Assignment 45

Alex Clemmer
Student number: u0458675

1

In general, an NP-complete problem is a problem for which there is an algorithm $\in P$ that verifies the problem's solution, but for which there is no known algorithm $\in P$ that searches for the solution. RSA basically relies on the idea that there is an algorithm $\in P$ for multiplying numbers, but no algorithm $\in P$ that searches for factors. If this were not the case, then we would easily be able to do things like find the primes that make up a private key. Obviously this is a pretty bad result.

2

Easy problems can reduce to harder problems pretty easily. *Any* problem $\in P$ can reduce to an NP-complete problem—that is, if a problem can be solved and checked in deterministic polynomial time, then it can also be solved and checked in nondeterministic polynomial time, since a nondeterministic machine's only advantage is being able to explore an arbitrary number of solutions in the search space concurrently.

It is *not* known to be true, however, that any NP-complete problem can reduce down to a problem $\in P$.

3

All NP-complete problems are bijectively reducible to each other. So if one problem is worst-case exponential, then all NP-complete problems can be reduced to that. And since that problem can be reduced to all other NP-complete problems, all NP-complete problems by extension become exponentially solvable.

4

Same reason as above. Say one NP-complete problem is polynomially solvable. Since we can reduce all NP-complete problems to that problem, and since we can reduce that problem to all other NP-complete problems, then they all must be polynomially solvable.

5

One popular strategy is to use randomized algorithms, which often come very close to being correct, which is often enough. Another way to do things is to restrict your problem set—yacc uses only a small subset of all CFGs, but the upside is that parsing and lexing is a lot faster to do in general. It depends on the context, really.