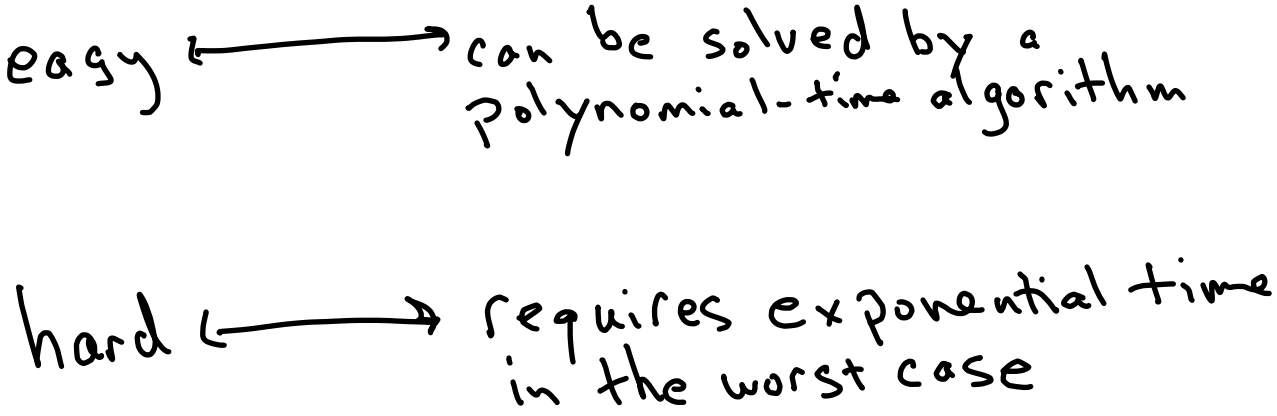
Section 19.3:
Easy and Hard Problems

Oversimplification



Polynomial-Time Algorithms

MergeSort ($O(n \log n)$ time)

Kosaraju's SCC Algorithm ($O(m+n)$ time)

Dijkstra's shortest path algorithm ($O((m+n) \log n$ time)

Kruskal/Prim (")

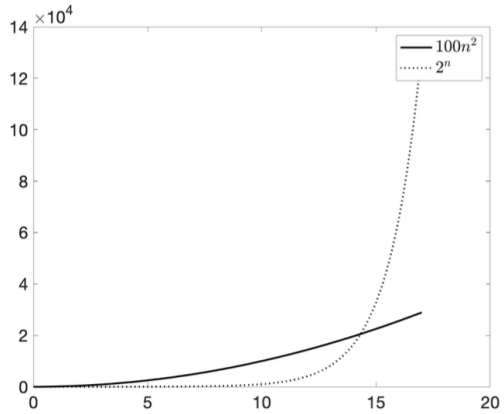
Needleman-Wunsch sequence alignment ($O(mn)$ time)

Floyd-Warshall all-pairs shortest paths ($O(n^3)$ time)

Polynomial-time algorithm: runs in time $O(n^d)$ for some constant d (independent of n).

Polynomial vs. Exponential Time

- gulf gets more dramatic as technology advances and we tackle larger input sizes



- polynomial time $\Leftrightarrow$ increasing computational power increases multiplicatively the input size you can accommodate in a fixed amount of time (e.g., 1 hour).

Easy Problems

Definition: A problem is polynomial-time solvable if there is a polynomial-time algorithm that solves it.

Note: not polynomial-time solvable $\Rightarrow$ not even an $O(n^{100})$- or $O(n^{10000})$-time algorithm for it!!

Hard Problems

Weak evidence: A poly-time TSP algorithm would solve a problem that has resisted the efforts of thousands of geniuses.

Strong evidence: A poly-time TSP algorithm would solve thousands of problems that have resisted the efforts of tens (hundreds!) of thousands of geniuses.

The $P \neq NP$ Conjecture

Definition: A problem is NP-hard if a polynomial-time algorithm solving it would refute the P≠NP conjecture.

P≠NP Conjecture (Informal Version): Checking an alleged solution is easier than coming up with one from scratch.

Subtleties

1. All bets off if "P ≠ NP" conjecture is false.
2. Only shows super-polynomial time is required.
   ["Exponential Time Hypothesis (ETH)" = conjecture that exponential time required]
3. A few problems seem to be "in between" poly-time solvable and NP-hard.