

# Complexity Theory (The cartoon version)

---

Nathan Sprague  
JMU Department of Computer Science

# The Questions

---

- Are there computational problems that cannot be solved?
- Are there computational problems that *can* be solved, but don't have efficient solutions?

# Are There Non-Computable Problems?

- Yes.
- The classic example is the Halting Problem
  - Given a program, determine whether it will halt or execute forever.
- Many other questions we might ask about the behavior of programs are also non-computable
  - See [Rice's Theorem](#)

# Problem Classes

- **Decision Problem** – a problem with a yes/no answer.
- The class **P**:
  - A decision problem is in **P** if it has a polynomial time algorithm.
- The Class **NP**:
  - A decision problem is in **NP** if it has a polynomial time verification algorithm
- Examples...
- Every problem in **P** is also in **NP**

# “Complete” Problems

- Informally: If a problem is **complete** for some class, it is as hard as any problem in that class.
  - If a problem is NP-Complete it is as hard as any problem in NP.
- More formally:
  - A problem  $X$  is complete if an instance of any problem from the class can be (efficiently) transformed into an instance of problem  $X$
  - (Such transformations are called reductions)
  - This implies that a solution for  $X$  will provide a solution for any problem in the class

# Are There NP-Complete Problems?

- Yes
- **Cook-Levin Theorem** tells us that the Boolean Satisfiability problem is NP-Complete.
- It is also the case that *many* problems in NP seem *really* hard: No polynomial time algorithms have been found despite decades of work.
- What can we conclude about the Boolean Satisfiability Problem?

# Are There Other NP-Complete Problems?

- Yes. Lots.
- If we can show a reduction from some new problem to *any* NP-Complete problem, then we know that the new problem is NP-Complete.
- [List of NP-Complete problems from Wikipedia](#)