Homework 7 – Due Friday, April 16, 2021 at 11:59 AM

Reminder Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to the course staff if asked. You must also identify your collaborators and write "Collaborators: none" if you worked by yourself. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Note You may use various generalizations of the Turing machine model we have seen in class, such as TMs with two-way infinite tapes, stay-put, or multiple tapes. If you choose to use such a generalization, state clearly and precisely what model you are using.

Problems There are 4 required problems.

1. (**Odd-Length TM**) Let $OL_{\text{TM}} = \{\langle M \rangle | \ M \text{ is a TM that accepts all strings whose length is a odd number and rejects all other strings}\}$.

   (a) Your first goal in this problem is to use a mapping reduction from $\overline{A_{\text{TM}}}$ to show that OL_{TM} is not Turing-recognizable. Describe what the inputs and outputs of this mapping reduction should be. (As in lecture 17 check-in, question 1.) (3 points)

   (b) Describe a TM computing a mapping reduction from $\overline{A_{\text{TM}}}$ to OL_{TM} and explain why this TM is correct. (8 points)

   (c) Explain why part (b) implies that OL_{TM} is not Turing-recognizable. (3 points)

   (d) Use a (different) mapping reduction to prove that $\overline{OL_{\text{TM}}}$ is not Turing-recognizable (i.e., OL_{TM} is not co-Turing-recognizable). (12 points)

2. (**Asymptotic Notation**) Use to the formal definitions of O and o notation to prove the following statements.

   (a) Prove that $n^2(3\log_7 n + n) = O(n^3)$ by showing that there exists a constant $c > 0$ and a natural number n_0 such that $n^2(3\log_7 n + n) \leq cn^3$ for every $n \geq n_0$. (6 points)

   (b) Prove that $3n = o(n^2)$ by showing that for every constant $c > 0$, there exists a natural number n_0 such that $3n \leq cn^2$ for every $n \geq n_0$. (6 points)

   (c) Prove that $3^{\sqrt{n}} = 2^{o(n)}$ using the fact that $f(n) = o(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) = 0$. (6 points)

3. (**Review of asymptotic notation**) This problem will be graded automatically by Gradescope. Please enter your answers manually by completing the assignment Homework 7-Problem 3. For each of the following, select *true* or *false* using the radio buttons on Gradescope. All logarithms are base 2 unless otherwise stated. (1 point each)

(a) $1337 = O(n^2)$

(b) $n^{10} = O(n^9 \log n)$

(c) $n \log n + 10n = o(n^2)$

(d) $4^n = O(2^n)$

(e) $4^n = 2^{O(n)}$

(f) $2^{2^n} = O(2^{n^2})$

(g) $2n = o(n^2)$

(k) $10^6 = o(n)$

(l) $2 \log n = o(\log n)$

(m) $\frac{1}{3} = o(1)$

(n) $2^n = o(4^n)$

(o) $n^5 = O(32^{\log_2 n})$

(p) $\log n = O(\log(\log n))$

(q) $3^{2^n} = o(2^{3^n})$

4. (**Polynomial-Time Algorithms**)

   (a) Let $A = \{w \in \{0,1\}^* \mid w$ contains at least as many 0's as 1's$\}$. Give an implementation-level description of a basic, single-tape Turing machine M that decides A. Briefly explain why your TM correctly decides A. Analyze the running time and space usage of M to show that $A \in TIME(n \log n)$ and $A \in SPACE(n)$. (12 points)

   (b) An undirected graph $G = (V, E)$ is *triangle-free* if for every triple of vertices u, v, w, it is **not** the case that $(u, v), (v, w)$, and (w, u) are all edges in the graph. Let $TF = \{\langle G \rangle \mid G$ is triangle-free$\}$. Show that $TF \in \mathsf{P}$ by i) giving a high-level description of a polynomial-time algorithm deciding TF, ii) analyzing the correctness of your algorithm, and iii) explaining why your algorithm runs in polynomial time.

   You don't need to specify the exact polynomial runtime that your algorithm runs in, since this may depend on implementation details that are suppressed in a high-level description. Just give a convincing argument that the runtime is polynomial as in the examples in Chapter 7.2 of Sipser. (12 points)

   (c) The *Fibonacci sequence* is defined by the following recurrence: $F_0 = 0, F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for every $n \geq 2$. Prove that there is **no** polynomial-time algorithm that takes as input a natural number n (written in binary) and outputs (i.e., writes to its tape) the number F_n (again, written in binary). (8 points)

   Hint: How big can the numbers F_n get as a function of n?

   (d) Give a high-level description of a polynomial-time algorithm that takes as input a natural number n (written in **unary**) and outputs the number F_n (written in **binary**). Explain why your algorithm is correct and why it runs in polynomial time. (10 points)

   Hint: You can use without proof the fact that Turing machines can perform basic arithmetic operations on binary numbers, like addition, in polynomial time.