We know:

\[ \text{L} \neq \text{NL} \neq \text{P} \neq \text{NP} \neq \text{PSPACE} \neq \text{EXP} \neq \text{NEXP} \]

Today we prove these separations:

The method we will prove is based on diagonalization, where we designed a new machine that did the opposite of the i-th machine \( M_i \) on input \( \langle M_i \rangle \).

We will do something along the same lines for listing all space-bounded (or time-bounded) TMs.

The construction is similar in the two cases but easier for space.

The general idea is that a bit more space will let \( M_1 \) do more, but that only works for "nice" space bounds.
**Space Hierarchy Theorem**

**Def:** A function \( f: \mathbb{N} \rightarrow \mathbb{N} \) is space constructible if

\[ f(n) > \log(n) + \log \log(n) \text{ and} \]

the map \( 1^n \rightarrow \) binary representation of \( \langle f(n), < f(n) \rangle \) is computable by an \( O(f(n)) \)-space TM.

The general idea of a list \( L \) is that for a space constructible \( f(n) \) and any \( \langle M \rangle \) we can simulate \( M \) on input \( x \) using space \( O(f(n)) \) s.t.

The simulation does what \( M \) does if

- \( M \) doesn't use more than \( f(1x) \) storage
- \( M \) doesn't run for more than \( 2^f(1x) \) steps (which implies that \( M \) doesn't run forever)

"On input \( \langle M \rangle \) and \( x \):

- Use space constructibility of \( f \) to compute the binary string \( < f(1x) > \) on the work tape
  (pretend each symbol of \( x \) is a 1)
- Mark off \( f(1x) \) cells on a separate sector of the work tape
- Create a counter \( 2^{f(1x)} \) using another \( f(1x) \) cells.
- Simulate \( M \) on input \( x \) keeping track of the \( n \) of steps
  - Subtract 1 from counter each step
  - Stop simulation if \( A \) moves off the marked cells & reject
  - Stop when counter reaches 0 & reject"

Using this idea we prove
Theorem: If \( f(n) \) is space constructible, then there is a language \( A \) decidable using space \( O(f(n)) \) but not \( o(f(n)) \).

Proof: Define \( A \) as the language decided by the following TM for a "diagonal language".

On input \( x \):
1. Use space constructibility of \( f \) to compute \( f(<M|x|>) \) on the work tape.
2. Mark off \( f(<M|>) \) cells on the work tape.
3. If \( x \) is not of the form \( <M|O_1^n> \), then reject.
4. Simulate \( M \) on input \( x \) and count steps.
   - If more than \( 2f(<M|>) \) steps, \( M \) stops and accept.
   - If more than \( f(<M|>) \) cells are used and \( steps \leq f(<M|>) \), then accept.
5. If \( M \) accepts, then reject.
   - If \( M \) rejects, then accept.

Claim: \( A \) is different from every language decided using space \( o(f(n)) \).

Suppose not. Then \( A = L(M_i) \) for some \( M_i \) that uses space \( g(n) = o(f(n)) \).

Consider whether \( A \) includes \( <M_i> \):
- If \( M_i \) runs on input \( <M_i> \), using \( \leq f(<M_i|O_2^n>) \) cells and time at most \( 2f(<M_i|O_2^n>) \), then we get a contradiction since we flipped the answer in defining \( A \).
However, even though \( g(n) = o(f(n)) \)
\( n = \langle M, i \rangle \) might be small enough that \( f(n) < g(n) \), in which case \( \text{P} \) wouldn't be a language.

To get around this, we flip an infinite number of values for each \( M_i \) and not just the diagonal.

We use
\[
* = \langle M_i, 0^k \rangle \text{ for all integer } k
\]
which will allow us to tell which
machine is associated.

Now, for any input \( * = \langle M_i, 0^k \rangle \) such that
\( k \) makes
\[
\langle M_i, 0^{2^k} \rangle \geq_g \langle M_i, 0^k \rangle
\]
is good enough,
and we get a contradiction.

---

If \( S(1) \) is \( o(S(1)) \) then
\[
\text{SPACE}(S(1)) \subsetneq \text{SPACE}(S(1))
\]

Note: most natural functions are space constructible
\( n^k, \log n, n \log n, \text{ etc.} \)

- \( \log n \):
  - On input \( \langle n \rangle \), count \# of bits
  - onto work tape: gets \( n \) in binary
  - which takes \( \log n \) bits.
  - Now count \# of bits in that: \( \log n \) in binary

\( \text{Con} = \text{NL} \subseteq \text{SPACE}(\log^2 n) \subsetneq \text{PSpace} \)
Time Hierarchy

Definition: A language is time constructible if \( f(n) \) is computable in time \( O(f(n)) \).

Theorem: If \( L \) is time constructible, there is a language decidable in time \( O(f(n)) \) but not \( O(f(n)^{log_2 n}) \).

Proof idea: Essentially the same as the one for bounded space except that on input \( x \), we use time constructibility to compute \( t(|x|) \) in binary and use it as a timer for the computation. Count down to 0 (subtract 1 per step) reject if it exceeds the time.

Unlike with space complexity, we have to count the number of steps to update the timer.
The timer takes \( \log(t(|x|)) \) bits to represent and update.

In the course, we used multi-tape TM for this. The book used 1-tape TM. The proof is different in the two cases.
Multi-type TM version: We need a fixed # of tapes for the machine defining \( A \) but the other TMs \( M_i \) might use more tapes.

We use simulation of \( k \)-tape TM by 2-tape TM.

Each step becomes \( O(t(n)/\log(t(n))) \) steps & it keeps track of a counter.

1-tape version: Maintain the counter like a pocket watch that is carried by the TM near the read head:

Think of it as on a separate branch of the tape.

Shift the timer letter right at each time step:

\( O(\log(t(n))) \) steps per original step.

If \( t(n) \) is \( O(t(n)/\log(t(n))) \) then both can be done in \( O(t(n)) \) step.

Can we use these diagonalization arguments to prove \( P \neq NP \)?
Idea why "NO": These diagonalization arguments work by "simulation".

We could get separated even if both machines could get access to the answer to some hard problem.

"Oracle TM1" $M^B$ gets to ask question: "Is $y$ in $B$" in 2 step language.

Simulation also works for oracle machine.

$P^B$ = languages decidable in poly time using such an oracle TM.

$NP^B$ = same except nondet oracle TM.

If diagonalization showed $P \neq NP$ it would also show $P^B \neq NP^B$ for every $B$.

But: $\exists B: P^B = NP^B$

Example $B = TQBF$.

$P_{TQBF} = \text{PSpace-Lelem} = NP_{\text{Space-Lelem}} = NP_{TQBF}$

Feit. Also $\exists A$, one can prove $PA \neq NP_A$ (complicated see 9.2 in text).
Q: Why wouldn't SAT work?

NP SAT can decide formulas of the form

\[ \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \forall (x_1 \ldots x_n, y_1 \ldots y_m) \]

Here's how: NP machine guesses \( x_1, \ldots, x_n \), \( y_1, \ldots, y_m \).
Then call SAT oracle on

\[ \exists \phi(x_1, \ldots, x_n, y_1, \ldots, y_m) \]

(if this is not SAT then

\[ \forall y_1, \ldots, \forall y_m \forall (x_1, \ldots, x_n) \]

it true

so the whole formula would be true)

\[ \therefore \exists_x = \text{NP} \text{SAT} \]

However \( \text{P} \text{NP} \subseteq \exists_x \cap \Pi_2^0 \)

we don't know if \( \exists_x = \Pi_2^0 \) (researcher question)

so it is possible that \( \exists_x \cap \Pi_2^0 \neq \exists_x \)

in which case \( \text{P} \text{SAT} \neq \text{NP} \text{SAT} \).