## Algorithmically unsolvable problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulate problem</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Halting problem</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Program correctness</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Program equivalence</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Integral roots of a polynomial</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Goodstein’s theorem</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Generalized ((3n + 1)) problem</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Post correspondence problem</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>
### Algorithmically solvable problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search in a sorted array</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>Search in an unsorted array</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Integer addition</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Generate primes</td>
<td>( O(n \log \log n) )</td>
</tr>
<tr>
<td>Sorting</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Fast Fourier transform</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Integer multiplication</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>Linear programming</td>
<td>( O(n^{3.5}) )</td>
</tr>
<tr>
<td>Primality test</td>
<td>( O(n^{10}) )</td>
</tr>
<tr>
<td>Satisfiability problem</td>
<td>( O(2^n) )</td>
</tr>
<tr>
<td>Traveling salesperson problem</td>
<td>( O((n - 1)!) )</td>
</tr>
<tr>
<td>Sudoku, chess, checkers, go</td>
<td>expo. class</td>
</tr>
</tbody>
</table>
Polynomial and exponential functions

- $n!$ (factorial)
- $2^n$ (exponential)
- $n^2$ (quadratic)
- $n \log(n)$
- $\sqrt{n}$
- $\log(n)$

Graph showing the growth of these functions for $n = 20, 40, 60, 80, 100$. The exponential function grows much faster than the polynomial 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 = \( \mathcal{O}(n^2) \)
- **Polynomial-time algorithms exist**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustive search</td>
<td>( \mathcal{O}(n^2) )</td>
<td>poly.</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>( \mathcal{O}(n \log^2 n) )</td>
<td>poly.</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>( \mathcal{O}(n \log n) )</td>
<td>poly.</td>
</tr>
</tbody>
</table>
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 $t = \sum_{i=2}^{n} nC_i$

- Polynomial-time algorithms exist

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustive search</td>
<td>$\sum_{i=2}^{n} (i \cdot nC_i)$</td>
<td>expo.</td>
</tr>
<tr>
<td>Bellman–Ford algorithm</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>Dijkstra’s algorithm</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>Dijkstra’s algorithm with binary heap</td>
<td>$O((</td>
<td>E</td>
</tr>
<tr>
<td>Thorup’s algorithm</td>
<td>$O(</td>
<td>E</td>
</tr>
</tbody>
</table>
Hard problems with easy solutions

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Search space</th>
<th>Best algorithm</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Greedy Algorithms</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum spanning tree</td>
<td>$\mathcal{O}(n^{n-2})$</td>
<td>$\mathcal{O}(</td>
<td>E</td>
</tr>
<tr>
<td>Shortest path</td>
<td>$\sum_{i=2}^{n}(i \cdot n \choose i)$</td>
<td>$\mathcal{O}(</td>
<td>E</td>
</tr>
<tr>
<td><strong>Iterative Improvement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Match $n$ boys with $n$ girls</td>
<td>$\mathcal{O}(n!)$</td>
<td>$\mathcal{O}(n^2)$</td>
<td>poly.</td>
</tr>
<tr>
<td>Linear programming</td>
<td>expo.</td>
<td>$\mathcal{O}(n^{3.5})$</td>
<td>poly.</td>
</tr>
</tbody>
</table>
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 = $\mathcal{O}((n - 1)!)$
- It is unknown if polynomial-time algorithms exist

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustive search</td>
<td>$\mathcal{O}((n - 1)!)$</td>
<td>expo.</td>
</tr>
<tr>
<td>Held-Karp algorithm</td>
<td>$\mathcal{O}(n^22^n)$</td>
<td>expo.</td>
</tr>
</tbody>
</table>
Hard problems with no known easy solutions

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Search space</th>
<th>Best algorithm</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability</td>
<td>$\mathcal{O}(2^n)$</td>
<td>$\mathcal{O}(2^n)$</td>
<td>expo.</td>
</tr>
<tr>
<td>Traveling salesman</td>
<td>$\mathcal{O}((n - 1)!)$</td>
<td>$\mathcal{O}(n^2 2^n)$</td>
<td>expo.</td>
</tr>
<tr>
<td>Sudoku</td>
<td>expo.</td>
<td>–</td>
<td>expo.</td>
</tr>
<tr>
<td>Chess</td>
<td>expo.</td>
<td>–</td>
<td>expo.</td>
</tr>
<tr>
<td>Checkers</td>
<td>expo.</td>
<td>–</td>
<td>expo.</td>
</tr>
<tr>
<td>Go</td>
<td>expo.</td>
<td>–</td>
<td>expo.</td>
</tr>
</tbody>
</table>
Problem = Search for a needle in a haystack

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Search for a needle in a haystack.</td>
</tr>
</tbody>
</table>
Inefficient algorithm

Solution

- Search through the entire haystack (search space).
Efficient algorithm

Solution

- Use a giant magnet.
Easy problems and possibly hard problems

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

- 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 \( P_{\text{old}} \) poly.-time reduces to problem \( P_{\text{new}} \), denoted by \( P_{\text{old}} \leq_p P_{\text{new}} \), if any instance of problem \( P_{\text{old}} \) can be solved using the following:
  1. poly. number of standard computational steps.
  2. poly. number of calls to function that solves problem \( P_{\text{new}} \).
- \( P_{\text{old}} \leq_p P_{\text{new}} \) means \( P_{\text{new}} \) is at least as hard as \( P_{\text{old}} \).

\[
\text{PROBLEM-OLD}(\text{input-old}) \quad \triangleright \quad P_{\text{old}} \leq_p P_{\text{new}}
\]

1. \( \text{input-new} \leftarrow f(\text{input-old}) \)
2. \( \text{output-new} \leftarrow \text{PROBLEM-NEW}(\text{input-new}) \)
3. \( \text{output-old} \leftarrow g(\text{output-new}) \)
4. return \( \text{output-old} \)
What is polynomial-time reduction?

Consequences

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

- If $P_{\text{new}}$ can be solved in polynomial time, then $P_{\text{old}}$ can be solved in polynomial time.
- If $P_{\text{old}}$ cannot be solved in polynomial time, then $P_{\text{new}}$ cannot be solved in polynomial time.
- If $P_{\text{new}} \leq_p P_{\text{old}}$, then
  - $P_{\text{old}}$ can be solved in polynomial time
  - iff $P_{\text{new}}$ can be solved in polynomial time.
### What are the complexity classes?

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P</strong> = Problems solvable in polynomial time.</td>
</tr>
<tr>
<td><strong>NP</strong> = Problems with solutions that can be verified/checked in polynomial time.</td>
</tr>
<tr>
<td><strong>PSPACE</strong> = Problems solvable with polynomial space.</td>
</tr>
<tr>
<td><strong>EXP</strong> = Problems solvable in exponential time.</td>
</tr>
<tr>
<td><strong>EXPSPACE</strong> = Problems solvable with exponential space.</td>
</tr>
</tbody>
</table>

---

**Diagram:**

- **RL**
- **CFL**
- **P**
- **NP**
- **PSPACE**
- **EXP**
- **EXPSPACE**
- **TDL**
What are the complexity classes?

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
</table>
| **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?

Less time

More time

NP
NP-Hard
NP-Complete
P
EXP
EXP-Hard
EXP-Complete

More time

Less time
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: $P = NP$?

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)

- 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) \((x \lor y \lor z) \land (x \lor \overline{y}) \land (y \lor \overline{z}) \land (z \lor \overline{x}) \land (\overline{y} \lor \overline{y} \lor \overline{z})\)
  
  No solution exists.

  (ii) \((\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)\)

  Solution is \((x_1, x_2, x_3, x_4) = (T, T, F, F)\)

- Applications.
  
  Circuit design, image analysis, software engineering, artificial intelligence, and automatic theorem proving

- SAT is NP-Complete. ▶ Cook-Levin theorem
Problem: Traveling salesperson (TSP)

- **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.
### 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](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.