

## Homework 9 – Due Thursday, April 16 at 11:59 PM

**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. Collaboration is not allowed on problems marked “INDIVIDUAL.”

**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. **You may describe Turing machines at a high-level on this assignment.**

**Problems** There are 4 required problems and 1 bonus problem.

1. (**INDIVIDUAL: Loopy TM**) Consider the following computational problem: Given the description of a Turing machine  $M$ , is it the case that  $M$  loops forever on *every* input  $w$ ?

- (a) Formulate this problem as language  $LOOP_{TM}$ .
- (b) Prove that  $LOOP_{TM}$  is undecidable. You may do this by either writing out a complete contradiction proof or by giving a mapping reduction – it’s your choice. In either case, make sure to explain why the TM computing your reduction is correct.

2. (**Review of asymptotic notation**) Use the formal definitions of  $O$  and  $o$  notation to prove the following statements. You may also use the fact that  $f(n) = o(g(n))$  if  $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ , as well as assume basic math facts like “ $n \leq n^2$  for all  $n \geq 1$ ” without proof.

- (a) Prove that  $3n + 2n \log n = O(n \log^2 n)$ . (The notation  $\log^2 n$  means  $(\log n)^2$ .)
- (b) Prove that  $3^{n^{1/3}} = 2^{o(n)}$ .

3. (**Asymptotic notation examples**) For each of the following statements, select true or false.

- (a)  $3^n = 2^{O(n)}$  True / False
- (b)  $2^{2^n} = O(2^{2n})$  True / False
- (c)  $n! = O(n^n)$  True / False
- (d)  $2^n = 3^{o(n)}$  True / False
- (e)  $2n = o(n^2)$  True / False
- (f)  $2^{332} = O(n)$  True / False
- (g)  $16n = O(n)$  True / False
- (h)  $n^4 = O(n^2 \log n)$  True / False
- (i)  $n \log^2 n = o(n^2 \log n)$  True / False
- (j)  $3^n = O(2^n)$  True / False

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

- (a) Let  $A = \{w \in \{0, 1\}^* \mid w \text{ contains at least as many 0's as 1's}\}$ . Give a high-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  $A \in SPACE(n)$ .
- (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  $TF = \{\langle G \rangle \mid G \text{ is triangle-free}\}$ . Show that  $TF \in P$  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.

5. (**Bonus problem**) 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  $n \geq 2$ .

- (a) 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).

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

- (b) 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.

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