Where we are, where we are going

The theoretical divide between efficient and inefficient:

\[ L \in P \implies \text{efficiently solvable.} \]

\[ L \notin P \implies \text{not efficiently solvable.} \]
What is efficient in theory and in practice?

**In practice:**

- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n^3)$
- $O(n^5)$
- $O(n^{10})$
- $O(n^{100})$

**In theory:**

- Polynomial time
- Otherwise

---

- Poly-time is **not** meant to mean “efficient in practice”.
- Poly-time: extraordinarily better than brute force search.
- Poly-time: mathematical insight into problem’s structure.
- Robust to notion of what is an *elementary step*, *what model we use*, *reasonable encoding of input*, *implementation details*.
- Nice closure property: Plug in a poly-time alg. into another poly-time alg. $\rightarrow$ poly-time
What is efficient in theory and in practice?

**Brute-Force Algorithm:** Exponential time

usually the “magic” happens here

what we care about most in 15-251

**Algorithmic Breakthrough:** Polynomial time

what we care about more in 15-451

Blood, sweat, and tears: Linear time

---

**Summary:** Poly-time vs not poly-time is a *qualitative* difference, not a *quantitative* one.

---

What is NP?

**EXP**

**DECIDABLE LANGUAGES**

**NP:**

A class/set between P and EXP.

(set of languages we would love to solve efficiently)
What is NP?

P = NP asks whether these two sets are equal.

How would you show P = NP?

How would you show P ≠ NP?

Boolean circuits are related to the P vs NP question in multiple ways.

Boolean Circuits
Some preliminary questions

What is a Boolean circuit?
- It is a computational model for computing decision problems (or computational problems).

We already have TMs. Why Boolean circuits?
- The definition is simpler.
- Easier to understand, usually easier to reason about.
- Boolean circuits can efficiently simulate TMs.
  (efficient decider TM \(\Rightarrow\) efficient/small circuits.)
- Circuits are good models to study parallel computation.
- Real computers are built with digital circuits.

Dividing a problem according to length of input

\[ \Sigma = \{0, 1\} \]

\[ L \subseteq \{0, 1\}^* \]

\[ f : \{0, 1\}^* \rightarrow \{0, 1\} \]

\[ L_n = \{w \in L : |w| = n\} \]

\[ \{0, 1\}^n = \text{all strings of length } n \]

\[ f^n : \{0, 1\}^n \rightarrow \{0, 1\} \]

for \( x \in \{0, 1\}^n \),

\[ f^n(x) = f(x) \]

\[ L = L_0 \cup L_1 \cup L_2 \cup \cdots \]

\[ f = (f^0, f^1, f^2, \ldots) \]

Dividing a problem according to length of input

A TM is a finite object (finite number of states) but can handle any input length.

Input \(\rightarrow TM\) \(\rightarrow output\)

computes \(L\)

Imagine a model where we allow the TM to change with input length.

\(TM_0\)

\(L_0\)

\(TM_1\)

\(L_1\)

\(TM_2\)

\(L_2\)

\(TM_3\)

\(L_3\)

\(\cdots\)
Dividing a problem according to length of input

So one machine does not compute \( L \).

You use a family of machines:

\[(M_0, M_1, M_2, \ldots)\]

(Imagine having a different Python function for each input length.)

Is this a reasonable/realistic model of computation?!!

Boolean circuits work this way.
Need a separate circuit for each input length.
(but we still love them)

---

**Boolean Circuit Definition**

---

**Picture of a circuit**
Poll 1: What does this circuit compute?

(sometimes circuits are drawn upside down)

How does a circuit decide a language?

How do we measure the complexity of a circuit?
Given \( f : \{0, 1\}^* \rightarrow \{0, 1\} \), write

\[
    f = (f^0, f^1, f^2, \ldots) \quad \text{where} \quad f^n : \{0, 1\}^n \rightarrow \{0, 1\}
\]

Construct a circuit for each input length.

\[\begin{array}{cccc}
C_0 & C_1 & C_2 & C_3 \\
\hline
f^0 & f^1 & f^2 & f^3 \\
\end{array}\]

A circuit family \( \mathcal{C} \) is a collection of circuits \( \{C_0, C_1, C_2, \ldots\} \)
where each \( C_n \) takes \( n \) input variables.

How can a circuit compute a language?

Circuit size and complexity

Definition (size of a circuit):

Definition (size of a circuit family):

Definition (circuit complexity):
Let $f : \{0, 1\}^* \to \{0, 1\}$ be the parity decision problem.

\[
f(x) = x_1 + \ldots + x_n \mod 2 \quad \text{(where } n = |x|)\\
f(x) = x_1 \oplus \ldots \oplus x_n
\]

What is the circuit complexity of this function?
The big picture

Computability with respect to circuits

Theorem 1:

A universal exponential upper bound for all decision problems.
(We know this is not true in the TM model.)

The big picture

Limits of efficient computability with respect to circuits

Theorem 2 (Shannon’s Theorem):

The big picture

Circuits can efficiently “simulate” TMs

Theorem 3:

poly-time TM $\implies$ poly-size circuits
Consequence of Theorem 3

poly-time TM $\Rightarrow$ poly-size circuits

no poly-size circuits $\Rightarrow$ no poly-time TM

To show $P \neq NP$:

Find $h$ in NP whose circuit complexity is more than any $n^k$.

Consequence of Theorem 3

So we can just work with circuits instead

This is awesome in 2 ways:

1. Circuits: clean and simple definition of computation.
   "Just" a composition of (AND), (OR), (NOT) gates.

2. Restrict the circuit.
   Make it less powerful.
   e.g. (i) restrict depth
   (ii) restrict types of gates

Poll 3

How many different functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ are there?
- $n$
- $2^n$
- $n^2$
- $2^n$
- $2^{2^n}$
- none of the above
- beats me
Theorem 2: Some functions are hard

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

**Proof:**

**Proof (continued):**
Theorem 2: Some functions are hard

Proof (continued):

That was due to Claude Shannon (1949).

Father of *Information Theory*.

Claude Shannon (1916 - 2001)

A non-constructive argument.

In fact, it is easy to show that *almost all* functions require exponential size circuits.

Concluding Remarks

Boolean circuits: another model of computation. (arguably simpler definition, easier to reason about)

*no* poly-size circuits $\implies$ *no* poly-time TM (can attack P vs NP problem with circuits)

CIRCUIT-SAT decision problem:

Given as input the description of a circuit, output True if the circuit is “satisfiable”.

Whether CIRCUIT-SAT is in P or not is intimately related to the P vs NP question!