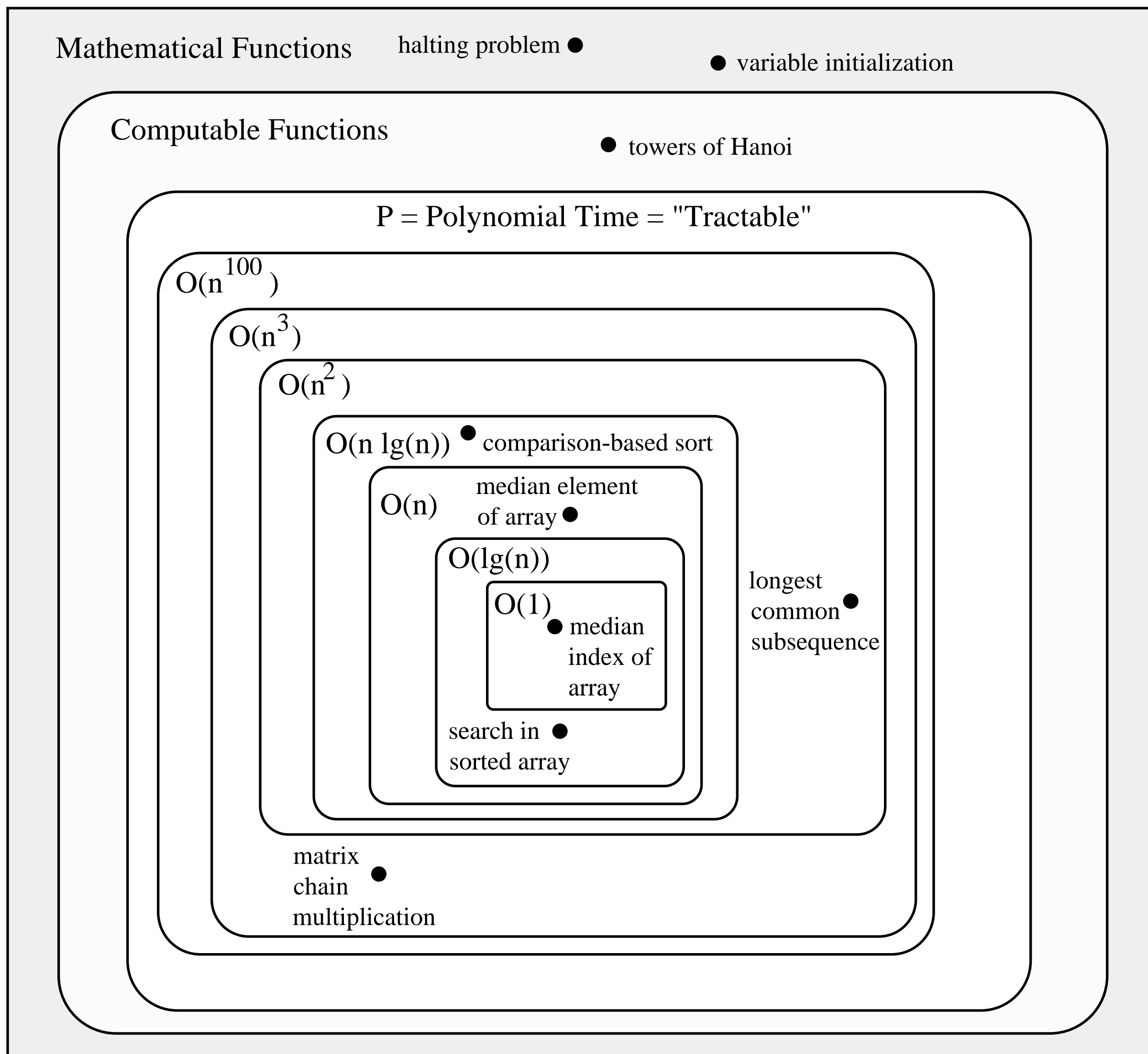
Wellesley College **CS231 Algorithms** **November 22, 1996**
Handout #32

P & NP

Reading: Sections 36.1 -- 36.2

Computational Classes

Suppose we classify problems according to the best possible worst-case time solution. Diagramatically can depict this classification scheme via nested classes:



Notes:

- Computable functions not in P are considered "intractible" (e.g. towers of Hanoi).

- The size of the class of mathematical functions is a larger infinity than the size of the class of computable functions. Consider boolean functions on integers, i.e.. functions that map integers to booleans. If N is the "size" of the integers, there are 2^N = an uncountable infinity of such functions. But there are only a countable infinity of programs (i.e., there is a process for enumerating them).

Unclassified Problems

Surprisingly, there are a large number of important problems that remain unclassified along the tractible/intractible dimension. Some examples:

- Travelling Salesperson Problem (TSP): given a weighted complete graph, find the least weight ***tour*** = cycle that visits each vertex exactly once.
- Hamiltonian Graphs (HAM): Determine if a graph has a ***hamiltonian cycle*** = a cycle that visits each each vertex exactly once.
- Graph Coloring: What is the smallest number of colors that can be used to color the vertices of a graph such that each edge connects vertices of different colors.
- Vertex Cover: What is the smallest set of vertices that "covers" all the edges?
- Clique: What is the size of the largest complete subgraph of a graph?
- Satisfiability (SAT): Is a boolean formula (a formula constructed out of boolean variables, ands, ors, and nots) ***satisfiable*** ? I.e., is there some assignment of variables to values that makes the formula true?

Most people believe that all the above are intractible, but no one knows for sure.

Interestingly, it turns out that these problems are related in the sense that some of these problems can be reduced to instances of others. (This is the topic of the next lecture.)

Languages

Here, we formalize a notion of a language that will allow us to talk about problems more precisely.

An ***alphabet*** is a set of symbols.

$_1$ = {0, 1}

$_2$ = {A, B, ..., Z, (,), , }

A ***language*** is a set of strings over an alphabet.

$_1^*$ = { , 0, 1, 00, 01, 10, 11, 000, 001, ...} Note: stands for the empty string

L_1 = {1, 01, 10, 001, 010, 111, ...} Strings with odd number of ones

L_2 = {00, 01, 10, 11} Strings of length two

There are several operations for combining languages:

Union: L_1 L_2

Intersection: L_1 L_2

Concatenation: $L_2 L_1$ = {001, 011, 101, 111, 0001, 0101, 1001, 1101, ...}

Note: L^n = L L ... L (n times)

Star: $L_2^* = L_2^0$ L_2^1 L_2^2 L_2^3 ...
= { , 00, 01, 10, 11, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, ...}

Complement:

$_1^{*C}$ = { }

L_1^C = { , 0, 00, 11, 000, 011, 101, 110, 0000, ...}

L_2^C = { , 0, 1, 000, 001, 010, 011, 100, 101, 110, 111, 0000, ...}

Problems

Can encode ***instances*** of a problem and solutions of instances as strings in an appropriate language.

For example, a shortest path problem between A and B in a given graph might be encoded as

```
(((A, B, C, D, E, F),
          ((A,C), (C,E), (E,D), (D,B)))
  A, B)
```

The solution might be encoded as `(A, C, E, D, B)`

A problem can be formalized as a set of ***pairs of problem instances and solutions***.

Shortest-Path =

```
{...,
  <(((A, B, C, D, E, F),
      ((A,C), (C,E), (E,D), (D,B)))
    A, B),
   (A, C, E, D, B)>,
 ...}
```

A **decision problem** is one whose solutions are all yes or no:

Shortest-Path-Less-Than-k =

```
{...,
  <((V, E), A, B, 4), no)>
  <((V, E), A, B, 5), yes)>
  <((V, E), A, B, 6), yes)>
 ...}
```

Can represent decision problems as languages by encoding instances for which solution is yes:

$L_{\text{Shortest-Path-Less-Than-k}}$ =

```
{...,
 <((V, E), A, B, 5), yes)>,
 <((V, E), A, B, 6), yes)>,
...}
```

Definitions

Algorithm A ***accepts*** string s if A(s) returns 1.

Algorithm A ***rejects*** string s if A(s) returns 0.

Language ***accepted*** by A is L_A = {s in {0,1}* | A(s) = 1}

A language is ***decided*** by A if A(s) returns 0 for all s in L_A^C.

A accepts (decides) language L ***in polynomial time*** if
  for every length-n string s
    there exists a k
      such that A accepts (decides) s in $O(n^k)$ time.

P = {L | there exists an A such that A decides L in polynomial time}

A verification algorithm A(s, c) verifies s if there is a certificate c such that A(s,c) returns 1.

The language verified by A is L_A = {s in {0,1}* | there exists c such that A(s, c) = 1}

NP = {L | there exists an A such that A verifies L in polynomial time.}
  Note: certificate for each s must be polynomial in |s|.

co-NP = {L | L^C is in NP}

Open Problems

- Is P = NP?
- Is NP = co-NP?

Possible scenarios: