Assignment 1: Computational Complexity

Due date: Thursday 30 Jan in Class.
You can return it on 28 or 30 Jan in class or during office hours.

The grade of different exercises only show their "relative" grade. Namely, different assignment sets might have different total grades when you add up the grades of their exercises, but they will indeed have the same total grade for us at the end.

1. (15 points) Assignment 15 of Chapter 1. For part (a) do not get into details about Turing machine's model, and rely on the fact that any polynomial time algorithm that you know how to implement in, say, C (or any other well known programming language) can be implemented in Turing machines also in polynomial time. So, describe your algorithms (whenever you need them) in pseudocodes as long as they are clearly easy to implement in polynomial time.

2. (5 points) Read Section 2.1.1 (upper half of page 41) and write the proof of Claim 2.4 in your own words. Try to prove it before reading the proof first, but it is OK to read the proof before writing it. Do not copy-paste. Understand it first, and then write it in your own words.

3. Read the definition of the problems SAT and 3SAT from Section 2.3.1. Note that both of these languages are in NP and that 3SAT is a *special case* of SAT. Therefore, if we prove 3SAT to be NP complete, SAT is also proved to be NP complete (make sure you verify this). The book, first proves SAT to be NP hard and then reduces 3SAT to SAT. Our goal here is to prove that 3SAT (and thus SAT) is in fact NP hard by reducing CircSAT to 3SAT.

   (a) (5 points) Suppose x is a boolean variable. Write a 3SAT instance X that uses variable x and two other "dummy" boolean variables x_1, x_2 such that if $x = false$ there is no way to make X satisfiable no matter x_1, x_2 are set, but if $x = true$, then there is a way to make $X = true$ a well. Hint: write down all 8 possible cases and decide which ones you like to make X true, then choose the right clauses among these 8 ones.

   (b) (5 points) Write a 3SAT instance X using only the variables x, y, z such that X is satisfied if and only if $z = x \wedge y$. Also: solve the same problem for $z = x \vee y$. Hint: Again go over all 8 possible cases.

   (c) (5 points) Do the same for $x = NOT(y)$. Note that here z is a "dummy" variable in X and we want $X = true$ if and only if $y = NOT(x)$.

   (d) (10 points) Reduce CircSAT to 3SAT. Hint: use extra variables (other than the actual input variables) for each wire.

4. (10 points) Suppose $L \in \mathbf{P}$ (i.e. L can be decided in polynomial time) and let L' be "non-trivial" language (i.e. there exists some $x \in L'$ and some $y \notin L'$). Prove that $L \leq_p L'$.

5. (10 points) Assignment 2.9 of the book. Hint: for the last part you can make use of the previous exercise and the existence of a language that is not solvable by any Turing machine, let alone in polynomial time (e.g. Halting problem).

6. (10 points) Assignment 2.10 of the book.

7. First read my note here: `https://piazza.com/class/hqio85joi0y49n?cid=9`
This proof is written for the single-tape Turing machine, where a single tape is used for all the work, and is also assumed to be one directional (i.e. the simplest form of Turing machines).

   (a) (5 points) Suppose we would like to prove the Cook-Levin theorem directly, using the same ideas, for the 3-tape Turing machine that has an input (read only) tape, a work-tape, and an output tape (as defined in class and the book). If we use a similar encoding method as explained in items 1,2,3,4 of my note above, which item among the properties 5,6,8 fails to hold anymore, and why?

   (b) (10 points) A multi-tape Turing machine is called "oblivious" if the location of its tapes *only* depends on the time and input length. Namely, given the time t one can compute efficiently where the headers would be pointing at, at this particular time t, if we are given the input length n. (See Remark 1.7 and Claim 1.8 from the book for more discussion on this, but you will *not* need them to solve this exercise).

   Our goal here is to show how we can resolve the issue that you detect in Part (a) of this exercise for the special case of *oblivious* 3-tape Turing machines. Suppose $L \in NP$ is defined using an oblivious 3-tape polynomial-time Turing machine (which as we know is equivalent to defining it using a single-tape Turing machine, but we somehow "forget" this useful fact here!) as its witness verifier. Namely, suppose there is an oblivious 3-tape witness verifier (polynomial time) Turing machine M for L such that an input $x \in \{0,1\}^n$ is $x \in L$ if and only if there is a witness $w \in \{0,1\}^{poly(n)}$ such that (x, w) is accepted by the oblivious (3-tape polynomial time) Turing machine M.[1]

   Give a direct proof of the Cook-Levin theorem for the language L by giving a reduction f (very similar to my note above, in fact simpler!) that maps any $x \in \{0,1\}$ to a circuit $C = f(x)$ of size $poly(n)$ such that: $x \in L$ if and only if there exists w such that $C(w) = 1$.

   Hint: The circuit C_t of the layer t is even simpler in case of Oblivious Turing machines!

8. **Extra Grade:** (15 points) Assignment 9 from Chapter 1. The term "time constructible" is used in this exercise which defined in page 16. Read this simple definition, however, for this exercise you do not need to use the fact that $T(n)$ is time constructible. HINT: Try simulating RAM Turing machines with ordinary (single tape) Turin machines. Be careful that in the RAM Turing model, the machine can read and write in memory cells whose address is *exponentially* long. So you cannot simply use the tape of the single-tape Turing machine to reflect the memory tape of the RAM Turing machine. Find a general (simple) way to keep track of how a RAM Turing machine performs its computation using a simple (single tape) Turing machine. Start by a general idea of how to simulate the RAM machine using a *two-tape* Turing machine with a work-tape. Then try to handle as much details as you can till you feel the ideas are convincing enough.

[1]Here by $poly(n)$ I refer to any function of n that is bounded by a polynomial (namely, it is at most n^c for some constant c and large enough n.