EXPTIME is the set of all decision problems that can be solved in exponential time. i.e., in time \(O(2^{p(n)})\) where \(p(n)\) is a polynomial in the input size.

Observe that \(\text{NP} \subseteq \text{EXPTIME}\).

The idea is to generate all possible certificates of an appropriate length and check them for correctness using the given certificate verification algorithm. An example, for Hamiltonian Cycle, we could generate all possible certificates and check each one in turn.

We do not know if there are problems in \(\text{NP}\) that cannot be solved in polynomial time (because the \(\text{P} \neq \text{NP}\) conjecture is not yet resolved). However, it is possible to prove that there exist problems in \(\text{EXPTIME} \setminus \text{P}\).

One such problem is the Bounded Halting problem. Here an instance \(I = (A, x, t)\), where \(A\) is a program, \(x\) is an input to \(A\), and \(t\) is a positive integer (in binary). The question to be solved is if \(A(x)\) halts after at most \(t\) computation steps.

The Bounded Halting problem can be solved in time \(O(t)\), but this is not a polynomial time algorithm because size(I) = \(|A| + |x| + \log t\).

Actually, it can be proven that Bounded Halting is \(\text{EXPTIME}\)-complete. This implies that it is in \(\text{EXPTIME} \setminus \text{P}\), since it is known that \(\text{EXPTIME} \neq \text{P}\).

The \(t\) is exponential in \(\log t\).

(And \(\log t\) might be the largest item in the input size, in which case \(O(t)\) would be exponential in the input size.)

UNDECIDABILITY

Problems that are impossible to solve

DECIDABLE VS UNDECIDABLE PROBLEMS

We say an algorithm \(A\) "solves" a decision problem \(\Pi\) if, for every instance \(I\), \(A(I)\) has finite runtime and returns the correct answer.

If an algorithm \(A\) solves decision problem \(\Pi\), then we say \(\Pi\) is decidable.

Formally, \(\Pi\) is decidable IFF there exists some algorithm \(A\) such that, for every instance \(I\), \(A(I)\) returns the correct answer in finite time.

If \(\Pi\) is not possible to design an algorithm \(A\) that solves decision problem \(\Pi\), then we say \(\Pi\) is undecidable.

Formally, \(\Pi\) is undecidable IFF there cannot exist an algorithm \(A\) such that, for every instance \(I\), \(A(I)\) returns the correct answer in finite time.

Equivalently, \(\Pi\) is undecidable IFF, for every algorithm \(A\), there exists some input \(I\) such that \(A(I)\) does not return the correct answer in finite time.

I.e., for some input, \(A(I)\) either runs forever or returns the wrong answer.
HALTING: AN UNDECIDABLE PROBLEM

The Halting problem is decidable IFF there exists an algorithm \( H_{\text{halt}} \) that, for every instance \( I = (A,x) \), \( H_{\text{halt}}(I) \) has finite runtime and correctly answers the question: “would a call to \( A(x) \) halt in finite time?”

For example, you could run \( H_{\text{halt}}(\text{BFS}, G) \) to determine whether \( \text{BFS}(G) \) will halt in finite time, which it will, so \( H_{\text{halt}}(\text{BFS}, G) \) returns yes.

UNDECIDABILITY OF THE HALTING PROBLEM

We suppose \( H_{\text{halt}} \) exists, to obtain a contradiction…

Since \( A \) is a string (of code), and its input \( x \) is also a string…

Suppose \( \text{Strange}(\text{Strange}) \) does not halt. Then, it must spin in the while loop forever. This means \( H_{\text{halt}}(\text{Strange}, \text{Strange}) = \text{true} \).

So, \( \text{Strange}(\text{Strange}) \) halts ---- contradiction!

If we have \( H_{\text{AllSolver}} \) then we have \( H_{\text{halt}} \), but this is impossible, so \( H_{\text{AllSolver}} \) cannot exist, so the \( H_{\text{All}} \) problem is undecidable!
Every problem in NP transforms to 3-SAT transforms to

SAT

3-SAT

Clique

Vertex Cover

(target) Subset Sum

0-1 Knapsack

Let’s show this.

RECALL: 0-1 KNAPSACK PROBLEM

Problem 7.18

Subset Sum

Instance: A list of sizes \( S = \{s_1, \ldots, s_n\} \) and a target sum \( T \).

Question: Does there exist a subset \( J \subseteq \{1, \ldots, n\} \) such that \( \sum_{i \in J} s_i \leq M \) and \( \sum_{i \in J} p_i \geq T \)?

Target Subset Sum \( \leq_p \) 0-1 Knapsack

Problem 7.18

Such that \( I \) contains a subset that sums to \( T \), (or better) profit can be obtained in knapsack input \( f(I) \).

TARGET SUBSET SUM \( \leq_p \) 0-1 KNAPSACK

Can I obtain profit \( T \) (or better) by taking (whole) items with total weight \( \leq M \)?

RECALL: 0-1 KNAPSACK PROBLEM

Problem 7.3

0-1 Knapsack Doc

Instance: a list of profits \( P = \{p_1, \ldots, p_n\} \), a list of weights \( W = \{w_1, \ldots, w_n\} \); a capacity \( M \); and a target profit \( T \).

Question: Is there an \( n \)-tuple \( \{x_1, x_2, \ldots, x_n\} \in \{0, 1\}^n \) such that \( \sum_{i=1}^{n} w_i x_i \leq M \) and \( \sum_{i=1}^{n} p_i x_i \geq T \)?

How should we poly-transform (Target) Subset-Sum input into (Target) 0-1 Knapsack input?

0-1 Knapsack Doc

Instance: a list of profits \( P = \{p_1, \ldots, p_n\} \), a list of weights \( W = \{w_1, \ldots, w_n\} \); a capacity \( M \); and a target profit \( T \).

Question: Is there an \( n \)-tuple \( \{x_1, x_2, \ldots, x_n\} \in \{0, 1\}^n \) such that \( \sum_{i=1}^{n} w_i x_i \leq M \) and \( \sum_{i=1}^{n} p_i x_i \geq T \)?

Claim: \( I \) contains a subset that sums to \( T \), (or better) profit can be obtained in knapsack input \( f(I) \).

Subset Sum \( \leq_p \) 0-1 Knapsack

Let \( I \) be an instance of Subset Sum consisting of ints \( \{x_1, \ldots, x_n\} \) and target sum \( T \).

Define

\[
\begin{align*}
p_i &= w_i, & 1 \leq i \leq n \\
w_i &= w_i, & 1 \leq i \leq n \\
M &= T 
\end{align*}
\]

Then define \( f(I) \) to be the instance of 0-1 Knapsack consisting of profits \( \{p_1, \ldots, p_n\} \), weights \( \{w_1, \ldots, w_n\} \); capacity \( M \) and target profit \( T \).

Exercise: Prove the correctness of this transformation.
Every problem in NP transforms to 3-SAT.

EXERCISE: GIVE A POLY-TRANSFORMATION

This exercise: Show how to transform Hamiltonian Cycle input into TSP-Decision input (in poly time). Such that: \( I \) contains a Hamiltonian cycle \( \iff \) \( f(I) \) contains a TSP cycle of weight at most \( T \).

Let \( I \) be an instance of Hamiltonian Cycle consisting of a graph \( G = (V, E) \).
For the complete graph \( K_n \), where \( n = |V| \), define edge weights as follows:
\[
w(x, y) = \begin{cases} 1 & \text{if } x, y \in E \\ 2 & \text{if } x, y \notin E.
\end{cases}
\]
Then define \( f(I) \) to be the instance of TSP-Dec consisting of the graph \( K_n \), edge weights \( w \), and target \( T = n \).
Exercise: Prove the correctness of this transformation.
**FUN AND GAMES**

We prove NP-hardness results for six of Nintendo's largest video games foundational: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. Our results apply to generalized versions of Super Mario Bros. • The Lost Levels, and Super Mario World; Donkey Kong Country • 1 • 2; all Legend of Zelda games; all Metroid games; and all Pokémon mini-puzzle games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

![Figure 11: Clause gadget for Super Mario Bros.](image)

**FACTUALLY INCORRECT MEMES**

• There's also an old video meme about proving that Super Mario Bros is NP complete
• (Long before it was legitimately proved NP hard ☺)
• Whereas the stuff on the previous slide is real math, the stuff in this video is just a meme, and is very wrong, but you may find it funny…

**SUMMARY OF COMPLEXITY CLASSES**

- $P$ (Polytime)
  - E.g., decision problem variants of BFS, Dijkstra's, some DP algorithms
- $NP$ (Non-deterministic polytime)
  - All of $P$ and $NP$; we can cover, detect SAT subsumption
- $NPC$ (NP-complete)
  - Decision problems $\Pi \in NP$ s.t. every $\Pi'$ $\in NP$ can be transformed to $\Pi$ in poly-time
  - Equivalently: decision problems that can be solved in poly-time if you have access to a non-deterministic oracle that returns a yes-certificate if one exists
- $NP$-hard (at least as hard as NPC)
  - Problems $\Pi$ s.t. every $\Pi'$ $\in NP$ can be reduced to $\Pi$ in poly-time

* Note: $P$, $NP$ and $NPC$ problems are decidable

**POLYTIME 2-SAT**

(If we have time)

**2-SAT EXAMPLES**

- $\left(p \lor q \right) \land \left(\neg p \lor v \lor r \right) \land \left(\neg r \lor v \lor \neg q \right)$
- Satisfiable: $p = 0, q = 1, r \in \{0,1\}$
- $\left(p \lor q \right) \land \left(\neg p \lor v \lor r \right) \land \left(\neg r \lor v \lor \neg q \right)$
  - $p \lor q$ is equivalent to $q \Rightarrow \neg p$
  - $p \lor q$ is equivalent to $\neg p \Rightarrow q$
  - $p \lor q$ is equivalent to $\neg q \Rightarrow p$

2-SAT can be solved in polynomial time. Suppose we are given an instance $I$ of 2-SAT on a set of boolean variables $X = \{x_1, \ldots, x_n\}$

1. For every clause $x \lor \neg y$ (where $x$ and $y$ are literals), construct two directed edges $xy$ and $\neg x \neg y$. We get a directed graph on vertex set $X \cup \overline{X}$.
2. Determine the strongly connected components of this directed graph.
3. $I$ is a yes-instance if and only if there is no strongly connected component containing $x$ and $\overline{x}$ for any $x \in X$.

Suppose no variable $x$ is in the same SCC as $\overline{x}$, then to get a satisfying assignment do the following:

For each $x$, if $x$ is path from $x$ to $\overline{x}$, then set $x = \text{false}$ else set $x = \text{true}$.