Recap: Time Complexity

Time complexity class \( \text{TIME}(t(n)) \) = all languages that are decidable by an \( O(t(n)) \) (deterministic, single-tape) Turing machine.

A \( t(n) \) multitape TM has an equivalent \( O(t^2(n)) \) single-tape TM.

\[
\text{multi-tape polynomial} \implies \text{single-tape polynomial}
\]

Every \( t(n) \) nondeterministic TM has an equivalent \( 2^{O(t(n))} \) deterministic single-tape TM.

\[
\text{nondeterministic polynomial} \implies \text{single-tape exponential}
\]

general-case construction, could be better in some cases

Checking vs. Proving vs. Disproving

**Hamiltonian Path**: directed path containing each node once
Decidable \( \implies \) complement is decidable
  - Finding: hard (NP-complete, will revisit)
  - Checking: easy (traverse path, mark all graph nodes)
    Complexity: polynomial \( O(n^2) \)
  - Disproving: hard (try all paths)
  - Checking witness easier than finding
  - Complement may not have witness

Polynomial Verifiers

**Def.** A verifier for a language \( A \) is an algorithm \( V \), where
\[
A = \{ w | V \text{ accepts } (w, c) \text{ for some string } c \}
\]
A polynomial-time verifier runs polynomial in the length of \( v \).
A language is polynomially verifiable if it has a polynomial time verifier.

Why no specified constraint on the certificate \( c \)?
If verifier is polynomial, certificate must be polynomial too, otherwise no time to read the entire certificate!

The Class NP

**Def.** \( \text{NP} \) is the class of languages that have polynomial-time verifiers.

For \( \text{HAMPATH} \), the certificate can be simply the path.

\[
\text{COMPOSITES} = \{ x \mid x = pq, \text{ for integers } p, q, > 1 \}
\]
Certificate \( c \): a divisor of \( x \). \( \implies \text{COMPOSITES} \) is in \( \text{NP} \).

Certificate for \( \text{PRIMES} \): ??? (negation is asymmetric)
We now know \( \text{PRIMES} \) is in \( P \) (Agrawal-Kayal-Saxena 2002) \( \implies \text{COMPOSITES} \) is also in \( P \)

NP = Nondeterministic Polynomial Time

**Theorem:** A language is in \( \text{NP} \) iff it is decided by some nondeterministic polynomial time Turing machine

\[
\text{polynomial verifier} \iff \text{polynomial-time NTM}
\]

\( \implies \) Assume verifier \( V \) running in time \( n^k \).

NTM \( N \) is:
1. Nondeterministically choose string \( c \) of length \( < n^k \)
   \( (n^k \text{ steps, } \Sigma \text{ branches at each}) \)
2. Run \( V \) on input \((w, c)\) (poly-time)
3. Accept/reject based on \( V \)

\( \Leftarrow \) Construct verifier that treats each symbol of \( c \) as a description of the nondeterministic choice to make by \( N \).
1. Simulate \( N \) on \( w \) according to choices \( c \)
2. If this branch of \( N \) accepts, accept, else reject
### The Complexity Class \( \text{NTIME} \)

**Def.** \( \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine } \} \)

\[
\text{NP} = \bigcup_k \text{NTIME}(n^k)
\]

### The class \( \text{co-NP} \)

For **CLIQUE** and **SUBSET-SUM**, no obvious way to quickly verify a NO answer. Can’t say that the **complements** of these sets are in NP. (No obvious certificate for complement of these problems)

**co-NP**: languages whose complements are in NP.

\( \text{co-NP} \neq \text{NP} \text{ ??} \text{ – don’t know} \)

**Examples:**
- Is a formula valid? counterexample: falsifying assignment
  - this is actually \( \text{SAT} \)
- PRIMES is in \( \text{co-NP} \): counterexample: proper divisor
  - we know it’s also in \( \text{P} \)

### Showing that a problem is in \( \text{NP} \)

- **Construct a poly-time verifier** (certificate \( c \) is usually the solution)
- **Have a NTM nondeterministically generate \( c \) and then check it**

**CLIQUE**

A clique of an undirected graph is a subgraph with all nodes connected.

Does graph \( G \) have a complete subgraph with \( k \) nodes?

Certificate: set of nodes that should form clique.

**SUBSET-SUM**

Given a collection (multiset) of integers, and a target integer \( t \), is there a subcollection of numbers adding up to \( t \)?

Certificate: the subset which should add up to \( t \)

### \( \text{P} \) versus \( \text{NP} \)

Recall: decide vs. verify

\( P = \text{can decide membership in polynomial time} \)

\( \text{NP} = \text{can verify membership in polynomial time (given certificate)} \)

Trivially, \( P \subseteq \text{NP} \). We may have \( P = \text{NP} \) or \( P \subset \text{NP} \)

We’ve seen poly-time \( \text{NTM} \Rightarrow \text{exponential-time DTM} \). Thus

\[
\text{NP} \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k})
\]

We don’t know if there is a stronger deterministic-time bound.

### Polynomial-time Reductions

**Def.** A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a **polynomial time computable function** if some polynomial time Turing machine exists that when started with any \( w \), halts with just \( f(w) \) on the tape.

**Def.** Language \( A \) is **polynomial-time (mapping) reducible** to language \( B \) (\( A \lesssim_p B \)) if a polynomial-time computable function \( f : \Sigma^* \rightarrow \Sigma^* \) exists, where for all \( w \),

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

- Reduction (function) goes one-way (construct B-problem from A)
- Equivalence proof goes both ways
  - YES maps to YES not enough
  - also need NO mapped to NO

![Polynomial-time Reductions Diagram](image-url)