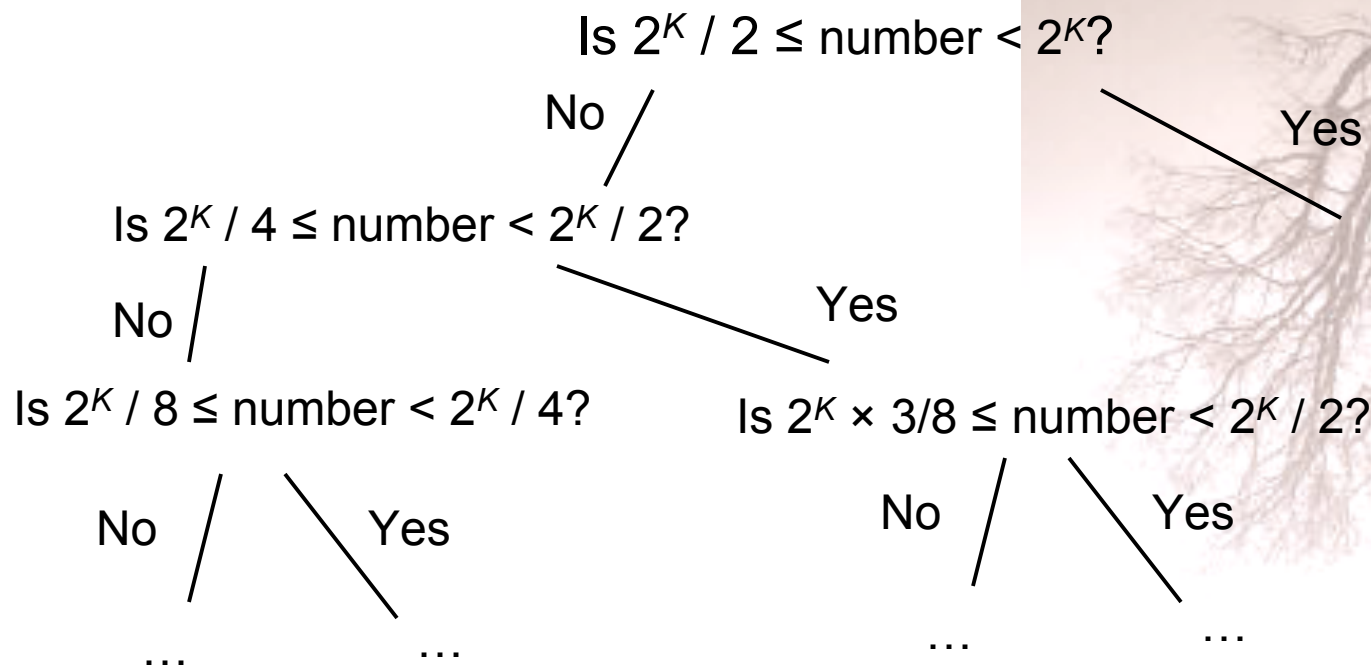
Binary search and binary representation of numbers

- Say we know $0 \leq \text{number} < 2^K$



Binary representations (cont'd)

- In general, each number uniquely represented by a sequence of yes/no answers to these questions.
- Correspond to paths down this “tree”:





Binary representation of n (*the more standard definition*)

$$n = 2^k b_k + 2^{k-1} b_{k-1} + \dots + 2 b_2 + b_1$$

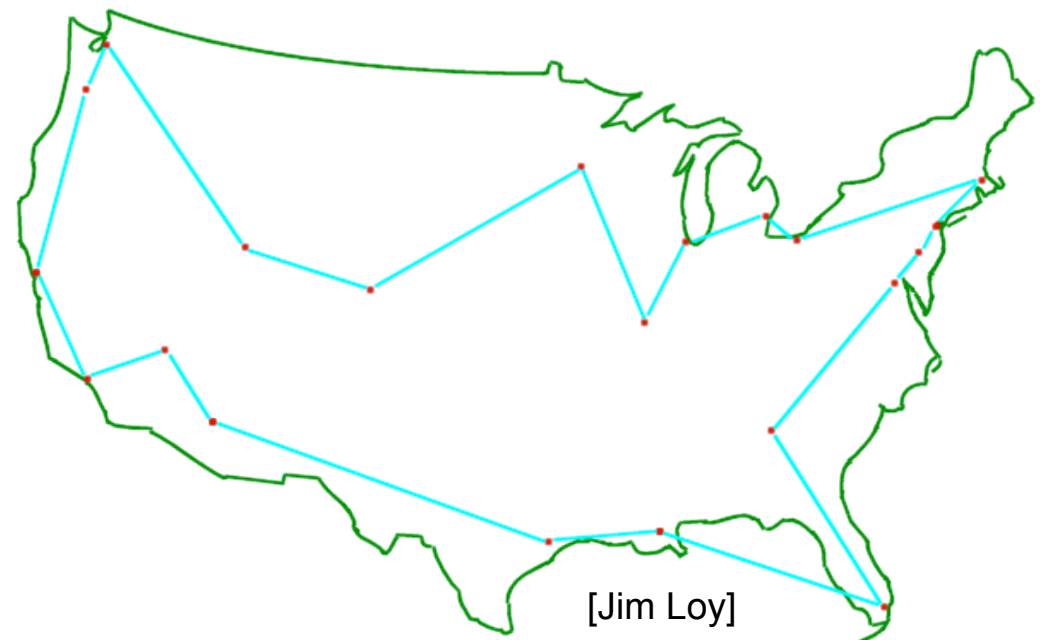
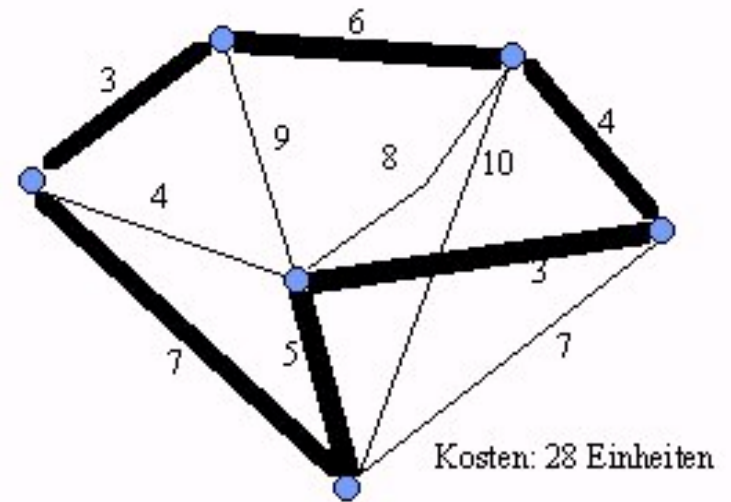
where the b 's are either 0 or 1)

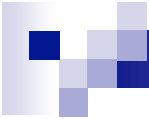
The binary representation of n is:

$$[n]_2 = b_k b_{k-1} \dots b_2 b_1$$

Efficiency of Effort: A lens on the world

- QWERTY keyboard
- “UPS Truck Driver’s Problem” (a.k.a. Traveling Salesman Problem or TSP)
- Handwriting Recognition
- CAPTCHA’s
- Quantum Computing





Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

Peter W. Shor[†]

SIAM J.
Computing
26(5) 1997

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

Can n particles do 2^n “operations” in a single step?
Or is Quantum Mechanics not quite correct?