What's It All About?

- Continuous mathematics—*calculus*—considers objects that vary continuously
  - distance from the wall
- Discrete mathematics considers *discrete* objects, that come in *discrete* bundles
  - number of babies: can't have 1.2

The mathematical techniques for discrete mathematics differ from those for continuous mathematics:

- counting/combinatorics
- number theory
- probability
- logic

We'll be studying these techniques in this course.

Why is it computer science?

This is basically a mathematics course:

- no programming
- lots of theorems to prove

So why is it computer science?

Discrete mathematics is the mathematics underlying almost all of computer science:

- Designing high-speed networks
- Finding good algorithms for sorting
- Doing good web searches
- Analysis of algorithms
- Proving algorithms correct

This Course

We will be focusing on:

- Tools for discrete mathematics:
  - computational number theory (handouts)
    - the mathematics behind the RSA cryptosystems
  - counting/combinatorics (Chapter 4)
  - probability (Chapter 6)
    - randomized algorithms for primality testing, routing
  - logic (Chapter 7)
    - how do you *prove* a program is correct
- Tools for proving things:
  - induction (Chapter 2)
  - (to a lesser extent) recursion

First, some background you'll need but may not have ...

Sets

You need to be comfortable with set notation:

$$S = \{m | 2 \le m \le 100, m \text{ is an integer}\}$$

S is
the set of
all m
such that
m is between 2 and 100
and
m is an integer.

Important Sets

(More notation you need to know and love ...)

- N (occasionally $I\!N$): the nonnegative integers $\{0, 1, 2, 3, \ldots\}$
- N^+: the positive integers $\{1, 2, 3, \ldots\}$
- Z: all integers $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- Q: the rational numbers $\{a/b : a, b \in Z, b \neq 0\}$
- R: the real numbers
- Q^+, R^+: the positive rationals/reals

Set Notation

- $|S|$ = *cardinality of* (number of elements in) S
  - $|\{a, b, c\}| = 3$
- **Subset**: $A \subset B$ if every element of A is an element of B
  - Note: Lots of people (including me, but not the authors of the text) usually write $A \subset B$ only if A is a *strict* or *proper* subset of B (i.e., $A \neq B$). I write $A \subseteq B$ if $A = B$ is possible.
- Power set: $\mathcal{P}(S)$ is the set of all subsets of S (sometimes denoted 2^S).
  - E.g., $\mathcal{P}(\{1, 2, 3\}) =$ $\{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
  - $|\mathcal{P}(S)| = 2^{|S|}$

Set Operations

- **Union:** $S \cup T$ is the set of all elements in S or T
  - $S \cup T = \{x | x \in S \text{ or } x \in T\}$
  - $\{1, 2, 3\} \cup \{3, 4, 5\} = \{1, 2, 3, 4, 5\}$
- **Intersection:** $S \cap T$ is the set of all elements in both S and T
  - $S \cap T = \{x | x \in S, x \in T\}$
  - $\{1, 2, 3\} \cap \{3, 4, 5\} = \{3\}$
- **Set Difference:** $S - T$ is the set of all elements in S not in T
  - $S - T = \{x | x \in S, x \notin T\}$
  - $\{3, 4, 5\} - \{1, 2, 3\} = \{4, 5\}$
- **Complementation:** $\overline{S}$ is the set of elements not in S
  - What is $\overline{\{1, 2, 3\}}$?
  - Complementation doesn't make sense unless there is a *universe*, the set of elements we want to consider.
  - If U is the universe, $\overline{S} = \{x | x \in U, x \notin S\}$
  - $\overline{S} = U - S$.

Venn Diagrams

Sometimes a picture is worth a thousand words (at least if we don't have too many sets involved).

A Connection

Lemma: For all sets S and T, we have

$$S = (S \cap T) \cup (S - T)$$

Proof: We'll show (1) $S \subset (S \cap T) \cup (S - T)$ and (2) $(S \cap T) \cup (S - T) \subset S$.

For (1), suppose $x \in S$. Either
(a) $x \in T$ or (b) $x \notin T$.

If (a) holds, then $x \in S \cap T$.

If (b) holds, then $x \in S - T$.

In either case, $x \in (S \cap T) \cup (S - T)$.

Since this is true for all $x \in S$, we have (1).

For (2), suppose $x \in (S \cap T) \cup (S - T)$. Thus, either (a) $x \in (S \cap T)$ or $x \in (S - T)$. Either way, $x \in S$.

Since this is true for all $x \in (S \cap T) \cup (S - T)$, we have (2).

Two Important Morals

1. One way to show $S = T$ is to show $S \subset T$ and $T \subset S$.
2. One way to show $S \subset T$ is to show that for every $x \in S$, x is also in T.

Relations

- **Cartesian product**:
  $S \times T = \{(s, t) : s \in S, t \in T\}$
  - $\{1, 2, 3\} \times \{3, 4\} =$
    $\{(1, 3), (2, 3), (3, 3), (1, 4), (2, 4), (3, 4)\}$
  - $|S \times T| = |S| \times |T|$.
- A *relation* on S and T (or, on $S \times T$) is a subset of $S \times T$
- A *relation* on S is a subset of $S \times S$
  - *Taller than* is a relation on people: (Joe,Sam) is in the Taller than relation if Joe is Taller than Sam
  - *Larger than* is a relation on R:
    $$L = \{(x, y) | x, y \in R, x > y\}$$
  - *Divisibility* is a relation on N:
    $$D = \{(x, y) | x, y \in N, x | y\}$$

Reflexivity, Symmetry, Transitivity

- A relation R on S is *reflexive* if $(x, x) \in R$ for all $x \in S$.
  - $\leq$ is reflexive; $<$ is not
- A relation R on S is *symmetric* if $(x, y) \in R$ implies $(y, x) \in R$.
  - "sibling-of" is symmetric (what about "sister of")
  - $\leq$ is not symmetric
- A relation R on S is *transitive* if $(x, y) \in R$ and $(y, z) \in R$ implies $(x, z) \in R$.
  - $\leq$, $<$, $\geq$, $>$ are all transitive;
  - "parent-of" is not transitive; "ancestor-of" is

Pictorially, we have:

Transitive Closure

[[NOT DISCUSSED ENOUGH IN THE TEXT]]

The *transitive closure* of a relation R is the least relation R^* such that

1. $R \subset R^*$
2. R^* is transitive (so that if $(u,v),(v,w) \in R^*$, then so is (u,w)).

Example: Suppose $R = \{(1,2),(2,3),(1,4)\}$.

- $R^* = \{(1,2),(1,3),(2,3),(1,4)\}$
- we need to add $(1,3)$, because $(1,2),(2,3) \in R$

Note that we don't need to add (2,4).

- If (2,1), (1,4) were in R, then we'd need (2,4)
- (1,2), (1,4) doesn't force us to add anything (it doesn't fit the "pattern" of transitivity.

Note that if R is already transitive, then $R^* = R$.

Equivalence Relations

- A relation R is an *equivalence relation* if it is reflexive, symmetric, and transitive
  - $=$ is an equivalence relation
  - $Parity$ is an equivalence relation on N; $(x,y) \in Parity$ if $x - y$ is even

Functions

We think of a function $f : S \to T$ as providing a mapping from S to T. But ...

Formally, a *function* is a relation R on $S \times T$ such that for each $s \in S$, there is a unique $t \in T$ such that $(s,t) \in R$.

If $f : S \to T$, then S is the *domain* of f, T is the *range*; $\{y : f(x) = y \text{ for some } x \in S\}$ is the *image*.

We often think of a function as being characterized by an algebraic formula

- $y = 3x - 2$ characterizes $f(x) = 3x - 2$.

It ain't necessarily so.

- Some formulas don't characterize functions:
  - $x^2 + y^2 = 1$ defines a circle; no unique y for each x
- Some functions can't be characterized by algebraic formulas
  - $f(n) = \begin{cases} 0 & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$

Function Terminology

Suppose $f : S \rightarrow T$

- f is *onto* (or *surjective*) if, for each $t \in T$, there is some $s \in S$ such that $f(s) = t$.
  - ◦ if $f : R^+ \rightarrow R^+$, $f(x) = x^2$, then f is onto
  - ◦ if $f : R \rightarrow R$, $f(x) = x^2$, then f is *not* onto
- f is *one-to-one* (1-1, *injective*) if it is not the case that $s \neq s'$ and $f(s) = f(s')$.
  - ◦ if $f : R^+ \rightarrow R^+$, $f(x) = x^2$, then f is 1-1
  - ◦ if $f : R \rightarrow R$, $f(x) = x^2$, then f is *not* 1-1.

- a function is *bijective* if it is 1-1 and onto.
  - ◦ if $f : R^+ \rightarrow R^+$, $f(x) = x^2$, then f is bijective
  - ◦ if $f : R \rightarrow R$, $f(x) = x^2$, then f is *not* bijective.

If $f : S \rightarrow T$ is bijective, then $|S| = |T|$.

Inverse Functions

If $f : S \rightarrow T$, then f^{-1} maps an element in the range of f to all the elements that are mapped to it by f.

$$f^{-1}(t) = \{s | f(s) = t\}$$

- if $f(2) = 3$, then $2 \in f^{-1}(3)$.

f^{-1} is not a function from range(f) to S.

It is a function if f is one-to-one.

- In this case, $f^{-1}(f(x)) = x$.

Functions You Should Know (and Love)

- *Absolute value*: Domain = R; Range = $\{0\} \cup R^+$

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

  - ◦ $|3| = |-3| = 3$
- *Floor function*: Domain = R; Range = Z

$$\lfloor x \rfloor = \text{ largest integer not greater than } x$$

  - ◦ $\lfloor 3.2 \rfloor = 3$; $\lfloor \sqrt{3} \rfloor = 1$; $\lfloor -2.5 \rfloor = -3$
- *Ceiling function*: Domain = R; Range = Z

$$\lceil x \rceil = \text{ smallest integer not less than } x$$

  - ◦ $\lceil 3.2 \rceil = 4$; $\lceil \sqrt{3} \rceil = 2$; $\lceil -2.5 \rceil = -2$
- *Factorial function*: Domain = Range = N

$$n! = n(n-1)(n-2)...3 \times 2 \times 1$$

  - ◦ $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
  - ◦ By convention, $0! = 1$

Exponents

Exponential with base a: Domain $= R$, Range$=R^+$

$$f(x) = a^x$$

- Note: a, the *base*, is fixed; x varies
- You probably know: $a^n = a \times \cdots \times a$ (n times)

How do we define $f(x)$ if x is not a positive integer?

- **Want:** (1) $a^{x+y} = a^x a^y$; (2) $a^1 = a$

This means

- $a^2 = a^{1+1} = a^1 a^1 = a \times a$
- $a^3 = a^{2+1} = a^2 a^1 = a \times a \times a$
- ...
- $a^n = a \times \ldots \times a$ (n times)

We get more:

- $a = a^1 = a^{1+0} = a \times a^0$
  - Therefore $a^0 = 1$
- $1 = a^0 = a^{b+(-b)} = a^b \times a^{-b}$
  - Therefore $a^{-b} = 1/a^b$

- $a = a^1 = a^{\frac{1}{2}+\frac{1}{2}} = a^{\frac{1}{2}} \times a^{\frac{1}{2}} = (a^{\frac{1}{2}})^2$
  - Therefore $a^{\frac{1}{2}} = \sqrt{a}$
- Similar arguments show that $a^{\frac{1}{k}} = \sqrt[k]{a}$
- $a^{mx} = a^x \times \cdots \times a^x (m \text{ times}) = (a^x)^m$
  - Thus, $a^{\frac{m}{n}} = (a^{\frac{1}{n}})^m = (\sqrt[n]{a})^m$.

This determines a^x for all x rational. The rest follows by continuity.

Computing a^n quickly

What's the best way to compute a^{1000}?

One way: multiply $a \times a \times a \times a \ldots$

- This requires 999 multiplications.

Can we do better?
How many multiplications are needed to compute:

- a^2
- a^4
- a^8
- a^{16}
- ...

Write 1000 in binary: 1111101000

- How many multiplications are needed to calculate a^{1000}?

Logarithms

Logarithm base a: Domain $= R^+$; Range $= R$

$$y = \log_a(x) \Leftrightarrow a^y = x$$

- $\log_2(8) = 3$; $\log_2(16) = 4$; $3 < \log_2(15) < 4$

The key properties of the log function follow from those for the exponential:

1. $\log_a(1) = 0$ (because $a^0 = 1$)
2. $\log_a(a) = 1$ (because $a^1 = a$)
3. $\log_a(xy) = \log_a(x) + \log_a(y)$

   Proof: Suppose $\log_a(x) = z_1$ and $\log_a(y) = z_2$.

   Then $a^{z_1} = x$ and $a^{z_2} = y$.
   Therefore $xy = a^{z_1} \times a^{z_2} = a^{z_1+z_2}$.
   Thus $\log_a(xy) = z_1 + z_2 = \log_a(x) + \log_a(y)$.
4. $\log_a(x^r) = r \log_a(x)$
5. $\log_a(1/x) = -\log_a(x)$ (because $a^{-y} = 1/a^y$)
6. $\log_b(x) = \log_a(x)/\log_a(b)$

Examples:

- $\log_2(1/4) = -\log_2(4) = -2$.
- $\log_2(-4)$ undefined
- $$\begin{aligned} & \log_2(2^{10}3^5) \\ = \; & \log_2(2^{10}) + \log_2(3^5) \\ = \; & 10\log_2(2) + 5\log_2(3) \\ = \; & 10 + 5\log_2(3) \end{aligned}$$

Limit Properties of the Log Function

$$\lim_{x\to\infty} \log(x) = \infty$$
$$\lim_{x\to\infty} \frac{\log(x)}{x} = 0$$

As x gets large $\log(x)$ grows without bound.

But x grows MUCH faster than $\log(x)$.

In fact, $\lim_{x\to\infty}(\log(x)^m)/x = 0$

Polynomials

$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_kx^k$ is a *polynomial* function.

- $a_0, \ldots, a_k$ are the *coefficients*

You need to know how to multiply polynomials:

$$\begin{aligned} & (2x^3 + 3x)(x^2 + 3x + 1) \\ = \; & 2x^3(x^2 + 3x + 1) + 3x(x^2 + 3x + 1) \\ = \; & 2x^5 + 6x^4 + 2x^3 + 3x^3 + 9x^2 + 3x \\ = \; & 2x^5 + 6x^4 + 5x^3 + 9x^2 + 3x \end{aligned}$$

Exponentials grow MUCH faster than polynomials:

$$\lim_{x\to\infty} \frac{a_0 + \cdots + a_kx^k}{b^x} = 0 \text{ if } b > 1$$

Why Rates of Growth Matter

Suppose you want to design an algorithm to do sorting.

- The naive algorithm takes time $n^2/4$ on average to sort n items
- A more sophisticated algorithm times time $2n\log(n)$

Which is better?

$$\lim_{n\to\infty}(2n\log(n)/(n^2/4)) = \lim_{n\to\infty}(8\log(n)/n) = 0$$

For example,

- if $n = 1,000,000$, $2n\log(n) = 40,000,000$ — this is doable
  $n^2/4 = 250,000,000,000$ — this is not doable

Algorithms that take exponential time are hopeless on large datasets.

Sum and Product Notation

$$\sum_{i=0}^{k} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k$$

$$\sum_{i=2}^{5} i^2 = 2^2 + 3^2 + 4^2 + 5^2 = 54$$

Can limit the set of values taken on by the *index* i:

$$\sum_{\{i:2\leq i\leq 8 | i \text{ even}\}} a_i = a_2 + a_4 + a_6 + a_8$$

Can have double sums:

$$\begin{aligned} & \Sigma_{i=1}^{2} \Sigma_{j=0}^{3} a_{ij} \\ = \ & \Sigma_{i=1}^{2} (\Sigma_{j=0}^{3} a_{ij}) \\ = \ & \Sigma_{j=0}^{3} a_{1j} + \Sigma_{j=0}^{3} a_{2j} \\ = \ & a_{10} + a_{11} + a_{12} + a_{13} + a_{20} + a_{21} + a_{22} + a_{23} \end{aligned}$$

Product notation similar:

$$\prod_{i=0}^{k} a_i = a_0 a_1 \cdots a_k$$

Changing the Limits of Summation

This is like changing the limits of integration.

- $\Sigma_{i=1}^{n+1} a_i = \Sigma_{i=0}^{n} a_{i+1} = a_1 + \cdots + a_{n+1}$

Steps:

- Start with $\Sigma_{i=1}^{n+1} a_i$.
- Let $j = i - 1$. Thus, $i = j + 1$.
- Rewrite limits in terms of j: $i = 1 \rightarrow j = 0$; $i = n+1 \rightarrow j = n$
- Rewrite body in terms of $a_i \rightarrow a_{j+1}$
- Get $\Sigma_{j=0}^{n} a_{j+1}$
- Now replace j by i (j is a dummy variable). Get

$$\sum_{i=0}^{n} a_{i+1}$$

Matrix Algebra

An $m \times n$ *matrix* is a two-dimensional array of numbers, with m rows and n columns:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- A $1 \times n$ matrix $[a_1 \ldots a_n]$ is a *row vector*.
- An $m \times 1$ matrix is a *column vector*.

We can add two $m \times n$ matrices:

- If $A = [a_{ij}]$ and $B = [b_{ij}]$ then $A + B = [a_{ij} + b_{ij}]$.

$$\begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} + \begin{bmatrix} 3 & 7 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 9 & 9 \end{bmatrix}$$

Another important operation: *transposition*.

- If we transpose an $m \times n$ matrix, we get an $n \times m$ matrix by switching the rows and columns.

$$\begin{bmatrix} 2 & 3 & 9 \\ 5 & 7 & 12 \end{bmatrix}^T = \begin{bmatrix} 2 & 5 \\ 3 & 7 \\ 9 & 12 \end{bmatrix}$$

Matrix Multiplication

Given two vectors $\vec{a} = [a_1, \ldots, a_k]$ and $\vec{b} = [b_1, \ldots, b_k]$, their *inner product* (or *dot product*) is

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{k} a_i b_i$$

- $[1, 2, 3] \cdot [-2, 4, 6] = (1 \times -2) + (2 \times 4) + (3 \times 6) = 24.$

We can multiply an $n \times m$ matrix $A = [a_{ij}]$ by an $m \times k$ matrix $B = [b_{ij}]$, to get an $n \times k$ matrix $C = [c_{ij}]$:

- $c_{ij} = \Sigma_{r=1}^{m} a_{ir} b_{rj}$
- this is the inner product of the ith row of A with the jth column of B

- $\begin{bmatrix} 2 & 3 & 1 \\ 5 & 7 & 4 \end{bmatrix} \times \begin{bmatrix} 3 & 7 \\ 4 & 2 \\ -1 & -2 \end{bmatrix} = \begin{bmatrix} 17 & 18 \\ 39 & 41 \end{bmatrix}$

$$\begin{aligned}
17 &= (2 \times 3) + (3 \times 4) + (1 \times -1) \\
&= (2, 3, 1) \cdot (3, 4, -1) \\
18 &= (2 \times 7) + (3 \times 2) + (1 \times -2) \\
&= (2, 3, 1) \cdot (7, 2, -2) \\
39 &= (5 \times 3) + (7 \times 4) + (4 \times -1) \\
&= (5, 7, 4) \cdot (3, 4, -1) \\
41 &= (5 \times 7) + (7 \times 2) + (4 \times -2) \\
&= (5, 7, 4) \cdot (7, 2, -2)
\end{aligned}$$

Why is multiplication defined in this strange way?

- Because it's useful!

Suppose

$$\begin{array}{ll}
z_1 = 2y_1 + 3y_2 + y_3 & y_1 = 3x_1 + 7x_2 \\
z_2 = 5y_1 + 7y_2 + 4y_3 & y_2 = 4x_1 + 2x_2 \\
 & y_3 = -x_1 - 2x_2
\end{array}$$

Thus, $\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 7 & 4 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$ and $\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 3 & 7 \\ 4 & 2 \\ -1 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

Suppose we want to express the z's in terms of the x's:

$$\begin{aligned}
z_1 &= 2y_1 + 3y_2 + y_3 \\
&= 2(3x_1 + 7x_2) + 3(4x_1 + 2x_2) + (-x_1 - 2x_2) \\
&= (2 \times 3 + 3 \times 4 + (-1))x_1 + (2 \times 7 + 3 \times 2 + (-2))x_2 \\
&= 17x_1 + 18x_2
\end{aligned}$$

Similarly, $z_2 = 39x_1 + 41x_2$.

$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 7 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 & 7 \\ 4 & 2 \\ -1 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Iteration 1: $\mathit{num} = 84$, $\mathit{denom} = 33$, $q = 2$, $\mathit{rem} = 18$
Iteration 2: $\mathit{num} = 33$, $\mathit{denom} = 18$, $q = 1$, $\mathit{rem} = 15$
Iteration 3: $\mathit{num} = 18$, $\mathit{denom} = 15$, $q = 1$, $\mathit{rem} = 3$
Iteration 4: $\mathit{num} = 15$, $\mathit{denom} = 3$, $q = 5$, $\mathit{rem} = 0$
Iteration 5: $\mathit{num} = 3$, $\mathit{denom} = 0 \Rightarrow \gcd(84, 33) = 3$

Procedure Calls

It is useful to extend our algorithmic language to have procedures that we can call repeatedly. For example, we may want to have a procedure for computing gcd or factorial, that we can call with different arguments. Here's the notation used in the book:

procedure Name(variable list)
    procedure body (includes a **return** statement)
endpro

- The **return** statement returns control to the portion of the algorithm from where the procedure was called

Example:

procedure Factorial(n)
    $\mathit{fact} \leftarrow 1$
    $m \leftarrow n$
    repeat until $m = 1$
      $\mathit{fact} \leftarrow \mathit{fact} \times m$
      $m \leftarrow m - 1$
    endrepeat
    return *fact*
endpro

Recursion

Recursion occurs when a procedure calls itself.

Classic example: Towers of Hanoi

Problem: Move all the rings from pole 1 and pole 2, moving one ring at a time, and never having a larger ring on top of a smaller one.

How do we solve this?

- Think recursively!
- Suppose you could solve it for $n-1$ rings? How could you do it for n?

Solution

- Move top $n-1$ rings from pole 1 to pole 3 (we can do this by assumption)
  - Pretend largest ring isn't there at all
- Move largest ring from pole 1 to pole 2
- Move top $n-1$ rings from pole 3 to pole 2 (we can do this by assumption)
  - Again, pretend largest ring isn't there

This solution translates to a recursive algorithm:

- Suppose robot$(r \to s)$ is a command to a robot to move the top ring on pole r to pole s
- Note that if $r, s \in \{1, 2, 3\}$, then $6-r-s$ is the other number in the set

procedure H(n, r, s) [Move n disks from r to s]
  if $n = 1$ **then** robot$(r \to s)$
    else $H(n-1, r, 6-r-s)$
      robot$(r \to s)$
      $H(n-1, 6-r-s, s)$
  endif
  return
endpro

Tree of Calls

Suppose there are initially three rings on pole 1, which we want to move to pole 2:

Analysis of Algorithms

For a particular algorithm, we want to know:

- How much time it takes
- How much space it takes

What does that mean?

- In general, the time/space will depend on the input size
  - The more items you have to sort, the longer it will take
- Therefore want the answer as a function of the input size
  - What is the best/worst/average case as a function of the input size.

Given an algorithm to solve a problem, may want to know if you can do better.

- What is the *intrinsic complexity* of a problem?

This is what *computational complexity* is about.

Towers of Hanoi: Analysis

procedure H(n, r, s) [Move n disks from r to s]
  if $n = 1$ **then** robot$(r \to s)$
    else $H(n-1, r, 6-r-s)$
      robot$(r \to s)$
      $H(n-1, 6-r-s, s)$
  endif
  return
endpro

Let h_n = # moves to move n rings from pole r to pole s.

- Clearly $h_1 = 1$
- Algorithm shows that $h_n = 2h_{n-1} + 1$
  - $h_2 = 3$; $h_3 = 7$; $h_4 = 15$; ...
  - $h_n = 2^n - 1$

We'll prove this formally later, when we also show that this is optimal.