6.045

Finish K-Complexity,
Start Time Complexity
Kolmogorov Complexity

Definition: The *shortest description of* $x$, denoted as $d(x)$, is the lexicographically shortest string $<M,w>$ such that $M(w)$ halts with only $x$ on its tape.

Definition: The *Kolmogorov complexity of* $x$, denoted as $K(x)$, is $|d(x)|$. 
There Exist Incompressible Strings

**Theorem:** For all $n$, there is an $x \in \{0,1\}^n$ such that $K(x) \geq n$

"There are incompressible strings of every length"

**Proof:**

(Number of binary strings of length $n) = 2^n$

but

(Number of descriptions of length $< n) \leq (Number of binary strings of length $< n)\)

$= 1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1$

Therefore, there is at least one $n$-bit string $x$ that does not have a description of length $< n$. 
Random Strings Are Incompressible!

Theorem: For all $n$ and $c \geq 1$,
\[
\Pr_{x \in \{0,1\}^n}[ K(x) \geq n-c ] \geq 1 - 1/2^c
\]

"Most strings are highly incompressible"

Proof: (Number of binary strings of length $n$) = $2^n$
but (Number of descriptions of length < $n-c$)
\leq (Number of binary strings of length < $n-c$)
= $2^{n-c} - 1$

Hence the probability that a random $x$ satisfies $K(x) < n-c$
is at most $(2^{n-c} - 1)/2^n < 1/2^c$. 
Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. 01000110110000010100111001011101110000

2. 123581321345589144233377610987

3. 126241207205040403203628803628800
Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. 01000110110000010100111001011101110000

2. 1235813213455891442333777610987

3. 126241207205040403203628803628800
Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. 01000110110000010100111001011101110000

2. 123581321345589144233377610987

3. 126241207205040403203628803628800
Kolmogorov Complexity: Try it!

Give short algorithms for generating the strings:

1. 01000110110000010100111001011101110000

2. 123581321345589144233377610987

3. 126241207205040403203628803628800

This seems hard to determine in general. Why?
Computing Compressibility?

Can an algorithm perform optimal compression?
Can algorithms tell us if a given string is compressible?

COMPRESS = \{ (x,c) \mid K(x) \leq c \}

**Theorem:** COMPRESS is undecidable!

**Idea:** If decidable, we could design an algorithm that prints the shortest incompressible string of length \( n \)

*But such a string could then be succinctly described, by providing the algorithm code and \( n \) in binary!*

Berry Paradox: “The smallest integer that cannot be defined in less than thirteen words.”
Computing Compressibility?

\[ \text{COMPRESS} = \{ (x,c) \mid K(x) \leq c \} \]

**Theorem:** COMPRESS is undecidable!

**Proof:** Suppose it is decidable. Consider the TM:

\[ M = \text{"On input } x \in \{0,1\}^*, \text{ let } N = 2^{|x|}. \]

For all \( y \in \{0,1\}^* \) in lexicographical order,

If \( (y, N) \notin \text{COMPRESS} \) then print \( y \) and halt.

\( M(x) \) prints the lex. first string \( y' \) with \( K(y') > 2^{|x|} \).

\( <M,x> \) is a description of \( y' \), \( |<M,x>| \leq (2|M|+1) + |x| \)

So \( 2^{|x|} < K(y') \leq c + |x| \). CONTRADICTION for large \( x \)!
Proving Theorems With K-Complexity

**Theorem:** \( L = \{x \ x \ | \ x \in \{0, 1\}^*\} \) is not regular.

**Proof:** Suppose \( L \) is recognized by a DFA \( D \).

Let \( n \geq 0 \) and choose an \( x \in \{0, 1\}^n \) such that \( K(x) \geq n \)

Let \( q_x \) be the state of \( D \) reached after reading in \( x \)

Define a TM \( M(D, q, n) \):

- Find a path \( P \) in DFA \( D \) of length \( n \) that starts from state \( q \) and ends in a final state.
- Print the \( n \)-bit string along path \( P \), and halt.

**Claim:** The pair \( <M, (D, q_x, n)> \) is a description of \( x \)!

So \( n \leq K(x) \leq |<M, (D, q_x, n)>| \leq c + (\log n) \)

**CONTRADICTION** for large \( n \)!
Computational Complexity Theory
Computational Complexity Theory

What can and can’t be computed with limited resources on computation, such as time, space, and so on

Captures many of the significant issues in practical problem solving

The field is rich with important open questions that no one has any idea how to begin answering!

We’ll start with: Time complexity
Let $f, g : \mathbb{N} \to \mathbb{N}$.

We say that $f(n) \leq O(g(n))$ if there are $c, n_0 \in \mathbb{N}$ so that for every integer $n \geq n_0$

$$f(n) \leq c \cdot g(n)$$

We say $g(n)$ is an upper bound on $f(n)$ if

$$f(n) \leq O(g(n))$$

$$5n^3 + 2n^2 + 22n + 6 \leq O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$
\[ 2n^{4.1} + 200283n^4 + 2 \leq O(n^{4.1}) \]

\[ 3n \log_2 n + 5n \log_2 \log_2 n \leq O(n \log_2 n) \]

\[ n \log_{10} n^{78} \leq O(n \log_{10} n) \]

\[ \log_{10} n = \log_2 n / \log_2 10 \]

\[ O(n \log_2 n) \leq O(n \log_{10} n) \leq O(n \log n) \]

Big-O isolates the “dominant” term of a function
A Simpler Big-O Definition

Let $f, g : \mathbb{N} \rightarrow \mathbb{N}$ We say $f(n) \leq O(g(n))$ if there is a $c \in \mathbb{N}$ so that for all $n \in \mathbb{N}$, $f(n) \leq c \ g(n) + c$
Measuring Time Complexity of a TM

We measure time complexity by counting the steps taken for a Turing machine to halt on an input.

Example: Let $A = \{ 0^k1^k \mid k \geq 0 \}$

Here’s a TM for $A$. On input $x$ of length $n$:

1. Scan across the tape and **reject**
   - $O(n)$
   - if $x$ is not of the form $0^i1^j$

2. Repeat the following if both 0s and 1s remain on the tape:
   - $O(n^2)$
   - Scan across the tape, crossing off a single 0 and a single 1

3. If 0s remain after all 1s have been crossed off, or vice-versa, **reject**. Otherwise **accept**.
Let $M$ be a TM that halts on all inputs. 

*(We will only consider decidable languages now!)*

**Definition:**

The **running time or time complexity of $M$** is the function $T : \mathbb{N} \rightarrow \mathbb{N}$ such that

$$T(n) = \text{maximum number of steps taken by } M \text{ over all inputs of length } n$$
Time-Bounded Complexity Classes

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

We showed: \( A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2) \)

Is there a faster Turing machine?
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n \log n)

M(w) := \text{If } w \text{ is not of the form } 0^*1^*, \text{ reject.}

Repeat until all bits of w are crossed out:
  If (parity of 0’s) \neq (parity of 1’s), \text{ reject.}
  Cross out every other 0. Cross out every other 1.
Once all bits are crossed out, \text{ accept.}

00000000000001111111111111
x0x0x0x0x0x0xx11x11x11x11x11x
xxx0xxx0xxx0xxxx1xxxx1xxxx1xxx
xxxxxxx0xxxxxxx1xxxxxxx1xxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME(n log n)}

M(w) := If w is not of the form 0^*1^*, reject.
Repeat until all bits of w are crossed out:
  If (parity of 0’s) \neq (parity of 1’s), reject.
  Cross out every other 0. Cross out every other 1.
Once all bits are crossed out, accept.

For a fixed w = 0^k1^k:
Let \( zero_i \) be number of 0s left in w, after iteration \( i \)
Let \( ones_i \) be number of 1s left in w, after iteration \( i \)
  Start with \( zero_0 = k, ones_0 = k \)
  Key Observation:
\[ zero_{i+1} = \text{floor}(zero_i/2), \quad ones_{i+1} = \text{floor}(ones_i/2) \]
Number of iterations \( \leq O(\log n) \)
It can be proved that there is no (one-tape) Turing Machine that can decide A in less than $O(n \log n)$ time!

**Puzzle:**

Let $f(n) = O \left( \frac{n \log n}{\alpha(n)} \right)$ where $\alpha(n)$ is unbounded.

Prove: $\text{TIME}(f(n))$ contains only regular languages(!)

For example, $\text{TIME}(n \log \log n)$ contains only regular languages!
In particular, define the class
\[ \text{REGULAR} = \{ L \mid L \text{ is a regular language} \}. \]

**Proposition:** \( \text{REGULAR} \subseteq \text{TIME}(n) \)

Assuming the Puzzle is true, then for every \( f(n) = O \left( \frac{n \log n}{\alpha(n)} \right) \) where \( \alpha(n) \) is unbounded,

\[ \text{TIME}(f(n)) = \text{TIME}(n) \]

A “collapse” of complexity classes!
Two Tapes Can Be More Efficient

Theorem: \( A = \{ 0^k1^k \mid k \geq 0 \} \) can be decided in \( O(n) \) time with a *two-tape TM*.

Proof Idea:

Sweep over all 0s, copy them over on the second tape.
Sweep over all 1s. For each 1, cross off a 0 from the second tape.
Different models of computation can yield different running times for the same language!

Let’s revisit some of the key concepts from computability theory...
**Theorem:** Let $t : \mathbb{N} \to \mathbb{N}$ satisfy $t(n) \geq n$, for all $n$. Then every $t(n)$ time multi-tape TM has an equivalent $O(t(n)^2)$ time one-tape TM.

Our simulation of multitape TMs by one-tape TMs achieves this!

**Corollary:** Suppose language $A$ can be decided by a multi-tape TM in $p(n)$ time, for some polynomial $p$. Then $A$ can also be decided by a one-tape TM in $q(n)$ time, for some polynomial $q$. 

Theorem: For every $t(n)$ time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM

Finite State Control
Theorem: For every $t(n)$ time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM.
Theorem: For every $t(n)$ time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM
Theorem: For every $t(n)$ time multi-tape TM, there is an equivalent $O(t(n)^2)$ time one-tape TM.
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 $O(|M| \cdot t)$ steps
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 \( n \), \( g(n) > n^2 f(n)^2 \), \( \text{TIME}(f(n)) \varsubsetneq \text{TIME}(g(n)) \)

**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, \textit{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 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 \( g(n) > n^2 f(n)^2 \),
\[ \textbf{TIME}(f(n)) \not\subset \textbf{TIME}(g(n)) \]

**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. Simulate \( U(M, M, 1^t) \) in \( O(g(|M|)) \) time

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

Set \( g(n) \) so that \( g(|M|) > |M|^2 f(|M|)^2 \) for all \( n \). \( \textbf{QED} \)

**Remark:** Time hierarchy also holds for multitape TMs!
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?

**A Better Time Hierarchy Theorem**

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

This is an open question!
$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$

Polynomial Time

The analogue of “decidability”
The EXTENDED Church-Turing Thesis

Everyone’s Intuitive Notion of Efficient Algorithms

= Polynomial-Time Turing Machines

A controversial thesis! Polynomial algorithms include $n^{100}$ time algorithms, randomized algorithms, quantum algorithms, and more.
Nondeterminism and NP

The analogue of “recognizability”
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
read  write  move

$q_\text{accept}$

$q_\text{reject}$

$0 \rightarrow 0, R$

$\square \rightarrow \square, R$

$\square \rightarrow \square, R$

$0 \rightarrow 0, R$

$0 \rightarrow 0, R$
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 $q_0w$,
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: $\text{NTIME}(t(n)) = \{ L | \text{L is decided by a } O(t(n)) \text{ time nondeterministic Turing machine} \}$

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

- **Logical operations**
  - \( \neg x \land y \lor z \)
- **Parentheses**
  - \( \phi = (\neg x \land y) \lor z \)
- **A satisfying assignment** is a setting of the variables that makes the formula true:
  - \( x = 1, y = 1, z = 1 \) is a satisfying assignment for \( \phi \)

**Boolean variables (0 or 1)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg(x \lor y) \land (z \land \neg x) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Boolean formula is **satisfiable** if there exists a true/false setting to the variables that makes the formula true

**YES** \[ a \land b \land c \land \neg d \]

**NO** \[ \neg(x \lor y) \land x \]

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

\[(x_1 \lor \neg x_2 \lor x_1)\]

\[(x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)\]

\[(x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)\]

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)\]

3SAT = \{ \phi | \phi \text{ is a satisfiable 3cnf-formula} \}
3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}

**Theorem:** \(3SAT \in \text{NTIME}(n^2)\)

**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

\[
\begin{align*}
(\overline{x} \lor y \lor x) & \\
(0 \lor \overline{y} \lor 0) & \quad (1 \lor \overline{y} \lor 1) \\
(0 \lor \overline{0} \lor 0) & \quad (0 \lor \overline{1} \lor 0)
\end{align*}
\]
NP = $\bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

Nondeterministic Polynomial Time
Theorem: \( L \in NP \iff \) There is a constant \( k \) and polynomial-time TM \( V \) such that

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

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

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

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

(2) If \( L \in NP \) then

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

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 \textbf{NP} if and only if there are polynomial-length proofs for membership in L.

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

\[ \text{SAT} = \{ \phi \mid \exists y \text{ such that } \phi \text{ is a Boolean formula and } y \text{ is a satisfying assignment to } \phi \} \]
**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
A Hamiltonian path traverses through each node exactly once.
Assume a reasonable encoding of graphs (example: the adjacency matrix is reasonable)

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

**Theorem:** \( \text{HAMPATH} \in \text{NP} \)

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 $\in$ 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.

\[ P = \text{the problems that can be efficiently solved} \]

\[ NP = \text{the problems where proposed solutions can be efficiently verified} \]

Is \( P = NP \)?

Can problem solving be automated?
P = NP?
If $P = NP$...

Mathematicians/creators may be out of a job

Cryptography as we know it may be impossible

In principle, every aspect of daily life could be efficiently and globally optimized...

... life as we know it would be different

Conjecture: $P \neq NP$
Next Episode:

NP-Complete Problems