0. (1 point) Submit your homework in accordance to the guidelines at
https://www.cc.gatech.edu/~rpeng/CS4510_S20/HomeworkGuidelines.pdf.

1. (8 points)

Exercise 7.14 from the text book (3rd edition), modified with $k$ as part of the input.

A permutation on the set $\{1 \ldots k\}$ is a one-to-one, onto function on this set. When $p$ is a
permutation, $p^t$ means the composition of $p$ with itself $t$ times.

Consider the language

$$\text{PERM-POWER} = \{ \langle k, p, q, t \rangle \mid p = q^t \}.$$

(a) (2 points) discuss a suitable encoding scheme for $\langle k, p, q, t \rangle$ over the alphabet $\Sigma = \{0, 1, \#, \}$

(b) (6 points) show that $\text{PERM-POWER} \in P$ by giving the implementation details of a Turing
machine that decides it in polynomial time.
Note that because $t$ may have up to $\Theta(n)$ bits, the value of $t$ may be as large as $2^{\Theta(n)}$. This means a poly-time solution should run in time (poly) logarithmic in the value of $t$.

2. (8 points)
Exercise 7.22 from the textbook.
Consider the language

$$\text{DOUBLE-SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a SAT instance with at least two satisfying assignments} \}.$$ 

Note that this is not 3-SAT. On the other hand, you may use the fact that SAT is NP-complete.

(a) (2 points) Show that \text{DOUBLE-SAT} $\in$ NP.
(b) (4 points) Show that \text{DOUBLE-SAT} is NP-hard.
(c) (2 points) Give an implementation description of the polytime mapping function used in your reduction.

3. (9 points)
The goal of this problem is to show that our formal definitions of P and NP imply that $P = NP$ cannot have a non-constructive proof. Throughout this problem we assume a consistent, binary, encoding of both Turing machines and 3-SAT instances.

Note that the complexities are intentionally not tight to address potential overheads from scanning the tape linearly in order to look for certain symbols.

(a) (3 points) Show that for any constant $c_1$, there is a constant $c_2$ such that if $M_1$ is a Turing machine encoded by at most $c_1$ bits that correctly decides 3-SAT in $O(n^{10})$ time (aka. 3-SAT $\in$ TIME($n^{10}$)), then there is a Turing machine $M_2$ encoded by at most most $c_2$ bits that runs in $O(n^{100})$ time such that if the 3-SAT instance has a satisfying assignment, $M_2$ accepts and writes a satisfying assignment at the end of the tape.

(b) (3 points) Let $M_1 \ldots M_k$ be Turing machines show that the language

$$\{ \langle k, M_1 \ldots M_k, t, w \rangle \mid w \text{ is an instance of 3-SAT and one of } M_i \text{ writes a satisfying assignment to } w \text{ in at most } t \text{ steps} \}$$

can be decided by a Turing machine in $O(k^{2t^{10}}n^{10})$ time.

(c) (3 points) Under the assumption that there exists an unknown Turing machine encodable in $c_1$ bits that decides 3-SAT is in $O(n^{10})$ time, give implementation details for a Turing Machine $M$ that decides 3-SAT in $O(n^{10000})$ time.

Hint: the constant factor in the big-O is able to hide any function related $c_1$ and $c_2$, which independent of input sizes, and thus constants.