Turing Machine Homework—Sample Answers

Here are some sample answers to the questions given as homework. Note that, since the questions were fairly informal, open, and philosophical, these are not necessarily the only "correct" answers—others could surely provide further insight—but these are what I had in mind when crafting the questions.

1. Like a DFA, a Turing Machine also reads its input; but unlike a DFA, it can move back and forth across the input tape, modify its contents, react with further modifications, etc. One of the whole points about a TM is that it can go for an **unpredictably** long time, including possibly forever. So there is no running time estimate in terms of O-notation to give, unless you want to allow something like "$O(\infty)$" as a way of saying: "Bigger than any function of n", for purposes of comparison with other algorithms' running times.

2. One way to make these reduction situations easier to think about is in terms of a more direct comparison between problems. It's technically a bit inaccurate (or at least: ill-defined), but we might say that when P reduces to Q, it means that P is **easier than** Q, say "$P \leq Q$" in symbols (where we mean the "or equal" part, too). Recall that what it *does* mean is that if we can solve Q, then we can solve P. And being able to solve a problem means being able to compute an answer, given (a representation of) *any* problem instance. So the idea is that solving Q somehow answers P as well, or leads in some straightforward way to a solution for P. So P is easier, because solving Q always solves P, too; whereas solving P might be great by itself, but it's of no help in solving Q.

3. OK, back to your roommate and their claim: if they think they've discovered that their graph problem reduces to the Halting Problem, well, that's claiming that it is "easier than" (in the sense above) an undecidable, or unsolvable problem. That seems like no big news: it means that it is either solvable, even easily solvable, or it's possible "equally" unsolvable … but that isn't narrowing anything down. Besides, "If you can solve the unsolvable, then you can solve this problem" isn't really making a big claim about how easy or hard it is to solve "this problem". On the other hand, if the Halting Problem is reducible to the graph problem, it means that the graph problem must be unsolvable: after all, solving it would allow you to solve the unsolvable, which is impossible. So, *that* (i.e., the second claim) is big news: it means that the interesting graph problem is actually not solvable by computation, which might be news to those who are interested in it.

4. One simple thought experiment is this: could you write a compiler for your language in the other language? Most sophisticated programmers would realize that this is true, as most languages are ultimately implemented in assembly language, often via C, or something similar. If so, then clearly the most sophisticated languages can be implemented in the most sparse and simple ones (and obviously *vice versa)*. And thus any language that claims to be strictly more powerful is just wrong: we can always implement it in our favorite language and then just use the implementation to compute what the original language did. OK, now that James and Simone are duly silenced, return to studying!

5. Your classmate seems very confused: it sounds like they have a plan to look for while loops in a Java program (which is easy enough), but then to "evaluate the condition" to see if it's true. But you can't just evaluate the condition once, in isolation: you have to evaluate it *every* time, in what are presumably changing conditions (since most of us *want* our while loops to stop). But then, even if you could tell that some particular program *did* loop by this analysis, that wouldn't mean that you could tell *every* program that looped: what about recursive programs? Or any programs whose conditions actually change … .