1 | sources

1.1 | gentle introductions

1.1.1 | `https://en.wikipedia.org/wiki/Computational_complexity_theory`

1.1.2 | `https://complexityzoo.net/Petting_Zoo`

2 | overview

2.1 | computational complexity theory studies how "difficult" a problem is

2.1.1 | importantly, not how "good" an algorithm is... this field deals with all algorithms that solve a given problem

2.2 | key concepts

2.2.1 | types of problems

2.2.2 | Turing machines

2.2.3 | reducibility

2.2.4 | complexity classes

2.2.5 | hierarchy

2.3 | key problems

2.3.1 | P vs NP

3 | flows

3.1 | Wikipedia computational complexity theory

3.1.1 | computational problems

1. problem instances

   A problem describes the problem. the actual "numbers" that describe a specific problem is called a problem instance. sorting a list is a problem, sorting *this* list is a problem instance.

2. representing problem instances

   formally strings of characters from alphabets. The input size is the length of the string. Different representations can be chosen but it should be trivial (fast) to convert from one to the other.

3. decision problems (most basic type)

   Generally, given an input, the output is either yes (accept) or no (reject). For example, deciding whether a graph is connected or not.

   (a) it can be thought of as a "formal language" toexpand

4. function problems

Very general: a function problem 'is a computational problem where a single output (of a total function) is expected for every input, but the output is more complex than that of a decision problem'. Basically calculate a non-binary function.

Examples: traveling salesman, integer factorization.

However, all function problems can be modeled as decision problems: For some function $f(*args) \to ans$, it can be modeled as the decision problem of whether $(*args, ans)$ is a valid output.