Outline
• The complexity class P
• Examples of problems in P
• 2-SAT, 3-SAT
• The complexity class EXP
• Time Hierarchy Theorem
• hardness and completeness – an EXP-complete problem

Extended Church-Turing Thesis
• the belief that TMs formalize our intuitive notion of an efficient algorithm is:
  The "extended" Church-Turing Thesis
  everything we can compute in time $t(n)$ on a physical computer can be computed on a Turing Machine in time $t(n)^{O(1)}$ (polynomial slowdown)
• quantum computers challenge this belief

Time Complexity
• interested in a coarse classification of problems. For this purpose,
  – treat any polynomial running time as "efficient" or "tractable"
  – treat any exponential running time as inefficient or "intractable"
Key definition: "P" or "polynomial-time" is $P = \bigcup_{k \geq 1} TIME(n^k)$

Examples of languages in P
• Recall: positive integers $x$, $y$ are relatively prime if their Greatest Common Divisor (GCD) is 1.
• will show the following language is in P:
  RELPRIME = \{<x, y> : x and y are relatively prime\}
• what is the running time of the algorithm that tries all divisors up to $\min(x, y)$?
Euclid’s Algorithm

- possibly earliest recorded algorithm

Example run on input <10, 22>:

\begin{align*}
\text{x, y} &= 10, 22 \\
\text{x, y} &= 22, 10 \\
\text{x, y} &= 10, 2 \\
\text{x, y} &= 2, 0
\end{align*}

reject

Example run on input <24, 5>:

\begin{align*}
\text{x, y} &= 24, 5 \\
\text{x, y} &= 5, 4 \\
\text{x, y} &= 1, 0
\end{align*}

accept

Claim: value of x reduced by ½ at every execution of (2) except possibly first one.

Proof:
- after (2) x < y
- after (3) x > y
- if x/2 ≥ y, then x mod y < y ≤ x/2
- if x/2 < y, then x mod y = x − y < w/2

every 2 times through loop, (x, y) each reduced by ½
loops ≤ 2max{\log_2 x, \log_2 y} = O(n = |\langle x, y \rangle|); poly time for each loop

2SAT

- This is a disguised version of the language 2SAT = \{formulas in Conjunctive Normal Form with 2 literals per clause for which there exists a satisfying truth assignment\}
- CNF = “AND of ORs”
  \begin{align*}
  (A, b) \land (E, D) \land (d, C) \land (b, a)
  \end{align*}
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

A puzzle

- Find an efficient algorithm to solve the following problem:
- Input: sequence of pairs of symbols
e.g. \langle A, b \rangle, \langle E, D \rangle, \langle d, C \rangle, \langle b, a \rangle
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

A puzzle

- Find an efficient algorithm to solve the following problem:
- Input: sequence of pairs of symbols
e.g. \langle A, b \rangle, \langle E, D \rangle, \langle d, C \rangle, \langle b, a \rangle
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.
2SAT

Theorem: There is a polynomial-time algorithm deciding 2SAT ("2SAT ∈ P").

Proof: algorithm described on next slides.

Algorithm for 2SAT

• Build a graph with separate nodes for each literal.
  - add directed edge (x, y) iff formula includes clause (¬x v y) (equiv. to x ⇒ y)

  e.g. (x1 v ¬x2) ∧ (x5 v x4) ∧ (¬x4 v x3) ∧ (¬x2 v ¬x1)

Claim: formula is unsatisfiable iff there is some variable x with a path from x to ¬x and a path from ¬x to x in derived graph.

• Proof (⇒)
  - edges represent implication ⇒. By transitivity of ⇒, a path from x to ¬x means x ⇒ ¬x, and a path from ¬x to x means ¬x ⇒ x.

• Proof (⇐)
  - to construct a satisfying assign. (if no x with a path from x to ¬x and a path from ¬x to x):
    - pick unassigned literal s with no path from s to ¬s
    - assign it TRUE, as well as all nodes reachable from it; assign negations of these literals FALSE
    - note: path from s to t and s to ¬t implies path from ¬t to ¬s and t to ¬s, implies path from s to ¬s
    - note: path s to t (assigned FALSE) implies path from ¬t (assigned TRUE) to ¬s, so s already assigned at that point.

Algorithm for 2SAT

• Algorithm:
  - build derived graph
  - for every pair x, ¬x check if there is a path from x to ¬x and from ¬x to x in the graph

• Running time of algorithm (input length n):
  - O(n) to build graph
  - O(n) to perform each check
  - O(n) checks
  - running time O(n^2). 2SAT ∈ P.