

CSE 332

Data Structures and Parallelism

**P vs. NP:
The Million \$ Problem**

Definition (Complexity Class)

A **complexity class** is a set of problems limited by some resource constraint (time, space, etc.)

Today, we will talk about three: P, NP, and EXP

Definition (The Class P)

P is the set of **decision problems** with a polynomial time (in terms of the input) algorithm.

We've spent pretty much this entire course talking about problems in P.

For example:

CONN

Input(s): Graph G

Output: true iff G is connected

CONN \in P

dfs solves **CONN** and takes $\mathcal{O}(|V| + |E|)$, which is the size of the input string (e.g., the graph).

2-COLOR \in P

We showed this earlier!

How About These? Are They in P?

- 3-COLOR?
- CIRCUITSAT?
- LONG-PATH?
- FACTOR?

We have no idea!

There are a lot of open questions about P...

But Is There Something NOT in P?

YES: The Halting Problem!

YES: Who wins a game of $n \times n$ chess?

As one might expect, there is another complexity class EXP:

Definition (The Class EXP)

EXP is the set of **decision problems** with an exponential time (in terms of the input) algorithm.

Generalized **CHESS** \in EXP.

Notice that $P \subseteq \text{EXP}$. That is, all problems with polynomial time worst-case solutions also have exponential time worst-case solutions.

But a digression first...

Remember **Finite State Machines**?

You studied two types:

- DFAs (go through a single path to an end state)
- NFAs (go through **all possible paths** simultaneously)

NFAs “try everything” and if any of them work, they return true. This idea is called **Non-determinism**. It’s what the “N” in NP stands for.

Definition #1 of NP:

Definition (The Class NP)

NP is the set of **decision problems** with a **non-deterministic** polynomial time (in terms of the input) algorithm.

Unfortunately, this isn’t particularly helpful to us. So, we’ll turn to an equivalent (but more usable) definition.

Definition (Certifier)

A **certifier** for problem **X** is an algorithm that takes as input:

- A String s , which is an instance of **X** (e.g., a graph, a number, a graph and a number, etc.)
- A String w , which acts as a “certificate” or “witness” that $s \in \mathbf{X}$

And returns:

- false (regardless of w) if $s \notin \mathbf{X}$
- true for **at least one** String w if $s \in \mathbf{X}$

Definition #2 of NP:

Definition (The Class NP)

NP is the set of **decision problems** with a polynomial time **certifier**.

A consequence of the fact that the certifier must run in polynomial time is that the valid “witness” must have **polynomial length** or the certifier wouldn’t be able to read it.

Okay, this makes no sense, example plx?

We claim **3-COLOR** \in NP. To prove it, we need to find a **certifier**.

Certificate?

We get to choose what the certifier interprets the certificate as. For **3-COLOR**, we choose:

An assignment of colors to vertices (e.g., $v_1 = \text{red}, v_2 = \text{blue}, v_3 = \text{red}$)

Certifier

```
1 checkColors(G, assn) {
2   if (assn isn't an assignment or G isn't a graph) {
3     return false;
4   }
5   for (v : V) {
6     for (w : v.neighbors()) {
7       if (assn[v] == assn[w]) {
8         return false;
9       }
10  }
11  return true;
12 }
```

For this to work, we need to check a couple things:

- 1 Length of the certificate? $\mathcal{O}(|V|)$
- 2 Runtime of the certifier? $\mathcal{O}(|V| + |E|)$

CONN

Input(s): Number n ; Number m

Output: true iff n has a factor f , where $f \leq m$

We claim **FACTOR** \in NP. To prove it, we need to find a **certifier**.

Certificate?

Some factor f with $f \leq m$

Certifier

```
1 checkFactor((n, m), f) {
2   if (n, m, or f isn't a number) {
3     return false;
4   }
5   return f <= m && n % f == 0;
6 }
```

For this to work, we need to check a couple things:

- 1 Length of the certificate? $\mathcal{O}(\text{bits of } m)$
- 2 Runtime of the certifier? $\mathcal{O}(\text{bits of } n)$

Let $X \in P$. We claim $X \in NP$. To prove it, we need to find a **certifier**.

Certificate?

We don't need one!

Certifier

```
1 runX(s, _) {  
2   return XAlgorithm(s)  
3 }
```

For this to work, we need to check a couple things:

- 1 Length of the certificate? $\mathcal{O}(1)$.
- 2 Runtime of the certifier? Well, $X \in P \dots$

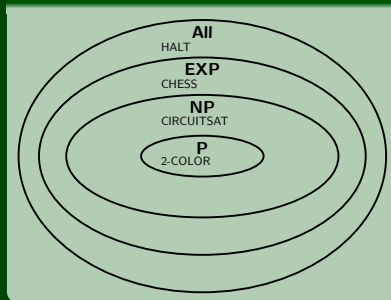
In other words, if $X \in P$, then there is a polynomial time algorithm that solves X .

So, the “verifier” just runs that program. . .

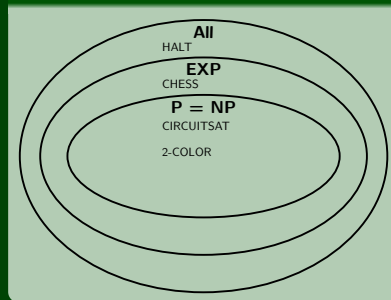
Finally, we can define P vs. NP...

Is finding a solution harder than certification/verification?

If $P \neq NP$



If $P = NP$



Another way of looking at it. If $P = NP$:

- We can solve **3-COLOR**, **TSP**, **FACTOR**, **SAT**, etc. efficiently
- If we can solve **FACTOR** quickly, there goes RSA... oops

Cook-Levin Theorem

Three Equivalent Statements:

- **CIRCUITSAT** is “harder” than any other problem in NP.
- **CIRCUITSAT** “captures” all other languages in NP.
- **CIRCUITSAT** is **NP-Hard**.

But we already proved that **3-COLOR** is “harder” than **CIRCUITSAT**!
So, **3-COLOR** is **also NP-Hard**.

Definition (NP-Complete)

A decision problem is **NP-Complete** if it is a member of NP and it is **NP-Hard**.

Is there an **NP-Hard** problem, **X**, where **X** is **not NP-Complete**?

Yes. The halting problem!

Some **NP-Complete** Problems

CIRCUITSAT, TSP, 3-COLOR, LONG-PATH, HAM-PATH, SCHEDULING, SUBSET-SUM, ...

Interestingly, there are a bunch of problem we don't know the answer for:

Some Problems Not Known To Be **NP-Complete**

FACTOR, GRAPH-ISOMORPHISM, ...