	UNIVERSALITY	COMPUTABILITY	INTRACTABILITY
QUESTION	**Is there** a universal computer?	What **can't** computers do?	What can computers do **efficiently**?
BIG IDEAS			What are **P** problems? Give examples. What are **NP** problems? Give examples.
TERMINOLOGY & EXAMPLES	**Church-Turing Thesis**: **Extended Church-Turing Thesis**:	**Halting problem**: **Post's Correspondence Problem**:	What makes an NP problem **NP-Complete**?
PEOPLE	**Ada Lovelace**, 1840's: **Alan Turing**, 1940's:	**David Hilbert**, 1928: **Alonzo Church**, 1930's:	**Stephen Cook**, 1971: **Richard Karp**, 1972:

	UNIVERSALITY	COMPUTABILITY	INTRACTABILITY
QUESTION	**Is there** a universal computer?	What **can't** computers do?	What can computers do **efficiently**?
BIG IDEAS	There exist universal (i.e., general purpose) computing machines. No general purpose machine is more powerful than another. A universal Turing machine can simulate any Turing machine.	Some problems will never be computable, no matter how advanced computers get. You can't tell if a program has an infinite loop. You can't tell if two programs give the same output in all cases. You can't tell if some parts of the code will ever be executed, in a complicated program.	**"P"** problems are the ones that can be **solved** in all cases in polynomial time. Examples: search/sort, int division. **"NP"** problems are the ones where, if you have a potential solution, you can **check** if it's valid in polynomial time**.** All **P** problems are in **NP**, but not necessarily vice versa.
TERMINOLOGY & EXAMPLES	**Church-Turing Thesis** -- Turing machines are as capable as any other computation machine. (This is a thesis because it cannot be proved.) **Extended Church-Turing Thesis** -- Turing machines are as capable as any other machine, regardless of hardware. (Quantum computing may someday disprove this. Or, not.)	**Halting problem** -- Impossible to decide if a program has an infinite loop. Proven by Turing, who got his PhD here at Princeton. **Post's Correspondence Problem** -- A puzzle game based on a set of cards with strings on the top and bottom. Impossible to decide if you can line up two infinite supplies of cards to make the same string on top and bottom.	Some NP problems are particularly hard, like SAT. If you can reduce SAT to that problem, it's an **"NP-Complete"** problem. TSP is known to be NP-Complete. **PRIME** is a problem in NP, which was recently discovered to be in P. **FACTOR** is in NP and the assumption that FACTOR isn't in P is used *heavily* in online security.
PEOPLE	**Ada Lovelace**, 1840's, first to publish the idea of a general purpose computer & world's first programmer, using punch cards. **Alan Turing**, 1940's, invented TM's. Cracked Axis powers' Enigma Machine code, helped Allies win war.	**David Hilbert**, 1928, posed the Entscheidungsproblem ("Decision problem"): can you decide if an arbitrary statement is true/false given logic rules? **Alonzo Church**, 1930's, formulated Lambda Calculus to address Hilbert's decision problem. Church and Turing proved the decision problem undecidable.	**Stephen Cook**, 1971, proved that if you solve SAT in polynomial time you could solve any problem in NP in polynomial time. This is the basis of NP-Completeness. **Richard Karp**, 1972, found 21 other NP problems which SAT could be reduced to, expanding the NP-Completeness family.