Problem 1.1. Give an example of a function that is *not* time constructible. Define the analogous notion of *space constructibility.* Give an example of a function that is space constructible, and one that is not space constructible.

Problem 1.2. Consider an alternate notion of Turing machines which can *delete* the current symbol as well as *insert* a symbol in their tapes, in addition to only overwriting. Define carefully the transition function and the computation of such machines. Argue that for every $f : \{0,1\}^* \to 0, 1^*$ and function $T : \mathbb{N} \to \mathbb{N}$, if f is computed by a delete- and insert-enabled TM in time $T(n)$, then it is computed by a "normal" TM in time at most $O(T(n)^2)$. (Assume for simplicity there is one working tape.)
Optional: Can you improve the running time of the "normal" TM by having a bigger alphabet? (Hint: Consider an extension of the alphabet by having an additional marked copy of each symbol in the original alphabet.)

Problem 1.3. Prove that the following languages are in P:

1. $\mathsf{CONNECTED}$: The set of all connected graphs. That is, $G \in \mathsf{CONNECTED}$ if there is a path between every two pair of vertices u and v.
2. $\mathsf{TRIANGLE}$: The set of all graphs that contain a "triangle": vertices u, v, w with edges (u, v), (v, w), (w, u).
3. Let

   $$\mathsf{MODEXP} = \{\langle a, b, c, p\rangle \mid a, b, c, \text{ and } p \text{ are binary integers s.t. } a^b \equiv c(\mod p)\}$$

   (Note that the obvious algorithm does not run in polynomial time. Hint: Try it first where b is a power of 2.)

You can give a short description or pseudocode for the algorithm. Do not give a Turing machine!

Problem 1.4. Show that P and NP are closed under union and intersection: given L_1 and L_2 in P (respectively, NP), the languages $L_1 \cup L_2$ and $L_1 \cap L_2$ are also in P (respectively, NP).

Problem 1.5. Show that P and NP are closed under concatenation: given L_1 and L_2 in P (respectively, NP), the language $L_1 \cdot L_2 = \{w \mid \exists u, v : w = u \cdot v \text{ and } u \in L_1, v \in L_2\}$ is also in P (respectively, NP).

Problem 1.6. Show the following languages are NP-complete:

1. $$\begin{aligned} \mathit{HALFCLIQUE} = \{G \mid & G \text{ is an undirected graph having a clique of at least } n/2 \text{ nodes,} \\ & \text{where } n \text{ is the number of nodes of G}\}. \end{aligned}$$

2. $\mathit{LPATH} = \{\langle G, s, t, k\rangle \mid \text{graph } G \text{ contains a simple path from } s \text{ to } t \text{ of length at least } k\}$.

3. Would your answer to (2) change if k is given in unary?

Remember to show two properties: the language belongs to NP and that it is NP-hard. You may take any language shown to be NP-hard in class or in Arora-Barak as a starting point for a reduction.

Problem 1.7. Let

$$\begin{aligned} \mathit{CNF}_k = \{\varphi \mid & \varphi \text{ is a satisfiable cnf-formula} \\ & \text{where each clause has at most } k \text{ variables}\}. \end{aligned}$$

Show that CNF_2 is in P and CNF_3 is NP-complete.

Problem 1.8. Show that if P = NP, then there is a polynomial time algorithm that takes a graph as input and finds a largest clique contained in that graph.

Problem 1.9. Look up an example of an NP-complete problem on the web that is *not* described in Arora-Barak's textbook in Chapter 2 (not even in the exercises). State the problem you have found.