Lecture 16: Time Complexity and P vs NP
Time-Bounded Complexity Classes

Definition:
\( \text{TIME}(t(n)) = \{ L' \mid \text{there is a Turing machine } M \)
\( \text{with time complexity } O(t(n)) \text{ so that } L' = L(M) \} \)
\( = \{ L' \mid L' \text{ is a language decided by a Turing machine with } \leq c \ t(n) + c \text{ running time} \} \)

We showed: \( A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n \log n) \)

Puzzle: Show \( A \not\in \text{TIME}( (n \log n)/\log\log n) \)
An Efficient Universal TM

Theorem: There is a (one-tape) Turing machine $U$ which takes as input:
- the code of an arbitrary TM $M$
- an input string $w$
- and a string of $t$ 1s, $t > |w|$

such that $U(M, w, 1^t)$ halts in $O(|M|^2 t^2)$ steps
and $U$ accepts $(M, w, 1^t) \iff M$ accepts $w$ in $t$ steps

The Universal TM with a Clock

Idea: Make a multi-tape TM $U'$ that does the above, and runs in $O(|M| t)$ steps. Convert to one-tape TM.
The Time Hierarchy Theorem

Intuition: If you get more time to compute, then you can solve strictly more problems.

Theorem: For all “reasonable” \( f, g : \mathbb{N} \rightarrow \mathbb{N} \) where for all \( n, g(n) > n^2 f(n)^2 \), \( \text{TIME}(f(n)) \not\subseteq \text{TIME}(g(n)) \)

Proof Idea: Diagonalization with a clock
Make a TM \( N \) that on input \( M \), simulates the TM \( M \) on input \( M \) for \( f(|M|) \) steps, then flips the answer.

We will show \( L(N) \) cannot have time complexity \( f(n) \)
The Time Hierarchy Theorem

Theorem: For “reasonable” f, g where \(g(n) > n^2 f(n)^2\),
\[ \text{TIME}(f(n)) \subsetneq \text{TIME}(g(n)) \]

Proof Sketch: Define a TM N as follows:

\[ N(M) = \text{Compute } t = f(|M|) \]

Run \(U(M, M, 1^t)\) and output the opposite answer.

Claim: \(L(N)\) does not have time complexity \(f(n)\).

Proof: Assume \(N'\) runs in \(f(n)\) time, and \(L(N') = L(N)\).

By assumption, \(N'(N')\) runs in \(f(|N'|)\) time and outputs the \textit{opposite} answer of \(U(N', N', 1^{f(|N'|)})\)

So \(N'(N')\) accepts \(\iff U(N', N', 1^{f(|N'|)})\) rejects

\(\iff N'(N')\) rejects in \(f(|N'|)\) steps \[U\text{ is universal}\]

This is a contradiction!
The Time Hierarchy Theorem

Theorem: For “reasonable” $f, g$ where $g(n) > n^2 f(n)^2$, $\text{TIME}(f(n)) \not\subset \text{TIME}(g(n))$

Proof Sketch: Define a TM $N$ as follows:

$N(M) = \text{Compute } t = f(|M|)$

Run $U(M, M, 1^t)$ and output the opposite answer.

So, $L(N)$ does not have time complexity $f(n)$.

For what functions $g(n)$ will $N$ run in $O(g(n))$ time?

1. Compute $t = f(|M|)$ in $O(g(|M|))$ time [“reasonable”]
2. Run $U(M, M, 1^t)$ in $O(g(|M|))$ time

Recall: $U(M, w, 1^t)$ halts in $O(|M|^2 t^2)$ steps

So set $g(n)$ so that $g(|M|) > |M|^2 f(|M|)^2$ for all $n$. QED

Remark: Time hierarchy also holds for multitape TMs!
A Better Time Hierarchy Theorem

Theorem: For “reasonable” f, g where
\[ g(n) > f(n) \log^2 f(n), \quad \text{TIME}(f(n)) \subsetneq \text{TIME}(g(n)) \]

Corollary: \[ \text{TIME}(n) \subsetneq \text{TIME}(n^2) \subsetneq \text{TIME}(n^3) \subsetneq \ldots \]

There is an infinite hierarchy of increasingly more time-consuming problems

Question: Are there important everyday problems that are high up in this time hierarchy? A natural problem that needs exactly \( n^{10} \) time?

THIS IS AN OPEN QUESTION!
$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$

Polynomial Time

The analogue of “decidability” in the world of complexity theory
The EXTENDED Church-Turing Thesis

Everyone’s Intuitive Notion of Efficient Algorithms = Polynomial-Time Turing Machines

A controversial (dead?) thesis!

Counterexamples include $n^{100}$ time algorithms, randomized algorithms, quantum algorithms, ...
Nondeterminism and NP
Nondeterministic Turing Machines

...are just like standard TMs, except:

1. The machine may proceed according to several possible transitions (like an NFA)

2. The machine accepts an input string if there exists an accepting computation history for the machine on the string
Definition: A nondeterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
Defining Acceptance for NTMs

Let N be a nondeterministic Turing machine

An accepting computation history for N on w is a sequence of configurations \( C_0, C_1, \ldots, C_t \) where

1. \( C_0 \) is the start configuration \( q_0 w \),
2. \( C_t \) is an accepting configuration,
3. Each configuration \( C_i \) yields \( C_{i+1} \)

Def. \( N(w) \) accepts in \( t \) time \( \iff \) Such a history exists

\( N \) has time complexity \( T(n) \) if for all \( n \), for all inputs of length \( n \) and for all histories, \( N \) halts in \( T(n) \) time
Definition: $\text{NTIME}(t(n)) =$

$\{ L \mid \text{L is decided by a } O(t(n)) \text{ time nondeterministic Turing machine} \}$

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

Is $\text{TIME}(t(n)) = \text{NTIME}(t(n))$ for all $t(n)$?

THIS IS AN OPEN QUESTION!

What can be done in “short” NTIME that cannot be done in “short” TIME?
Boolean Formulas

A satisfying assignment is a setting of the variables that makes the formula true.

\[ \phi = (\neg x \land y) \lor z \]

\( x = 1, \ y = 1, \ z = 1 \) is a satisfying assignment for \( \phi \).

Boolean variables (0 or 1)

\[ \neg (x \lor y) \land (z \land \neg x) \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

logical operations
parentheses
A Boolean formula is satisfiable if there exists a true/false setting to the variables that makes the formula true

\[
\text{YES } \quad a \land b \land c \land \neg d
\]

\[
\text{NO } \quad \neg(x \lor y) \land x
\]

\[
\text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \}
\]

(How are we encoding formulas? In a “reasonable” way!)

Encoding: takes formula \( \phi \) of \( n \) symbols, and outputs \( O(n^c) \) bits

Decoding: takes \( O(n^c) \) bits and \( i \), and outputs \( i \)-th symbol of \( \phi \)
A 3cnf-formula has the form:

$$(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)$$

Ex: $$(x_1 \lor \neg x_2 \lor x_1)$$

$$\quad (x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)$$

$$\quad (x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)$$

$$\quad (x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)$$

3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}
Theorem: $3SAT \in NTIME(n^c)$ for some constant $c > 1$

Proof Idea: On input $\phi$:

1. Check if the formula is in 3cnf

2. For each variable $v$ in $\phi$, nondeterministically substitute either 0 or 1 in place of $v$

3. Evaluate the formula and accept iff $\phi$ is true
\[ \text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k) \]

**Nondeterministic Polynomial Time**

The analogue of “recognizability”
Theorem: \( L \in \text{NP} \iff \) There is a constant \( k \) and

polynomial-time TM \( V \) such that

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

Proof: (1) If \( L = \{ x \mid \exists y \ |y| \leq |x|^k \text{ and } V(x,y) \text{ accepts } \} \)

then \( L \in \text{NP} \)

Given the poly-time TM \( V \), our NP machine for \( L \) is:

\( N(x): \) Nondeterministically guess \( y \). Run \( V(x,y) \)

(2) If \( L \in \text{NP} \) then

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

Let \( N \) be a nondet. poly-time TM that decides \( L \).

Define a TM \( V(x,y) \) which accepts

\( \iff y \) encodes an accepting computation history of \( N \) on \( x \)
Moral: A language L is in NP if and only if there are polynomial-length proofs for membership in L

\[
3\text{SAT} = \{ \phi \mid \exists y \text{ such that } \phi \text{ is in } 3\text{cnf and } y \text{ is a satisfying assignment to } \phi \}
\]

\[
\text{SAT} = \{ \phi \mid \exists y \text{ such that } \phi \text{ is a Boolean formula and } y \text{ is a satisfying assignment to } \phi \}
\]
NP = Problems with the property that, once you have the answer, it is “easy” to verify the answer.

SAT is in NP because a satisfying assignment is a polynomial-length proof that a formula is satisfiable.

When $\phi \in \text{SAT}$, I can prove that fact to you with a short proof you can quickly verify.
The Hamiltonian Path Problem

A Hamiltonian path traverses through each node exactly once
Assume a reasonable encoding of graphs (example: the adjacency matrix is reasonable)

HAMPATH = \{ (G,s,t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} 

Theorem: HAMPATH \in \text{NP}

A Hamiltonian path P in G from s to t is a proof that (G,s,t) is in HAMPATH

Given P (as a permutation on the nodes) can easily check that it is a path through all nodes exactly once
The k-Clique Problem

k-clique = complete subgraph on k nodes
CLIQUE = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-clique} \} 

Theorem: CLIQUE ∈ NP

A k-clique in G is a proof that \( (G, k) \) is in CLIQUE

Given a subset S of k nodes from G, we can efficiently check that all possible edges are present between the nodes in S
A language is in NP if and only if there are “polynomial-length proofs” for membership in the language

P = the problems that can be *efficiently solved*

NP = the problems where *proposed solutions can be efficiently verified*

**Is P = NP?**

*can problem solving be automated?*
If $P = NP$...

Mathematicians/creators may be out of a job

Cryptography as we know it may be impossible – no “one-way” functions!

In principle, every aspect of daily life could be efficiently and globally optimized...
... life as we know it would be different

Conjecture: $P \neq NP$
Is SAT solvable in $O(n)$ time on a multitape TM?

Logic circuits of $6n$ gates for SAT?

If yes, then not only is $P=NP$, but there would be a “dream machine” that could crank out short proofs of theorems, quickly optimize all aspects of life... recognizing quality work is all you need to produce

THIS IS AN OPEN QUESTION!
Polynomial Time Reducibility

$f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function
if there is a poly-time Turing machine $M$ that on
every input $w$, halts with just $f(w)$ on its tape

Language $A$ is poly-time reducible to language $B$,
written as $A \leq_p B$,
if there is a poly-time computable $f : \Sigma^* \rightarrow \Sigma^*$ so that:

$$w \in A \iff f(w) \in B$$

$f$ is a polynomial time reduction from $A$ to $B$

Note there is a $k$ such that for all $w$, $|f(w)| \leq k|w|^k$
f converts any string w into a string f(w) such that
\[ w \in A \iff f(w) \in B \]
Theorem: If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

Proof: Let $M_B$ be a poly-time TM that decides $B$. Let $f$ be a poly-time reduction from $A$ to $B$.

We build a machine $M_A$ that decides $A$ as follows:

$M_A = \text{On input } w,$

1. Compute $f(w)$

2. Run $M_B$ on $f(w)$, output its answer

$$w \in A \iff f(w) \in B$$
Theorem: If $A \leq_p B$ and $B \in NP$, then $A \in NP$

Proof: Analogous...
Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

Theorem: If $A \leq_p B$ and $B \in NP$, then $A \in NP$

Corollary: If $A \leq_p B$ and $A \notin P$, then $B \notin P$
Definition: A language $B$ is NP-complete if:

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

On homework, you showed
A language $L$ is recognizable iff $L \leq_m A_{\text{TM}}$

$A_{\text{TM}}$ is “complete for recognizable languages”: $A_{\text{TM}}$ is recognizable, and for all recognizable $L$, $L \leq_m A_{\text{TM}}$
Suppose $L$ is NP-Complete...

If $L \in P$, then $P = NP$!

If $L \notin P$, then $P \neq NP$!
Suppose $L$ is NP-Complete...

Then assuming the conjecture $P \neq NP$,

$L$ is not decidable in $n^k$ time, for every $k$.
There are thousands of NP-complete problems!

Your favorite topic certainly has an NP-complete problem somewhere in it

Even the other sciences are not safe: biology, chemistry, physics have NP-complete problems too!
The Cook-Levin Theorem: SAT and 3SAT are NP-complete

1. 3SAT ∈ NP
   A satisfying assignment is a “proof” that a 3cnf formula is satisfiable

2. 3SAT is NP-hard
   Every language in NP can be polynomial-time reduced to 3SAT (complex logical formula)

Corollary: 3SAT ∈ P if and only if P = NP
Theorem (Cook-Levin): 3SAT is NP-complete

Proof Idea:

(1) $3\text{SAT} \in \text{NP}$ (done)

(2) Every language $A$ in NP is polynomial time reducible to $3\text{SAT}$ (this is the challenge)

We give a poly-time reduction from $A$ to SAT

The reduction converts a string $w$ into a 3cnf formula $\phi$ such that $w \in A$ iff $\phi \in 3\text{SAT}$

For any $A \in \text{NP}$, let $N$ be a nondeterministic TM deciding $A$ in $n^k$ time

$\phi$ will simulate $N$ on $w$
Deterministic Computation

accept or reject

Nondeterministic Computation

accept

\[ n^k \]

\[ \exp(n^k) \]
Let $L(N) \in \text{NTIME}(n^k)$. A tableau for $N$ on $w$ is an $n^k \times n^k$ matrix whose rows are the configurations of some possible computation history of $N$ on $w$.

<table>
<thead>
<tr>
<th></th>
<th>$q_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>...</th>
<th>$w_n$</th>
<th>$\square$</th>
<th>...</th>
<th>$\square$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
</tbody>
</table>

Each “cell” contains an $\sigma \in Q \cup \Gamma \cup \{\#\}$
A tableau is accepting if the last row of the tableau is an accepting configuration

N accepts w if and only if there is an accepting tableau for N on w

Given w, we’ll construct a 3cnf formula $\phi$ with $O(|w|^{2k})$ clauses, describing logical constraints that any accepting tableau for N on w must satisfy

The 3cnf formula $\phi$ will be satisfiable if and only if there is an accepting tableau for N on w
Variables of formula $\phi$ will encode a tableau

Let $C = Q \cup \Gamma \cup \{ \# \}$

Each cell of a tableau contains a symbol from $C$

$cell[i,j] = \text{symbol in the cell at row } i \text{ and column } j$

$= \text{the } j\text{th symbol in the } i\text{th configuration}$

For every $i$ and $j$ ($1 \leq i, j \leq n^k$) and for every $s \in C$
we make a Boolean variable $x_{i,j,s}$ in $\phi$

Total number of variables $= |C|n^{2k}$, which is $O(n^{2k})$

The $x_{i,j,s}$ variables represent the cells of a tableau

We will enforce the condition: for all $i, j, s$,

$x_{i,j,s} = 1 \iff cell[i,j] = s$
Idea: Make \( \phi \) so that every *satisfying assignment* to the variables \( x_{i,j,s} \) corresponds to an *accepting tableau* for \( N \) on \( w \) (an assignment to all cell\([i,j]\)’s of the tableau)

The formula \( \phi \) will be the AND of four CNF formulas:

\[
\phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}}
\]

\( \phi_{\text{cell}} \) : for all \( i, j \), there is a *unique* \( s \in C \) with \( x_{i,j,s} = 1 \)

\( \phi_{\text{start}} \) : the first row of the table equals the *start* configuration of \( N \) on \( w \)

\( \phi_{\text{accept}} \) : the last row of the table has an accept state

\( \phi_{\text{move}} \) : every row is a configuration that yields the configuration on the next row
\( \phi_{\text{start}} : \) the first row of the table equals the \textit{start} configuration of \( N \) on \( w \)

\[
\phi_{\text{start}} = x_{1,1,\#} \land x_{1,2,q_0} \land x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land x_{1,n+3,\Box} \land \ldots \land x_{1,n^k-1,\Box} \land x_{1,n^k,\#}
\]

<table>
<thead>
<tr>
<th>#</th>
<th>q_0</th>
<th>w_1</th>
<th>w_2</th>
<th>\ldots</th>
<th>w_n</th>
<th>\Box</th>
<th>\ldots</th>
<th>\Box</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
</tbody>
</table>

\( O(n^k) \) clauses
\( \phi_{\text{accept}} : \) the last row of the table has an accept state

\[
\phi_{\text{accept}} = \bigvee_{1 \leq j \leq n^k} x_{n^k,j, \text{q}_{\text{accept}}}
\]

<table>
<thead>
<tr>
<th></th>
<th>q₀</th>
<th>w₁</th>
<th>w₂</th>
<th>...</th>
<th>wₙ</th>
<th>⬠</th>
<th>...</th>
<th>⬠</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[q₀ accept]
\( \phi_{\text{accept}} : \text{the last row of the table has an accept state} \)

\[
\phi_{\text{accept}} = \bigvee_{1 \leq j \leq n^k} x_{n^k,j, q_{\text{accept}}}
\]

How can we convert \( \phi_{\text{accept}} \) into a 3-cnf formula?

The clause \((a_1 \lor a_2 \lor \ldots \lor a_t)\) is equivalent to

\((a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land \ldots \land (\neg z_{t-3} \lor a_{t-1} \lor a_t)\)

where \(z_i\) are new variables.

This produces \(O(t)\) new 3cnf clauses.

\(O(n^k)\) clauses
\( \phi_{\text{cell}} : \) for all \( i, j \), there is a unique \( s \in C \) with \( x_{i,j,s} = 1 \)

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right) \right]
\]

for all \( i, j \)

- at least one \( x_{i,j,s} \) is set to 1
- at most one \( x_{i,j,s} \) is set to 1

\( O(n^{2k}) \) clauses
ϕ_{move} : every row is a configuration that yields the configuration on the next row

Key Question: If one row yields the next row, how many cells can be different between the two rows?

Answer: AT MOST THREE CELLS!
\( \phi_{\text{move}} \): every row is a configuration that yields the configuration on the next row

Key Question: If one row yields the next row, how many cells can be different between the two rows?

Answer: AT MOST THREE CELLS!
$\phi_{\text{move}}$: every row is a configuration that yields the configuration on the next row

Idea: check that every $2 \times 3$ “window” of cells is legal (consistent with the transition function of N)
\[ \phi_{\text{move}} = \bigwedge (\text{the } (i, j) \text{ window is "legal"}) \]

\[ 1 \leq i, j \leq n^k \]

the \((i, j)\) window is "legal" =

\[ \equiv \bigwedge (\neg x_{i,j,a_1} \lor x_{i,j+1,a_2} \lor x_{i,j+2,a_3} \lor x_{i+1,j,a_4} \lor x_{i+1,j+1,a_5} \lor x_{i+1,j+2,a_6}) \]

ISN'T "legal"

\[ O(n^{2k}) \] clauses