On the Limits of Computing

- **Reasons for Failure**
  1. Runs too long
     - Real time requirements
     - Predicting yesterday's weather
  2. Non-computable!
  3. Don't know the algorithm

- **Complexity, N**
  - Time
  - Space

- **Tractable and Intractable**

---

On the Limits of Computing

- **Intractable Algorithms**
  - Computer "crawls" or seems to come to halt for large N
  - Large problems *essentially unsolved*
  - May never be able to compute answer for some obvious questions

- **Chess**
  - Here N is number of moves looking ahead
  - We *have* an Algorithm!
    - Layers of look-ahead: If I do this, then he does this, ....
    - Problem Solved (?!)
  - Can Represent Possibilities by Tree
  - Assume 10 Possibilities Each Move
    - $t = A \cdot 10^N$ or $O(A^N)$

- **Exponential !!!**

---

Exponential Algorithms

- **Recognizing Exponential Growth**
  - Things get BIG very rapidly
  - Numbers seem to EXPLODE
  - KEY: at each *add* step, work *multiplies* rather than *adds*

- **Exponential = $O(A^N) = Intractable**

- **Traveling Salesperson Example**
  - Visit N Cities in *Optimal* Order
  - Optimize for minimum:
    - Time
    - Distance
    - Cost

- **N factorial (N!) Possibilities**
  - N! is (very) roughly $N^N$
    - Sterling's approximation: $N! \approx \sqrt{2\pi N}(N/e)^N$

- **Typical of some very practical problems**

---

Traveling Salesperson Examples

- **3 cities 2! = 2 possible routes (1 of interest)**
  - abc
  - acb

- **4 cities 3! = 6 possible routes (3 of interest)**
  - abcd
  - abdc
  - acbd
  - acdb
  - adbc
  - adcb

- (Only half usually of interest because just reverse of another path)
**Traveling Salesperson Examples**

5 cities $4! = 24$ possible routes  
(12 of interest)

- abede
- abced
- abdce
- abdec
- abced
- abedc
- acbde
- acbed
- acdbe
- acdeb
- acebd
- acedb

**Towers of Hanoi**

$t = 0.00549 \times 2^N$

(for a very old PC)

N  t  
5  .17 sec  
10  5.62 sec  
15  3.00 min  
20  1.6 hour  
25  2.13 day  
30  68.23 day  
35  5.98 year  
40  191.3 year  
45  6120 year  
50  196 K year  
55  6.27 M year  
60  201 M year  
65  6.42 G year  
70  205 G year  

What would a faster computer do for these numbers?

**Intractable Algorithms**

- Other Games
- More hardware not the answer!
- Predicting Yesterday's Weather
- Actual Examples for Time Complexity

**Existence of Noncomputable Functions**

- Approach
  - Matching up Programs and Functions
  - E.g., assume 3 functions, only 2 programs
  - Without details, conclude one function has no program
- Have: Uncountable Infinity of Functions Mapping int to int
  - How can we show that is true?
  - Functions can be seen as columns in tables
  - Put all functions into a huge (infinite!) table
  - Show that even that cannot hold them all
  - Can you identify the functions in the following table?
Table of All Integer to Integer Functions

1 1 2 6 0 0 8 2 1 4 . ..
2 4 4 7 0 1 8 4 1 7 . ..
3 9 6 8 0 0 8 6 2 10 . ..
4 16 8 9 1 1 8 16 3 13 . ..
5 25 10 10 1 0 8 10 5 16 . ..
6 36 12 11 1 1 8 36 8 19 . ..
7 49 14 12 1 0 8 14 13 22 . ..
8 64 16 13 1 1 8 64 21 25 . ..
9 81 18 14 1 0 8 18 34 28 . ..
. . . . . . . . . . . . . . . . . . . . . . . . . . .

A Function NOT in this (inclusive?) Table

1+1 1 2 6 0 0 8 2 1 4 . ..
2 4+1 4 7 0 1 8 4 1 7 . ..
3 9 6+1 8 0 0 8 6 2 10 . ..
4 16 8 9+1 1 1 8 16 3 13 . ..
5 25 10 10 1+1 0 8 10 5 16 . ..
6 36 12 11 1+1 8 36 8 19 . ..
7 49 14 12 1 0 8+1 14 13 22 . ..
8 64 16 13 1 1 8 64+1 21 25 . ..
9 81 18 14 1 0 8 18 34+128 . ..
10 100 20 15 1 1 8 100 55 31+1 . ..
. . . . . . . . . . . . . . . . . . . . . . . . . . .

Existence of Noncomputable Functions

- All Programs Can be Ordered (thus Countable)
  - By size, shortest program first
  - Just use alphabetical order
- Try to Draw Lines Between Functions and Programs
  - Could draw lines from every program to every function
  - But, have proved functions uncountable...
  - Thus, There Must be Functions With NO Programs!
- Hard to come up with function that computer can't produce
  - Possible example: true random generator
    (No algorithm can produce truly random number sequence)
  - Use Table
  - Program must be of finite size; Requires infinite table

Noncomputable Programs

- Programs that Read Programs
  - What programs have we used that read in programs?
  - Express programs as a single string (formatting messed up)
  - Therefore, could write program to see if there is an if statement in the program: answers YES or NO
  - How about, Does program halt?
  - Lack of while (and functions) guarantees a halt
  - Not very sophisticated
  - Not Halting for All Possible Inputs is usually considered a Bug
- Solving the Halting Problem
  - Write specific code to check out more complicated cases
  - Gets more and more involved...
The Halting Problem: Does it Halt?

- Consider Following Program: Does it halt for all input?
  ```java
  // input an integer value for k
  while (k > 1)
  {
    if ((k/2) == k) // is k even?
      k = k / 2;
    else
      k = 3 * k + 1;
  }
  ```

- Try It!
  - e.g. 17: 52 26 13, 40 20 10 5, 16 8 4 2 1
  - For a long time, no one knew whether this quit for all inputs.

Proving Noncomputability

- Mathematicians have proven that no one, finite program can check this property for all possible programs
- Examples of non-computable problems
  - Equivalence: Define by same input > same output
  - Use variation of above program; not sure it ends
  - Cannot generally prove equivalence
- Use Proof by Contradiction (Indirect Proof)
- Proving non-computability
  - Sketch of proof

Noncomputability Proof

- **Assume Existence of Function `halt`:**
  ```java
  String halt(String p, String x);
  ```
  - Inputs: `p = program`, `x = input data`
  - Returns: "Halts" or "Does not halt"
- **Can now write:**
  ```java
  String selfhalt(String p);
  ```
  - Inputs: `p = program`
  - Returns: "Halts on self" or "Does not halt on self"
  - Uses: `halt(p, p)`
  - i.e.: asking if halts when program `p` uses itself as data

Noncomputability Proof.2

- **Now write function `contrary`:**
  ```java
  void contrary()
  {  
    TextField program = new TextField(1000);
    String p, answer;
    p = program.getText();
    answer = selfhalt(p);
    if (answer.equals("Halts on self"))
    {  
      while (true) // infinite loop
        answer = "x";
    }  
    else
      return;  // i.e., halts
  }  
  ```
  - "Feed it" this program.
Noncomputability Proof.3

- Paradox!
  - If $\text{halt}$ program decides it halts, it goes into infinite loop and goes on forever.
  - If $\text{halt}$ program decides it doesn't halt, it quits immediately.
- Therefore $\text{halt}$ cannot exist!

- Whole classes of programs on program behavior are non-computable
  - Equivalence
  - Many other programs that deal with the behavior of a program.

Living with Noncomputability

- What Does It All Mean?
  - Not necessarily a very tough constraint unless you get “too greedy”.
  - Programs can't do everything.
    - Beware of people who say they can!