ECE250:
Algorithms and Data Structures

NP Completeness

Ladan Tahvildari, PEng, SMIEEE

Associate Professor

Software Technologies Applied Research (STAR) Group

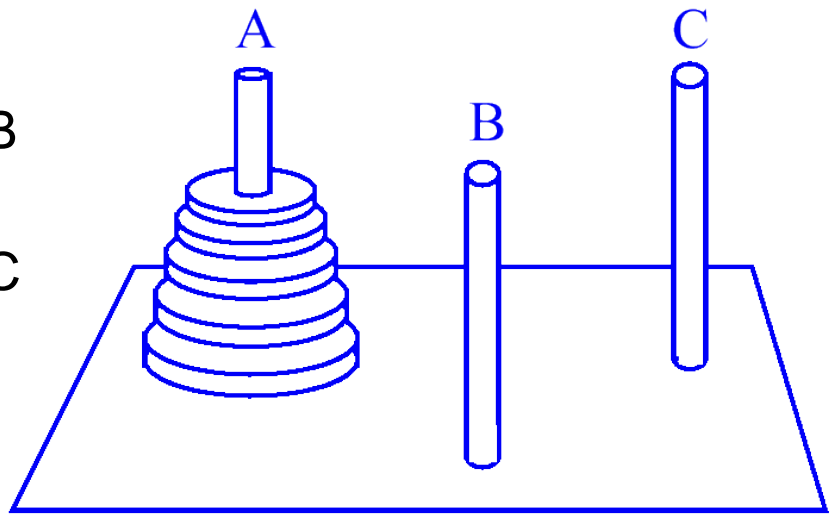
Dept. of Elect. & Comp. Eng.

University of Waterloo

Materials from CLRS: Chapter 34

Towers of Hanoi

- **Goal:** Transfer all *n* disks from peg A to peg C
- **Rules:**
  - move one disk at a time
  - never place larger disk above smaller one
- Recursive solution:
  - transfer $n - 1$ disks from A to B
  - move largest disk from A to C
  - transfer $n - 1$ disks from B to C
- Total number of moves:
  - $T(n) = 2T(n - 1) + 1$



Towers of Hanoi (con't)

Recurrence relation:

$$T(n) = 2\,T(n-1) + 1$$

$$T(1) = 1$$

- Solution by repeated substitution:

$$T(n) = 2\,(2\,T(n-2) + 1) + 1 =$$
$$= 4\,T(n-2) + 2 + 1 =$$
$$= 4\,(2\,T(n-3) + 1) + 2 + 1 =$$
$$= 8\,T(n-3) + 4 + 2 + 1 = \ldots$$
$$= 2^i\,T(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 2^0$$

- the expansion stops ($n - i = 1$) when $i = n - 1$

$$T(n) = 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0$$

Towers of Hanoi (cont')

- This is a ***Geometric Series***, so that we have

  $T(n) = 2^n - 1 = O(2^n)$

- The running time of this algorithm is ***exponential (k^n)*** rather than ***polynomial (n^k)***

- Good or bad news?
  - the Tibetan priests were confronted with a tower problem of 64 rings...
  - Assuming the priests move **one ring per second**, it would take **~585 billion years** to complete the process!

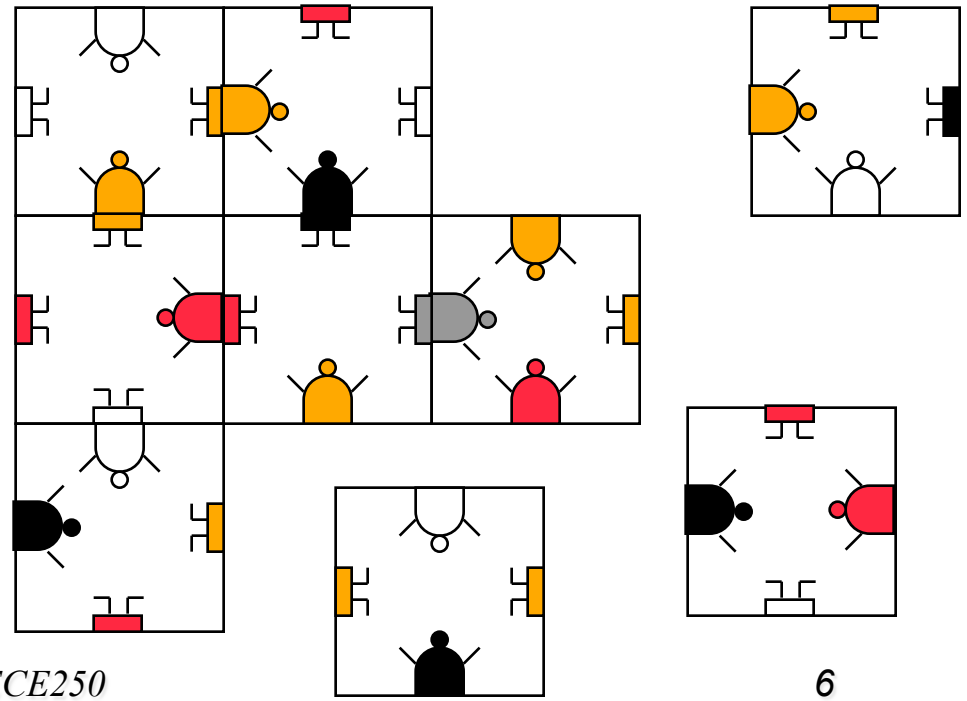
Decision Problems

- Are such long running times linked to the size of the solution of an algorithm?
  - No! To show that, we in the following consider only TRUE/FALSE or yes/no problems – ***decision problems***
- We can usually transform an *optimization problem* into an **easier** *decision problem*:
  - *Optimization* problem *O* : "Find a shortest path between vertices *u* and *v* in a graph *G*."
  - Related *decision* problem *D* : "Determine if there is a path between vertices *u* and *v* in a graph *G* shorter than *k.*"
  - If we have an easy way to solve *O*, we can use it to solve *D*.
  - If we show that *D* is hard, we know that *O* must be also hard.

Monkey Puzzle

An example of a decision problem

- Nine square cards with imprinted "monkey halves"
- The goal is to arrange the cards in 3x3 square with matching halves...

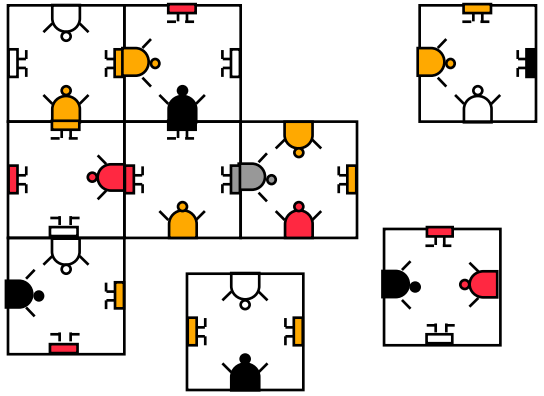


More about Monkey Puzzle …

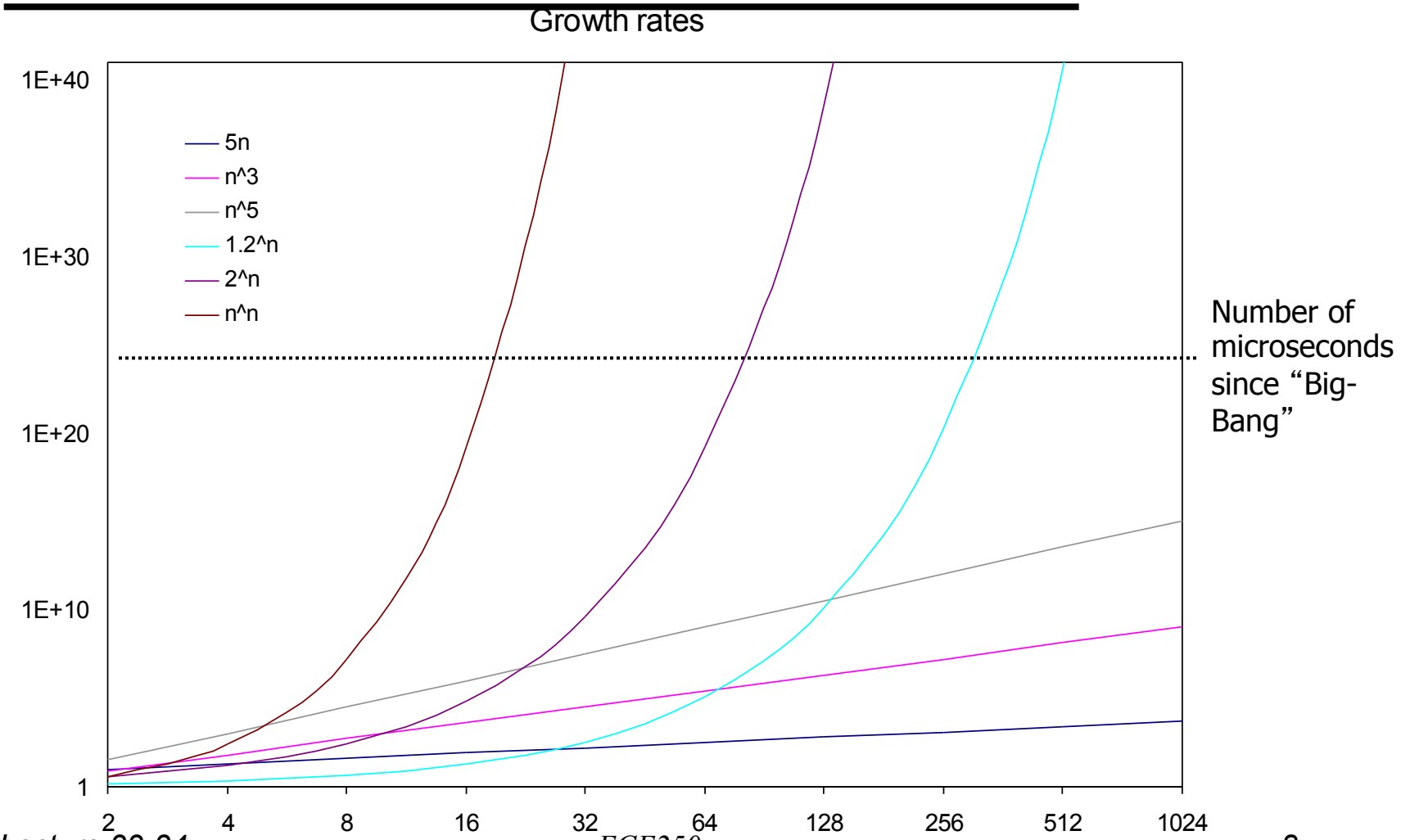
- Assumption: orientation is fixed

- *Does any MxM arrangement exist that fulfills the matching criterion?*

- Brute-force algorithm would take ***n*! time** to verify whether a solution exists
  - assuming n = 25, it would take 490 billion years on a one-million-per-second arrangements computer to verify whether a solution exists



Reasonable vs. Unreasonable

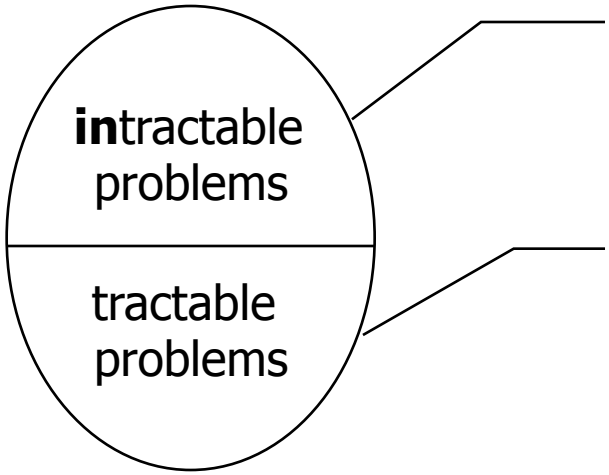


Reasonable vs. Unreasonable

	function/ n	10	20	50	100	300
Polynomial	n^2	1/10,000 second	1/2,500 second	1/400 second	1/100 second	9/100 second
Polynomial	n^5	1/10 second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
Exponential	2^n	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75 digit-number of centuries
Exponential	n^n	2.8 hours	3.3 trillion years	a 70 digit-number of centuries	a 185 digit-number of centuries	a 728 digit-number of centuries

Reasonable vs. Unreasonable

- ”Good”, reasonable algorithms
  - algorithms bound by a polynomial function n^k
  - **Tractable Problems**
- ”Bad”, unreasonable algorithms
  - algorithms whose running time is above n^k
  - **Intractable Problems**



problems **not** admitting reasonable algorithms

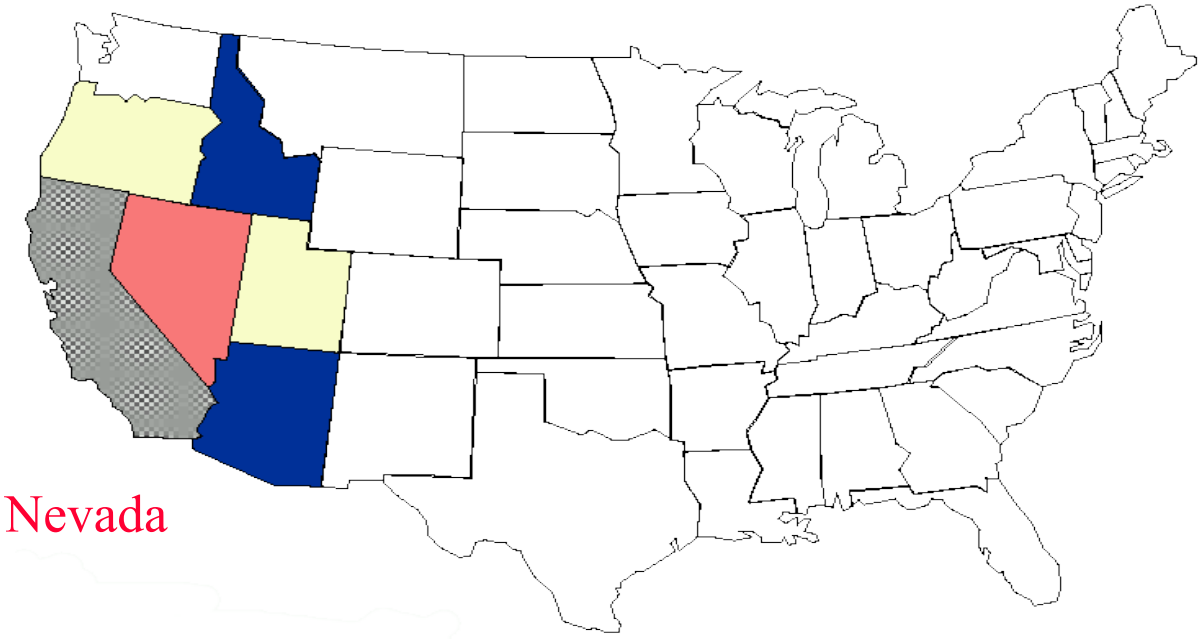
problems admitting reasonable (polynomial-time) algorithms

So What!

- Computers become faster every day
  - insignificant (a constant) compared to exp. running time
- Maybe the Monkey puzzle is just one specific problem, we could simply ignore
  - the monkey puzzle falls into a category of problems called NPC (NP complete) problems (~1000 problems)
  - all admit **unreasonable** solutions
  - **not known** to admit **reasonable** ones…

Coloring Problem (COLOR)

- **3-Color**: given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color



NO instance
Impossible to 3-color Nevada and bordering states!

Coloring Problem (cont')

- Any map can be **4-colored**
- Maps that contain no points that are the junctions of an odd number of states can be **2-colored**
- No polynomial algorithms are known to determine whether a map can be **3-colored** – it's an NP-complete problem

P Complexity Class

- Definition of P:
  - Set of all decision problems solvable in polynomial time on *real* computers

- Examples:
  - SHORTEST PATH: Is the shortest path between *u* and *v* in a graph shorter than *k*?
  - RELPRIME: Are the integers *x* and *y* relatively prime?
    - YES: (*x, y*) = (34, 39).
  - LCS: Given two strings *x* and *y*, is the length of their longest common subsequence > *k*?
    - YES: (*x, y, k*) = (”CGTTAG”, ”GGTACG”, 3)

NP [Non-deterministic Polynomial time]

- Definition of NP:
  - Set of all decision problems solvable in polynomial time on a *non-deterministic* computer
- Definition important because it links many fundamental problems
- Useful alternative definition
  - Set of all decision problems with efficient verification algorithms
    - Efficient = polynomial number of steps on deterministic time
    - Verifier: algorithm for decision problem with extra input

NP (cont')

- NP = set of decision problems with efficient verification algorithms

- Why doesn't this imply that all problems in NP can be solved efficiently?

  - BIG PROBLEM: need to know certificate ahead of time
    - real computers can simulate by guessing all possible certificates and verifying
    - naïve simulation takes exponential time unless you get "lucky"

NP-Completeness

- Informal definition of **NP-hard** class:
  - A problem with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently
- **NP-complete** problems are NP problems that are NP-hard
  - ”Hardest computational problems” in NP

Informally, a problem is NP-complete iff it has two important features:
- It is hard to find solutions of the problem.
  But it is relatively easy to check its correctness of a solution.

NP-Completeness (cont')

- Each NPC problem's faith is tightly coupled to all the others (complete set of problems)

- Finding a **polynomial time algorithm for one NPC problem** would **automatically** yield a polynomial time algorithm **for all NP problems**

- Proving that one NP-complete problem has an **exponential lower bound** woud **automatically** proof that **all other NP-complete** problems have exponential lower bounds

Reducibility

- *How can we prove such a statement?*

- **Polynomial Time Reduction**:
  - given two problems
  - it is an algorithm running in polynomial time that reduces one problem to the other such that
    - given input X to the first and asking for a yes/no answer
    - we transform X into input Y to the second problem such that its answer matches the answer of the first problem

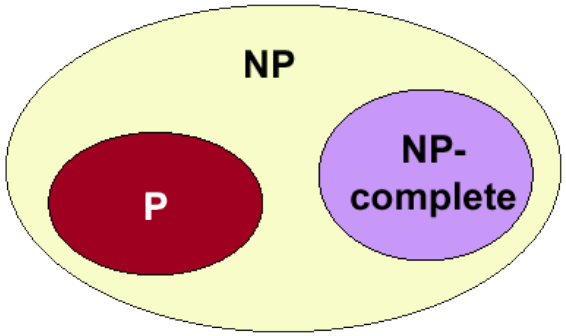
Reduction Example

- Reduction is a general technique for showing that one problem is harder (easier) than another
  - For problems *A* and *B*, we can often show: if *A* can be solved efficiently, then so can *B*
  - In this case, we say *B* reduces to *A* (*B* is "easier" than *A*, or, *B* cannot be ”worse” than *A*)
- NP-complete problem *A*:
  - It is NP problem
  - For any NP problem *B* can be *polynomialy* reduced to *A*

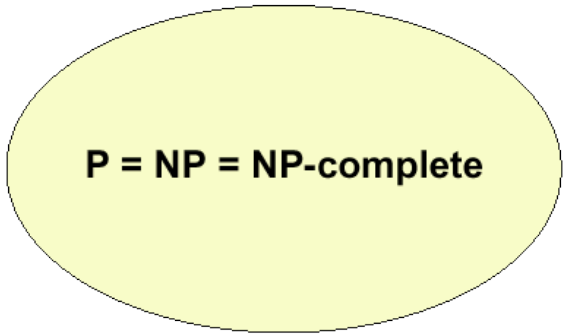
The Main Question

- Does P = NP?
  - Is the original DECISION problem as easy as VERIFICATION?

- Most important open problem in theoretical computer science. Clay institute of mathematics offers **one-million** dolar prize for solution!
  - http://www.claymath.org/millennium/



If P ≠ NP



If P = NP

The Main Question (cont')

- If P=NP, then:
  - Efficient algorithms for 3- COLOR, TSP, and factoring.
  - Modern cryptography breaks down on conventional machines

- If no, then:
  - Can't hope to write efficient algorithms for NP-complete problems
  - But maybe efficient algorithm still exists for testing the primality of a number – i.e., there are some problems that are NP, but not NP-complete

The Main Question (cont')

- Probably no, since:
  - Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP-complete problems without success
  - Consensus opinion: $P \neq NP$
- But maybe yes, since:
  - No success in proving $P \neq NP$ either

Dealing with NP-Completeness

- Hope that a worst case doesn't occur
  - Complexity theory deals with worst case behavior. The instance(s) you want to solve may be "easy"
    - TSP where all points are on a line or circle
    - 13,509 US city TSP problem solved (Cook et. al., 1998)
- Change the problem
  - Develop a heuristic, and hope it produces a good solution.
  - Design an approximation algorithm: algorithm that is guaranteed to find a high- quality solution in polynomial time
    - active area of research, but not always possible
- Keep trying to prove P = NP.

The Big Picture

- Summarizing: it is not known whether NP problems are tractable or intractable

- But, there exist provably intractable problems
  - Even worse – there exist problems with running times unimaginably worse than exponential!

- More bad news: there are **provably non-computable (un-decidable)** problems
  - There are no (and there will not ever be!!!) algorithms to solve these problems