Called a traveling-salesman tour.

Computing the best tour is an infamously difficult computational problem.
Traveling-salesman problem

http://xkcd.com/399/  (Randall Munro)

What are those strange symbols and why is this funny?
This is about algorithms.
Algorithms

I do algorithm design for a living.
The goal: clever methods for solving computational problems.
Staring at the ceiling, she asked me what I was thinking about.

I should have made something up.

The Bellman-Ford algorithm makes terrible pillow talk.

http://xkcd.com/69/  (Randall Munro)
1337: Part 2

http://xkcd.com/342/
(Randall Munro)

Again with the strange symbols?
Computational Problems

A computational problem is specified by specifying

- what kind of inputs are acceptable
- what kind of outputs are acceptable

Exactly what I call an input-output spec.

Some computational problems:

- **input**: number \( a \)
  - **output**: the product \( a \cdot a \)

- **input**: numbers \( a \) and \( b \)
  - **output**: the product \( a \cdot b \)

- **input**: a Sudoku puzzle
  - **output**: the solution (values of missing entries)

- **input**: coordinates of tennis balls
  - **output**: shortest tour visiting all the balls

Example input and output

- **input**: coordinates of tennis balls
  - **output**: shortest tour visiting all the balls
A *computational problem* is specified by specifying
• what kind of inputs are acceptable
• what kind of outputs are acceptable

It is an *optimization* problem if the output is chosen among many candidates by selecting the candidate with the smallest/biggest value (however value is defined).

• **input**: coordinates of tennis balls
• **output**: shortest tour visiting all the balls
Algorithms

An algorithm for a computational problem … Not a formula Not a program/app.

A precisely specified computational method that, for any legal input, is guaranteed to produce a legal output.

Example: Exponentiation using multiplication

\[ 3^N = 3 \cdot 3 \cdot 3 \cdot \ldots \cdot 3 \]  

\( N \) times

\[ 3^{16} = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \]

It seems that computing \( 3^{16} \) by multiplying requires fifteen multiplications.…

Generalizing gives you an algorithm for exponentiation.

However, squaring a number requires one multiplication: \( x^2 = x \cdot x \)

Use repeated squaring:

\[ 3^{16} = (((3^2)^2)^2)^2 \]

Two ways of computing \( 3^{16} \): naive (takes fifteen mults) and clever (four mults)

Four multiplications instead of fifteen doesn’t seem like a big deal...

However, suppose the exponent is large:

\[ 3^{1048576} = ((((((((3^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2)^2) \]

Practical need: In, need exponents that are hundreds of digits.
4096-bit RSA requires exponents with over a thousand digits.

**Practical need:** In encryption, need exponents that are *hundreds* of digits.

Algorithms for exponentiating 3

**input:** a number $b$

**output:** $3^b$

Measure the input $b$ by number $n$ of digits of $b$

<table>
<thead>
<tr>
<th>Naive algorithm</th>
<th>Clever algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>N</strong></td>
<td><strong>number of multiplications</strong></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1024</td>
<td>1023</td>
</tr>
<tr>
<td>1025</td>
<td>1024</td>
</tr>
<tr>
<td>1026</td>
<td>1025</td>
</tr>
<tr>
<td>1027</td>
<td>1026</td>
</tr>
<tr>
<td>1028</td>
<td>1027</td>
</tr>
</tbody>
</table>
Algorithms for exponentiating 3

**input:** a number \( b \)

**output:** \( 3^b \)

Measure the input \( b \) by number \( n \) of digits of \( b \)

<table>
<thead>
<tr>
<th>Naive algorithm</th>
<th>Clever algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of digits</td>
<td>number of multiplications</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>98</td>
</tr>
<tr>
<td>3</td>
<td>998</td>
</tr>
<tr>
<td>4</td>
<td>9998</td>
</tr>
<tr>
<td>5</td>
<td>99998</td>
</tr>
<tr>
<td>6</td>
<td>999998</td>
</tr>
<tr>
<td>7</td>
<td>9999998</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>...</td>
</tr>
<tr>
<td>101</td>
<td>...</td>
</tr>
<tr>
<td>102</td>
<td>...</td>
</tr>
<tr>
<td>103</td>
<td>...</td>
</tr>
<tr>
<td>104</td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithms for exponentiating $3$

**input**: a number $b$

**output**: $3^b$

Measure the input $b$ by number $n$ of digits of $b$

### Naive algorithm

<table>
<thead>
<tr>
<th>number of digits</th>
<th>number of multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>98</td>
</tr>
<tr>
<td>3</td>
<td>9998</td>
</tr>
<tr>
<td>4</td>
<td>99998</td>
</tr>
<tr>
<td>5</td>
<td>999998</td>
</tr>
<tr>
<td>6</td>
<td>9999998</td>
</tr>
<tr>
<td>7</td>
<td>99999998</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>98765432109876</td>
</tr>
</tbody>
</table>

### Clever algorithm

<table>
<thead>
<tr>
<th>number of digits</th>
<th>number of multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>98765432109876</td>
</tr>
</tbody>
</table>

---

Graph:
- **Horizontal axis**: input size $n$
- **Vertical axis**: number of computational steps

- $y = 10^n$
- $y = n/0.3$
We use formulas to describe how many operations are needed depending on size of input:

\[ O(10^n) \quad O(n^2) \quad O(n) \]

horizontal axis = input size \( n \)

vertical axis = number of computation steps

- \( O(10^n) \): bad
- \( O(n^2) \): pretty good
- \( O(n) \): best
Analysis of algorithms: Predicting how long a given algorithm will take

Why?
• Help you decide whether a computational task is feasible.
• Help you decide which of several algorithms is best for task.
• Help you decide when you have an algorithm that suffices.
• Helps you focus on what part of a big complicated program will be the computational bottleneck.

How?
Start with a *model* of the computer.
The model captures *some* aspects of how computer works.
Every model is an approximation of the thing it models.
We use models because they are simpler than reality, easier to work with.
More faithful models might give you more accurate predictions but harder to work with. Need a good compromise. Depends on use.
We will define basic operations:
• cons
• car
• cdr
• binding a formal argument to an actual argument
• ...

Measure computation time by how many basic operations are performed.
Because I know the Rules of Evaluation, I can pretend to be the computer, so I can add up the number of operations.

As you will see, the exact number of operations is not vital.
How many operations for evaluating the following?

\((\ast\ (\ +\ a\ b)\ (\ +\ c\ d))\)

For now, let’s assume each of
- multiplying two numbers
- adding two numbers
is a single operation.

Each of
- evaluating a variable
- binding a formal to an actual
is one operation.

<table>
<thead>
<tr>
<th>variable-evaluations: 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindings: 6</td>
</tr>
<tr>
<td>additions: 2</td>
</tr>
<tr>
<td>multiplications: 1</td>
</tr>
<tr>
<td>Total: 16</td>
</tr>
</tbody>
</table>
Quiz 1: Write the len procedure

input: a list $L$
output: number of items in $L$
(You’ve seen this before.)

(check-expect
(define len
  (lambda (L)
    (cond
      ((empty? L) 0)
      ((cons? L) (+ 1 (len (rest L))))))))

or

(define len
  (lambda (L)
    (if (empty? L) 0
      (+ 1 (len (rest L))))))
Quiz 2: Write the English2cow procedure:

input: a list $L$
output: a list of the same length as $L$ all of whose elements are the symbol moo

(check-expect
  (English2cow (quote (How are you?)))
  (quote (moo moo moo)))

Solution:
(define English2cow (lambda (L)
  (if (empty? L)
      empty
      (cons (quote moo) (English2cow (rest L))))))
Analyzing a recursive procedure

Analyze English2cow

What does “analyze” mean?
We want a formula that predicts how long the procedure will take (number of operations).
That depends on length of input list!

Let’s make a table...

(define English2cow (lambda (L)
(if (empty? L)
  empty
  (cons (quote moo) (English2cow (rest L))))))

Unit cost

• if
• quote
• cons
• rest
• empty?
• Binding