BILL, RECORD LECTURE!!!!

BILL RECORD LECTURE!!!

DTIME, P, EXP, and of Course NP

Exposition by William Gasarch—U of MD

Our Goals for Complexity Theory

We want to prove that

Our Goals for Complexity Theory

We want to prove that

1. Some languages L have **a fast program to decide them**

Our Goals for Complexity Theory

We want to prove that

1. Some languages L have **a fast program to decide them**
2. (Spoiler Alert: $L \in \mathrm{P}$.)

Our Goals for Complexity Theory

We want to prove that

1. Some languages L have **a fast program to decide them**
2. (Spoiler Alert: $L \in \mathrm{P}$.)
3. Some languages L **unlikely to have a fast program to decide them**

Our Goals for Complexity Theory

We want to prove that

1. Some languages L have **a fast program to decide them**
2. (Spoiler Alert: $L \in \mathrm{P}$.)
3. Some languages L **unlikely to have a fast program to decide them**
4. (Spoiler Alert: L is NP-complete.)

Our Goals for Complexity Theory

We want to prove that

1. Some languages L have **a fast program to decide them**
2. (Spoiler Alert: $L \in \mathrm{P}$.)
3. Some languages L **unlikely to have a fast program to decide them**
4. (Spoiler Alert: L is NP-complete.)

We first look at some problems of interest.

Variants of SAT

We define several variants of SAT:

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.
4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.

Variants of SAT

We define several variants of SAT:

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.
4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.
5. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals.

Sample Problems

How hard are the following problems:

Sample Problems

How hard are the following problems:

1. **HAM** Given a graph G does it have a Ham Cycle?
   (A cycle that has every vertex exactly once.)

Sample Problems

How hard are the following problems:

1. **HAM** Given a graph G does it have a Ham Cycle?
   (A cycle that has every vertex exactly once.)
2. **EUL** Given a graph G does it have a Euler Cycle?
   (A cycle that has every edge exactly once.)

Sample Problems

How hard are the following problems:

1. **HAM** Given a graph G does it have a Ham Cycle?
   (A cycle that has every vertex exactly once.)
2. **EUL** Given a graph G does it have a Euler Cycle?
   (A cycle that has every edge exactly once.)
3. **CLIQ** Given G and k, is there a set of k vertices that all know each other?

Sample Problems

How hard are the following problems:

1. **HAM** Given a graph G does it have a Ham Cycle? (A cycle that has every vertex exactly once.)
2. **EUL** Given a graph G does it have a Euler Cycle? (A cycle that has every edge exactly once.)
3. **CLIQ** Given G and k, is there a set of k vertices that all know each other?

To even ask these questions we need (1) a standard way to describe sets and a (2) model of computation.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding **We are busy people!**

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding **We are busy people!**Not getting into details of coding a fml into a string. HW?

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding **We are busy people!**Not getting into details of coding a fml into a string. HW?
4. A **set of formulas** is a set of strings, all of which are interpreted as formulas.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding **We are busy people!**Not getting into details of coding a fml into a string. HW?
4. A **set of formulas** is a set of strings, all of which are interpreted as formulas.
5. An ordered pair of **Graph,Number**: Use 00 for 0, 11 for 1, and 01 for a separator. The number you can code in usual binary.

Representing Elements of Sets

All elements (graphs, formulas, pairs of graphs and numbers) are represented by binary strings.

1. A **graph** is represented by an adjacency matrix. An n-node graph is an n^2-long string.
2. A **set of graphs** (like HAMC) is a set of strings, all of square length, all interpreted as a n adjacency matrix for a graph.
3. A **formula** is represented by coding **We are busy people!**Not getting into details of coding a fml into a string. HW?
4. A **set of formulas** is a set of strings, all of which are interpreted as formulas.
5. An ordered pair of **Graph,Number**: Use 00 for 0, 11 for 1, and 01 for a separator. The number you can code in usual binary.
6. A **set of ordered pairs: Graphs and Numbers** ....

Length of the Input

Def The **length of an input** is simply the length of the string that represents it.

Length of the Input

Def The **length of an input** is simply the length of the string that represents it.

We Sometimes Cheat We may take the length of a formula to be the number of vars. We may take the length of a graph to be the number of vertices. These notions of length are poly-related to the actual length and hence is fine for our purposes.

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

We are busy people!

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

We are busy people!

We are not going to bother defining Turing Machines Until we Need to!

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

We are busy people!

We are not going to bother defining Turing Machines Until we Need to!

Here is all you need to know:

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

We are busy people!

We are not going to bother defining Turing Machines Until we Need to!

Here is all you need to know:

1. Everything computable is computable by a Turing machine.

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

We are busy people!

We are not going to bother defining Turing Machines Until we Need to!

Here is all you need to know:

1. Everything computable is computable by a Turing machine.
2. Turing machines compute with discrete steps so one can talk about how many steps a computation takes.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.
3. The computation $M(x)$ will halt in $\leq T(|x|)$ steps.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.
3. The computation $M(x)$ will halt in $\leq T(|x|)$ steps.

What do you think of this definition? Discuss.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.
3. The computation $M(x)$ will halt in $\leq T(|x|)$ steps.

What do you think of this definition? Discuss.

Its Terrible!

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.
3. The computation $M(x)$ will halt in $\leq T(|x|)$ steps.

What do you think of this definition? Discuss.

Its Terrible!
The definition depends on the details of the Turing Machine definition. This should not be what we care about.

Time Classes

If we want to solve (say) HAMC we expect that the more vertices in the graph, the longer it will take. We want to measure **how fast** the time increases.

Def Let $T(n)$ be a computable function (think increasing). Let A be a set of strings. A is in $\mathrm{DTIME}(T(n))$ if there is a Turing Machine M such that

1. If $x \in A$ then $M(x)$ will halt and output 1.
2. If $x \notin A$ then $M(x)$ will halt and output 0.
3. The computation $M(x)$ will halt in $\leq T(|x|)$ steps.

What do you think of this definition? Discuss.

Its Terrible!
The definition depends on the details of the Turing Machine definition. This should not be what we care about.

So what to do?

How to use DTIME

So what do so with such a terrible definition?

How to use DTIME

So what do so with such a terrible definition?

- Prove theorems about $\mathrm{DTIME}(T(n))$ where the model does not matter. I might do this later in the course.

How to use DTIME

So what do so with such a terrible definition?

- Prove theorems about $\mathrm{DTIME}(T(n))$ where the model does not matter. I might do this later in the course.
- Define time classes where the model does not matter.

Models of Computation

There are many models of computation.

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).
2. Generalized Grammars (they look like CFG's with more stuff).

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).
2. Generalized Grammars (they look like CFG's with more stuff).
3. Variants of the two above.

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).
2. Generalized Grammars (they look like CFG's with more stuff).
3. Variants of the two above.
4. Others

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).
2. Generalized Grammars (they look like CFG's with more stuff).
3. Variants of the two above.
4. Others

Fact If A is in $\mathrm{DTIME}(T(n))$ on a Turing Machine then A can be computed by a generalized grammar in time $\mathrm{DTIME}((T(n))^5)$.
(I made that up, but something like it is true.)

Models of Computation

There are many models of computation.

1. Turing Machines (they look like DFA's with more stuff).
2. Generalized Grammars (they look like CFG's with more stuff).
3. Variants of the two above.
4. Others

Fact If A is in $\mathrm{DTIME}(T(n))$ on a Turing Machine then A can be computed by a generalized grammar in time $\mathrm{DTIME}((T(n))^5)$.
(I made that up, but something like it is true.)
Fact For any two commonly used models of comp, they are equivalent **within poly time.**

Polynomial Time and Other Classes

Def

Polynomial Time and Other Classes

Def

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.

Polynomial Time and Other Classes

Def

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.
2. $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$.

Polynomial Time and Other Classes

Def

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.
2. $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$.
3. PF is the set of a **functions** computable in poly time.

Polynomial Time and Other Classes

Def

1. $\mathrm{P} = \mathrm{DTIME}(n^{O(1)})$.
2. $\mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$.
3. PF is the set of a **functions** computable in poly time.

These definitions are model independent.

Why Polynomial Time? Reason I

Consider **3SAT**.

Why Polynomial Time? Reason I

Consider **3SAT**.

1. $\mathrm{3SAT} \in \mathrm{EXP}$, time 2^n, by brute force.

Why Polynomial Time? Reason I

Consider $\mathbf{3SAT}$.

1. $\mathrm{3SAT} \in \mathrm{EXP}$, time 2^n, by brute force.
2. If I came up with a $(1.618)^n$ algorithm that's **just brute force** with some tricks.

Why Polynomial Time? Reason I

Consider **3SAT**.

1. $3\mathrm{SAT} \in \mathrm{EXP}$, time 2^n, by brute force.
2. If I came up with a $(1.618)^n$ algorithm that's **just brute force** with some tricks. (There is such an algorithm.)

Why Polynomial Time? Reason I

Consider **3SAT**.

1. $3\mathrm{SAT} \in \mathrm{EXP}$, time 2^n, by brute force.
2. If I came up with a $(1.618)^n$ algorithm that's **just brute force** with some tricks. (There is such an algorithm.)
3. If I came up with an n^{1000} algorithm then it's **NOT brute force**. I would have found something **very clever**. Not practical, but that cleverness can probably be exploited to get a practical algorithm.

Why Polynomial Time? Reason II

A contrast to quadratic time.

Why Polynomial Time? Reason II

A contrast to quadratic time.

1. Quadratic Time. Different models of comp yield diff notions.

Why Polynomial Time? Reason II

A contrast to quadratic time.

1. Quadratic Time. Different models of comp yield diff notions.
2. P. Different models of comp yield same P.

Why Polynomial Time? Reason II

A contrast to quadratic time.

1. Quadratic Time. Different models of comp yield diff notions.
2. P. Different models of comp yield same P.
3. Quadratic time not closed under composition: if $f(n), g(n)$ are quadratic then $f(g(n))$ is quartic, not quadratic.

Why Polynomial Time? Reason II

A contrast to quadratic time.

1. Quadratic Time. Different models of comp yield diff notions.
2. P. Different models of comp yield same P.
3. Quadratic time not closed under composition: if $f(n), g(n)$ are quadratic then $f(g(n))$ is quartic, not quadratic.
4. P is closed under composition: if $f(n), g(n)$ are poly then $f(g(n))$ is poly.

$\mathbf{3SAT, HAM, EUL, CLIQ, 3COL}$ All Walk into a Bar

We rewrite 3SAT, HAM, EUL.

$\mathbf{3SAT}, \mathbf{HAM}, \mathbf{EUL}, \mathbf{CLIQ}, \mathbf{3COL}$ All Walk into a Bar

We rewrite $\mathrm{3SAT}$, HAM, EUL.

$$\mathrm{3SAT} = \{\phi : (\exists \vec{b})[\phi(\vec{b}) = T]\}$$

$\mathbf{3SAT, HAM, EUL, CLIQ, 3COL}$ All Walk into a Bar

We rewrite $\mathrm{3SAT}$, HAM, EUL.

$$\mathrm{3SAT} = \{\phi : (\exists \vec{b})[\phi(\vec{b}) = T]\}$$

$$\mathrm{HAM} = \{G : (\exists v_1, \ldots, v_n)[v_1, \ldots, v_n \text{ is a Ham Cycle}]\}.$$

3SAT, HAM, EUL, CLIQ, 3COL All Walk into a Bar

We rewrite 3SAT, HAM, EUL.

$$\mathrm{3SAT} = \{\phi : (\exists \vec{b})[\phi(\vec{b}) = T]\}$$

$$\mathrm{HAM} = \{G : (\exists v_1, \ldots, v_n)[v_1, \ldots, v_n \text{ is a Ham Cycle}]\}.$$

$$\mathrm{EUL} = \{G : (\exists v_1, \ldots, v_n)[v_1, \ldots, v_n \text{ is an Eul Cycle}]\}.$$

3SAT, HAM, EUL, CLIQ, 3COL (cont)

We rewrite CLIQ, 3COL.

$$\mathrm{CLIQ} = \{(G, k) : (\exists v_1, \ldots, v_k)[v_1, \ldots, v_k \text{ are a Clique}]\}.$$

3SAT, HAM, EUL, CLIQ, 3COL (cont)

We rewrite CLIQ, 3COL.

$$\mathrm{CLIQ} = \{(G, k) : (\exists v_1, \ldots, v_k)[v_1, \ldots, v_k \text{ are a Clique}]\}.$$

$$\mathrm{3COL} = \{G : \text{ there is a 3-coloring } \rho \text{ of } G \}.$$

(ρ assigns R,W,B to the vertices, no two adjacent verts have same color.)

3SAT, HAM, EUL, CLIQ, 3COL (cont)

We rewrite CLIQ, 3COL.

$$\mathrm{CLIQ} = \{(G, k) : (\exists v_1, \ldots, v_k)[v_1, \ldots, v_k \text{ are a Clique}]\}.$$

$$\mathrm{3COL} = \{G : \text{ there is a 3-coloring } \rho \text{ of } G\ \}.$$

(ρ assigns R,W,B to the vertices, no two adjacent verts have same color.)

Why is this interesting?

We Look At *CLIQ*

$$\mathrm{CLIQ} = \{(G,k) : (\exists v_1,\ldots,v_k)[v_1,\ldots,v_k \text{ are a Clique}]\}.$$

We Look At *CLIQ*

$$\mathrm{CLIQ} = \{(G,k) : (\exists v_1,\ldots,v_k)[v_1,\ldots,v_k \text{ are a Clique}]\}.$$

If $(G,k) \in \mathrm{CLIQ}$ then the $(v_1,\ldots,v_k)$ is a **witness** of this.
Note $(v_1,\ldots,v_k)$ is short: length is poly in the length of (G,k).

We Look At *CLIQ*

$$\mathrm{CLIQ} = \{(G, k) : (\exists v_1, \ldots, v_k)[v_1, \ldots, v_k \text{ are a Clique}]\}.$$

If $(G, k) \in \mathrm{CLIQ}$ then the $(v_1, \ldots, v_k)$ is a **witness** of this.
Note $(v_1, \ldots, v_k)$ is short: length is poly in the length of (G, k).

Note Verifying a witness is fast:
If $(v_1, \ldots, v_k)$ is a **potential witness** then **verifying** that $(v_1, \ldots, v_k)$ is a witness is **fast**: time poly in the length of (G, k).

We Look At *CLIQ*

$$\mathrm{CLIQ} = \{(G,k) : (\exists v_1,\ldots,v_k)[v_1,\ldots,v_k \text{ are a Clique}]\}.$$

If $(G,k) \in \mathrm{CLIQ}$ then the $(v_1,\ldots,v_k)$ is a **witness** of this.
Note $(v_1,\ldots,v_k)$ is short: length is poly in the length of (G,k).

Note Verifying a witness is fast:
If $(v_1,\ldots,v_k)$ is a **potential witness** then **verifying** that $(v_1,\ldots,v_k)$ is a witness is **fast**: time poly in the length of (G,k).

$\mathrm{3SAT}$, HAM, EUL are similar.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x : (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x : (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x : (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in A$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.

NP

Def A is in NP if there exists a set $B \in \mathrm{P}$ and a polynomial p such that

$$A = \{x : (\exists y)[|y| = p(|x|) \wedge (x, y) \in B]\}.$$

Intuition. Let $A \in \mathrm{NP}$.

- If $x \in A$ then there is a SHORT (poly in $|x|$) proof of this fact, namely y, such that x can be VERIFIED in poly time. So if I wanted to convince you that $x \in A$, I could give you y. You can verify $(x, y) \in B$ easily and be convinced.
- If $x \notin A$ then there is NO proof that $x \in A$.

Note $\mathrm{3SAT}$, HAM, EUL, CLIQ are all in NP.

Our Plan for NP

3SAT, HAM, EUL, CLIQ are all in NP.

Our Plan for NP

$\mathrm{3SAT}, \mathrm{HAM}, \mathrm{EUL}, \mathrm{CLIQ}$ are all in NP.

1. This does not mean that any of these problems are easy.

Our Plan for NP

$\mathrm{3SAT}$, HAM, EUL, CLIQ are all in NP.

1. This does not mean that any of these problems are easy.
2. This does not mean that any of these problems are hard.

Our Plan for NP

3SAT, HAM, EUL, CLIQ are all in NP.

1. This does not mean that any of these problems are easy.
2. This does not mean that any of these problems are hard.
3. 3SAT, HAM, CLIQ (but NOT EUL) are **equivalent** and hence one of the following holds:

Our Plan for NP

3SAT, HAM, EUL, CLIQ are all in NP.

1. This does not mean that any of these problems are easy.
2. This does not mean that any of these problems are hard.
3. 3SAT, HAM, CLIQ (but NOT EUL) are **equivalent** and hence one of the following holds:
   - 3SAT, HAM, CLIQ are all in P.

Our Plan for NP

3SAT, HAM, EUL, CLIQ are all in NP.

1. This does not mean that any of these problems are easy.
2. This does not mean that any of these problems are hard.
3. 3SAT, HAM, CLIQ (but NOT EUL) are **equivalent** and hence one of the following holds:
   - 3SAT, HAM, CLIQ are all in P.
   - None of 3SAT, HAM, CLIQ are in P.

Ind Set

We will do an example with another problem.
Def Let G be a graph. An **Ind Set** is a set of vertices, no pair of which has an edge between the two of them.

Ind Set

We will do an example with another problem.
Def Let G be a graph. An **Ind Set** is a set of vertices, no pair of which has an edge between the two of them.

$$\mathrm{IS} = \{(G, k) : G \text{ has an Ind Set of size } k \}.$$

If $\mathrm{IS} \in \mathrm{P}$ then $3\mathrm{SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

1. **Input** ϕ, a formula in 3-CNF form.

If $\mathrm{IS} \in \mathrm{P}$ then $3\mathrm{SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

1. **Input** ϕ, a formula in 3-CNF form.
2. **Output** (G, k) such that

$$\phi \in 3\mathrm{SAT} \text{ iff } (G, k) \in \mathrm{IS}.$$

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

1. **Input** ϕ, a formula in 3-CNF form.
2. **Output** (G, k) such that

$$\phi \in \mathrm{3SAT} \text{ iff } (G, k) \in \mathrm{IS}.$$

3. The algorithm runs in time $p(|\phi|)$ (p is a poly).

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

1. **Input** ϕ, a formula in 3-CNF form.
2. **Output** (G, k) such that

$$\phi \in \mathrm{3SAT} \text{ iff } (G, k) \in \mathrm{IS}.$$

3. The algorithm runs in time $p(|\phi|)$ (p is a poly).
4. Produces (G, k) where $|(G, k)| \leq q(|\phi|)$ (q is a poly).

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

We will give an algorithm that does the following:

1. **Input** ϕ, a formula in 3-CNF form.
2. **Output** (G, k) such that

$$\phi \in \mathrm{3SAT} \text{ iff } (G, k) \in \mathrm{IS}.$$

3. The algorithm runs in time $p(|\phi|)$ (p is a poly).
4. Produces (G, k) where $|(G, k)| \leq q(|\phi|)$ (q is a poly).

Call this algorithm **ALG**. On next slide we use **ALG** to show that $\mathrm{IS} \in \mathrm{P}$ implies $\mathrm{3SAT} \in \mathrm{P}$.

If $\mathrm{IS} \in \mathrm{P}$ then $3\mathrm{SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G,k)|)$.

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G,k)|)$.

1. **Input** ϕ, a formula in 3-CNF form of length L.

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G, k)|)$.

1. **Input** ϕ, a formula in 3-CNF form of length L.
2. Compute **ALG** on ϕ to get (G, k). Takes time $p(|\phi|)$ and produces (G, k) where $|(G, k)| \leq q(|\phi|)$.

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G,k)|)$.

1. **Input** ϕ, a formula in 3-CNF form of length L.
2. Compute **ALG** on ϕ to get (G,k). Takes time $p(|\phi|)$ and produces (G,k) where $|(G,k)| \le q(|\phi|)$.
3. Run M on (G,k) (takes time $r(q(|\phi|)))$. Recall that

$$\phi \in \mathrm{3SAT} \text{ iff } (G,k) \in \mathrm{IS}.$$

If $\mathrm{IS} \in \mathrm{P}$ then $\mathrm{3SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G,k)|)$.

1. **Input** ϕ, a formula in 3-CNF form of length L.
2. Compute **ALG** on ϕ to get (G,k). Takes time $p(|\phi|)$ and produces (G,k) where $|(G,k)| \le q(|\phi|)$.
3. Run M on (G,k) (takes time $r(q(|\phi|))$). Recall that

$$\phi \in \mathrm{3SAT} \text{ iff } (G,k) \in \mathrm{IS}.$$

So just output the output of $M(G,k)$.

If $\mathrm{IS} \in \mathrm{P}$ then $3\mathrm{SAT} \in \mathrm{P}$: Plan

Assume $\mathrm{IS} \in \mathrm{P}$ via program M which runs in $r(|(G, k)|)$.

1. **Input** ϕ, a formula in 3-CNF form of length L.
2. Compute **ALG** on ϕ to get (G, k). Takes time $p(|\phi|)$ and produces (G, k) where $|(G, k)| \le q(|\phi|)$.
3. Run M on (G, k) (takes time $r(q(|\phi|)))$. Recall that

$$\phi \in 3\mathrm{SAT} \text{ iff } (G, k) \in \mathrm{IS}.$$

   So just output the output of $M(G, k)$.

This is an algorithm for $3\mathrm{SAT}$ that takes time

$$p(|\phi|) + r(q(|\phi|))$$

How We Present **ALG**

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \quad \wedge \quad (\neg x \vee \neg y \vee z) \quad \wedge \quad (\neg x \vee y \vee \neg z)$$

How We Present **ALG**

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \qquad \wedge \qquad (\neg x \vee \neg y \vee z) \qquad \wedge \qquad (\neg x \vee y \vee \neg z)$$

From that one example and my verbiage you will be able to write down a general algorithm.

How We Present **ALG**

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \quad \wedge \quad (\neg x \vee \neg y \vee z) \quad \wedge \quad (\neg x \vee y \vee \neg z)$$

From that one example and my verbiage you will be able to write down a general algorithm.
HW?

How We Present **ALG**

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \qquad \wedge \qquad (\neg x \vee \neg y \vee z) \qquad \wedge \qquad (\neg x \vee y \vee \neg z)$$

From that one example and my verbiage you will be able to write down a general algorithm.
HW? No.

How We Present **ALG**

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \quad \wedge \quad (\neg x \vee \neg y \vee z) \quad \wedge \quad (\neg x \vee y \vee \neg z)$$

From that one example and my verbiage you will be able to write down a general algorithm.
HW? No.
Your Programming Project!

How We Present ALG

On the next slide we just show what **ALG** does on

$$(x \vee y \vee \neg z) \quad \wedge \quad (\neg x \vee \neg y \vee z) \quad \wedge \quad (\neg x \vee y \vee \neg z)$$

From that one example and my verbiage you will be able to write down a general algorithm.
HW? No.
Your Programming Project! Not this semester.

GO TO Other Slide Packet

GO TO next slide packet just for the IS reduction

GO TO Other Slide Packet

GO TO next slide packet just for the IS reduction

AND the 3-COL reduction.

GO TO Other Slide Packet

GO TO next slide packet just for the IS reduction

AND the 3-COL reduction.

We will come back here later.

Reductions

We now generalize what we did for $3\mathrm{SAT}$ and CLIQ.

Reductions

We now generalize what we did for $\mathrm{3SAT}$ and CLIQ.
Def Let X, Y be sets. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

Reductions

We now generalize what we did for $\mathrm{3SAT}$ and CLIQ.
Def Let X, Y be sets. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

(Example: Our function that took ϕ to (G, k).)

Reductions

We now generalize what we did for $3\mathrm{SAT}$ and CLIQ.

Def Let X, Y be sets. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

(Example: Our function that took ϕ to (G, k).)

We express this by writing $X \leq Y$.

Reductions

We now generalize what we did for $\mathrm{3SAT}$ and CLIQ.
Def Let X, Y be sets. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

(Example: Our function that took ϕ to (G, k).)
We express this by writing $X \leq Y$.

Reductions are transitive.
Lemma (HW) If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$. (We use that if $f(n), g(n)$ are poly then $f(g(n))$ is poly.)

Reductions

We now generalize what we did for $3\mathrm{SAT}$ and CLIQ.
Def Let X, Y be sets. A **reduction** from X to Y is a polynomial-time computable function f such that

$$x \in X \text{ iff } f(x) \in Y.$$

(Example: Our function that took ϕ to (G, k).)
We express this by writing $X \leq Y$.

Reductions are transitive.
Lemma (HW) If $X \leq Y$ and $Y \in \mathrm{P}$ then $X \in \mathrm{P}$. (We use that if $f(n), g(n)$ are poly then $f(g(n))$ is poly.)

Contrapositive If $X \leq Y$ and $X \notin \mathrm{P}$ then $Y \notin \mathrm{P}$.

Def of NP-Complete

Def A set Y is **NP-complete (NPC)** if the following hold:

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Def of NP-Complete

Def A set Y is **NP-complete (NPC)** if the following hold:

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \leq Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Def of NP-Complete

Def A set Y is **NP-complete (NPC)** if the following hold:

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \le Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Honesty When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.

Def of NP-Complete

Def A set Y is **NP-complete (NPC)** if the following hold:

- $Y \in \mathrm{NP}$
- If $X \in \mathrm{NP}$ then $X \le Y$.

Easy Lemma If Y is NP-complete and $Y \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Honesty When I first saw the definition of NP-completeness I thought (1) there are no NP-complete sets or (2) there are no natural NP-complete sets.

The condition:

$$\text{for EVERY } X \in \mathrm{NP},\ X \le Y$$

seemed very hard to meet.

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is $\mathbf{NP}$-complete

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is $\mathbf{NP}$-complete

Thoughts on this:

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual Turing Machines. SAT is the **first** NP-complete problem. You could not use some other problem. $3\mathrm{SAT}$ was the second by an easy reduction.

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual Turing Machines. SAT is the **first** NP-complete problem. You could not use some other problem. $3\mathrm{SAT}$ was the second by an easy reduction.
2. Once we have $3\mathrm{SAT}$ is NP-complete we will NEVER use Turing machines again. To show Y NPC: (1) $Y \in \mathrm{NP}$, (2) $A \le Y$ for a known A that is NPC, often 3SAT.

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual Turing Machines. SAT is the **first** NP-complete problem. You could not use some other problem. $3\mathrm{SAT}$ was the second by an easy reduction.
2. Once we have $3\mathrm{SAT}$ is NP-complete we will NEVER use Turing machines again. To show Y NPC: (1) $Y \in \mathrm{NP}$, (2) $A \le Y$ for a known A that is NPC, often 3SAT.
3. Thousands of problems are NP-complete. If any are in P then they are all in P.

SAT is NP-Complete

Cook (1971) and Levin (1973) independently showed:

CNF-SAT is NP-complete

Thoughts on this:

1. The proof is not hard, but it involves looking at actual Turing Machines. SAT is the **first** NP-complete problem. You could not use some other problem. $3\mathrm{SAT}$ was the second by an easy reduction.
2. Once we have $3\mathrm{SAT}$ is NP-complete we will NEVER use Turing machines again. To show Y NPC: (1) $Y \in \mathrm{NP}$, (2) $A \leq Y$ for a known A that is NPC, often 3SAT.
3. Thousands of problems are NP-complete. If any are in P then they are all in P.
4. Most Computer Scientists and Mathematicians think $\mathrm{P} \neq \mathrm{NP}$.

History: HAM and EUL

1736 Euler shows the Konigsberg bridge problem is unsolvable by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

History: HAM and EUL

1736 Euler shows the Konigsberg bridge problem is unsolvable by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

History: HAM and EUL

1736 Euler shows the Konigsberg bridge problem is unsolvable by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.

History: HAM and EUL

1736 Euler shows the Konigsberg bridge problem is unsolvable by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.
They didn't have the notion of algorithms to state what they wanted more rigorously.

History: HAM and EUL

1736 Euler shows the Konigsberg bridge problem is unsolvable by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Note Mathematicians wanted a **characterization of HAM graphs similar to the characterization of EUL graphs**.
They didn't have the notion of algorithms to state what they wanted more rigorously.

The theory of NP-completeness enabled mathematicians to **state** what they wanted rigorously ($\mathrm{HAM} \in \mathrm{P}$) and also gave the basis for proving likely it **cannot** be done (since HAM is NP-Complete).

$\mathbf{SAT}$, $\mathbf{HAM}$, $\mathbf{CLIQ}$, $\mathbf{3COL}$ Walk into a Bar

$\mathbf{SAT}, \mathbf{HAM}, \mathbf{CLIQ}, \mathbf{3COL}$ Walk into a Bar

1. SAT is NP-complete by Cook-Levin Theorem.

SAT, HAM, CLIQ, 3COL Walk into a Bar

1. SAT is NP-complete by Cook-Levin Theorem.
2. CLIQ is NP-complete. We proved this by showing $3SAT \leq CLIQ$.

SAT, HAM, CLIQ, 3COL Walk into a Bar

1. SAT is NP-complete by Cook-Levin Theorem.
2. CLIQ is NP-complete. We proved this by showing $3\mathrm{SAT} \le \mathrm{CLIQ}$.
3. 3COL is NP-complete. We may prove this later.

SAT, HAM, CLIQ, 3COL Walk into a Bar

1. SAT is NP-complete by Cook-Levin Theorem.
2. CLIQ is NP-complete. We proved this by showing $3\mathrm{SAT} \le \mathrm{CLIQ}$.
3. 3COL is NP-complete. We may prove this later.
4. HAM is NP-complete. Just take my word for it.

So What Do We Know?

So What Do We Know?

1. We **do not know** that $3\mathrm{SAT} \notin \mathrm{P}$.

So What Do We Know?

1. We **do not know** that $3SAT \notin P$.
2. We **do not know** that $CLIQ \notin P$.

So What Do We Know?

1. We **do not know** that $\mathrm{3SAT} \notin \mathrm{P}$.
2. We **do not know** that $\mathrm{CLIQ} \notin \mathrm{P}$.
3. We **do know** that $\mathrm{3SAT} \in \mathrm{P}$ IFF $\mathrm{CLIQ} \in \mathrm{P}$.

So What Do We Know?

1. We **do not know** that $3\mathrm{SAT} \notin \mathrm{P}$.
2. We **do not know** that $\mathrm{CLIQ} \notin \mathrm{P}$.
3. We **do know** that $3\mathrm{SAT} \in \mathrm{P}$ IFF $\mathrm{CLIQ} \in \mathrm{P}$.
4. We **believe** $3\mathrm{SAT} \notin \mathrm{P}$, hence we **believe** $\mathrm{CLIQ} \notin \mathrm{P}$.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. The NP-complete problems have been worked on for a long time (many predating the definition of P and NP) and none have been shown to be in P.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. The NP-complete problems have been worked on for a long time (many predating the definition of P and NP) and none have been shown to be in P.
2. Intuitively **coming up with a proof** seems harder than **verifying a proof**.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. The NP-complete problems have been worked on for a long time (many predating the definition of P and NP) and none have been shown to be in P.
2. Intuitively **coming up with a proof** seems harder than **verifying a proof**.
3. $\mathrm{P} \neq \mathrm{NP}$ has great explanatory power. See next slide.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1-\epsilon)\ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1 - \epsilon) \ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**.
3. These two proofs have nothing to do with each other yet give matching upper and lower bounds.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1-\epsilon) \ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**.
3. These two proofs have nothing to do with each other yet give matching upper and lower bounds.
4. There are many other approx problems where $\mathrm{P} \neq \mathrm{NP}$ explains why they cannot be improved.

My Opinions

My opinions

My Opinions

My opinions

1. 1.1 IF $\mathrm{P} = \mathrm{NP}$ that might be proven in the next decade.

My Opinions

My opinions

1. 1.1 IF $\mathrm{P} = \mathrm{NP}$ that might be proven in the next decade.
   1.2 IF $\mathrm{P} \neq \mathrm{NP}$ this will not be proven until the year 2525.

My Opinions

My opinions

1. 1.1 IF $\mathrm{P} = \mathrm{NP}$ that might be proven in the next decade.
   1.2 IF $\mathrm{P} \neq \mathrm{NP}$ this will not be proven until the year 2525.
2. $\mathrm{P} \neq \mathrm{NP}$. In fact, SAT requires $2^{\Omega(n)}$ time.

What Do Theorists Think of P vs NP?

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.
First I'll poll you, then I'll show you what the polls said.

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.

First I'll poll you, then I'll show you what the polls said.

Poll of 452 students: Do you think $P \neq NP$?

What Do Theorists Think of P vs NP?

I have done three polls of what theorists think of P vs NP and other issues.
First I'll poll you, then I'll show you what the polls said.
Poll of 452 students: Do you think P $\neq$ NP?

	P$\neq$NP	P$=$NP	Ind	DK	other
2002	61 (61%)	9 (9%)	4 (4%)	22 (22%)	7 (7%))
2012	126 (83%)	12 (9%)	5 (3%)	1 (0.66%)	8 (5.1%)
2019	109 (88%)	15 (12%)	0	0	0

BILL, STOP RECORDING LECTURE!!!!

BILL STOP RECORDING LECTURE!!!