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 $f(*args) \rightarrow ans$, it can be modeled as the decision problem of whether $(*args, ans)$ is a valid output.

   (a) but does this really work? how can a decision TM be used to compute the function output efficiently? toexpand

5. size of an instance

   Size is usually the length of the input. The complexity is a function of the input size, usually representing the worst case time or space (or any other complexity measure) required for any input size.

3.1.2 | **machine models and complexity measures**

1. Turing machine

   standard Turing machine stuff. its very general. Many types of turing machines (probabilistic, non-deterministic, quantum, etc) are used to define different complexity classes.

2. other machine models toexpand

   Other non-standard Turing machines are used, but the idea is that they aren't actually any better, somehow?

3. Complexity Measures

   Usually time or space, but any complexity measure that satisfies Blum's complexity axioms can be used. Examples include: communication complexity, circuit complexity.

   Also constant factors don't really matter. And its usually the worst case.

   Importantly, complexity measures are also a function of the type of Turing machine used, since some Turing machines are better in some scenarios.

   (a) blums complexity axioms toexpand

4. best/worst/average case

   We generally talk about worst case complexity, but some algorithms have good average-case which is good enough (eg. quicksort). Generally, best-case < average-case < amortized analysis < worst-case.

5. upper and lower bounds for problems

   Importantly, this is **not an upper or lower bound for an algorithm**. Instead, for problems in general, it's relatively easy to decide an upper bound (which is just the worst case complexity of any correct algorithm), but a lower bound is difficult (since it must involve algorithms that haven't been discovered yet).

3.1.3 | **complexity classes**