6.6 Intractability

- introduction
- search problems
- P vs. NP
- classifying problems
- NP-completeness
- coping with intractability
6.6 Intractability

- introduction
- search problems
- P vs. NP
- classifying problems
- NP-completeness
- coping with intractability
Questions about computation

Q. What is a general-purpose computer?
Q. Are there limits on the power of digital computers?
Q. Are there limits on the power of machines we can build?

David Hilbert  Kurt Gödel  Alan Turing  Alonzo Church  John von Neumann
A simple model of computation: DFAs

Tape.
- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves one cell at a time.

Q. Is there a more powerful model of computation?
A. Yes.
A universal model of computation: Turing machines

Tape.
- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves one cell at a time.

Q. Is there a more powerful model of computation?
A. No! most important scientific result of 20th century?
Church-Turing thesis (1936)

Turing machines can compute any function that can be computed by a physically harnessable process of the natural world.

Remark. "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

Use simulation to demonstrate that models are equivalent.
- Android simulator on iPhone.
- iPhone simulator on Android.

Implications.
- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

Bottom line. Turing machine is a simple and universal model of computation.
Church-Turing thesis: evidence

- 8 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent.

<table>
<thead>
<tr>
<th>model of computation</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enhanced Turing machines</td>
<td>multiple heads, multiple tapes, 2D tape, nondeterminism</td>
</tr>
<tr>
<td>untyped lambda calculus</td>
<td>method to define and manipulate functions</td>
</tr>
<tr>
<td>recursive functions</td>
<td>functions dealing with computation on integers</td>
</tr>
<tr>
<td>unrestricted grammars</td>
<td>iterative string replacement rules used by linguists</td>
</tr>
<tr>
<td>extended L-systems</td>
<td>parallel string replacement rules that model plant growth</td>
</tr>
<tr>
<td>programming languages</td>
<td>Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel</td>
</tr>
<tr>
<td>random access machines</td>
<td>registers plus main memory, e.g., TOY, Pentium</td>
</tr>
<tr>
<td>cellular automata</td>
<td>cells which change state based on local interactions</td>
</tr>
<tr>
<td>quantum computer</td>
<td>compute using superposition of quantum states</td>
</tr>
<tr>
<td>DNA computer</td>
<td>compute using biological operations on DNA</td>
</tr>
</tbody>
</table>
A question about algorithms

Q. Which algorithms are useful in practice?
   • Measure running time as a function of input size $N$.
   • Useful in practice ("efficient") = polynomial time for all inputs.

Ex 1. Sorting $N$ items takes $N \log N$ compares using mergesort. Useful.
Ex 2. Finding best TSP tour on $N$ points takes $N!$ steps using brute search. Not useful, since not bounded by a polynomial (brute search is $N^N$)

Theory. Definition is broad and robust.
Practice. Poly-time algorithms scale to huge problems.

constants $a$ and $b$ tend to be small, e.g., $3N^2$
Exponential growth
dwarfs technological change.

- Suppose you have a giant parallel computing device...
- With as many processors as electrons in the universe...
- And each processor has power of today's supercomputers...
- And each processor works for the life of the universe...

<table>
<thead>
<tr>
<th>quantity</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>electrons in universe †</td>
<td>$10^{79}$</td>
</tr>
<tr>
<td>supercomputer instructions per second †</td>
<td>$10^{13}$</td>
</tr>
<tr>
<td>age of universe in seconds †</td>
<td>$10^{17}$</td>
</tr>
</tbody>
</table>

† estimated

- Will not help solve 1,000 city TSP problem via brute force.
Questions about problems

Q. Which problems can we solve in practice?
A. Those with poly-time algorithms.

Q. Which problems have poly-time algorithms?
A. Not so easy to know. Focus of today's lecture.

many known poly–time algorithms for sorting
no known poly–time algorithm for TSP
Def. A problem is intractable if it can't be solved in polynomial time.

Goal. Prove that a problem is intractable.

Two problems that provably require exponential time.

- Given a constant-size program, does it halt in at most $K$ steps?
- Given $N$-by-$N$ checkers board position, can the first player force a win?

Frustrating news. Very few successes.
6.6 Intractability

- introduction
- search problems
- \( P \) vs. \( NP \)
- classifying problems
- \( NP \)-completeness
- coping with intractability
Four fundamental problems

**LSOLVE.** Given a system of linear equations, find a solution.

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36 \\
x_0 &= -1 \\
x_1 &= 2 \\
x_2 &= 2
\end{align*}
\]

Variables are real numbers

**LP.** Given a system of linear inequalities, find a solution.

\[
\begin{align*}
48x_0 + 16x_1 + 119x_2 &\leq 88 \\
5x_0 + 4x_1 + 35x_2 &\geq 13 \\
15x_0 + 4x_1 + 20x_2 &\geq 23 \\
x_0, x_1, x_2 &\geq 0 \\
x_0 &= 1 \\
x_1 &= 1 \\
x_2 &= \frac{1}{3}
\end{align*}
\]

Variables are real numbers

**ILP.** Given a system of linear inequalities, find a 0-1 solution.

\[
\begin{align*}
x_1 + x_2 &\geq 1 \\
x_0 + x_2 &\geq 1 \\
x_0 + x_1 + x_2 &\leq 2 \\
x_0 &= 0 \\
x_1 &= 1 \\
x_2 &= 1
\end{align*}
\]

Variables are 0 or 1

**SAT.** Given a system of boolean equations, find a binary solution.

\[
\begin{align*}
(x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\
(x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\
(x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true} \\
x_0 &= \text{false} \\
x_1 &= \text{false} \\
x_2 &= \text{true}
\end{align*}
\]

Variables are true or false
Four fundamental problems

**LSOLVE.** Given a system of linear equations, find a solution.
**LP.** Given a system of linear inequalities, find a solution.
**ILP.** Given a system of linear inequalities, find a 0-1 solution.
**SAT.** Given a system of boolean equations, find a binary solution.

Q. Which of these problems have poly-time algorithms?
- **LSOLVE.** Yes. Gaussian elimination solves $N$-by-$N$ system in $N^3$ time.
- **LP.** Yes. Ellipsoid algorithm is poly-time. but was open problem for decades
- **ILP, SAT.** No poly-time algorithm known or believed to exist!

but we still don't know for sure
Search problems


Requirement. Must be able to efficiently check that $S$ is a solution.

poly-time in size of instance $I$
Search problems

**Search problem.** Given an instance $I$ of a problem, find a solution $S$.

**Requirement.** Must be able to efficiently check that $S$ is a solution.

---

**LSOLVE.** Given a system of linear equations, find a solution.

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}
\]

instance $I$

\[
\begin{align*}
x_0 &= -1 \\
x_1 &= 2 \\
x_2 &= 2
\end{align*}
\]

solution $S$

To check solution $S$, plug in values and verify each equation.
Search problems


Requirement. Must be able to efficiently check that $S$ is a solution.

LP. Given a system of linear inequalities, find a solution.

\[
\begin{align*}
48x_0 &+ 16x_1 + 119x_2 \leq 88 \\
5x_0 &+ 4x_1 + 35x_2 \geq 13 \\
15x_0 &+ 4x_1 + 20x_2 \geq 23 \\
x_0, \ x_1, \ x_2 &\geq 0
\end{align*}
\]

To check solution $S$, plug in values and verify each inequality.

\[
\begin{align*}
x_0 &= 1 \\
x_1 &= 1 \\
x_2 &= \frac{1}{3}
\end{align*}
\]
Search problems

Requirement. Must be able to efficiently check that $S$ is a solution.

ILP. Given a system of linear inequalities, find a binary solution.

\[
\begin{align*}
x_0 + x_2 & \geq 1 \\
x_0 + x_2 & \geq 1 \\
x_0 + x_1 + x_2 & \leq 2
\end{align*}
\]

\[
\begin{align*}
x_0 & = 0 \\
x_1 & = 1 \\
x_2 & = 1
\end{align*}
\]

To check solution $S$, plug in values and verify each inequality.
Search problems


Requirement. Must be able to efficiently check that $S$ is a solution.

SAT. Given a system of boolean equations, find a boolean solution.

\[
\begin{align*}
(x'_1 \text{ or } x'_2) & \text{ and } (x_0 \text{ or } x_2) = true \\
(x_0 \text{ or } x_1) & \text{ and } (x_1 \text{ or } x'_2) = false \\
(x_0 \text{ or } x_2) & \text{ and } (x'_0) = true
\end{align*}
\]

instance $I$  \hspace{2cm}  solution $S$

\[
\begin{align*}
x_0 &= false \\
x_1 &= false \\
x_2 &= true
\end{align*}
\]

To check solution $S$, plug in values and verify each equation.
Search problems


Requirement. Must be able to efficiently check that $S$ is a solution.

FACTOR. Given an $n$-bit integer $x$, find a nontrivial factor.

\[ \text{input size} = \text{number of bits} \]

\[
\begin{array}{c}
147573952589676412927 \\
193707721 \\
\end{array}
\]

instance $I$ \hspace{1cm} solution $S$

To check solution $S$, long divide 193707721 into 147573952589676412927.
6.6 Intractability

- introduction
- search problems
- $P$ vs. $NP$
- classifying problems
- $NP$-completeness
- coping with intractability
NP

Def. NP is the class of all search problems.

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
<th>poly-time algorithm</th>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSOLVE ((A, b))</td>
<td>Find a vector (x) that satisfies (Ax = b)</td>
<td>Gaussian elimination</td>
<td>[\begin{align*} 0x_0 + x_1 + 1x_2 &amp;= 4 \ 2x_0 + 4x_1 - 2x_2 &amp;= 2 \ 0x_0 + 3x_1 + 15x_2 &amp;= 36 \end{align*}]</td>
<td>(x_0 = -1), (x_1 = 2), (x_2 = 2)</td>
</tr>
<tr>
<td>LP ((A, b))</td>
<td>Find a vector (x) that satisfies (Ax \leq b)</td>
<td>ellipsoid</td>
<td>[\begin{align*} 48x_0 + 16x_1 + 119x_2 &amp;\leq 88 \ 5x_0 + 4x_1 + 35x_2 &amp;\geq 13 \ 15x_0 + 4x_1 + 20x_2 &amp;\geq 23 \ x_0, x_1, x_2 &amp;\geq 0 \end{align*}]</td>
<td>(x_0 = 1), (x_1 = 1), (x_2 = \frac{7}{5})</td>
</tr>
<tr>
<td>ILP ((A, b))</td>
<td>Find a binary vector (x) that satisfies (Ax \leq b)</td>
<td>???</td>
<td>[\begin{align*} x_1 + x_2 &amp;\geq 1 \ x_0 + x_2 &amp;\geq 1 \ x_0 + x_1 + x_2 &amp;\leq 2 \end{align*}]</td>
<td>(x_0 = 0), (x_1 = 1), (x_2 = 1)</td>
</tr>
<tr>
<td>SAT ((\Phi, b))</td>
<td>Find a boolean vector (x) that satisfies (\Phi(x) = b)</td>
<td>???</td>
<td>((x_1' \text{ or } x_2') \text{ and } (x_0 \text{ or } x_2) = \text{true}) ((x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x_2') = \text{false}) ((x_0 \text{ or } x_2) \text{ and } (x_1') = \text{true})</td>
<td>(x_0 = \text{false}), (x_1 = \text{false}), (x_2 = \text{true})</td>
</tr>
<tr>
<td>FACTOR ((x))</td>
<td>Find a nontrivial factor of the integer (x)</td>
<td>???</td>
<td>(147573952589676412927)</td>
<td>(193707721)</td>
</tr>
</tbody>
</table>

Significance. What scientists and engineers aspire to compute feasibly.
**Def.** P is the class of search problems solvable in poly-time.

Note: classic definition limits P to yes-no problems

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
<th>poly-time algorithm</th>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
</table>
| LSOLVE       | Find a vector $x$ that satisfies $Ax = b$        | Gaussian elimination     | $\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}$ | $\begin{align*}
x_0 &= -1 \\
x_1 &= 2 \\
x_2 &= 2
\end{align*}$ |
| LP           | Find a vector $x$ that satisfies $Ax \leq b$     | ellipsoid                | $\begin{align*}
48x_0 + 16x_1 + 119x_2 &\leq 88 \\
5x_0 + 4x_1 + 35x_2 &\geq 13 \\
15x_0 + 4x_1 + 20x_2 &\geq 23 \\
x_0, x_1, x_2 &\geq 0
\end{align*}$ | $\begin{align*}
x_0 &= 1 \\
x_1 &= 1 \\
x_2 &= 1/5
\end{align*}$ |
| SORT         | Find a permutation that puts array $a$ in order  | mergesort                | $\begin{align*}
2.3 8.5 1.2 \\
9.1 2.2 0.3
\end{align*}$ | $\begin{align*} 5 &2 &4 &0 &1 &3 \end{align*}$ |
| STCONN       | Find a path in a graph $G$ from $s$ to $t$      | depth-first search       | $\begin{align*}$ | $\begin{align*}$ |

**Significance.** What scientists and engineers do compute feasibly.
**Nondeterminism**

Nondeterministic machine can **guess** the desired solution.

**Ex.** `int[] a = new int[N];`
- Java: initializes entries to 0.
- Nondeterministic machine: initializes entries to the solution!

**ILP.** Given a system of linear inequalities, **guess** a 0-1 solution.

\[
\begin{align*}
x_1 + x_2 &\geq 1 \\
x_0 + x_2 &\geq 1 \\
x_0 + x_1 + x_2 &\leq 2
\end{align*}
\]

**Ex. Turing machine.**
- Deterministic: state, input determines next state.
- Nondeterministic: more than one possible next state.

**NP.** Search problems solvable in poly time on a nondeterministic TM.
Extended Church-Turing thesis

\[ P = \text{search problems solvable in poly-time in the natural world.} \]

Evidence supporting thesis. True for all physical computers.

Natural computers? No successful attempts (yet).

Ex. Computing Steiner trees with soap bubbles

STEINER: Find set of lines of minimal length connecting N given points

Implication. To make future computers more efficient, suffices to focus on improving implementation of existing designs.
P vs. NP

Does \( P = NP \) ?

Copyright © 1990, Matt Groening

Copyright © 2000, Twentieth Century Fox
Automating creativity

Q. Being creative vs. appreciating creativity?

Ex. Mozart composes a piece of music; our neurons appreciate it.
Ex. Wiles proves a deep theorem; a colleague referees it.
Ex. Boeing designs an efficient airfoil; a simulator verifies it.
Ex. Einstein proposes a theory; an experimentalist validates it.

Computational analog. Does P = NP?
The central question

P. Class of search problems solvable in poly-time.
NP. Class of all search problems.

Does P = NP? [Can you always avoid brute-force searching and do better]

Two worlds.

If P = NP... Poly-time algorithms for SAT, ILP, TSP, FACTOR, ...
If P ≠ NP... Would learn something fundamental about our universe.

Overwhelming consensus. P ≠ NP.
The central question

**P.** Class of search problems solvable in poly-time.

**NP.** Class of all search problems.

**Does P = NP?** [Can you always avoid brute-force searching and do better]

**Millennium prize.** $1 million for resolution of P = NP problem.
6.6 Intractability

- introduction
- search problems
- P vs. NP
- classifying problems
- NP-completeness
- coping with intractability
A key problem: satisfiability

**SAT.** Given a system of boolean equations, find a solution.

\[
x'_1 \text{ or } x_2 \text{ or } x_3 = \text{true} \\
x_1 \text{ or } x'_2 \text{ or } x_3 = \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x'_3 = \text{true} \\
x'_1 \text{ or } x'_2 \text{ or } x_4 = \text{true}
\]

Key applications.

- Automatic verification systems for software.
- Electronic design automation (EDA) for hardware.
- Mean field diluted spin glass model in physics.
- ...
Exhaustive search

Q. How to solve an instance of SAT with \( n \) variables?
A. Exhaustive search: try all \( 2^n \) truth assignments.

Q. Can we do anything substantially more clever?
Conjecture. No poly-time algorithm for SAT.
Classifying problems

Q. Which search problems are in P?
A. No easy answers (we don't even know whether P = NP).

Problem $X$ poly-time reduces to problem $Y$ if $X$ can be solved with:
- Polynomial number of standard computational steps.
- Polynomial number of calls to $Y$.

Consequence. If SAT poly-time reduces to $Y$, then we conclude that $Y$ is (probably) intractable.
SAT poly-time reduces to ILP

**SAT.** Given a system of boolean equations, find a solution.

\[
x' \lor x_2 \lor x_3 = \text{true}
\]
\[
x_1 \lor x'_2 \lor x_3 = \text{true}
\]
\[
x'_1 \lor x'_2 \lor x'_3 = \text{true}
\]
\[
x'_1 \lor x'_2 \lor x_4 = \text{true}
\]

**ILP.** Given a system of linear inequalities, find a 0-1 solution.

\[
1 \leq (1 - x_1) + x_2 + x_3
\]
\[
1 \leq x_1 + (1 - x_2) + x_3
\]
\[
1 \leq (1 - x_1) + (1 - x_2) + (1 - x_3)
\]
\[
1 \leq (1 - x_1) + (1 - x_2) + x_4
\]

solution to this ILP instance gives solution to original SAT instance

can to reduce any SAT problem to this form
More poly-time reductions from boolean satisfiability

Conjecture. SAT is intractable.
Implication. All of these problems are intractable.
Still more reductions from SAT

Aerospace engineering. Optimal mesh partitioning for finite elements.
Biology. Phylogeny reconstruction.
Chemical engineering. Heat exchanger network synthesis.
Chemistry. Protein folding.
Civil engineering. Equilibrium of urban traffic flow.
Economics. Computation of arbitrage in financial markets with friction.
Electrical engineering. VLSI layout.
Environmental engineering. Optimal placement of contaminant sensors.
Game theory. Nash equilibrium that maximizes social welfare.
Mathematics. Given integer $a_1,\ldots,a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta) \, d\theta$
Mechanical engineering. Structure of turbulence in sheared flows.
Medicine. Reconstructing 3d shape from biplane angiocardiogram.
Operations research. Traveling salesperson problem.
Physics. Partition function of 3d Ising model.
Politics. Shapley-Shubik voting power.
Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris.
Statistics. Optimal experimental design.

plus over 6,000 scientific papers per year
6.6 Intractability

- introduction
- search problems
- P vs. NP
- classifying problems
- NP-completeness
- coping with intractability
NP-completeness

Def. An NP problem is **NP-complete** if every problem in NP poly-time reduce to it.


Extremely brief proof sketch:
- Convert non-deterministic TM notation to SAT notation.
- If you can solve SAT, you can solve any problem in NP.

Corollary. Poly-time algorithm for SAT iff \( P = NP \).
You NP-complete me
Implications of Cook-Levin theorem

All of these problems (and many, many more) poly-time reduce to SAT.

Stephen Cook
'82 Turing award

Leonid Levin

3-COLOR reduces to SAT
Implications of Karp + Cook-Levin

All of these problems are NP-complete; they are manifestations of the same really hard problem.
Implications of NP-Completeness

Implication. [SAT captures difficulty of whole class NP]

- Poly-time algorithm for SAT iff $P = NP$.
- No poly-time algorithm for some NP problem $\implies$ none for SAT.

Remark. Can replace SAT with any of Karp's problems.

Proving a problem NP-complete guides scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: 3D-ISING proved NP-complete. a holy grail of statistical mechanics

search for closed formula appears doomed
Two worlds (more detail)

Overwhelming consensus (still). \( P \neq NP \).

Why we believe \( P \neq NP \).

“We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device.” — Avi Wigderson
Summary

**P.** Class of search problems solvable in poly-time.

**NP.** Class of all search problems, some of which seem wickedly hard.

**NP-complete.** Hardest problems in NP.

**Intractable.** Problem with no poly-time algorithm.

Many fundamental problems are NP-complete.

- SAT, ILP, HAMILTON-PATH, ...
- 3D-ISING, ...

Use theory a guide:

- A poly-time algorithm for an NP-complete problem would be a stunning breakthrough (a proof that P = NP).
- You will confront NP-complete problems in your career.
- Safe to assume that P ≠ NP and that such problems are intractable.
- Identify these situations and proceed accordingly.
6.6 Intractability

- introduction
- search problems
- P vs. NP
- classifying problems
- NP-completeness
- coping with intractability
Exploiting intractability

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.

- To use: multiply two $n$-bit integers. [poly-time]
- To break: factor a $2n$-bit integer. [unlikely poly-time]

Multiply = EASY

$$23 \times 67 \quad 1,541$$

Factor = HARD
Exploiting intractability

**Challenge.** Factor this number.

740375634795617128280467960974295731425931888892312890849362
326389727650340282662768919964196251178439958943305021275853
701189680982867331732731089309005525051168770632990723963807
86710086096962537934650563796359

RSA–704
($30,000 prize if you can factor)

**Can't do it?** Create a company based on the difficulty of factoring.

*RSA algorithm*

*RSA sold for $2.1 billion*

*or design a t-shirt*
Exploiting intractability

**FACTOR.** Given an \( n \)-bit integer \( x \), find a nontrivial factor.

**Q.** What is complexity of FACTOR?
**A.** In NP, but not known (or believed) to be in P or NP-complete.

**Q.** What if \( P = NP \)?
**A.** Poly-time algorithm for factoring; modern economy collapses.

**Proposition.** [Shor 1994] Can factor an \( n \)-bit integer in \( n^3 \) steps on a "quantum computer."

**Q.** Do we still believe the extended Church-Turing thesis???
COURSE WRAP-UP

- Course goals
- What you can do next
# Farewell to Algorithms

What you learned

<table>
<thead>
<tr>
<th>topic</th>
<th>data structures and algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data types</strong></td>
<td>stack, queue, bag, union-find, priority queue</td>
</tr>
<tr>
<td><strong>sorting</strong></td>
<td>quicksort, mergesort, heapsort, radix sorts</td>
</tr>
<tr>
<td><strong>searching</strong></td>
<td>BST, red-black BST, hash table</td>
</tr>
<tr>
<td><strong>graphs</strong></td>
<td>BFS, DFS, Prim, Kruskal, Dijkstra</td>
</tr>
<tr>
<td><strong>strings</strong></td>
<td>KMP, regular expressions, tries, data compression</td>
</tr>
</tbody>
</table>
What you can do next

CIS 320 - Introduction to Algorithms
CIS 330 - Design Principles of Information Systems
CIS 331 - Intro to Networks and Security
CIS 341 - Compilers and Interpreters
CIS 350 - Software Design/Engineering
CIS 390 - Robotics
CIS 391 - Introduction to Artificial Intelligence
CIS 450 - Database and Information Systems
CIS 460 - Computer Graphics
CIS 500 - Software Foundations
CIS 519 - Introduction to Machine Learning
Algorithms pervade the modern world
Algorithms are integral to disciplines beyond computer science

You could help to unlock the secrets of life and of the universe.

“Computer models mirroring real life have become crucial for most advances made in chemistry today.... Today the computer is just as important a tool for chemists as the test tube.”

— Royal Swedish Academy of Sciences
(Nobel Prize in Chemistry 2013)

Martin Karplus, Michael Levitt, and Arieh Warshel
“I will, in fact, claim that the difference between a bad programmer and a good one is whether the programmer considers code or data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”

— Linus Torvalds (creator of Linux)
You are now better prepared for your job interviews
Farewell to Algorithms

My hope for what you got out of 121:

- You are now a more proficient programmer
- You have a toolkit of algorithms that you can use in your programs
- You have an understanding of why selecting one data structure or algorithm over another is advantageous for large data sets
- You have confidence going into computer science job interviews
Thank you!

I really enjoyed teaching you!

Please stay in touch!

- Email me if you use 121 knowledge to get a job.  ccb@upenn.edu
- Take my classes.  NETS 213 "Crowdsourcing and Human Computation", CIS 525 "Machine Translation"
- Learn about my research.  Stop by my office (Levine 506).  I’m always eager to talk about my research and explore potential undergraduate research opportunities.

Thank you for making this such a good experience for me.