CSC363: Assignment #4

Due on Friday, March 26, 2021 by 11:59pm

Total Marks: 20 $(+4)$

Question 1

(*4 marks*) Prove that a polynomial function $f(n)$ of degree d is in $O(n^d)$.

(*Hint: Write out* $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$*, where* k *is the degree of* f*. Let* c *be the sum of all of the positive coefficients.*)

Question 2

(*Designed by Paul, 4 marks*) Let $A, B \subseteq \Sigma^*$ be languages. Define the *concatenation* of A and B to be

$$AB = \{w \in \Sigma^* : w = w_1 w_2 \text{ for some } w_1 \in A \text{ and } w_2 \in B\}.$$

For example, if $A = \{\text{“sushi”}, \text{“chungus”}\}$ and $B = \{\text{“juice”}, \text{“fish”}\}$, then

$$AB = \{\text{“sushijuice”}, \text{“sushifish”}, \text{“chungusjuice”}, \text{“chungusfish”}\}.$$

Show that if $A, B \in \mathrm{P}$, then $AB \in \mathrm{P}$ as well, by constructing a poly-time Turing machine that decides whether a given string is in AB. You may use pseudocode as long as you briefly justify its runtime.

Question 3

(*Designed by Eric, 3 marks*) Let $A, B \subseteq \Sigma^*$ be languages.

Show that if $A \in \mathrm{P}$, and B is neither empty nor all of Σ^*, then $A \leq_p B$. You may use pseudocode as long as you briefly justify its runtime.

Question 4

(*Designed by Yousef, 6 marks*) Let $A, B \subseteq \Sigma^*$ be languages.

We say $A \leq_T^p B$ if there is a B-oracle poly-time Turing machine[1] M such that $M(x)$ accepts if and only if $x \in A$. In this case, we say A is **polynomial-time Turing reducible** to B.[2]

Define

$$\text{LOOP} = \{(M, w_1, w_2, w_3) : M \text{ is a Turing machine that doesn't halt on at least 2 of the } w_i \}$$

and

$$\text{HP} = \{(M, w) : M \text{ is a Turing machine that halts on } w\}.$$

(a) (*0 marks*) Guess what HP stands for. You may ask your friends (or the internet).

(b) (*3 marks*) Show that LOOP $\leq_T^p \overline{\text{HP}}$.

(c) (*3 marks*) Show that $\overline{\text{HP}} \leq_T^p$ LOOP.

You may use pseudocode as long as you briefly justify its runtime.

[1] That is, the number of times we ask "is this in B?" is polynomial, and the rest of the machine runs in polynomial time as well.

[2] This is analogous to Turing reducibility. $A \leq_T B$ says "given an oracle to B, we can compute A". $A \leq_T^p B$ says "given an oracle to B (and assuming this oracle takes $O(1)$ time to decide whether something is in B or not), we can compute A in polynomial time".

The difference between $\leq_T^p$ and $\leq_p$ is analogous to the difference between $\leq_T$ and $\leq_m$. In general, to prove $A \leq_p B$, one can describe a polynomial-time procedure to check if something is in A, given access to an oracle for B (which can be assumed to run in $O(1)$ time).

Question 5

(*Designed by Daniel, 3 marks + 4 bonus marks*) You will apply everything you have learned in this course up until now to the field of cryptography. If you accept the challenge, then you will prove a weaker version of the following claim:

If P = NP then there are no pseudorandom number generators.

Formally, a **number generator** G is a poly-time computable function (where n is *a variable*):

$$G : \{0,1\}^n \to \{0,1\}^{n+1}$$

G takes in a n-bit string (you can think of this as the *random seed*) and extends it to a $(n+1)$-bit string[3], for any $n \in \mathbb{N}$.

Now let D be a Turing machine that halts on all inputs. We define the following:

- $p_D(n)$ is the probability that if an input $x \in \{0,1\}^n$ is randomly generated, and D is given $G(x)$ as input, then D accepts. In other words, $p_D(n)$ is the probability that if we feed D a string from our number generator, it will accept.
- $r_D(n)$ is the probability that if an input $\alpha \in \{0,1\}^{n+1}$ is randomly generated, and D is given α as input, then D accepts. In other words, $r_D(n)$ is the probability that if we feed D a truly random string of length $n+1$, it will accept.

We say that G is **pseudorandom**[4] if for every *deterministic poly-time* Turing machine D and every $c > 0$, for large enough[5] n we have

$$|p_D(n) - r_D(n)| \leq \frac{1}{n^c}.$$

This is saying that as the size of the input gets larger, no (reasonably efficient) Turing machine can distinguish a pseudo-randomly generated string from a truly random string with a significant success rate.

Consider the language $L_G = \{G(x) : x \in \{0,1\}^*\}$, and notice that $L_G = range(G)$, which is the set of all strings that can be outputted by G. You can also partition the language L_G into infinitely many finite sets based on the length of the input. This may be useful to you when answering these questions.

(a) (*3 marks*) Consider the number generator which bit shifts the input to the left once:

$$F(x) = x0$$

Show that F is not pseudorandom.

(b) (*Bonus 4 marks, you should finish the other questions first before attempting*)
Let $H : \{0,1\}^n \to \{0,1\}^{n+1}$ be some arbitrary number generator.
Show that if P = NP then H is not pseudorandom.

We suggest you break it up into three parts:

i) Show that $L_H \in$ NP.
ii) Use the assumption to obtain a deterministic poly-time Turing Machine D that decides L_H.
iii) Show that H is not pseudorandom using D.

[3] In the complete definition, a number generator can output a string of any length with the condition that it is longer than the input. We are keeping things relatively simple for this question by restricting the output to be exactly one more bit than the input, thus restricting the range.

[4] In our definition of a pseudorandom number generator, this might be called a Cryptographically Secure Pseudo-Random Number Generator in other resources, or CSPRNG for short, if this interests you further.

[5] By this we mean there is some $n_0 \in \mathbb{N}$ so that for all $n \geq n_0$, the following statement holds.