$NP = \text{Non-deterministic Polynomial Time}$

is a large family of problems
expect some are not solvable in polynomial time.

$NP$-hard: Problems at least as hard as everything in $NP$.

If can solve any one of them in poly time, then can solve every $NP$ problem in poly time.

$NP$-complete: $NP$-hard and in $NP$
especially essentially equivalent to each other
The hardest problems in $NP$.

Idea behind $NP$: (motivation before definition)
Problems for which it might be hard to find the answer. But once found is easy to check.

Like systems of equations:
takes work to find solution, but easy to check.

Linear equations are in polynomial time,
but Systems of Polynomial Equations are hard.
Abbreviate SPE.
SPE: Have variables, say $x_1, \ldots, x_n$, and polynomials $p_1(x), \ldots, p_k(x)$.  

Problem 1: find $x$ s.t. $p_i(x) = 0$ for all $i$? 

$\rightarrow$ Problem 2: Does there exist $x$ s.t. $p_i(x) = 0$ for all $i$?

Problem 1: Given this $x$, can efficiently check if satisfies all equations. If $x$ is rational, and the coefficients of the polynomials are rational, can check.

i: $x^2 - 2 = 0$, $x^p - 2 = 0 \forall x, y, p$  

ii: Size$(x)$ could be much larger than size$(p)$, is inefficient from perspective of $p_i \ldots p_k$  

iii: What if is no solution?  

no solution is an answer.  
How would we check that?

We go with Problem 2, which has just yes/no answers.  
If “yes”, is an $x$ that you can (try to) check.  
If “no”, there might not be.

Problems with yes/no answers are called decision problems
An \( \mathsf{NP} \)-complete problem: \( \exists \mathsf{013}\text{-SPE} \)

Does there exist \( x \in \mathsf{013}^n \) s.t. \( P_i(x) = 0 \), \( \forall i \)?

Now, the solution can not be big: is in \( \mathsf{011}^n \).
Can evaluate \( P_i(x) \) in time polynomial in \( \text{size}(P_i) \), so can check \( x \) efficiently.

For yes answers and \( x \) proving answer is “yes” \( P_i(x) = 0 \) \( \forall i \leq k \), call \( x \) a witness, certificate, or proof.

For no answers there does not need to be.

Let \( q = (P_1, \ldots, P_k) \) specify an instance of the problem.
Write \( q \in \mathsf{013}\text{-SPE} \) if is valid problem with yes answer.

Def A problem \( Y \) (like \( \exists \mathsf{013}\text{-SPE} \)) is in \( \mathsf{NP} \) if \( \exists \) a polynomial-time algorithm \( A \) (witness checker) and constant \( c \) governing answer size such that for all \( q \) (problem instances)

\( q \in Y \) (valid and yes answer) \( \Rightarrow \exists w \) (witness) s.t. \( A(q,w) = \text{"yes"} \) and \( \text{size}(w) \leq c \cdot \text{size}(q)^c \)

if \( q \not\in Y \), \( \forall w \) s.t. \( \text{size}(w) \leq c \cdot \text{size}(q)^c \), \( A(q,w) = \text{"no"} \).
If answer is yes, A continues A
    " " " no, A will never say "yes"
If q is not valid problem, A says no
w could be larger than q, but rarely is.
size(w) ≤ poly(size(q))
Time of A is poly in size of q.

3.113 - SPE is in NP

Def. A problem Y is in P if

\[ \exists \text{ a polynomial-time algorithm } A \]
\[ q \in Y \Rightarrow A(q) = "yes" \]
\[ q \notin Y \Rightarrow A(q) = "no" \]

P ⊆ NP
Linear feasibility \( \exists x \text{ st. } Ax \leq b ? \) is in P

Optimization → Decision:
\[ \exists x \text{ st. } f(x) \leq t \text{ and } g(x) \leq 0 \]

Then search on \( x \)
\[ x_1 > 0 \quad x_1 \leq 2 \quad x_1 \leq q \quad x_1 \leq 8 \ldots \]
Can learn \( x \) by asking about its bits.
All this is in NP, if \( f \) and \( g \) poly time computable.
“Algorithm” is a bit vague. Program is more precise. Turing machines formalize this. We will use logic circuits.

\[ NP \text{ contains very hard problems} \]

\begin{itemize}
  \item Factoring.
  \item Break every public-key crypto scheme.
  \item Design anything given specs.
  \item Prove any theorem, as long as proof is not too long.
\end{itemize}

How do we prove something is \( NP \)-hard?

\textbf{Reductions.}

A \textit{Karp reduction from} \( Y \) to \( Z \) is a polynomial time algorithm \( A \) s.t.
\[
g \in Y \iff A(g) \in Z
\]

\( A \) transforms problem \( Y \) into problem \( Z \)

Given \( A \) and a Ptime algorithm for \( Z \), can solve \( Y \) in Ptime.

If \( Y \) is hard, then \( Z \) must also be hard.
A **Cook** reduction from $Y$ to $Z$ is a polynomial time algorithm $A$ that decides if $y \in Y$
using an oracle for $Z$ that decides membership in $Z$
in constant time.

Can solve $Y$ given a subroutine for $Z$

$Y$ is polynomial-time reducible to $Z$ 
if there exists a Cook reduction from $Y$ to $Z$.
Karp reductions are Cook reductions,
and always use Karp reductions
(For all known decision problems in $NP$
with $Y \leq_P Z$, is a Karp reduction)

$Z$ is $NP$-hard if $Y \in NP$, $Y \leq_P Z$.

Still seems like a lot to show,
$NP$-complete problems make this manageable.

$Z$ is $NP$-complete ($NPC$)
if $Z \in NP$ and $Z$ is $NP$-hard.
To show \( X \) is \( \text{NP-hard} \), just show \( Z \leq_p X \)
for some \( \text{NP-hard} \) \( Z \).

Then \( U \in \text{NP} \), \( Y \leq_p Z \) and \( Z \leq_p X \Rightarrow Y \leq_p X \)
\( \Rightarrow X \) is \( \text{NP-hard} \)

Once we know one such \( Z \), always work from it
and never have to write \( U \in \text{NP} \) again.

**Theorem** Circuit-Satisfiability (C-SAT) is \( \text{NP-complete} \).

**Problem:** given a logic circuit with one output,
is there an input that makes the output true?

**Example**

\[
\begin{align*}
\text{AND} & \quad \text{NOT} \\
\text{AND} & \quad \text{OR} \\
\text{AND} & \\
\text{AND} & \\
\end{align*}
\]

A binary boolean circuit has gates numbered 1,..., \( k \)
\( \text{s.t.} \) each is either

a. an input \( (\text{True} \ (\text{False} \ \bar{I}/0) \)

b. the negation \( (\text{NOT}) \) of a lower numbered gate
c. the AND or OR of two lower numbered gates.

Gate \( k \) is the output.

WOLOG inputs are gates 1 through \( n \), and \( k \geq n \).
Observation: can view every gate $g_j$ as a function of the inputs. If all these equations are satisfied, $y_j = g_j(x_1, x_n)$, for all $j$. $g_j(x_1, x_n)$
Why \( \text{C-SAT} \) is \( \text{NP} \)-complete:

\( \text{C-SAT} \in \text{NP} \): given an input can evaluate every gate. \( \checkmark \)

\( \text{C-SAT} \) is \( \text{NP} \)-hard because (roughly) for every algorithm \( A \) that runs in time \( T(n) \) on inputs of size \( n \),

for every \( n \) there is a circuit \( C_n \) with \( \leq O((T(n)+n)^2) \) gates s.t.

\( C_n(x) = A(x) \) for all \( x \in \{0,1\}^n \)

Anything a \( \text{ptime} \) algorithm can do a polynomial size circuit can, too.

Now, want to prove \( \Sigma_0^P \text{-SPE} \) is \( \text{NP} \)-complete and \( \text{SPE} \) is \( \text{NP} \)-hard.

We already argued \( \Sigma_0^P \text{-SPE} \in \text{NP} \).

Now need to prove \( \text{C-SAT} \leq_p \Sigma_0^P \text{-SPE} \)

Let \( \Phi \) be an input to \( \text{C-SAT} \). That is a circuit.

We need to translate into an instance of \( \Sigma_0^P \text{-SPE} \)

Let \( g_1, \ldots, g_k \) be gates in \( \Phi \)

\( g_1, \ldots, g_n \) are inputs \( x_1, \ldots, x_n \)
Variables will \( Y_1, \ldots, Y_k \)

- For \( k-n+1 \) equations
  - Force \( Y_j = g_j(Y_1, \ldots, Y_n) \)
  
  \[ g_j = \text{NOT}(g_i) \quad Y_j = 1 - Y_i \quad Y_j + Y_i = 0 \]
  
  \[ g_j = \text{AND}(g_h, g_i) \quad Y_j = Y_h \cdot Y_i \quad Y_j - Y_h Y_i = 0 \]
  
  \[ \text{OR}(g_h, g_i) \quad Y_j = Y_h + Y_i - Y_h Y_i \]

For output \( Y_k = 1 \quad Y_k = 0 \)

If all \( g \)s satisfied, and \( Y_1, Y_n \in \{0, 1\} \)

then \( Y_k = g_k(Y_1, \ldots, Y_n) \)

If \( \exists \; x_1, \ldots, x_n \) s.t. \( g_k(x_1, \ldots, x_k) = 1 \)

then \( \exists \; Y_1, \ldots, Y_n \) that satisfy all equations

\[ Y_j = g_j(x_1, \ldots, x_k) \quad Y_i = x_i \quad 1 \leq i \leq n \]

And conversely if \( Y_1, \ldots, Y_k \) sat all \( g \)s

then \( g(Y_1, \ldots, Y_n) = 1, \; g \in \text{C-SAT} \)

Equations are quadratic and each has at most 3 terms, and coeffs in \( \{-1, 0, 1\} \)
Thus \( \exists_{0,1}^1 \text{SPE} \leq_P \text{SPE} \)

so \( \text{SPE} \) \( \text{NP} \)-hard

Proof: same eqns, add \( x_i (\neg x_i) = 0, \forall i \)

only if \( x \in \exists_{0,1}^1 \).