Lecture 22:
Finish Randomized Complexity,
Summary of 6.045
Randomized / Probabilistic Complexity
Probabilistic TMs

A probabilistic TM $M$ is a nondeterministic TM where:

- Each nondeterministic step is called a coin flip.
- Each nondeterministic step has only two legal next moves (heads or tails).

The probability that $M$ runs on a branch $b$ is: \[ \Pr [ b ] = 2^{-k} \]

where $k$ is the number of coin flips that occur on branch $b$. 
**Definition.** A probabilistic TM $M$ decides a language $A$ with error $\varepsilon$ if for all strings $w$,

- If $w \in A$ then $\Pr[ M \text{ accepts } w ] \geq 1 - \varepsilon$
- If $w \notin A$ then $\Pr[ M \text{ doesn’t accept } w ] \geq 1 - \varepsilon$

**Theorem:** A language $A$ is in $NP$ if there is a nondeterministic polynomial time TM $M$ such that for all strings $w$:

- If $w \in A$ then $\Pr[ M \text{ accepts } w ] > 0$
- If $w \notin A$ then $\Pr[ M \text{ accepts } w ] = 0$
BPP = Bounded Probabilistic P

BPP = \{ L \mid L \text{ is recognized by a probabilistic polynomial-time TM with error at most } \frac{1}{3} \}

Why 1/3?

It doesn’t matter what error value we pick, as long as the error is smaller than 1/2.

When the error is smaller than 1/2, we can make it very small by repeatedly running the TM.
An arithmetic formula is like a Boolean formula, except it has $+, -, \text{ and } \ast$ instead of OR, NOT, AND.

$$\text{ZERO-POLY} = \{ p \mid p \text{ is an arithmetic formula over } \mathbb{Z} \text{ that is identically zero} \}$$

Identically zero means: all coefficients are 0

Two examples of formulas in ZERO-POLY:

$$(x + y) \cdot (x + y) - x \cdot x - y \cdot y - 2 \cdot x \cdot y$$
Abbreviate as: $(x + y)^2 - x^2 - y^2 - 2xy$

$$(x^2 + a^2) \cdot (y^2 + b^2) - (x \cdot y - a \cdot b)^2 - (x \cdot b + a \cdot y)^2$$

There is a rich history of polynomial identities in mathematics. Useful also in program testing!
Testing Univariate Polynomials

Let $p(x)$ be a polynomial in one variable over $\mathbb{Z}$

$$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_dx^d$$

Suppose $p$ is hidden in a “black box” – we can only see its inputs and outputs.

Want to determine if $p$ is identically 0

Simply evaluate $p$ on $d+1$ distinct values!

Non-zero degree $d$ polynomials have $\leq d$ roots.
But the zero polynomial has every value as a root.
Testing Multivariate Polynomials

Let $p(x_1,\ldots,x_n)$ be a polynomial in $n$ variables over $\mathbb{Z}$.

Suppose $p(x_1,\ldots,x_n)$ is given to us, but as a very complicated arithmetic formula.

Can we efficiently determine if $p$ is identically 0?

If $p(x_1,\ldots,x_n)$ is a product of $m$ polynomials, each of which is a polynomial in $t$ terms, $\prod_m(\Sigma_t \text{stuff})$

Then expanding the expression into a $\Sigma$ of $\prod$ could take $t^m$ time!

Big Idea: Evaluate $p$ on random values
Theorem (Schwartz-Zippel-DeMillo-Lipton)
Let $p(x_1,x_2,...,x_n)$ be a *nonzero* polynomial, where each $x_i$ has degree at most $d$. Let $F \subseteq \mathbb{Z}$ be finite.

If $a_1,...,a_m$ are selected randomly from $F$, then:

$$\Pr \left[ p(a_1, ..., a_m) = 0 \right] \leq \frac{dn}{|F|}$$

Low-deg. nonzero polynomials are nonzero on MANY inputs

Proof (by induction on $n$):

Base Case ($n = 1$):

$$\Pr \left[ p(a_1) = 0 \right] \leq \frac{d}{|F|}$$

Nonzero polynomials of degree $d$ have most $d$ roots, so at most $d$ elements in $F$ can make $p$ zero
Inductive Step \((n > 1)\): Assume true for \(n-1\) and prove for \(n\)

Let \(p(x_1,\ldots,x_n)\) be not identically zero.

Write: 
\[
p(x_1,\ldots,x_n) = p_0 + x_n p_1 + x_n^2 p_2 + \ldots + x_n^d p_d
\]
where \(x_n\) does not occur in any \(p_i(x_1,\ldots,x_{n-1})\)

Observe: At least one \(p_i\) is not identically zero

Suppose \(p(a_1,\ldots,a_n) = 0\). Let \(q(x_n) = p(a_1,\ldots,a_{n-1},x_n)\). Two cases:

1. \(q \equiv 0\). That is, for all \(j\), \(p_j(a_1,\ldots,a_{n-1}) = 0\) (including \(p_i\))

\[
\Pr[(1)] \leq \Pr[p_i(a_1,\ldots,a_{n-1}) = 0] \leq (n-1)d/|F| \quad \text{by induction}
\]

2. \(q\) is not identically zero, but \(q(a_n) = 0\).

Note \(q\) is a univariate degree-\(d\) polynomial!

\[
\Pr[(2)] \leq \Pr[q(a_n) = 0] \leq d/|F| \quad \text{by univariate case}
\]

\[
\Pr[(1) \text{ or } (2)] \leq \Pr[(1)] + \Pr[(2)] \leq nd/|F|
\]
ZERO-POLY = \{ p \mid p \text{ is an arithmetic formula over } \mathbb{Z} \text{ that is \textit{identically} zero}\}

Theorem: ZERO-POLY ∈ BPP

Proof: Suppose \( n = |p| \). Then \( p \) has \( k \leq n \) variables, and the \textit{degree} of each variable is at most \( n \).

Algorithm A: Given polynomial \( p \),
For all \( i = 1, \ldots, k \), choose \( r_i \) randomly from \( \{1, \ldots, 3n^2\} \)
If \( p(r_1, \ldots, r_k) = 0 \) then output \textit{zero}
else output \textit{nonzero}

Observe \( A \) runs in polynomial time.
If \( p \equiv 0 \), then \( \Pr[A(p) \text{ outputs } \textit{zero}] = 1 \)
If \( p \not\equiv 0 \), then by the Schwartz-Zippel lemma,
\( \Pr[A(p) \text{ outputs } \textit{zero}] = \Pr_r[p(r) = 0] \leq n^2 / 3n^2 \leq 1/3 \)
Checking Equivalence of Arithmetic Formulas

ZERO-POLY = \{ p \mid p \text{ is an arithmetic formula that is identically zero}\}

Theorem: \text{ZERO-POLY} \in \text{BPP}

EQUIV-POLY = \{ (p,q) \mid p \text{ and } q \text{ are arithmetic formulas computing the same polynomial}\}

Corollary: \text{EQUIV-POLY} \in \text{BPP}

Proof: (p,q) \text{ in EQUIV-POLY } \iff p-q \text{ in ZERO-POLY}
Therefore \text{EQUIV-POLY} \leq_p \text{ZERO-POLY}
and we get a BPP algorithm for EQUIV-POLY.

See Sipser 10.2 for an application to testing equivalence of simple programs!
Equivalence of Arithmetic Formulas

EQUIV-POLY = \{ (p,q) | p and q are arithmetic formulas computing the same polynomial \}

Corollary: EQUIV-POLY ∈ BPP

There is a big contrast with Boolean formulas!

EQUIV = \{ (\phi,\psi) | \phi and \psi are Boolean formulas computing the same function \}

We showed EQUIV is in coNP. It’s also coNP-complete!

TAUTOLOGY \leq_P EQUIV: map \phi to (\phi, T)
ZERO-POLY = \{ p \mid p \text{ is an arithmetic formula that is identically zero} \}

**Theorem:** ZERO-POLY \( \in \) BPP

It is not known how to solve ZERO-POLY efficiently *without* randomness!

Thm [KI’04, AvM’11] If ZERO-POLY \( \in \) P then **NEW LOWER BOUNDS FOLLOW** (not P \( \neq \) NP, but still breakthroughs!)
BPP = \{ L \mid L \text{ is recognized by a probabilistic polynomial-time TM with error at most } \frac{1}{3} \}

Is BPP \subseteq NP?

THIS IS AN OPEN QUESTION!
Is BPP $\subseteq$ PSPACE?

Yes! Run through all possible sequences of coin flips one at a time, and count the number of branches that accept.

Known: BPP $\subseteq$ NP$^{NP}$ and BPP $\subseteq$ coNP$^{NP}$, but BPP $\subseteq$ P$^{NP}$ is still open!
Is \( NP \subseteq BPP \)?

THIS IS AN OPEN QUESTION!
Is BPP = EXPTIME?

THIS IS AN OPEN QUESTION!?

It’s widely conjectured that P = BPP!

Certain lower bounds $\Rightarrow$ P = BPP
Is BPP = EXPTIME?

THIS IS AN OPEN QUESTION!?

It’s widely conjectured that P = BPP!

Certain lower bounds ⇒ P = BPP
Definition: A language $A$ is in $\text{RP (Randomized P)}$ if there is a nondeterministic polynomial time TM $M$ such that for all strings $x$:

- $x \notin A \Rightarrow \Pr[M(x) \text{ accepts}] = 0$
- $x \in A \Rightarrow \Pr[M(x) \text{ accepts}] > 2/3$

$\text{NONZERO-POLY} = \{ p \mid p \text{ is an arithmetic formula that is not identically zero} \}$

Theorem: $\text{NONZERO-POLY} \in \text{RP}$ (Our proof of $\text{ZERO-POLY}$ in $\text{BPP}$ shows this)
Is $\text{RP} \subseteq \text{NP}$?

Yes!

Being RP means that not only are there “nifty proofs” but in fact most strings are nifty proofs!
Is $\text{RP} \subseteq \text{BPP}$?

Yes!

RP has “one-sided error”
BPP has “two-sided error”
Review
Deterministic Finite Automata

transition: \textit{for every state and alphabet symbol}

start state \((q_0)\)

accept states \((F)\)

states

\(q_0\)

\(q_1\)

\(q_2\)

\(q_3\)
Deterministic Computation

accept or reject

Non-Deterministic Computation

NFAs

Are these equally powerful???
YES for finite automata
**DEFINITION**

- DFAs
- NFAs
- Regular Languages
- Regular Expressions
- GNFAs
Regular Languages are closed under all of the following operations:

**Union:** \( A \cup B = \{ w \mid w \in A \text{ or } w \in B \} \)

**Intersection:** \( A \cap B = \{ w \mid w \in A \text{ and } w \in B \} \)

**Complement:** \( \overline{A} = \{ w \in \Sigma^* \mid w \not\in A \} \)

**Reverse:** \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \} \)

**Concatenation:** \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)

**Star:** \( A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \} \)
L is regular

if and only if

(∃ DFA M)(∀ strings x)[M acc. x ⇔ x ∈ L]

“M gives the correct output on all strings”

L is NOT regular

if and only if

(∀ DFA M)(∃ string x_M)[M acc. x_M ⇔ x ∉ L]

“M gives the wrong output on x_M”

So the problem of proving L is NOT regular can be viewed as a problem about designing “bad inputs”
How to Confuse DFAs

Want to show: Language L is not regular

Proof: By contradiction. Assume L is regular.
So L has a DFA M with Q states, for some Q > 0.

YOU: Cleverly pick strings x, y where |y| > Q

Run M on xy. Pigeons tell us: Some state q of M is visited more than once, while reading in y.
Therefore, M is in state q after reading xy', and is in q after reading xy'', for distinct prefixes y' and y'' of y

YOU: Cleverly pick string z so that exactly one of xy'z and xy''z is in L

But M will give the same output on both! Contradiction!
DFA Minimization:

There is an *efficient algorithm* which, given any DFA $M$, will output the unique minimum-state DFA $M^*$ equivalent to $M$. If this were true for more general models of computation, that would be an engineering breakthrough!! *(Would imply $P=NP$, for example)*

Table-Filling Algorithm to find “distinguishable” pairs of states
Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$

$x \equiv_L y$ if and only if for all $z \in \Sigma^*$, $[xz \in L \iff yz \in L]$.

The Myhill-Nerode Theorem:
A language $L$ is regular if and only if the number of equivalence classes of $\equiv_L$ is finite.

Regular = “easy”

Not Regular = “hard”
The Myhill-Nerode Theorem gives us a (universal) way to prove that a given language is not regular:

L is not regular

*if and only if*

there are infinitely many equiv. classes of \(\equiv_L\)

L is not regular

*if and only if*

There are infinitely many strings \(w_1, w_2, \ldots\) so that for all \(w_i \neq w_j, w_i \text{ and } w_j \text{ are distinguishable to } L:\)

there is a \(z \in \Sigma^*\) such that

*exactly one* of \(w_i z\) and \(w_j z\) is in \(L\)
Streaming Algorithms

Have three components:

Initialize:

<variables and their assignments>

When next symbol seen is $\sigma$:

<pseudocode using $\sigma$ and vars>

When stream stops (end of string):

<accept/reject condition on vars>

(or: <pseudocode for output>)

Algorithm $A$ computes $L \subseteq \Sigma^*$ if $A$ accepts the strings in $L$, rejects strings not in $L$. 
L = \{ x \mid x \text{ has odd number of 1's} \}
Has streaming algorithms using $O(1)$ space
(that is, it has a DFA) \hspace{1cm} \text{“very easy”}

L = \{ x \mid x \text{ has more 1's than 0's} \}
Has streaming algorithms using $O(\log n)$ space,
no streaming algorithm uses much less \hspace{1cm} \text{“easy”}

L = \{ x \mid x \text{ is a palindrome} \}
Has streaming algorithms using $O(n)$ space,
no streaming algorithm uses much less \hspace{1cm} \text{“hard”}
For any \( L \subseteq \Sigma^* \) define \( L_n = \{ x \in L \mid |x| \leq n \} \)

A streaming distinguisher for \( L_n \) is a subset \( D_n \) of \( \Sigma^* \): for all distinct \( x, y \in D_n \), there is a \( z \) in \( \Sigma^* \) such that \( |xz| \leq n, |yz| \leq n \), and exactly one of \( xz \), \( yz \) is in \( L \).

**Streaming Theorem:** Suppose for all \( n \), there is a streaming distinguisher \( D_n \) for \( L_n \) with \( |D_n| \geq 2^{S(n)} \). Then all streaming algs for \( L \) must use at least \( S(n) \) space!

**Idea:** Use the set \( D_n \) to show that every streaming algorithm for \( L \) must enter at least \( 2^{S(n)} \) different memory states, over all inputs of length at most \( n \).

But if there are at least \( 2^{S(n)} \) distinct memory states, then the alg must be using at least \( S(n) \) bits of space!
Communication Complexity

A theoretical model of distributed computing

- **Function** \( f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\} \)
  - Two inputs, \( x \in \{0,1\}^* \) and \( y \in \{0,1\}^* \)
  - We assume \(|x| = |y| = n\). Think of \( n \) as HUGE

- **Two computers:** Alice and Bob
  - Alice *only* knows \( x \), Bob *only* knows \( y \)

- **Goal:** Compute \( f(x, y) \) by communicating as few bits as possible between Alice and Bob

*We do not count computation cost.* We only care about the number of bits communicated.
Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$

Def. $f_L: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$
for $x, y$ with $|x| = |y|$ as:
$$f_L(x, y) = 1 \iff xy \in L$$

**Theorem:** If $L$ has a streaming algorithm using $\leq s$ space, then $cc(f_L)$ is at most $2s + 1$.

**Lower bounds on cc**

⇒ **Lower bounds on streaming**
(even with multiple passes)
Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$

Def. $f_L : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$

for $x, y$ with $|x| = |y|$ as:

$$f_L(x, y) = 1 \iff xy \in L$$

Examples:

$L = \{ x \mid x \text{ has an odd number of } 1s \}$

$\Rightarrow f_L(x, y) = \text{PARITY}(x, y)$ has $\Theta(1)$ comm. compl.

$L = \{ x \mid x \text{ has more } 1s \text{ than } 0s \}$

$\Rightarrow f_L(x, y) = \text{MAJORITY}(x, y)$ has $\Theta(\log n)$ comm. compl.

$L = \{ xx \mid x \in \{0,1\}^* \}$

$\Rightarrow f_L(x, y) = \text{EQUALS}(x, y)$ has $\Theta(n)$ comm. compl.
Theorem: \( L \) is decidable iff both \( L \) and \( \overline{L} \) are recognizable.
Theorem: \( L \) is recognizable \( \iff \) There is a TM \( V \) halting on all inputs such that
\[
L = \{ x \mid \exists y \in \Sigma^* \ [ V(x,y) \text{ accepts} ] \}
\]
Decidable Computation

Recognizable Computation

Are these equally powerful???
NO for Turing Machines
Decidable = Recognizable \cap \text{Co-recognizable}

Church-Turing Thesis
$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \} \}

**Thm. $A_{TM}$ is undecidable:** (proof by contradiction)

Assume $H$ is a machine that decides $A_{TM}$

$$H(\langle M, w \rangle) = \begin{cases} 
  \text{Accept} & \text{if } M \text{ accepts } w \\
  \text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}$$

Define a new TM $D$ with the following spec:

$D(\langle M \rangle)$: Run $H$ on $\langle M, M \rangle$ and output the *opposite* of $H$

$$D(\langle D \rangle) = \begin{cases} 
  \text{Reject} & \text{if } D \text{ accepts } \langle D \rangle \\
  \text{Accept} & \text{if } D \text{ does not accept } \langle D \rangle 
\end{cases}$$

Set $M = D$?
$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \}$

**Thm.** $A_{TM}$ is undecidable: (proof by contradiction)

Assume $H$ is a machine that decides $A_{TM}$

$$H(\langle M, w \rangle) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}$$

Define a new TM $D$ with the following spec:

$D(\langle M \rangle)$: Run $H$ on $\langle M, M \rangle$ and output the *opposite* of $H$

$$D(\langle D \rangle) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } \langle D \rangle \\
\text{Accept} & \text{if } D \text{ does not accept } \langle D \rangle 
\end{cases}$$

Set $M = D$?
Mapping Reductions

\( f : \Sigma^* \rightarrow \Sigma^* \) is a **computable function** if there is a Turing machine \( M \) that halts with just \( f(w) \) written on its tape, for every input \( w \).

A language \( A \) is **mapping reducible** to language \( B \), written as \( A \leq^m B \), if there is a computable \( f : \Sigma^* \rightarrow \Sigma^* \) such that for every \( w \in \Sigma^* \),

\[
    w \in A \iff f(w) \in B
\]

\( f \) is called a **mapping reduction** (or many-one reduction) from \( A \) to \( B \).
Recursion Thm: For every computable \( t \), there is a computable \( r \) such that \( r(w) = t(R,w) \) where \( R \) is a description of a TM computing \( r \).

Moral: Suppose we can design a TM \( T \) of the form

\[
\text{“On input \((x,w)\), do bla bla with } x, \\
\text{do bla bla bla bla with } w, \text{ etc. etc.”}
\]

We can always find a TM \( R \) with the behavior:

\[
\text{“On input } w, \text{ do bla bla bla with code of } R, \\
\text{do bla bla bla bla with } w, \text{ etc. etc.”}
\]

We can use the operation:

\[
\text{“Obtain your own description”}
\]

in Turing machine pseudocode!
Limitations on Mathematics

For every consistent and interesting $F$,

**Theorem 1.** (Gödel 1931) $F$ is *incomplete*: There are mathematical statements in $F$ that are *true* but cannot be proved in $F$.

**Theorem 2.** (Gödel 1931) The consistency of $F$ cannot be proved in $F$.

**Theorem 3.** (Church-Turing 1936) The problem of checking whether a given statement in $F$ has a proof is undecidable.
For every consistent and interesting $F$,

Theorem 1. (Gödel 1931) $F$ is incomplete: There are mathematical statements in $F$ that are true but cannot be proved in $F$.

Theorem 2. (Gödel 1931) The consistency of $F$ cannot be proved in $F$.

Theorem 3. (Church-Turing 1936) The problem of checking whether a given statement in $F$ has a proof is undecidable.
Definition:
\[
\text{TIME}(t(n)) = \{ L' \mid \text{there is a Turing machine } M \\
\text{with time complexity } O(t(n)) \text{ so that } L' = L(M) \}
\]
\[
= \{ L' \mid \text{L' is a language decided by a Turing machine with } \leq c t(n) + c \text{ running time} \}
\]

The Time Hierarchy Theorem

Intuition: The more computing time you have, the more problems you can solve.

Theorem: For all “reasonable” \( f, g : \mathbb{N} \to \mathbb{N} \) where for all \( n, g(n) > n^2 f(n)^2 \), \( \text{TIME}(f(n)) \subsetneq \text{TIME}(g(n)) \).
Deterministic Poly-Time Computation

Non-Deterministic Poly-Time Computation

“easy”
accept or reject

“probably not easy”
accept

Are these equally powerful???
P = NP ????
Theorem: \( L \in \text{NP} \iff \) There is a constant \( k \) and polynomial-time TM \( V \) such that

\[
L = \{ x \mid \exists y \in \Sigma^* \ [ |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts} ] \}
\]

Moral: A language \( L \) is in \( \text{NP} \) if and only if there are polynomial-length ("nifty") proofs for membership in \( L \).

Theorem: \( L \) is recognizable \( \iff \) There is a TM \( V \) that halts on all inputs such that

\[
L = \{ x \mid \exists y \in \Sigma^* \ [ V(x,y) \text{ accepts} ] \}
\]
Definition: A language $B$ is NP-complete if:

1. $B \in \text{NP}$

2. Every $A$ in NP is poly-time reducible to $B$
   That is, $A \leq_p B$
   When this is true, we say “$B$ is NP-hard”

NP-complete problems: “very likely hard”
3SAT, SAT, CLIQUE, IS, VC, SUBSET-SUM, KNAPSACK, PARTITION, BIN-PACKING, ...
Definition: $\text{coNP} = \{ L \mid \neg L \in \text{NP} \}$

What does a coNP problem $L$ look like?

The instances $\textbf{not}$ in $L$ have $\textit{nifty proofs}$. Any NP problem $L$ can be written in the form:

$L = \{ x \mid \exists y \text{ of poly}(|x|) \text{ length so that } V(x,y) \text{ accepts} \}$

$\neg L = \{ x \mid \neg \exists y \text{ of poly}(|x|) \text{ length so that } V(x,y) \text{ accepts} \}
= \{ x \mid \forall y \text{ of poly}(|x|) \text{ length, } V(x,y) \text{ rejects} \}$

Instead of using an “existentially guessing” (nondeterministic) machine, we can define a “universally verifying” machine!
Complexity Classes With Oracles

Let $B$ be a language.

$P^B = \{ L \mid L \text{ can be decided by some polynomial-time TM with an oracle for } B \}$

$P^{NP} = \text{the class of languages decidable by some polynomial-time oracle TM with an oracle for some } B \text{ in NP}$

$NP^{NP} = \text{the class of languages decidable by some nondeterministic polynomial-time oracle TM with an oracle for some } B \text{ in NP}$
NP-complete problems:

NHALT, SAT, 3SAT, CLIQUE, VC, SUBSET-SUM, ...

coNP-complete problems:

UNSAT, TAUTOLOGY, NOHAMPATH, ...

PSPACE-complete problems:

SPACE-HALT, TQBF, GG

There are also NP\textsuperscript{NP}-complete and coNP\textsuperscript{NP} problems

(but you don’t need to know them for the final!)
Time Hierarchy
Poly-time Reductions
NP Completeness
coNP Completeness
Oracles: $P^{NP}$, $NP^{NP}$, $coNP^{NP}$

EXPTIME
PSPACE = NPSPACE
FACTORING
coNP
TAUT
NP
SAT
NP
MIN-FORMULA
BPP
P
NP
coNP
NP
EXPTIME
GG
TQBF
FACTORIZATION
P
NP
NP
PSPACE = NPSPACE
EXPTIME
What’s next?

A few possibilities...

6.046 – Design and Analysis of Algorithms
6.841/18.405 – Advanced Complexity Theory
18.408 – Topics in Theoretical Computer Science
18.416 – Randomized Algorithms
6.875 – Cryptography and Cryptanalysis

Many more! There’s a big theory group at MIT!
Time to let the credits roll...
You have been watching:

6.045

Filmed at the

MASSACHVSETTS INSTITVTE OF TECHNOLOGY

in front of

absolutely nobody

Starring:

Ryan Williams

as “the professor”

A Large Hand Sanitizer Station

as itself