Chapter 4 in the book. Sections 4.1 and 4.2 today!
Where we’ve been and where we’re going...

- **Functional Programming**
  - Short and powerful programs using recursion and higher-order functions

- **How computers work**
  - Representing data, building a small computer out of circuits, programming that computer in its native language

- **Writing bigger programs**
  - Object-oriented design, AI concepts, games!

- **From hard to impossible problems!**

- **End-of-course projects**
Stuff

• Exam 1 is next Thursday, February 27
• You are welcome to bring a page of notes front-and-back
• Prof. Lucas Bang will be proctoring the exam
• Sample problems (and link to their solutions) handed out today ;^)
• Book royalties returned to you soon ($1/book!)
A True Story...

Donald Gillies Sr.

A prime example of Gillies’ work!

How big is $2^{11213}$?
The Biological Challenge
From A. P. Jackson, “Cophylogeny of the Ficus Microcosm,” Biological Reviews, 79,
Docstrings vs. comments vs. strings

def zimzam(X):
    """ Takes a number X as input and ... """

def zimzam(X):
    # Takes a number X as input and...

def zimzam(X):
    " Takes a number X as input and ..."""
def foo():
    """ returns a Boolean """

def bar():
    if foo() == True:
        return True
    else:
        return False

def bar():
    return foo()
A very small issue...

def stringScore(string, scoreFunction, wordList, memo):
    if string == "": return 0
    elif (string, scoreFunction) in memo:
        ...

    best = max(useIt, loseIt)
    memo[(string, scoreFunction)] = best
    return best
Python trick of the day...

def somethingFancy():
    ...
    return [number, List] # returning a care package

carePackage = somethingFancy()
score = carePackage[0]
list = carepackage[1]

score, list = somethingFancy()

x, y, z = [1, 2, 3] ← In general, this is legit!
"Hard" Problems

Snowplows of Northern Minnesota (aka the "Vertex Cover Problem")

- Burrsburg
- Tundratown
- Frostbite City
- Shiversville
- Freezeapolis
“Hard” Problems

Snowplows of Northern Minnesota (aka the “Vertex Cover Problem”)

Brute-force? Greed? DP?
# $n^2$ and $n^3$ versus $2^n$  

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>$n = 10$</th>
<th>$n = 30$</th>
<th>$n = 50$</th>
<th>$n = 70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>1 sec</td>
<td></td>
</tr>
</tbody>
</table>
## $n^2$ and $n^3$ versus $2^n$

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>$n=10$</th>
<th>$n=30$</th>
<th>$n=50$</th>
<th>$n=70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## $n^2$ and $n^3$ versus $2^n$  

The Ran-O-Matic performs $10^9$ operations/sec  

<table>
<thead>
<tr>
<th></th>
<th>$n = 10$</th>
<th>$n = 30$</th>
<th>$n = 50$</th>
<th>$n = 70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td>37 thousand years</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### $n^2$ and $n^3$ versus $2^n$

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>$n = 10$</th>
<th>$n = 30$</th>
<th>$n = 50$</th>
<th>$n = 70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td>37 thousand years</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

37 thousand years -> 37 years
The Traveling Salesperson Problem

San Francisco

Claremont

New York

Moscow

Paris

Brute Force?  Greed?  DP?
“Hard” Problems

The Travelling Salesperson Problem

Claremont

Montclare

Montclear

Clearmont
“Hard” Problems

The Travelling Salesperson Problem

Claremont  Montclare  
Montclear  
Montclear  

Claremont  
Montclare  
Montclear  
Clearmont
The Hamiltonian Cycle Problem
Here as he walked by on the 16th of October 1843
Sir William Rowan Hamilton in a flash of genius discovered
the fundamental formula for quaternion multiplication
\[ i^2 = j^2 = k^2 = ijk = -1 \]
& cut it on a stone of this bridge
Snowplows and Travelling Salesperson Revisited!

Tens of thousands of other known problems go in this cloud!!

NP-hard problems

Snowplow Problem

Protein Folding

Travelling Salesperson Problem

Sodoku
I can’t find an efficient algorithm, I guess I’m just too dumb.

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

I can’t find an efficient algorithm, but neither can all these famous people!

It all starts with representing data in binary!

What is the number 4312?

\[
\begin{array}{cccc}
10^3 & 10^2 & 10^1 & 10^0 \\
4 & 3 & 1 & 2 \\
\end{array}
\]

What is this number in base 20?

\[
\begin{array}{cccc}
20^2 & 20^1 & 20^0 \\
1 & 3 & 2 \\
\end{array}
\]

What is this number in base 2?

\[
\begin{array}{cccc}
2^2 & 2^1 & 2^0 \\
1 & 1 & 0 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>Base 2</th>
<th>Base 10</th>
<th>Base 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>0010</strong></td>
<td>2</td>
<td><strong>2</strong></td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

There are 10 different kinds of people. Those who know base 2 (aka binary) and those who don’t!
Arbitrary Bases (base “$b$”)

When using base $b$, the digits permitted are:

What is 5 in…

- base 2?
- base 3?
- base 4?
- base 5?
- base 6?
- base 42?

We write:

$101_2 = 12_3 = 11_4 = 10_5 = 5_6 = 5_{10} = 5_{42}$

The subscript indicates the base.
Is There Such a Thing as Base 1?

Unary!

1^3 1^2 1^1 1^0

Now we’re using powers of 1 (Weird!)

Technically, we should use 0 as our digit!
Comparing Representations in Different Bases

Consider the number $10^9$ in base 1, 2, 3, 10, and 20:

- **Base 1**: \[ 11111111111111111111111111111111111111111111 \ldots \]
  
  At ten “1’s” per inch, this will be…

- **Base 2**: \[ 1110111001101011001010000000000 \]

- **Base 3**: \[ 212020020002101001 \]

- **Base 10**: \[ 10000000000 \]

- **Base 20**: \[ \text{FCA}0000 \]

Does FCA stand for “Friendly Cuddly Alien?”
Comparing Representations in Different Bases

Consider the number $10^9$ in base 1, 2, 3, 10, and 20:

Base 1: 11111111111111111111111111111111111111111111...

Base 2: 11101110011010110010100000000

Base 3: 212020020002101001

Base 10: 1000000000

Base 20: FCA0000

At ten “1’s” per inch, this will be **1578 miles long!**

Does FCA stand for “Friendly Cuddly Alien?”
Converting Between Bases

Convert $1101_2$ to base 10

Convert $25_{10}$ to base 2

The digits 0 and 1 are referred to as “bits” - that’s short for “binary digits”

Worksheet

What “algorithms” did you use to do these conversions? There are two entirely different ways to do this!
The “Power” of Shifting!

“Left Shifting”

\[
\begin{array}{cccc}
10^3 & 10^2 & 10^1 & 10^0 \\
\hline
4 & 2 & & \\
4 & 2 & 0 & \\
\end{array}
\]

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
\hline
1 & 1 & & \\
1 & 1 & 0 & \\
\end{array} = 3_{10}
\]

“Right Shifting”

\[
\begin{array}{cccc}
10^3 & 10^2 & 10^1 & 10^0 \\
\hline
4 & 5 & 7 & \\
4 & 5 & & \\
\end{array}
\]

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
\hline
1 & 0 & 1 & \\
1 & 0 & & \\
\end{array} = 5_{10}
\]
Base Conversion, Recursively!

\[
24_{10} = \_2
\]

\[
2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0
\]

\[
0
\]

\[
25_{10} = \_2
\]

\[
2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0
\]

\[
1
\]
Addition

Base 10 Addition

\[
\begin{array}{ccc}
10^2 & 10^1 & 10^0 \\
4 & 3 & \\
+ & 8 & 9 \\
\end{array}
\]
Addition

Base 10 Addition

\[
\begin{array}{c}
10^2 & 10^1 & 10^0 \\
4 & 3 & 12 \\
+ & 8 & 9 \\
\end{array}
\]

That’s a “10”
Addition

Base 10 Addition

<table>
<thead>
<tr>
<th>10²</th>
<th>10¹</th>
<th>10⁰</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Move the “1” to the tens place
Addition

Base 10 Addition

\[
\begin{array}{ccc}
10^2 & 10^1 & 10^0 \\
\hline
1 & 4 & 3 \\
8 & 9 & \\
13 & 2 & \\
\end{array}
\]

Done!
Addition

Base 10 Addition

<table>
<thead>
<tr>
<th>10^2</th>
<th>10^1</th>
<th>10^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Try it in base 2!

Base 2 Addition

<table>
<thead>
<tr>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Base 10 Multiplication

\[
\begin{array}{ccc}
10^2 & 10^1 & 10^0 \\
3 & 4 & 1 \\
\times & 1 & 0 & 2 \\
\hline
6 & 8 & 2 \\
0 & 0 & 0 \\
+ & 3 & 4 & 1 \\
\hline
3 & 4 & 7 & 8 & 2 \\
\end{array}
\]

Base 2 Multiplication

\[
\begin{array}{ccc}
2^2 & 2^1 & 2^0 \\
1 & 1 & 1 \\
\times & 1 & 0 & 1 \\
\hline
\end{array}
\]
Aside: Multiplication with Russian Peasants

Compute $21 \times 6$:

\[
\begin{array}{cc}
21 & 6 \\
10 & 12 \\
5 & 24 \\
2 & 48 \\
1 & 96 \\
\end{array}
\]
Aside: Multiplication with Russian Peasants

Compute $21 \times 6$:

\[
\begin{array}{cc}
21 & 6 \\
10 & 12 \\
5 & 24 \\
2 & 48 \\
1 & 96 \\
\end{array}
\]

$6 + 24 + 96 = 126$

(Translation: “Why does this work?”)
Aside: Multiplication with Russian Peasants

Compute \(21 \times 6:\)

\[
\begin{array}{c|c}
21 & 6 \\
10 & 12 \\
5 & 24 \\
2 & 48 \\
1 & 96 \\
\end{array}
\]

\[
\begin{array}{c|c}
10101 & 110 \\
1010 & 1100 \\
101 & 11000 \\
10 & 110000 \\
1 & 1100000 \\
\end{array}
\]

\[
6 + 24 + 96 = 126
\]

Я люблю двоичную систему!

(Translation: “I love binary!”)
Aside: Multiplication with Russian Peasants

\[
\begin{array}{c}
\text{110} \\
\times \quad \text{10101} \\
\text{110} \\
000 \\
11000 \\
000 \\
1100000 \\
\hline
1111110 \\
\end{array}
\]

\[
\begin{array}{c}
\text{10101} \\
\text{110} \\
\text{101} \\
\text{11000} \\
\text{10} \\
\text{110000} \\
1 \\
\hline
1111110 \\
\end{array}
\]
Aside: Multiplication with Russian Peasants

\[
\begin{array}{cccc}
& 110 \\
\times & 10101 \\
& 110 \\
& 000 \\
& 11000 \\
& 000 \\
& 1100000 \\
\hline
& 1111110
\end{array}
\]

\[
\begin{array}{cccc}
10101 & 110 \\
1010 & 1100 \\
101 & 11000 \\
10 & 110000 \\
1 & 1100000 \\
\hline
1111110
\end{array}
\]
Aside: Multiplication with Russian Peasants

\[
\begin{array}{c}
110 \\
\times \quad 10101 \\
\hline
110 \\
110 \\
000 \\
11000 \\
000 \\
1100000 \\
\hline
1111110
\end{array}
\]
Aside: Multiplication with Russian Peasants

\[
\begin{array}{c}
110 \\
\times \quad 10101 \\
\hline
110 \\
110 \\
000 \\
11000 \\
000 \\
\text{1100000} \\
\hline
\text{1111110} \\
\end{array}
\]