1 What if $P = NP$?
- The Great Collapse
- The Power of Nondeterminism
- The Demise of Creativity
Outline

1. What if $P = \textbf{NP}$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard
Outline

1. What if $P = \textbf{NP}$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard

3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
Outline

1. What if $P = \text{NP}$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard

3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
Outline

1. **What if P = NP?**
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. **Upper Bounds are Easy and Lower Bounds, Hard**

3. **Diagonalization and Time Hierarchy**
   - Time Hierarchy Theorem
The Meaning of $P$ vs $NP$

The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to find solutions than it is to check solutions. Intuition leads one to believe that it is.

Big Consequences of $P = NP$

We will see that $P = NP$ leads to a great many complexity classes to also be equal to $P$. One such example is $P = coNP$, since one can easily switch the outputs "yes" and "no" of polynomial-time algorithms.
The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to

The Meaning of \( P \) vs \( NP \)

The biggest consequence of the relationship between \( P \) and \( NP \) is whether it is harder to find solutions.
The Meaning of P vs NP

The biggest consequence of the relationship between P and NP is whether it is harder to find solutions than it is to check solutions.
The Meaning of $P$ vs $NP$

The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to **find solutions** than it is to **check solutions**.

Intuition leads one to believe that it is.
Difficulty

The Meaning of \( P \) vs \( NP \)

The biggest consequence of the relationship between \( P \) and \( NP \) is whether it is harder to find solutions than it is to check solutions.

Intuition leads one to believe that it is.

Big Consequences of \( P = NP \)

We will see that \( P = NP \) leads to a great many complexity classes to also be equal to \( P \). One such example is \( P = \text{coNP} \), since one can easily switch the outputs "yes" and "no" of polynomial-time algorithms.

Billy Hardy

\( P \) vs \( NP \)
The Meaning of $P$ vs $NP$

The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to find solutions than it is to check solutions.

Intuition leads one to believe that it is.

Big Consequences of $P = NP$

We will see that $P = NP$ leads to a great many complexity classes to also be equal to $P$. 
The Meaning of $P \text{ vs } NP$

The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to find solutions than it is to check solutions.

Intuition leads one to believe that it is.

Big Consequences of $P = NP$

We will see that $P = NP$ leads to a great many complexity classes to also be equal to $P$.

One such example is $P = coNP$. 

Billy Hardy

P vs NP
The Meaning of $P$ vs $NP$

The biggest consequence of the relationship between $P$ and $NP$ is whether it is harder to find solutions than it is to check solutions. Intuition leads one to believe that it is.

Big Consequences of $P = NP$

We will see that $P = NP$ leads to a great many complexity classes to also be equal to $P$. One such example is $P = coNP$, since one can easily switch the outputs “yes” and “no” of polynomial-time algorithms.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

More Complexity Classes

**NP and coNP**

$NP$ and $coNP$ can be thought of as $P$ problems which ask for existence (or lack thereof). This is because the definition of $NP$ is $\exists w : B(x, w)$ where $w$ is the witness and $B$ is in $P$, and $coNP$ is $\forall w : B(x, w)$.
More Complexity Classes

NP and coNP

NP and coNP can be thought of as P problems which ask for existence (or lack thereof).
NP and coNP

NP and coNP can be thought of as P problems which ask for existence (or lack there of).

This is because the definition of NP is $\exists w : B(x, w)$ where $w$ is the witness and $B$ is in P, and coNP is $\forall w : B(x, w)$. 
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

More Complexity Classes

**NP and coNP**

NP and coNP can be thought of as $P$ problems which ask for existence (or lack there of).

This is because the definition of NP is $\exists w : B(x, w)$ where $w$ is the witness and $B$ is in $P$, and coNP is $\forall w : B(x, w)$.

**Extending the Idea**
More Complexity Classes

**NP and coNP**

NP and coNP can be thought of as \( P \) problems which ask for existence (or lack thereof).

This is because the definition of NP is \( \exists w : B(x, w) \) where \( w \) is the witness and \( B \) is in \( P \), and coNP is \( \forall w : B(x, w) \).

**Extending the Idea**

One can extend this idea by adding more and more quantifiers.
### The Class $\Pi_2^P$

The class $\Pi_2^P$ consists of properties of the form:

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

**Smallest Boolean Circuit**

- **Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.
- **Query:** Is $C$ the smallest circuit that computes $f_C$?

**Logically:**

$$\forall C' < C : \exists x : f_{C'}(x) \neq f_C(x)$$

**Observation**

Obviously, Smallest Boolean Circuit is in $\Pi_2^P$. 

---

**Billy Hardy**

P vs NP
The Class $\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y: \exists z: B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.
The Class $\Pi_2^P$

$\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$
The Class $\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.
The Class $\Pi_2P$

$\Pi_2P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

Input: A Boolean circuit $C$ that computes a function $f_C$ of its input.

Query: Is $C$ the smallest circuit that computes $f_C$?

Logically:

$$\forall C' < C : \exists x : f_{C'}(x) \neq f_C(x)$$

Observation

Obviously Smallest Boolean Circuit is in $\Pi_2P$. 
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

The Class $\Pi_2^P$

$\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

**Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.
The Class $\Pi_2^P$

$\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $\mathbf{P}$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

**Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.

**Query:** Is $C$ the smallest circuit that computes $f_C$?
The Class $\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $\mathbf{P}$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

**Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.

**Query:** Is $C$ the smallest circuit that computes $f_C$?

Logically: $\forall C' : C' < C : \exists x : f_{C'}(x) \neq f_C(x)$
The Class $\Pi_2^P$

$\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

**Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.

**Query:** Is $C$ the smallest circuit that computes $f_C$?

Logically: $\forall C' < C : \exists x : f_{C'}(x) \neq f_C(x)$

Observation
The Class $\Pi_2^P$

$\Pi_2^P$

The class of properties $A$ of the form

$$A(x) = \forall y : \exists z : B(x, y, z)$$

where $B$ is in $P$, and where $|y|$ and $|z|$ are polynomial in $|x|$.

Smallest Boolean Circuit

**Input:** A Boolean circuit $C$ that computes a function $f_C$ of its input.

**Query:** Is $C$ the smallest circuit that computes $f_C$?

Logically:

$$\forall C' < C : \exists x : f_{C'}(x) \neq f_C(x)$$

Observation

Obviously Smallest Boolean Circuit is in $\Pi_2^P$. 
Further Classes

$\Sigma^P_k$ is the class of properties $A$ of the form:
$$A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Qy_k : B(x, y_1, \ldots, y_k),$$
where $B$ is in $P$, $|y_i| = \text{poly}(|x|)$ for all $i$, and $Q = \exists$ if $k$ is odd, otherwise $\forall$.

$\Pi^P_k$ is the class of properties $A$ of the form:
$$A(x) = \forall y_1 : \exists y_2 : \forall y_3 : \cdots : Qy_k : B(x, y_1, \ldots, y_k),$$
where $B$ is in $P$, $|y_i| = \text{poly}(|x|)$ for all $i$, and $Q = \forall$ if $k$ is odd, otherwise $\exists$. 
Further Classes

\( \Sigma_k P \)

\( \Sigma_k P \) is the class of properties \( A \) of the form

\[ A(x) = \begin{cases} \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k), & \text{where } B \text{ is in } P, |y_i| = \text{poly}(|x|) \text{ for all } i, \text{ and } Q = \exists \text{ if } k \text{ is odd, otherwise } \forall. \end{cases} \]
Further Classes

<table>
<thead>
<tr>
<th>$\Sigma_k P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_k P$ is the class of properties $A$ of the form</td>
</tr>
<tr>
<td>$A(x) = \exists y_1: \forall y_2: \exists y_3: \cdots: Qy_k : B(x, y_1, \ldots, y_k)$,</td>
</tr>
</tbody>
</table>
Further Classes

$\Sigma_k P$

$\Sigma_k P$ is the class of properties $A$ of the form

$$A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k),$$

where $B$ is in $P$, $|y_i| = \text{poly}(|x|)$ for all $i$, and $Q = \exists$ if $k$ is odd, otherwise $\forall$.
Further Classes

\[ \Sigma_k P \]

\( \Sigma_k P \) is the class of properties \( A \) of the form

\[ A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k), \]

where \( B \) is in \( P \), \( |y_i| = poly(|x|) \) for all \( i \), and \( Q = \exists \) if \( k \) is odd, otherwise \( \forall \).

\[ \Pi_k P \]

\( \Pi_k P \) is the class of properties \( A \) of the form

\[ A(x) = \forall y_1 : \exists y_2 : \forall y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k), \]

where \( B \) is in \( P \), \( |y_i| = poly(|x|) \) for all \( i \), and \( Q = \forall \) if \( k \) is odd, otherwise \( \exists \).
Further Classes

**ΣₖP**

ΣₖP is the class of properties A of the form

\[ A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Qy_k : B(x, y_1, \ldots, y_k), \]

where B is in P, |yᵢ| = poly(|x|) for all i, and Q = ∃ if k is odd, otherwise ∀.

**ΠₖP**

ΠₖP is the class of properties A of the form
Further Classes

\( \Sigma_k P \)

\( \Sigma_k P \) is the class of properties \( A \) of the form

\[
A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Qy_k : B(x, y_1, \ldots, y_k),
\]

where \( B \) is in \( P \), \( |y_i| = poly(|x|) \) for all \( i \), and \( Q = \exists \) if \( k \) is odd, otherwise \( \forall \).

\( \Pi_k P \)

\( \Pi_k P \) is the class of properties \( A \) of the form

\[
A(x) = \forall y_1 : \exists y_2 : \forall y_3 : \cdots : Qy_k : B(x, y_1, \ldots, y_k),
\]
Further Classes

\[ \Sigma_k P \]

\( \Sigma_k P \) is the class of properties \( A \) of the form

\[ A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k), \]

where \( B \) is in \( P \), \( |y_i| = poly(|x|) \) for all \( i \), and \( Q = \exists \) if \( k \) is odd, otherwise \( \forall \).

\[ \Pi_k P \]

\( \Pi_k P \) is the class of properties \( A \) of the form

\[ A(x) = \forall y_1 : \exists y_2 : \forall y_3 : \cdots : Q y_k : B(x, y_1, \ldots, y_k), \]

where \( B \) is in \( P \), \( |y_i| = poly(|x|) \) for all \( i \), and \( Q = \forall \) if \( k \) is odd, otherwise \( \exists \).
Further Classes

Understanding These Classes

These classes correspond to two-player games that last for \( k \) moves. For instance, consider a Chess game where white claims they can mate in \( k \) moves. This means there exists a move for white, such that for all of black's replies, there exists a move for white, ... until white has won. Given the initial position and the sequence of moves, it is easy to check whether white has mated black.
Further Classes

Understanding These Classes

These classes correspond to two-player games that last for $k$ moves.
Further Classes

Understanding These Classes

These classes correspond to two-player games that last for $k$ moves.

For instance, consider a Chess game where white claims they can mate in $k$ moves.
Further Classes

Understanding These Classes

These classes correspond to two-player games that last for $k$ moves.

For instance, consider a Chess game where white claims they can mate in $k$ moves.

This means there exists a move for white, such that for all of black’s replies, there exists a move for white, ... until white has won.
Understanding These Classes

These classes correspond to two-player games that last for $k$ moves.

For instance, consider a Chess game where white claims they can mate in $k$ moves.

This means there exists a move for white, such that for all of black’s replies, there exists a move for white, ... until white has won.

Given the initial position and the sequence of moves, it is easy to check whether white has mated black.
## Relationships of the Classes

### Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

\[
\Sigma_k \subseteq \Sigma_{k+1}, \quad \Sigma_k \subseteq \Pi_{k+1}, \quad \Pi_k \subseteq \Sigma_{k+1}, \quad \Pi_k \subseteq \Pi_{k+1}
\]

Nondeterminism

As before, each \( \exists \) can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true. So we can say,

\[
\Sigma_k \mathbf{P} = \mathbf{N}^{\Pi_{k-1}} \mathbf{P}.
\]

And since \( \Sigma_0 \mathbf{P} = \Pi_0 \mathbf{P} = \mathbf{P} \), we have

\[
\Sigma_1 \mathbf{P} = \mathbf{NP} \quad \text{and} \quad \Pi_1 \mathbf{P} = \mathbf{coNP}.
\]

Or, more generally,

\[
\Pi_k \mathbf{P} = \mathbf{co}^{\Sigma_{k-1}} \mathbf{P}.
\]

since the negation of \( \forall \) is \( \exists \), and vice versa.
Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so
One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

\[ \Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1} \]
Subsets
One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

$$\Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1}$$

Nondeterminism

As before, each $\exists$ can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true. So we can say,

$$\Sigma_k^P = \Pi_{k-1}^P$$

And since $\Sigma_0^P = \Pi_0^P = \mathbf{P}$, we have

$$\Sigma_1^P = \mathbf{NP} \text{ and } \Pi_1^P = \mathbf{coNP}$$

Or, more generally,

$$\Pi_k^P = \mathbf{co}\Sigma_k^P$$

since the negation of $\forall$ is $\exists$, and vice versa.
### Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

\[
\Sigma_k \subseteq \Sigma_{k+1}, \quad \Sigma_k \subseteq \Pi_{k+1}, \quad \Pi_k \subseteq \Sigma_{k+1}, \quad \Pi_k \subseteq \Pi_{k+1}
\]

### Nondeterminism

As before, each \(\exists\) can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.
What if $P = \text{NP}$?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

Relationships of the Classes

Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

\[
\Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1}
\]

Nondeterminism

As before, each $\exists$ can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.

So we can say,

\[
\Sigma_k P = \mathbf{N}\Pi_{k-1} P
\]
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

Relationships of the Classes

Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

$$
\Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1}
$$

Nondeterminism

As before, each $\exists$ can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.

So we can say,

$$
\Sigma_k P = \mathbf{N} \Pi_{k-1} P
$$

And since $\Sigma_0 P = \Pi_0 P = P$, 

Billy Hardy

$P$ vs $NP$
### Relationships of the Classes

#### Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

\[
\Sigma_k \subseteq \Sigma_{k+1}, \quad \Sigma_k \subseteq \Pi_{k+1}, \quad \Pi_k \subseteq \Sigma_{k+1}, \quad \Pi_k \subseteq \Pi_{k+1}
\]

#### Nondeterminism

As before, each \( \exists \) can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.

So we can say,

\[
\Sigma_k \mathbb{P} = \mathbb{N} \Pi_{k-1} \mathbb{P}.
\]

And since \( \Sigma_0 \mathbb{P} = \Pi_0 \mathbb{P} = \mathbb{P} \), we have

\[
\Sigma_1 \mathbb{P} = \mathbb{NP} \quad \text{and} \quad \Pi_1 \mathbb{P} = \mathbb{coNP}
\]
Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

$$\Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1}$$

Nondeterminism

As before, each $\exists$ can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.

So we can say,

$$\Sigma_k P = N \Pi_{k-1} P .$$

And since $\Sigma_0 P = \Pi_0 P = P$, we have

$$\Sigma_1 P = NP \text{ and } \Pi_1 P = coNP$$

Or, more generally,

$$\Pi_k P = co\Sigma_k P .$$
## Relationships of the Classes

### Subsets

One can easily add quantifiers with dummy variables inside or outside each of the problems in the classes, so

$$\Sigma_k \subseteq \Sigma_{k+1}, \Sigma_k \subseteq \Pi_{k+1}, \Pi_k \subseteq \Sigma_{k+1}, \Pi_k \subseteq \Pi_{k+1}$$

### Nondeterminism

As before, each $\exists$ can be thought of as a layer of nondeterminism that asks whether there is a witness that makes the statement inside that quantifier true.

So we can say,

$$\Sigma_k P = \mathbb{N}\Pi_{k-1} P$$

And since $\Sigma_0 P = \Pi_0 P = P$, we have

$$\Sigma_1 P = \text{NP} \text{ and } \Pi_1 P = \text{coNP}$$

Or, more generally,

$$\Pi_k P = \text{co}\Sigma_k P$$

since the negation of $\forall$ is $\exists$, and vice versa.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

Polynomial Hierarchy

These complexity classes are known, collectively, as the polynomial hierarchy. Taking their union over all $k$ gives the class

$$\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^P = \bigcup_{k=0}^{\infty} \Pi_k^P,$$

which consists of problems that can be phrased with any constant number of quantifiers.

Classes are Distinct

Analogous to the belief that $P \neq NP$ and $NP \neq \text{coNP}$, it is believed that the classes $\Sigma_k$ and $\Pi_k$ are all distinct.

In other words, whenever one adds a quantifier, or a layer of nondeterminism, a fundamentally deeper kind of problem is obtained.
Polynomial Hierarchy

These complexity classes are known, collectively, as the *polynomial hierarchy*. 
These complexity classes are known, collectively, as the *polynomial hierarchy*. 

Taking their union over all $k$ gives the class
Polynomial Hierarchy

These complexity classes are known, collectively, as the *polynomial hierarchy*. Taking their union over all \( k \) gives the class

\[
\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k \text{P} = \bigcup_{k=0}^{\infty} \Pi_k \text{P},
\]
Polynomial Hierarchy

These complexity classes are known, collectively, as the *polynomial hierarchy*. Taking their union over all $k$ gives the class

$$PH = \bigcup_{k=0}^{\infty} \Sigma_k P = \bigcup_{k=0}^{\infty} \Pi_k P,$$

which consists of problems that can be phrased with any constant number of quantifiers.
These complexity classes are known, collectively, as the *polynomial hierarchy*. 

Taking their union over all $k$ gives the class 

$$
\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k \text{P} = \bigcup_{k=0}^{\infty} \Pi_k \text{P},
$$

which consists of problems that can be phrased with any constant number of quantifiers.
Polynomial Hierarchy

These complexity classes are known, collectively, as the *polynomial hierarchy*. Taking their union over all \( k \) gives the class

\[
\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k \text{P} = \bigcup_{k=0}^{\infty} \Pi_k \text{P} ,
\]

which consists of problems that can be phrased with any constant number of quantifiers.

Classes are Distinct

Analogous to the belief that \( \text{P} \neq \text{NP} \) and \( \text{NP} \neq \text{coNP} \),
Polynomial Hierarchy

These complexity classes are known, collectively, as the polynomial hierarchy.

Taking their union over all $k$ gives the class

$$\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k \text{P} = \bigcup_{k=0}^{\infty} \Pi_k \text{P},$$

which consists of problems that can be phrased with any constant number of quantifiers.

Classes are Distinct

Analogous to the belief that $P \neq \text{NP}$ and $\text{NP} \neq \text{coNP}$, it is believed that the classes $\Sigma_k$ and $\Pi_k$ are all distinct.
Polynomial Hierarchy

These complexity classes are known, collectively, as the *polynomial hierarchy*. Taking their union over all $k$ gives the class

$$\text{PH} = \bigcup_{k=0}^{\infty} \Sigma_k \text{P} = \bigcup_{k=0}^{\infty} \Pi_k \text{P},$$

which consists of problems that can be phrased with any constant number of quantifiers.

Classes are Distinct

Analogous to the belief that $\text{P} \neq \text{NP}$ and $\text{NP} \neq \text{coNP}$, it is believed that the classes $\Sigma_k$ and $\Pi_k$ are all distinct.

In other words, whenever one adds a quantifier, or a layer of nondeterminism, a fundamentally deeper kind of problem is obtained.
The Great Collapse

If $P = NP$

If $P = NP$, then if $B(x, y) \in P$, then $A(x) = \exists y : B(x, y) \in P$, by definition.

Since $P = \text{coNP}$ as well, we also absorb $\exists$ and $\forall$.

By continually absorbing quantifiers, we get $\text{PH} = P$. 

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

Billy Hardy

P vs NP
If $P = NP$

If $P = NP$, then
If $P = NP$

If $P = NP$, then

\[ \text{if } B(x, y) \in P, \text{ then } A(x) = \exists y : B(x, y) \in P, \]

by definition.
If $P = NP$

If $P = NP$, then

$$\text{if } B(x, y) \in P, \text{ then } A(x) = \exists y : B(x, y) \in P,$$

by definition.

Since $P = coNP$ as well, we also absorb $\exists$ and $\forall$. 
If \( P = NP \), then

\[
  \text{if } B(x, y) \in P, \text{ then } A(x) = \exists y : B(x, y) \in P,
\]

by definition.

Since \( P = coNP \) as well, we also absorb \( \exists \) and \( \forall \).

By continually absorbing quantifiers, we get \( PH = P \)
The Great Collapse

Claim

If $\text{NP} = \text{coNP}$, then $\text{PH} = \text{NP}$.

Proof

1. Let $A(x) = \exists y : B(x, y)$ be in $\text{NP} = \Sigma_1^P$.
2. Then $C(x) = \forall z : A(x)$ is also in $\Pi_2^P$.
3. Since $A(x)$ is in $\text{coNP}$, so $A(x) = \forall y : B(x, y)$.
4. Then $C(x) = \forall z : \forall y : B(x, y) = A(x)$.
5. Thus $\text{NP} = \Pi_2^P$.
6. The inductive proof follows from here.
Claim

If $NP = coNP$, then $PH = NP$. 

Proof

1. Let $A(x) = \exists y : B(x, y)$ be in $NP = \Sigma_1^P$.
2. Then $C(x) = \forall z : A(x)$ is in $\Pi_2^P$.
3. $A(x)$ is also in $coNP$, so $A(x) = \forall y : B(x, y)$.
4. $C(x) = \forall z : \forall y : B(x, y)$.
5. $C(x) = \forall (z, y) : B(x, (z, y)) = A(x)$.
6. The inductive proof follows from here.

Billy Hardy

P vs NP
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

The Great Collapse

Claim

If $NP = coNP$, then $PH = NP$.

Proof
**The Great Collapse**

**Claim**
If $\text{NP} = \text{coNP}$, then $\text{PH} = \text{NP}$.

**Proof**

1. Let $A(x) = \exists y : B(x, y)$ be in $\text{NP} = \Sigma_1 \text{P}$
The Great Collapse

Claim
If \( \text{NP} = \text{coNP} \), then \( \text{PH} = \text{NP} \).

Proof
1. Let \( A(x) = \exists y : B(x, y) \) be in \( \text{NP} = \Sigma_1 \text{P} \)
2. \( C(x) = \forall z : A(x) \) is in \( \Pi_2 \text{P} \)
Claim

If $\text{NP} = \text{coNP}$, then $\text{PH} = \text{NP}$.

Proof

1. Let $A(x) = \exists y : B(x, y)$ be in $\text{NP} = \Sigma_1 \text{P}$
2. $C(x) = \forall z : A(x)$ is in $\Pi_2 \text{P}$
3. $A(x)$ is also in $\text{coNP}$, so $A(x) = \forall y : B(x, y)$
Claim
If $\text{NP} = \text{coNP}$, then $\text{PH} = \text{NP}$.

Proof
1. Let $A(x) = \exists y : B(x, y)$ be in $\text{NP} = \Sigma_1 \text{P}$
2. $C(x) = \forall z : A(x)$ is in $\Pi_2 \text{P}$
3. $A(x)$ is also in $\text{coNP}$, so $A(x) = \forall y : B(x, y)$
4. $C(x) = \forall z : \forall y : B(x, y)$
The Great Collapse

Claim

If $NP = coNP$, then $PH = NP$.

Proof

1. Let $A(x) = \exists y : B(x, y)$ be in $NP = \Sigma_1 P$
2. $C(x) = \forall z : A(x)$ is in $\Pi_2 P$
3. $A(x)$ is also in $coNP$, so $A(x) = \forall y : B(x, y)$
4. $C(x) = \forall z : \forall y : B(x, y)$
5. $C(x) = \forall (z, y) : B(x, (z, y)) = A(x)$ So $NP = \Pi_2 P$
Claim

If $\text{NP} = \text{coNP}$, then $\text{PH} = \text{NP}$.

Proof

1. Let $A(x) = \exists y : B(x, y)$ be in $\text{NP} = \Sigma_1^P$  
2. $C(x) = \forall z : A(x)$ is in $\Pi_2^P$  
3. $A(x)$ is also in $\text{coNP}$, so $A(x) = \forall y : B(x, y)$  
4. $C(x) = \forall z : \forall y : B(x, y)$  
5. $C(x) = \forall (z, y) : B(x, (z, y)) = A(x)$ So $\text{NP} = \Pi_2^P$  
6. The inductive proof follows from here.
Outline

1. What if $P = NP$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard

3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
What if $P = \text{NP}$? 
Upper Bounds are Easy and Lower Bounds, Hard 
Diagonalization and Time Hierarchy

The Great Collapse 
The Power of Nondeterminism 
The Demise of Creativity

The common misconception is that $P \text{ vs. } \text{NP}$ is about polynomial time. In fact, it is really about how powerful nondeterminism is in general. As in, whether finding solutions is inherently harder than checking them.

$\text{EXP}$ Recall that $\text{EXP}$ is the class of problems that one can solve in an exponential amount of time, where "exponential" is defined as $\text{EXP} = \bigcup_k \text{TIME}(2^{n^k}) = \text{TIME}(2^{\text{poly}(n)})$.

$\text{NEXP}$ Also recall that $\text{NEXP} = \text{NTIME}(2^{\text{poly}(n)})$ is the class of problems that one can check a solution in an exponential amount of time.

Not about Polynomial Time
Not about Polynomial Time

The common misconception is that $P$ vs. $NP$ is about polynomial time.
P vs. NP

Not about Polynomial Time

The common misconception is that P vs. NP is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.
**P vs. NP**

**Not about Polynomial Time**

The common misconception is that **P vs. NP** is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.
Not about Polynomial Time

The common misconception is that \( \text{P vs. NP} \) is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.
Not about Polynomial Time

The common misconception is that P vs. NP is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.

EXP

Recall that EXP is the class of problems that one can solve in an exponential amount of time,
What if $P = NP$?  
Upper Bounds are Easy and Lower Bounds, Hard 
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

**P vs. NP**

**Not about Polynomial Time**

The common misconception is that **P vs. NP** is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.

**EXP**

Recall that **EXP** is the class of problems that one can solve in an exponential amount of time, where “exponential” is defined as

\[
\text{EXP} = \bigcup_k \text{TIME}(2^{n^k}) = \text{TIME}(2^{\text{poly}(n)}).
\]
Not about Polynomial Time

The common misconception is that $P \text{ vs. } NP$ is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.

EXP

Recall that $EXP$ is the class of problems that one can solve in an exponential amount of time, where “exponential” is defined as

$$EXP = \bigcup_k \text{TIME}(2^{n^k}) = \text{TIME}(2^{poly(n)}).$$
P vs. NP

Not about Polynomial Time

The common misconception is that P vs. NP is about polynomial time.

In fact, it is really about how powerful nondeterminism is in general.

As in, whether finding solutions is inherently harder than checking them.

EXP

Recall that EXP is the class of problems that one can solve in an exponential amount of time, where “exponential” is defined as

\[ \text{EXP} = \bigcup_{k} \text{TIME}(2^{n^k}) = \text{TIME}(2^{\text{poly}(n)}). \]

NEXP

Also recall that NEXP = NTIME(2^{\text{poly}(n)}) is the class of problems that one can check a solution in an exponential amount of time.
The Relationship between EXP and NEXP

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on.

Proof:
Let problem $A$ be in $NEXP$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros. This takes $O(t(n))$ time, since $t(n)$ is time constructible. This new problem is in $NP$.
So we can solve it deterministically in time $poly(t(n)) = 2^{O(n^c)}$, since $P = NP$.
So $A$ is in $EXP$.
The inductive proof follows similarly.
P vs. NP

The Relationship between EXP and NEXP

In analogy to P ≠ NP, one can check whether EXP ≠ NEXP.
The Relationship between \textbf{EXP} and \textbf{NEXP}

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP^{EXP} \neq NEXP^{EXP}$, and so on.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

**P vs. NP**

### The Relationship between EXP and NEXP

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP^{EXP} \neq NEXP^{EXP}$, and so on.

### Claim
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

**P vs. NP**

The Relationship between **EXP** and **NEXP**

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP^{EXP} \neq NEXP^{EXP}$, and so on.

**Claim**

If $P = NP$
The Relationship between EXP and NEXP

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP^{EXP} \neq NEXP^{EXP}$, and so on.

Claim

If $P = NP$ then $EXP = NEXP$, 
The Relationship between $EXP$ and $NEXP$

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXPEXP \neq NEXPEXP$, and so on.

Claim

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, ...
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

---

**P vs. NP**

---

The Relationship between $\text{EXP}$ and $\text{NEXP}$

In analogy to $P \neq NP$, one can check whether $\text{EXP} \neq \text{NEXP}$.

Furthermore, the extension can be made to $\text{EXP}^{\text{EXP}} \neq \text{NEXP}^{\text{EXP}}$, and so on.

---

Claim

If $P = NP$ then $\text{EXP} = \text{NEXP}$, $\text{EXP}^{\text{EXP}} = \text{NEXP}^{\text{EXP}}$, and so on.
The Relationship between $\text{EXP}$ and $\text{NEXP}$

In analogy to $\text{P} \neq \text{NP}$, one can check whether $\text{EXP} \neq \text{NEXP}$.

Furthermore, the extension can be made to $\text{EXP}^{\text{EXP}} \neq \text{NEXP}^{\text{EXP}}$, and so on.

Claim

If $\text{P} = \text{NP}$ then $\text{EXP} = \text{NEXP}$, $\text{EXP}^{\text{EXP}} = \text{NEXP}^{\text{EXP}}$, and so on.

Proof
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

P vs. NP

The Relationship between $EXP$ and $NEXP$

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP^{EXP} \neq NEXP^{EXP}$, and so on.

Claim

If $P = NP$ then $EXP = NEXP$, $EXP^{EXP} = NEXP^{EXP}$, and so on.

Proof

- Let problem $A$ be in $NEXP$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$. 
**P vs. NP**

The Relationship between **EXP** and **NEXP**

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXPEXP \neq NEXPEXP$, and so on.

**Claim**

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on.

**Proof**

- Let problem $A$ be in **NEXP**, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
The Relationship between EXP and NEXP

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXPEXP \neq NEXPEXP$, and so on.

Claim

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on.

Proof

- Let problem $A$ be in $NEXP$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
- This takes $O(t(n))$ time, since $t(n)$ is time constructible.
The Relationship between \textbf{EXP} and \textbf{NEXP}

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXP EXP \neq NEXP EXP$, and so on.

\textbf{Claim}

If $P = NP$ then $EXP = NEXP$, $EXP EXP = NEXP EXP$, and so on.

\textbf{Proof}

- Let problem $A$ be in $NEXP$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
- This takes $O(t(n))$ time, since $t(n)$ is \textit{time constructible}
- This new problem is in $NP$. 
The Relationship between **EXP** and **NEXP**

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$.

Furthermore, the extension can be made to $EXPEXP \neq NEXPEXP$, and so on.

**Claim**

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on.

**Proof**

- Let problem $A$ be in **NEXP**, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
- This takes $O(t(n))$ time, since $t(n)$ is *time constructible*.
- This new problem is in **NP**.
- So we can solve it deterministically in time $poly(t(n)) = 2^{O(n^c)}$, since $P = NP$.
The Relationship between $\text{EXP}$ and $\text{NEXP}$

In analogy to $\text{P} \neq \text{NP}$, one can check whether $\text{EXP} \neq \text{NEXP}$.

Furthermore, the extension can be made to $\text{EXP}^{\text{EXP}} \neq \text{NEXP}^{\text{EXP}}$, and so on.

Claim

If $\text{P} = \text{NP}$ then $\text{EXP} = \text{NEXP}$, $\text{EXP}^{\text{EXP}} = \text{NEXP}^{\text{EXP}}$, and so on.

Proof

- Let problem $A$ be in $\text{NEXP}$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
- This takes $O(t(n))$ time, since $t(n)$ is time constructible
- This new problem is in $\text{NP}$.
- So we can solve it deterministically in time $\text{poly}(t(n)) = 2^{O(n^c)}$, since $\text{P} = \text{NP}$.
- So $A$ is in $\text{EXP}$.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

**P vs. NP**

### The Relationship between $EXP$ and $NEXP$

In analogy to $P \neq NP$, one can check whether $EXP \neq NEXP$. Furthermore, the extension can be made to $EXPEXP \neq NEXPEXP$, and so on.

### Claim

If $P = NP$ then $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on.

### Proof

- Let problem $A$ be in $NEXP$, so witnesses can be checked in time $t(n) = 2^{O(n^c)}$, for some constant $c$.
- Now pad the input, making it $t(n)$ bits long, by adding $t(n) - n$ zeros.
- This takes $O(t(n))$ time, since $t(n)$ is time constructible.
- This new problem is in $NP$.
- So we can solve it deterministically in time $poly(t(n)) = 2^{O(n^c)}$, since $P = NP$.
- So $A$ is in $EXP$.
- The inductive proof follows similarly.
What if P = NP?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

Time-Constructible

A function \( f \) is called time-constructible if there exists a positive integer \( n_0 \) and Turing machine \( M \) which, given a string \( 1^n \) consisting of \( n \) ones, stops after exactly \( f(n) \) steps for all \( n \geq n_0 \).
Time-Constructible

A function $f$ is called \textit{time-constructible} if there exists a positive integer $n_0$ and Turing machine $M$ which, given a string $1^n$ consisting of $n$ ones, stops after exactly $f(n)$ steps for all $n \geq n_0$. 
What if $P = NP$?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$\text{NTIME}(t(n)) \subseteq \text{TIME}(\text{poly}(t(n)))$$

Or for a class of superpolynomial functions such that $t(n) c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$\text{NTIME}(C) = \text{TIME}(C)$$

The Collapse
This applies not only to exponentials $2^{\text{poly}(n)}$, double exponential $2^{2^{\text{poly}(n)}}$ and so on.

So we have $\text{EXP} = \text{NEXP}$, $\text{EXPEXP} = \text{NEXPEXP}$, and so on up the hierarchy.

It is easy to show that any equality in the hierarchy propagates up, and inequality propagates down.
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$NTIME(t(n)) \subseteq TIME(poly(t(n)))$$
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$NTIME(t(n)) \subseteq TIME(poly(t(n)))$$

Or for a class of superpolynomial functions such that $t(n)^c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$NTIME(C) = TIME(C)$$
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$\text{NTIME}(t(n)) \subseteq \text{TIME}(\text{poly}(t(n)))$$

Or for a class of superpolynomial functions such that $t(n)^c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$\text{NTIME}(C) = \text{TIME}(C)$$

The Collapse
More Generally

We can say if \( P = NP \), then for any time-constructible function \( t(n) \geq n \),

\[
\text{NTIME}(t(n)) \subseteq \text{TIME}(\text{poly}(t(n)))
\]

Or for a class of superpolynomial functions such that \( t(n)^c \in C \) for any \( t(n) \in C \) and any constant \( c \), then

\[
\text{NTIME}(C) = \text{TIME}(C)
\]

The Collapse

This applies not only to exponentials \( 2^{\text{poly}(n)} \), double exponential \( 2^{2^{\text{poly}(n)}} \) and so on.
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$NTIME(t(n)) \subseteq TIME(poly(t(n)))$$

Or for a class of superpolynomial functions such that $t(n)^c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$NTIME(C) = TIME(C)$$

The Collapse

This applies not only to exponentials $2^{poly(n)}$, double exponential $2^{2^{poly(n)}}$ and so on.

So we have $EXP = NEXP$, $EXPEXP = NEXPEXP$, and so on up the hierarchy.
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$\text{NTIME}(t(n)) \subseteq \text{TIME}(\text{poly}(t(n)))$$

Or for a class of superpolynomial functions such that $t(n)^c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$\text{NTIME}(C) = \text{TIME}(C)$$

The Collapse

This applies not only to exponentials $2^{\text{poly}(n)}$, double exponential $2^{2^{\text{poly}(n)}}$ and so on.

So we have $\text{EXP} = \text{NEXP}$, $\text{EXP}^{\text{EXP}} = \text{NEXP}^{\text{EXP}}$, and so on up the hierarchy.

It is easy to show that any equality in the hierarchy propagates up,
More Generally

We can say if $P = NP$, then for any time-constructible function $t(n) \geq n$,

$$\text{NTIME}(t(n)) \subseteq \text{TIME}(\text{poly}(t(n)))$$

Or for a class of superpolynomial functions such that $t(n)^c \in C$ for any $t(n) \in C$ and any constant $c$, then

$$\text{NTIME}(C) = \text{TIME}(C)$$

The Collapse

This applies not only to exponentials $2^{\text{poly}(n)}$, double exponential $2^{2^{\text{poly}(n)}}$ and so on.

So we have $\text{EXP} = \text{NEXP}, \text{EXP}^{\text{EXP}} = \text{NEXP}^{\text{EXP}}$, and so on up the hierarchy.

It is easy to show that any equality in the hierarchy propagates up, and inequality propagates down.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

Proof Finding vs. Checking

Proof Checking

Input: A statement $S$ and a proof $P$.
Query: Is $P$ a valid proof of $S$?

SHORT PROOF

Input: A statement $S$ and an integer $n$ given in unary.
Query: Does $S$ have a proof of length $n$ or less?

Observation: Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. Furthermore, since $S$ can be a SAT formula, Short Proof is $NP$-complete.
Proof Finding vs. Checking

**PROOF CHECKING**

**Input:** A statement $S$ and a proof $P$
### Proof Finding vs. Checking

<table>
<thead>
<tr>
<th><strong>PROOF CHECKING</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A statement $S$ and a proof $P$</td>
</tr>
<tr>
<td><strong>Query:</strong> Is $P$ a valid proof of $S$?</td>
</tr>
</tbody>
</table>

**Observation:**

Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. Furthermore, since $S$ can be a SAT formula, Short Proof is $NP$-complete.
Proof Finding vs. Checking

**Proof Checking**

**Input:** A statement $S$ and a proof $P$

**Query:** Is $P$ a valid proof of $S$?

**Short Proof**

Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. Furthermore, since $S$ can be a SAT formula, Short Proof is $NP$-complete.
Proof Finding vs. Checking

**Proof Checking**

**Input:** A statement $S$ and a proof $P$

**Query:** Is $P$ a valid proof of $S$?

**Short Proof**

**Input:** A statement $S$ and an integer $n$ given in unary
Proof Finding vs. Checking

**Proof Checking**

*Input:* A statement $S$ and a proof $P$
*Query:* Is $P$ a valid proof of $S$?

**Short Proof**

*Input:* A statement $S$ and an integer $n$ given in unary
*Query:* Does $S$ have a proof of length $n$ or less?
Proof Finding vs. Checking

**Proof Checking**

**Input:** A statement $S$ and a proof $P$

**Query:** Is $P$ a valid proof of $S$?

**Short Proof**

**Input:** A statement $S$ and an integer $n$ given in unary

**Query:** Does $S$ have a proof of length $n$ or less?

**Observation**

Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. Furthermore, since $S$ can be a SAT formula, Short Proof is $NP$-complete.
Proof Finding vs. Checking

**Proof Checking**
- **Input:** A statement $S$ and a proof $P$
- **Query:** Is $P$ a valid proof of $S$?

**Short Proof**
- **Input:** A statement $S$ and an integer $n$ given in unary
- **Query:** Does $S$ have a proof of length $n$ or less?

**Observation**
- Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. 
### Proof Checking

**Input:** A statement $S$ and a proof $P$  
**Query:** Is $P$ a valid proof of $S$?

### Short Proof

**Input:** A statement $S$ and an integer $n$ given in unary  
**Query:** Does $S$ have a proof of length $n$ or less?

### Observation

Obviously Proof Checking is in $P$, which implies Short Proof is in $NP$. Furthermore, since $S$ can be a SAT formula, Short Proof is $NP$-complete.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization and Time Hierarchy

The Great Collapse

The Power of Nondeterminism

The Demise of Creativity

ELEGANT THEORY

Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs. So we can solve $SHORTPROOF$ in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

The consequences of $P = NP$ reach beyond Computer Science, as we can tweak $SHORTPROOF$ a bit.

Input: A set $E$ of experimental data and an integer $n$ given in unary

Query: Is there a theory $T$ of length $n$ or less that explains $E$?

Billy Hardy

P vs NP
Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs.
### Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs.

So we can solve SHORT PROOF in polynomial time precisely if we can do better than an exhaustive search.
Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs.

So we can solve SHORT PROOF in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

Input: A set $E$ of experimental data and an integer $n$ given in unary
Query: Is there a theory $T$ of length $n$ or less that explains $E$?
What if $P = NP$?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

**ELEGANT THEORY**

Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs.

So we can solve `SHORT PROOF` in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

The consequences of $P = NP$ reach beyond Computer Science, as we can tweak `SHORT PROOF` a bit.
Exhaustive Search

If there are \( k \) letters in the alphabet for proofs, there are \( k^n \) possible proofs.

So we can solve \textsc{Short Proof} in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

The consequences of \( P = NP \) reach beyond Computer Science, as we can tweak \textsc{Short Proof} a bit.
Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs. So we can solve SHORT PROOF in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

The consequences of $P = NP$ reach beyond Computer Science, as we can tweak SHORT PROOF a bit.

**Input:** A set $E$ of experimental data and an integer $n$ given in unary
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

The Great Collapse
The Power of Nondeterminism
The Demise of Creativity

ELEGANT THEORY

Exhaustive Search

If there are $k$ letters in the alphabet for proofs, there are $k^n$ possible proofs.

So we can solve SHORT PROOF in polynomial time precisely if we can do better than an exhaustive search.

Not Just Computer Science

The consequences of $P = NP$ reach beyond Computer Science, as we can tweak SHORT PROOF a bit.

ELEGANT THEORY

Input: A set $E$ of experimental data and an integer $n$ given in unary
Query: Is there a theory $T$ of length $n$ or less that explains $E$?
Outline

1. What if $P = NP$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity
2. Upper Bounds are Easy and Lower Bounds, Hard
3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
Proving $P \neq NP$

**Strategy**

The direct strategy is to prove that for some problem $A \in NP$, $A \not\in P$. So one must prove that every possible polynomial time algorithm that could solve $A$, fails. Which is not easy.

**Complexity Classes**

This leads us to realize that proving upper bounds on classes is easy compared to proving lower bounds. (One just has to find one such algorithm that solves $A$ in polynomial time to increase the upper bound on $P$.)
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Proving $P \neq NP$

Strategy

The direct strategy is to prove that for some problem $A \in \text{NP}$, $A \not\in \text{P}$.
Proving $P \neq NP$

Strategy

The direct strategy is to prove that for some problem $A \in \text{NP}, A \not\in \text{P}$. So one must prove that every possible polynomial time algorithm that could solve $A$, fails.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Proving $P \neq NP$

Strategy

The direct strategy is to prove that for some problem $A \in \text{NP}$, $A \notin \text{P}$.

So one must prove that every possible polynomial time algorithm that could solve $A$, fails.

Which is not easy.
What if \( P = NP \) ?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

**Proving** \( P \neq NP \)

**Strategy**

The direct strategy is to prove that for some problem \( A \in \text{NP} \), \( A \notin P \).

So one must prove that every possible polynomial time algorithm that could solve \( A \), fails.

Which is not easy.

**Complexity Classes**
Proving $P \neq NP$

### Strategy

The direct strategy is to prove that for some problem $A \in \text{NP}$, $A \notin P$.

So one must prove that every possible polynomial time algorithm that could solve $A$, fails.

Which is not easy.

### Complexity Classes

This leads us to realize that proving upper bounds on classes is easy compared to proving lower bounds.
Proving $P \neq NP$

### Strategy

The direct strategy is to prove that for some problem $A \in \text{NP}$, $A \not\in P$.

So one must prove that **every possible** polynomial time algorithm that could solve $A$, fails.

Which is not easy.

### Complexity Classes

This leads us to realize that proving upper bounds on classes is easy compared to proving lower bounds.

(One just has to find one such algorithm that solves $A$ in polynomial time to increase the upper bound on $P$)
Easy Lower Bounds

Sorting a List

As shown in Chapter 3, sorting a list of numbers has to be done in at least $O(n \cdot \log(n))$ time, when comparisons between list members are made. Better than $O(n \cdot \log(n))$ Radix sort achieves $O(mn)$ time, where $m$ is the number of bits used to represent the elements.
Easy Lower Bounds

Sorting a List

As shown in Chapter 3, sorting a list of numbers has to be done in at least $O(n \cdot \log(n))$ time,
Easy Lower Bounds

Sorting a List

As shown in Chapter 3, sorting a list of numbers has to be done in at least $O(n \cdot \log(n))$ time, when comparisons between list members are made.
Easy Lower Bounds

**Sorting a List**

As shown in Chapter 3, sorting a list of numbers has to be done in at least $O(n \cdot \log(n))$ time, when comparisons between list members are made.

**Better Than $O(n \cdot \log(n))$**
Sorting a List

As shown in Chapter 3, sorting a list of numbers has to be done in at least $O(n \cdot \log(n))$ time, when comparisons between list members are made.

Better Than $O(n \cdot \log(n))$

Radix sort achieves $O(mn)$ time, where $m$ is the number of bits used to represent the elements.
Outline

1. What if $P = NP$?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard

3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Proving Inequality of Classes

Technique
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Proving Inequality of Classes

Technique

While proving a particular problem can not be solved in polynomial time appears daunting.
What if $P = NP$?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

Proving Inequality of Classes

Technique

While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of $P$ is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.
Proving Inequality of Classes

Technique

While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of P is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.

Thus, for the class C which these problems belong to, we can conclude that
While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of P is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.

Thus, for the class C which these problems belong to, we can conclude that $P \neq C$. 
Proving Inequality of Classes

Technique

While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of P is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.

Thus, for the class C which these problems belong to, we can conclude that P ≠ C.

Our Goal

We will use the above technique to prove P ≠ EXPTIME.

Or, more generally, that \( \text{TIME}(g(n)) \subset \text{TIME}(f(n)) \), for \( g(n) \in o(f(n)) \).
What if $P = NP$?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

Proving Inequality of Classes

**Technique**

While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of $P$ is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.

Thus, for the class $C$ which these problems belong to, we can conclude that $P \neq C$.

**Our Goal**

We will use the above technique to prove $P \neq \text{EXPTIME}$.
Proving Inequality of Classes

Technique

While proving a particular problem can not be solved in polynomial time appears daunting.

Our actual strategy for proving problems are outside of $P$ is to construct artificial problems that any polynomial time algorithm gets incorrect in at least one case.

Thus, for the class $C$ which these problems belong to, we can conclude that $P \neq C$.

Our Goal

We will use the above technique to prove $P \neq \text{EXPTIME}$.

Or, more generally, that $\text{TIME}(g(n)) \subset \text{TIME}(f(n))$, for $g(n) \in o(f(n))$. 
Outline

1. What if \( P = NP \) ?
   - The Great Collapse
   - The Power of Nondeterminism
   - The Demise of Creativity

2. Upper Bounds are Easy and Lower Bounds, Hard

3. Diagonalization and Time Hierarchy
   - Time Hierarchy Theorem
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem $P_{REDICT}$, which will take in as input a problem $\Pi$ and $\Pi$'s input $x$.

$P_{REDICT}(\Pi, x)$

**Input:** A program $\Pi$ and an input $x$

**Output:** If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. Otherwise, return "don't know."

$P_{REDICT}$'s Behavior

Since $f$ is fixed, different values of $f$ we get different versions of $P_{REDICT}$. $P_{REDICT}$ captures $\Pi$'s behavior for precisely $f(|x|)$ steps or less. Not some constant times $f(|x|)$. 

Billy Hardy

P vs NP
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem PREDICT,
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem $\text{PREDICT}$, which will take in as input a problem $\Pi$ and
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem \textsc{Predict}, which will take in as input a problem $\Pi$ and $\Pi$'s input $x$. 
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem PREDICT, which will take in as input a problem $\Pi$ and $\Pi$’s input $x$.

PREDICT($\Pi, x$)
Problem Construction

General Problem
For a fixed function $f(n)$, we create the problem $\text{PREDICT}$, which will take in as input a problem $\Pi$ and $\Pi$’s input $x$.

$\text{PREDICT}(\Pi, x)$

Input: A program $\Pi$ and an input $x$
Problem Construction

**General Problem**
For a fixed function $f(n)$, we create the problem $PREDICT$, which will take in as input a problem $\Pi$ and $\Pi$’s input $x$.

**$PREDICT(\Pi, x)$**

**Input:** A program $\Pi$ and an input $x$

**Output:** If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. 

Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem $\text{PREDICT}$, which will take in as input a problem $\Pi$ and $\Pi$'s input $x$.

$\text{PREDICT}(\Pi, x)$

**Input**: A program $\Pi$ and an input $x$

**Output**: If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. Otherwise, return “don’t know.”
Problem Construction

General Problem
For a fixed function $f(n)$, we create the problem PREDICT, which will take in as input a problem $\Pi$ and $\Pi$’s input $x$.

PREDICT($\Pi, x$)

Input: A program $\Pi$ and an input $x$
Output: If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. Otherwise, return “don’t know.”

PREDICT’s Behavior
Problem Construction

General Problem

For a fixed function $f(n)$, we create the problem $\text{PREDICT}$, which will take in as input a problem $\Pi$ and $\Pi$’s input $x$.

$\text{PREDICT}(\Pi, x)$

Input: A program $\Pi$ and an input $x$
Output: If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. Otherwise, return “don’t know.”

$\text{PREDICT}$’s Behavior

- Since $f$ is fixed, different values of $f$ we get different versions of $\text{PREDICT}$.
Problem Construction

General Problem

For a fixed function \( f(n) \), we create the problem \textsc{Predict}, which will take in as input a problem \( \Pi \) and \( \Pi \)'s input \( x \).

\textsc{Predict}(\Pi, x)

\textbf{Input:} A program \( \Pi \) and an input \( x \)
\textbf{Output:} If \( \Pi \) halts within \( f(|x|) \) steps when given \( x \) as input, return its output \( \Pi(x) \). Otherwise, return “don’t know.”

\textsc{Predict}'s Behavior

- Since \( f \) is fixed, different values of \( f \) we get different versions of \textsc{Predict}.
- \textsc{Predict} captures \( \Pi \)'s behavior for \textit{precisely} \( f(|x|) \) steps or less.
General Problem

For a fixed function $f(n)$, we create the problem $\text{PREDICT}$, which will take in as input a problem $\Pi$ and $\Pi$'s input $x$.

$\text{PREDICT}(\Pi, x)$

**Input:** A program $\Pi$ and an input $x$

**Output:** If $\Pi$ halts within $f(|x|)$ steps when given $x$ as input, return its output $\Pi(x)$. Otherwise, return “don’t know.”

$\text{PREDICT}$’s Behavior

- Since $f$ is fixed, different values of $f$ we get different versions of $\text{PREDICT}$.
- $\text{PREDICT}$ captures $\Pi$’s behavior for *precisely* $f(|x|)$ steps or less. Not some constant times $f(|x|)$.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**Catch22($\Pi$)**

**Claim**

$\text{CATCH22}(\Pi)$

**Proof**

Assume the contrary. So $\exists \Pi_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.

So $\Pi_{22}(\Pi_{22})$ runs within $f(|\Pi_{22}|)$ steps.

So $\Pi_{22}(\Pi_{22}) = \Pi_{22}(\Pi_{22})$.

So there can not exist any program $\Pi_{22}$.
What if P = NP?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

Diagonalization

**CATCH22(Π)**

**Input:** A program Π

**Claim:** CATCH22(Π) cannot be solved in $f(n)$ steps or less.

**Proof**
Assume the contrary. So $\exists \ Π_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.

$\therefore Π_{22}(Π_{22})$ runs within $f(|Π_{22}|)$ steps.

$Π_{22}(Π_{22}) = Π_{22}(Π_{22})$

So there can not exist any program $Π_{22}$. 
What if $P = \text{NP}$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**Catch22($\Pi$)**

**Input:** A program $\Pi$

**Output:** If $\Pi$ halts within $f(|\Pi|)$ steps when given its own source code as input, return the *negation* of its output $\overline{\Pi(\Pi)}$. 

Claim: Catch22 cannot be solved in $f(n)$ steps or less.

Proof: Assume the contrary. So $\exists \Pi_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less. So $\Pi_{22}(\Pi_{22})$ runs within $f(|\Pi_{22}|)$ steps. So $\Pi_{22}(\Pi_{22}) = \overline{\Pi_{22}(\Pi_{22})}$, so there cannot exist any program $\Pi_{22}$. 

Billy Hardy  

P vs NP
Diagonalization

**CATCH22(Π)**

**Input:** A program Π

**Output:** If Π halts within $f(|Π|)$ steps when given its own source code as input, return the *negation* of its output $Π(Π)$. Otherwise, return “don’t know.”
**Diagonalization**

**CATCH22(Π)**

**Input:** A program Π

**Output:** If Π halts within $f(|Π|)$ steps when given its own source code as input, return the *negation* of its output $Π(Π)$. Otherwise, return “don’t know.”

**Claim**
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

$\text{CATCH22}(\Pi)$

**Input:** A program $\Pi$

**Output:** If $\Pi$ halts within $f(|\Pi|)$ steps when given its own source code as input, return the *negation* of its output $\overline{\Pi(\Pi)}$. Otherwise, return “don’t know.”

Claim

$\text{CATCH22}$ can not be solved in $f(n)$ steps or less.
CATCH22(Π)

Input: A program Π
Output: If Π halts within $f(|Π|)$ steps when given its own source code as input, return the *negation* of its output $Π(Π)$. Otherwise, return “don’t know.”

Claim

CATCH22 cannot be solved in $f(n)$ steps or less.

Proof
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**CATCH22($\Pi$)**

**Input:** A program $\Pi$

**Output:** If $\Pi$ halts within $f(|\Pi|)$ steps when given its own source code as input, return the *negation* of its output $\Pi(\Pi)$. Otherwise, return “don’t know.”

**Claim**

CATCH22 can not be solved in $f(n)$ steps or less.

**Proof**

Assume the contrary. So $\exists \Pi_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.
**Diagonalization**

### CATCH22(Π)

**Input:** A program Π  
**Output:** If Π halts within $f(|Π|)$ steps when given its own source code as input, return the *negation* of its output $Π(Π)$. Otherwise, return “don’t know.”

### Claim

**CATCH22** cannot be solved in $f(n)$ steps or less.

### Proof

- Assume the contrary. So $∃Π_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.
- So $Π_{22}(Π_{22})$ runs within $f(|Π_{22}|)$ steps.
Diagonalization

**CATCH22(Π)**

**Input:** A program Π

**Output:** If Π halts within $f(|Π|)$ steps when given its own source code as input, return the *negation* of its output $Π(Π)$. Otherwise, return “don’t know.”

**Claim**

*CATCH22* can not be solved in $f(n)$ steps or less.

**Proof**

- Assume the contrary. So $∃Π_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.
- So $Π_{22}(Π_{22})$ runs within $f(|Π_{22}|)$ steps.
- So $Π_{22}(Π_{22}) = \overline{Π_{22}(Π_{22})}$
**What if $P = NP$?**

Upper Bounds are Easy and Lower Bounds, Hard

**Diagonalization and Time Hierarchy**

---

**Diagonalization**

**CATCH22($\Pi$)**

**Input:** A program $\Pi$

**Output:** If $\Pi$ halts within $f(|\Pi|)$ steps when given its own source code as input, return the *negation* of its output $\Pi(\Pi)$. Otherwise, return “don’t know.”

---

**Claim**

CATCH22 can not be solved in $f(n)$ steps or less.

---

**Proof**

- Assume the contrary. So $\exists \Pi_{22}$ which runs on inputs $x$ in $f(|x|)$ steps or less.
- So $\Pi_{22}(\Pi_{22})$ runs within $f(|\Pi_{22}|)$ steps.
- So $\Pi_{22}(\Pi_{22}) = \overline{\Pi_{22}(\Pi_{22})}$
- So there can not exist any program $\Pi_{22}$. 
Diagonalization

Since $\text{ATCH}_22$ is just a special case of $\text{PREDICT}$ and takes one extra step to negate the result. This means $\text{PREDICT}$ cannot be solved in less than $f(n)$ steps, as well.

$\text{ATCH}_22$'s Running Time

We can solve $\text{ATCH}_22$ by running an interpreter on $\Pi'$'s source code for $f(|\Pi|)$ steps and seeing what happens. In order to ensure only at most $f(|\Pi|)$ steps occur, the interpreter takes $s(t)$ steps to run $t$ steps of $\Pi$.

By the previous proof, $s(t) > t$.

Assuming a random access machine, $s(t) = O(t)$.

So $\text{ATCH}_22$ can be solved in $s(f(n)) + O(f(n)) = O(s(f(n)))$ time.
Since \textsc{Catch22} is just a special case of \textsc{Predict} and takes one extra step to negate the result.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**PREDICT**

Since \texttt{CATCH22} is just a special case of \texttt{PREDICT} and takes one extra step to negate the result.

This means \texttt{PREDICT} can not be solved in less than $f(n)$ steps, as well.
What if P = NP?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**PREDICT**

Since CATCH22 is just a special case of PREDICT and takes one extra step to negate the result.

This means PREDICT can not be solved in less than $f(n)$ steps, as well.

**CATCH22’s Running Time**
**Diagonalization**

**PREDICT**

Since \( \text{CATCH22} \) is just a special case of \( \text{PREDICT} \) and takes one extra step to negate the result.

This means \( \text{PREDICT} \) can not be solved in less than \( f(n) \) steps, as well.

**CATCH22’s Running Time**

- We can solve \( \text{CATCH22} \) by running an interpreter on \( \Pi \)’s source code for \( f(|\Pi|) \) steps and seeing what happens.
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

**Diagonalization**

**PREDICT**

Since CATCH22 is just a special case of PREDICT and takes one extra step to negate the result.

This means PREDICT can not be solved in less than $f(n)$ steps, as well.

**CATCH22's Running Time**

- We can solve CATCH22 by running an interpreter on $\Pi$'s source code for $f(|\Pi|)$ steps and seeing what happens.
- In order to ensure only at most $f(|\Pi|)$ steps occur, the interpreter takes $s(t)$ steps to run $t$ steps of $\Pi$.  

**Time Hierarchy Theorem**
What if $P = NP$?

**Upper Bounds are Easy and Lower Bounds, Hard**

**Diagonalization and Time Hierarchy**

**Diagonalization**

**PREDICT**

Since $\text{CATCH22}$ is just a special case of $\text{PREDICT}$ and takes one extra step to negate the result.

This means $\text{PREDICT}$ can not be solved in less than $f(n)$ steps, as well.

**CATCH22’s Running Time**

- We can solve $\text{CATCH22}$ by running an interpreter on $\Pi$’s source code for $f(|\Pi|)$ steps and seeing what happens.
- In order to ensure only at most $f(|\Pi|)$ steps occur, the interpreter takes $s(t)$ steps to run $t$ steps of $\Pi$.
- By the previous proof, $s(t) > t$. 
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Diagonalization

**PREDICT**

Since **CATCH22** is just a special case of **PREDICT** and takes one extra step to negate the result.

This means **PREDICT** can not be solved in less than $f(n)$ steps, as well.

**CATCH22**'s Running Time

- We can solve **CATCH22** by running an interpreter on $\Pi$'s source code for $f(|\Pi|)$ steps and seeing what happens.
- In order to ensure only at most $f(|\Pi|)$ steps occur, the interpreter takes $s(t)$ steps to run $t$ steps of $\Pi$.
- By the previous proof, $s(t) > t$.
- Assuming a random access machine, $s(t) = O(t)$. 
Diagonalization

**PREDICT**

Since `CATCH22` is just a special case of `PREDICT` and takes one extra step to negate the result.

This means `PREDICT` can not be solved in less than \( f(n) \) steps, as well.

**CATCH22’s Running Time**

- We can solve `CATCH22` by running an interpreter on \( \Pi \)'s source code for \( f(|\Pi|) \) steps and seeing what happens.
- In order to ensure only at most \( f(|\Pi|) \) steps occur, the interpreter takes \( s(t) \) steps to run \( t \) steps of \( \Pi \).
- By the previous proof, \( s(t) > t \).
- Assuming a random access machine, \( s(t) = O(t) \).
- So `CATCH22` can be solved in \( s(f(n)) + O(f(n)) = O(s(f(n))) \) time.
What if $P = \text{NP}$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

**Time Hierarchy Theorem**

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps.

Then if $g(n) = o(f(n))$, $\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$.

**Proof**

Since $\text{ATCH}22$ can not be solved exactly $f(n)$ steps, it can not be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$.

This proves the Time Hierarchy Theorem.

**More Time Does Mean More Computation**

The Time Hierarchy Theorem proves:

- $P \subset \text{EXP} \subset \text{EXPEXP} \subset \cdots$
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Time Hierarchy Theorem

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps.

Proof

Since $\text{ATCH}^2$ cannot be solved exactly in $f(n)$ steps, it cannot be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$.

This proves the Time Hierarchy Theorem.

More Time Does Mean More Computation

The Time Hierarchy Theorem proves:

$$P \subset \text{EXP} \subset \text{EXPEXP} \subset \cdots$$
What if P = NP?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Time Hierarchy Theorem

Assume an interpreter can simulate \( t \) steps of an arbitrary program \( \Pi \) that runs in at most \( f(n) \) steps, while keeping track of the number of steps computed thus far in \( s(t) \) steps. Then if \( g(n) = o(f(n)) \),

\[
\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))
\]

This proves the Time Hierarchy Theorem.

More Time Does Mean More Computation

The Time Hierarchy Theorem proves:

\[ P \subset \text{EXP} \subset \text{EXP} \subset \cdots \]
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Time Hierarchy Theorem

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps. Then if $g(n) = o(f(n))$,

$$\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$$

Proof
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

### Time Hierarchy Theorem

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps. Then if $g(n) = o(f(n))$,

$$\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$$

### Proof

Since $\text{CATCH22}$ can not be solved exactly $f(n)$ steps, it can not be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$. 
What if $P = NP$?

Upper Bounds are Easy and Lower Bounds, Hard

Diagonalization and Time Hierarchy

Time Hierarchy Thereom

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps. Then if $g(n) = o(f(n))$,

$$\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$$

Proof

Since $\text{CATCH22}$ can not be solved exactly $f(n)$ steps, it can not be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$.

This proves the Time Hierarchy Thereom.
What if P = NP?
Upper Bounds are Easy and Lower Bounds, Hard
Diagonalization and Time Hierarchy

Time Hierarchy Theorem

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps. Then if $g(n) = o(f(n))$,

$$\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$$

Proof

Since $\text{CATCH22}$ can not be solved exactly $f(n)$ steps, it can not be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$.

This proves the Time Hierarchy Theorem.

More Time Does Mean More Computation
Time Hierarchy Theorem

Assume an interpreter can simulate $t$ steps of an arbitrary program $\Pi$ that runs in at most $f(n)$ steps, while keeping track of the number of steps computed thus far in $s(t)$ steps. Then if $g(n) = o(f(n))$,

$$\text{TIME}(g(n)) \subset \text{TIME}(s(f(n)))$$

Proof

Since $\text{CATCH22}$ can not be solved exactly $f(n)$ steps, it can not be solved in $O(g(n))$ steps for any $g(n) = o(f(n))$.

This proves the Time Hierarchy Thereom.

More Time Does Mean More Computation

The Time Hierarchy Thereom proves:

$$\text{P} \subset \text{EXP} \subset \text{EXP\text{EXP}} \subset \cdots$$