Theory of Computation
(Algorithmically Hard Problems)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

January 24, 2021

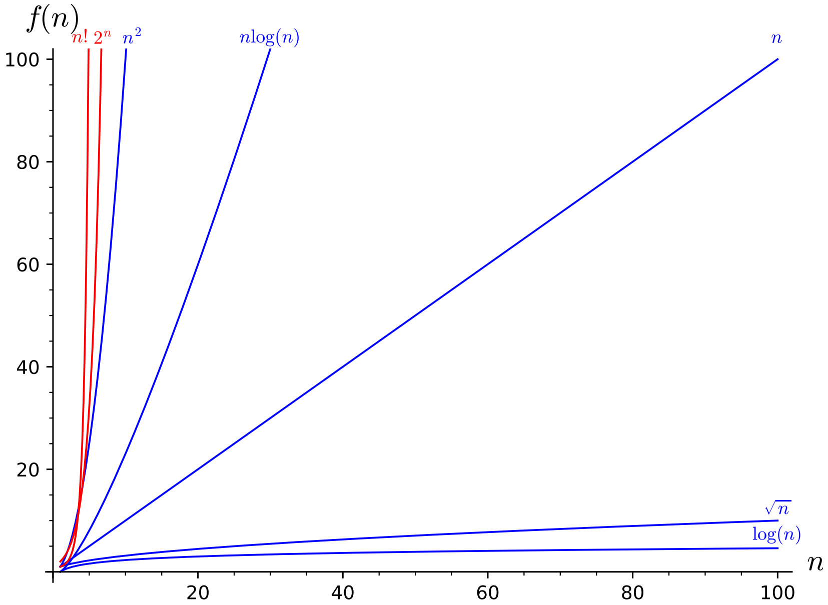
Algorithmically unsolvable problems

Problem	Running time
Simulate problem	∞
Halting problem	∞
Program correctness	∞
Program equivalence	∞
Integral roots of a polynomial	∞
Goodstein's theorem	∞
Generalized $(3n+1)$ problem	∞
Post correspondence problem	∞

Algorithmically solvable problems

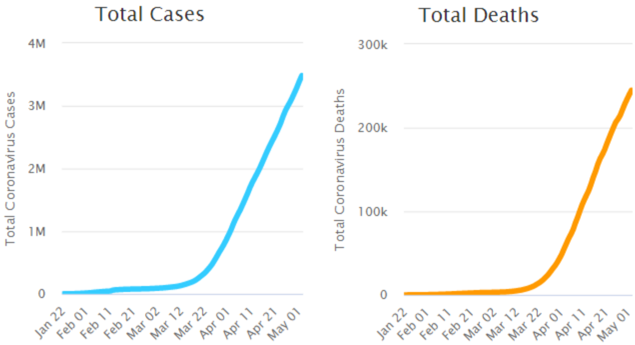
Problem	Running time
Search in a sorted array	$\mathcal{O}(\log n)$
Search in an unsorted array	$\mathcal{O}(n)$
Integer addition	$\mathcal{O}(n)$
Generate primes	$\mathcal{O}(n \log\log n)$
Sorting	$\mathcal{O}(n \log n)$
Fast Fourier transform	$\mathcal{O}(n \log n)$
Integer multiplication	$\mathcal{O}(n^2)$
Matrix multiplication	$\mathcal{O}(n^3)$
Linear programming	$\mathcal{O}(n^{3.5})$
Primality test	$\mathcal{O}(n^{10})$
Satisfiability problem	$\mathcal{O}(2^n)$
Traveling salesperson problem	$\mathcal{O}((n-1)!)$
Sudoku, chess, checkers, go	expo. class

Polynomial and exponential functions



Exponential functions

- Moore's law (Doubling of computing power every 18 months)
- Compound interest in banks
- Coronavirus



Source: `https://www.worldometers.info/coronavirus/`

Closest pair problem

Problem

- Find the two closest points in a set of n points.

Analysis

- Polynomial-sized search space
  Search space = Total #pairs of points $= \mathcal{O}\left(n^2\right)$
- Polynomial-time algorithms exist

Algorithm	Time complexity	Class
Exhaustive search	$\mathcal{O}\left(n^2\right)$	poly.
Divide-and-conquer	$\mathcal{O}\left(n \log^2 n\right)$	poly.
Divide-and-conquer	$\mathcal{O}\left(n \log n\right)$	poly.

Shortest path problem

Problem

- Given a weighted (with nonnegative weights) directed graph G, find a path between a source vertex s and a destination vertex t such that the sum of the weights of its constituent edges is minimized.

Analysis

- Exponential-sized search space
  Search space = Total #paths from s to t = $\sum_{i=2}^{n} {}^{n}C_i$
- Polynomial-time algorithms exist

Algorithm	Time complexity	Class
Exhaustive search	$\sum_{i=2}^{n}(i \cdot {}^{n}C_i)$	expo.
Bellman–Ford algorithm	$\mathcal{O}\left(\lvert V\rvert^2 \log \lvert V\rvert\right)$	poly.
Dijkstra's algorithm	$\mathcal{O}\left(\lvert V\rvert^2\right)$	poly.
Dijkstra's algorithm with binary heap	$\mathcal{O}\left((\lvert E\rvert + \lvert V\rvert) \log \lvert V\rvert\right)$	poly.
Thorup's algorithm	$\mathcal{O}\left(\lvert E\rvert + \log\log \lvert V\rvert\right)$	poly.

Hard problems with easy solutions

- Exponential-sized search space. Polynomial-time algorithms.

Problem	Search space	Best algorithm	Class
Greedy Algorithms			
Minimum spanning tree	$\mathcal{O}\left(n^{n-2}\right)$	$\mathcal{O}\left(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert\right)$	poly.
Shortest path	$\sum_{i=2}^{n}(i \cdot {}^{n}C_i)$	$\mathcal{O}\left(\lvert E\rvert + \log\log \lvert V\rvert\right)$	poly.
Iterative Improvement			
Match n boys with n girls	$\mathcal{O}\left(n!\right)$	$\mathcal{O}\left(n^2\right)$	poly.
Linear programming	expo.	$\mathcal{O}\left(n^{3.5}\right)$	poly.

Traveling salesperson problem (TSP)

Problem

- Given a list of cities, the distances between each pair of cities, and the origin city, what is the shortest possible route that starts from the origin city, visits each city, and returns to the origin city?

Analysis

- Exponential-sized search space
  Search space = Total #routes = $\mathcal{O}\left((n-1)!\right)$
- It is unknown if polynomial-time algorithms exist

Algorithm	Time complexity	Class
Exhaustive search	$\mathcal{O}\left((n-1)!\right)$	expo.
Held-Karp algorithm	$\mathcal{O}\left(n^2 2^n\right)$	expo.

Hard problems with no known easy solutions

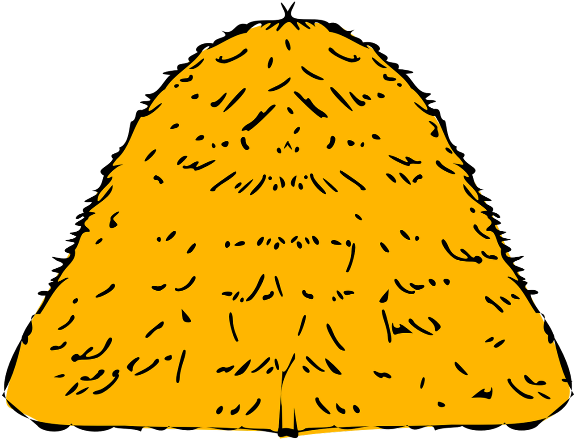
- Exponential-sized search space. No known polynomial-time algorithms.

Problem	Search space	Best algorithm	Class
Satisfiability	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	expo.
Traveling salesperson	$\mathcal{O}((n-1)!)$	$\mathcal{O}(n^2 2^n)$	expo.
Sudoku	expo.	–	expo.
Chess	expo.	–	expo.
Checkers	expo.	–	expo.
Go	expo.	–	expo.

Problem = Search for a needle in a haystack

Problem

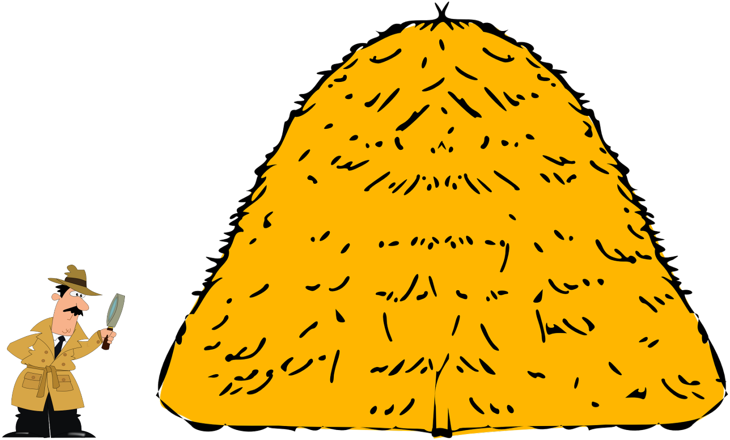
- Search for a needle in a haystack.



Inefficient algorithm

Solution

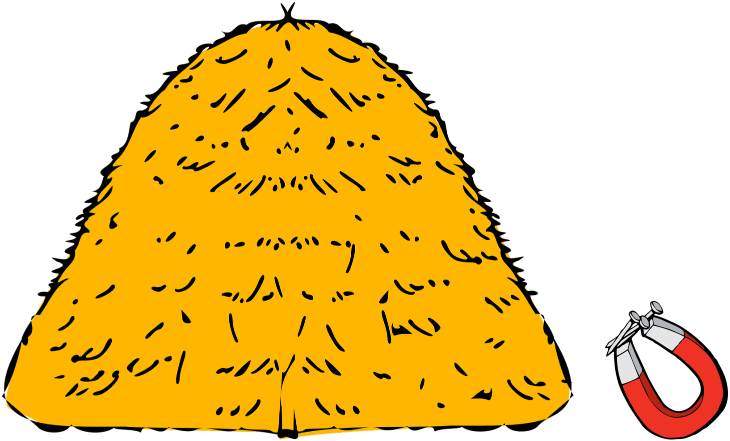
- Search through the entire haystack (search space).



Efficient algorithm

Solution

- Use a giant magnet.



Easy problems and possibly hard problems

Easy problems	Possibly hard problems
Shortest path	Longest path
Linear programming	Integer linear programming
Minimum spanning tree	Traveling salesperson
2-Satisfiability	3-Satisfiability
Min cut	Max cut
Planar 4-colorability	Planar 3-colorability
Bipartite vertex cover	Vertex cover
Bipartite matching	3D-matching
Independent set on trees	Independent set
Euler path	Rudrata path

- The problems on the right have escaped efficient algorithms for decades to centuries. Why?
- The problems on the right seem hard for the same reason – they are all equivalent.
- Each pair of those problems can be reduced to each other.

What is polynomial-time reduction?

Definition

- Reduction is a fantastic idea to solve one problem using another.
- Problem P_{old} poly.-time reduces to problem P_{new}, denoted by $P_{\text{old}} \leq_p P_{\text{new}}$, if any instance of problem P_{old} can be solved using the following:
  - (i) poly. number of standard computational steps.
  - (ii) poly. number of calls to function that solves problem P_{new}.
- $P_{\text{old}} \leq_p P_{\text{new}}$ means P_{new} is at least as hard as P_{old}.

Problem-Old(input-old) $\triangleright$ $P_{\text{old}} \leq_p P_{\text{new}}$

1. input-new $\leftarrow f$(input-old)
2. output-new $\leftarrow$ Problem-New(input-new)
3. output-old $\leftarrow g$(output-new)
4. return output-old

What is polynomial-time reduction?

Consequences

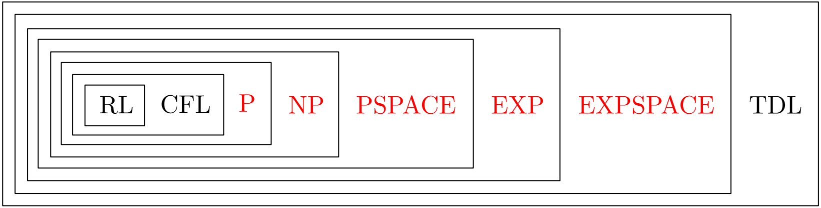
Suppose $P_{\mathsf{old}} \leq_p P_{\mathsf{new}}$.

- If P_{new} can be solved in polynomial time,
  then P_{old} can be solved in polynomial time.
- If P_{old} cannot be solved in polynomial time,
  then P_{new} cannot be solved in polynomial time.
- If $P_{\mathsf{new}} \leq_p P_{\mathsf{old}}$, then
  P_{old} can be solved in polynomial time
  iff P_{new} can be solved in polynomial time.

What are the complexity classes?

Definition

- P = Problems solvable in polynomial time.
- NP = Problems with solutions that can be verified/checked in polynomial time.
- PSPACE = Problems solvable with polynomial space.
- EXP = Problems solvable in exponential time.
- EXPSPACE = Problems solvable with exponential space.

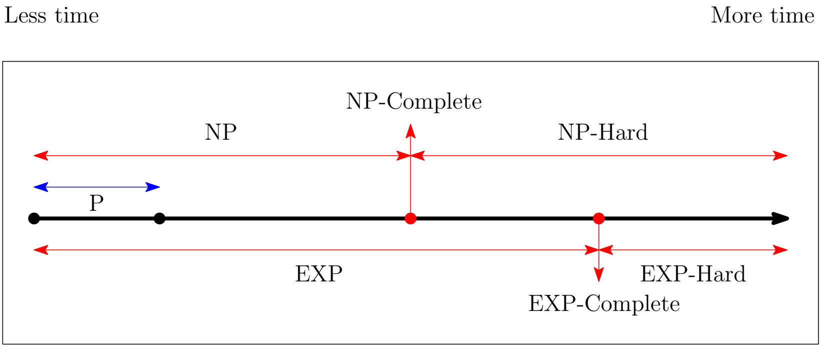


What are the complexity classes?

Definition

- NP = Problems solvable in poly time using nondeterminism
  = Problems with solutions that can be verified/checked
  in polynomial time.
- NP-Hard = Problems at least as hard as NP problems.
  Formally, a problem X is NP-Hard
  if every NP problem Y is polynomial-time reducible to X.
- NP-Complete = Hardest problems in NP.
  Formally, a problem X is NP-Complete
  if (i) X is in NP, and (ii) X is NP-Hard.

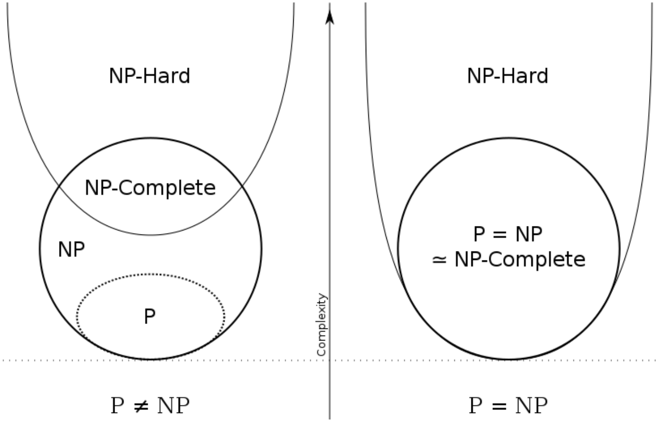
What are the complexity classes?



Unsolved problem: P = NP?

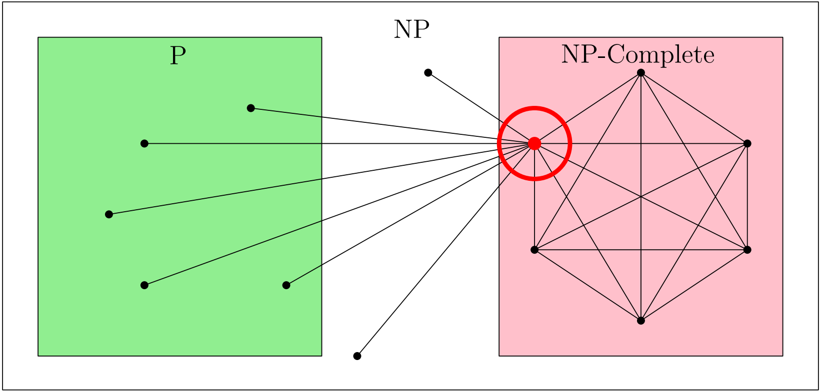
Problem

- The most famous unsolved problem in CS since 1970's.
- Nobody knows if poly-time algorithms exist for any NP problem.



Source: `https://en.wikipedia.org/wiki/P_versus_NP_problem`

Unsolved problem: P = NP?



https://en.wikipedia.org/wiki/List_of_NP-complete_problems

If you solve one NP-Complete problem in poly-time, you would have solved all NP problems (several thousands of them) in poly-time.

Problem: Satisfiability (SAT)

Problem

- Given a Boolean formula (or logical expression) in conjunctive normal form (CNF), find either a satisfying truth assignment or report that none exists.
- Examples.
  (i) $(x \vee y \vee z) \wedge (x \vee \overline{y}) \wedge (y \vee \overline{z}) \wedge (z \vee \overline{x}) \wedge (\overline{y} \vee \overline{y} \vee \overline{z})$
  No solution exists.
  (ii) $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
  Solution is $(x_1, x_2, x_3, x_4) = (\mathsf{T}, \mathsf{T}, \mathsf{F}, \mathsf{F})$
- Applications.
  Circuit design, image analysis, software engineering, artificial intelligence, and automatic theorem proving
- SAT is NP-Complete. ▷ Cook-Levin theorem

Problem: Traveling salesperson (TSP)

Problem

- Optimization problem. Given n vertices $1, \ldots, n$ and all distances between them, find a tour/cycle that visits every vertex exactly once, of minimum total cost, or report that no such tour exists.
- Search problem. Given n vertices $1, \ldots, n$, and all distances between them, as well as a cost b, find a tour/cycle that visits every vertex exactly once, of total cost less than or equal to b, or report that no such tour exists.
- Decision problem. Given n vertices $1, \ldots, n$, and all distances between them, as well as a cost b, check if there exists a tour/cycle that visits every vertex exactly once, of total cost less than or equal to b, or report that no such tour exists.
- Optimization problem $\equiv$ search problem $\equiv$ decision problem
- TSP is NP-Complete.

3-SAT is NP-Complete

Problem

- 3-SAT is SAT in which every clause has at most three literals.
- 3-SAT is NP-Complete.

Solution

1. Prove that 3-SAT is in NP.
   3-SAT is a special case of SAT.
2. Prove that 3-SAT is NP-Hard.
   Show that SAT poly-time reduces to 3-SAT.
   `https://cse.iitkgp.ac.in/~palash/2018AlgoDesignAnalysis/SAT-3SAT.pdf`

Independent-Set is NP-Complete

Problem

- Given a graph and a natural number m, find m vertices such that no two of which have an edge between them.
- Independent-Set is NP-Complete.

Solution

1. Prove that Independent-Set is in NP.
   Polynomial-time verification algorithm?
2. Prove that Independent-Set is NP-Hard.
   Show that 3-SAT poly-time reduces to Independent-Set.