Problem: we need to separate feasible algorithms from non-feasible ones.

Examples:
- $t \sim n^2$, $n^3$, feasible
- $t \sim 2^n$, not feasible

Definition:
\[ t_u(n) = \max_{x : \text{len}(x) = n} t_u(x) \]

An algorithm $u$ is called feasible if there exists a polynomial $p(n)$ such that
\[ \forall n : t_u(n) \leq p(n) \]

Imp fact: This definition doesn't fully reflect the notion of feasibility.

Example:
\[ t_u(n) = 10^{300} \cdot n \]

Common sense viewpoint: not feasible.

From our def: it is feasible (because $10^{300} \cdot n$ is polynomial)
\[ t_u(n) = 10^3 \cdot \exp(0.00001 \cdot n) \]

Common sense: feasible [because $\exp$ is small]

Our def: not feasible [because $\exp$ is not polynomial]

So, the definition has limitations. No one knows how to produce a better definition.

We can use the knowingly imperfect definition of feasibility.

What is a problem? In mathematics:

input $\rightarrow \square \rightarrow$ output $X \rightarrow$ statement $Y \rightarrow$ proof of $X$ or $\neg X$.

Comments:
1. Checking whether $Y$ is a correct proof $X$ is relatively easy.

2. The length of the proof should be feasible. [checking the steps followed in the proof]

[like 4 color problem, length is too much]
bounded by polynomial
of (\text{len}(x))

Condition \( R(x, y) \) is feasible means \( \text{len}(y) \leq P \text{e} \text{len}(x) \)

we have:
\* Feasible predicate relation \( R(x, y) \) true or false
\* Polynomial \( P \text{e} \)

Given: \( x \) binary seq.

we want: find \( y \) such that \( P(x, y) \) and \( \text{len}(y) \leq P \text{e} \text{len}(x) \)

In Physics:
\( x \) - observation
\( y \) - theory; that explains observations.

\( R(x, y) \) - check whether observations are consistent with the data.

Here, \( \text{len}(y) \leq \text{len}(x) \)

No of observations

Engineers:
\( x \) - specifications (cost, how much it weighs, ittolerate, coind etc)
\( y \) - spec' design [like constructing a bridge]

\( R(x, y) \) - feasible [now-a-days computer simulations help in testing the design like simulating a truck going over bridge]

But coming up with a good design is difficult.

\( \text{len}(y) \) is not so complex here.

Humanity:
\( x \) - emotions
\( y \) - a poem

\( R(x, y) \) - not formalizable.

In principle, every practical problem can be algorithmically solved by

trying all binary strings \( y \) of length \( \leq P \text{e} \text{len}(x) \).

Problem: we need \( 2^n \) time not feasible.

Good news: if we guess an answer that we can check it's correctness

in polynomial time.

A P.R. problem \( \equiv \) can be solved on a non-deterministic TM in

practical polynomial time.

Non-deterministic Polynomial = NP

NP \equiv \text{class of all practical problems}. 
CS problem: \( X - \) original list
\( Y - \) sorted list.

\( R(X,Y) \): checking if \( Y \) is same list as \( X \). \( Y \in \) \( Y \) is sorted.
\( \text{len}(Y) = \text{len}(X) \).

\( P \): class of all problems which can be solved by a feasible algorithm.
\( \text{example: sorting problem} \).

\( \Pi \)

Natural question: \( \Pi \subseteq P \)

Is it possible to solve any practical problem in reasonable time? Ans: we don't know.

\( \Pi \) hard problem: Hardest problems in \( \Pi \).

By, \( \Pi \) hard, we can reduce class of \( \Pi \) to a smaller set of problems.