“It ain’t no good if it ain’t snappy enough.”

(Efficient Computations)
Administrative stuff

- Readings avail. from course web page
- Feedback form on course web page; fully anonymous.
- HW1 extended - due Tues 2/23 instead.
- Reminder: Lab 3 Wed 7:30 Friend 007.
In what ways (according to Brian Hayes) is the universe like a cellular automaton?

What aspect(s) of the physical world are not represented well by a cellular automaton?
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

  (discussed also in pseudocode handout)

“Speed” of computer: number of elementary steps it can perform per second (Simplified definition)

  Do not consider this in “running time” of algorithm; technology-dependent.
Example: Find Min

- $n$ items, stored in array $A$
- Variables are $i$, $best$
- $best \leftarrow 1$
- Do for $i = 2$ to $n$
  
  {  
    {  
      $best \leftarrow i$
    }  
  }
Example: Find Min

- $n$ items, stored in array $A$
- Variables are $i$, $best$
- $best \leftarrow 1$
- Do for $i = 2$ to $n$
  
  ```
  \{ 
  \quad \text{if (} A[i] < A[best] \text{)} \text{ then} 
  \quad \{ \text{best} \leftarrow i \} 
  \}
  ```

- How many operations executed before the loop?
  - A: 0  B: 1  C: 2  D: 3
Example: Find Min

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

- How many operations per iteration of the loop?  
  - A: 0  
  - B: 1  
  - C: 2  
  - D: 3
Example: Find Min

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

- How many **times** does the loop run?
  - A: $n$  B: $n+1$  C: $n-1$  D: $2n$

“iterations”
Example: Find Min

- $n$ items, stored in array $A$
- Variables are $i$, $best$
- $best \leftarrow 1$
- Do for $i = 2$ to $n$
  \[
  \begin{array}{l}
    \text{if } (A[i] < A[best]) \text{ then } \\
    \quad \{ \text{best } \leftarrow i \}
  \end{array}
  \]

Uses at most $2(n - 1) + 1$ operations \hspace{1cm} (roughly $= 2n$)

1 assignment & 1 comparison = 2 operations per loop iteration

Number of iterations

Initialization
“20 Questions”:
I have a number between 1 and a million in mind. Guess it by asking me yes/no questions, and keep the number of questions small.

  Question 1: “Is the number bigger than half a million?” No

  Question 2: “Is the number bigger than a quarter million?” No

Strategy: Each question halves the range of possible answers.
Pseudocode: Guessing number from 1 to n

Lower ← 1
Upper ← n
Found ← 0
Do while (Found=0)
{
    Guess ← Round( (Lower + Upper)/2 )
    If (Guess = True Number)
    {
        Found ← 1
        Print(Guess)
    }
    If (Guess < True Number)
    {
        Lower ← Guess
    }
    else
    {
        Upper ← Guess
    }
}
Brief detour: Logarithms (CS view)

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

<table>
<thead>
<tr>
<th>( n )</th>
<th>16</th>
<th>1024</th>
<th>1048576</th>
<th>8388608</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log_2 n )</td>
<td>4</td>
<td>10</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>

John Napier
Next....

“There are only 10 types of people in the world – those who know binary and those who don’t.”
Binary search and binary representation of numbers

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

```
0 2^K
Is 2^K / 2 \leq \text{number} < 2^K?
No  Yes
Is 2^K / 4 \leq \text{number} < 2^K / 2?
No  Yes
Is 2^K \times 3/8 \leq \text{number} < 2^K / 2?
No  Yes
...  ...
```
Binary representations (cont’d)

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

  Is \(2^K / 2 \leq \text{number} < 2^K\)?
  - No
  - Is \(2^K / 4 \leq \text{number} < 2^K / 2\)?
    - No
    - Is \(2^K / 8 \leq \text{number} < 2^K / 4\)?
      - No
      - ... 
      - Yes
    - ... 
    - Yes
  - ... 
  - Yes
Binary representation of $n$  
(*the more standard definition*)

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

where the $b$’s are either 0 or 1)

The binary representation of $n$ is:  
$$[n]_2 = b_k \ b_{k-1} \ldots \ b_2 \ b_1$$
Efficiency of Selection Sort

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

\{
    Find cheapest bottle among those numbered $i$ to $n$
    
    Swap that bottle and the $i$'th bottle.
\}

- 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 \(n - 1\)

\[
S = 1 + 2 + \ldots + (n - 2) + (n - 1)
\]

\[
+ S = (n - 1) + (n - 2) + \ldots + 2 + 1
\]

\[
2S = n + n + \ldots + n + n
\]

\[
\text{So total time for selection sort is } \leq n(n - 1) + 3n
\]

(for large \(n\), roughly \(= n^2\))
Running times encountered in this lecture

<table>
<thead>
<tr>
<th></th>
<th>n= 8</th>
<th>n= 1024</th>
<th>n= 1048576</th>
<th>n=8388608</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log_2 n$</td>
<td>3</td>
<td>10</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>$n$</td>
<td>8</td>
<td>1024</td>
<td>1048576</td>
<td>8388608</td>
</tr>
<tr>
<td>$n^2$</td>
<td>64</td>
<td>1048576</td>
<td>1099511627776</td>
<td>70368744177664</td>
</tr>
</tbody>
</table>

Efficiency really makes a difference!
Efficiency of Effort:  
A lens on the world

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

[Jim Loy]
Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer

Peter W. Shor†

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?

Computational efficiency has a bearing on physical theories.