Exam 3 - Comments

True, False, or Unknown

1. For each of the following, circle one of the choices to indicate whether the statement is known to be *True*, is known to be *False*, or *Unknown* if its validity depends on something that is either currently unknown or not specified in the question.

Then, write a short justification to support your answer. When your answer is *Unknown*, your answer should make it clear what unknown the validity of the statement depends on (for example, that it is equivalent to a statement whose truth is currently unknown to anyone).

(a) The function, $XOR : \{0,1\}^* \rightarrow \{0,1\}$, which outputs the logical exclusive or of all the input bits, is in class P.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 5 words):

We saw a finite automaton (linear time) that implements XOR.

(b) The function, XOR (from the previous question), is in class NP.

Circle one:

True *False* *Unknown*

Justification (3 symbols):

$P \subseteq NP$.

(c) If a function A in NP has an exponential lower bound (e.g., requires $\Omega(2^n)$ time to compute), then no function in NP-Complete can be computed in polynomial time.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 3 sentences):

If we prove that some $A \in$ NP requires exponential time to compute, this means that $P \neq NP$ and that no function in NP-Complete can be computed in polynomial time.

Proving Uncomputability

2. In this question, your goal is to show that the function $REENTERS$ defined below is uncomputable.

> **Input:** A string w that describes a Turing Machine.
>
> **Output:** **1** if the machine described by w would re-enter its start state when executed on a tape that is initially all blank. Otherwise, **0**.

That is, a machine which computes $REENTERS$ outputs **1** when the input describes a Turing Machine which, when run on a blank tape, enters the start state as a result of some transition.

(a) Which strategy would show that $REENTERS$ is uncomputable? (Circle one, no explication needed.)

Use a machine that computes $REENTERS$ to compute $HALTS$.	Use a machine that computes $HALTS$ to compute $REENTERS$.

(b) Employ the strategy you chose in the previous question to show that $REENTERS$ is uncomputable.

We need to show that if there is a machine, M_R that computes $REENTERS$, we can use M_R to compute $HALTS$. Recall that $HALTS$ takes in a description of a Turing Machine w, and outputs **1** if w would halt running on the empty tape, and **0** otherwise. (A variation on $HALTS$ takes in both a machine description and an input x, and outputs **1** iff the machine described by w would halt on input x. You could use either verison here since we did not specific, but it is easy to convert between them by adding a sub-machine that writes x on the tape to the description of the machine.)

So, our goal is to implement $HALTS(w)$ as $REENTERS(w')$. We can do this by transforming the machine represented by w into the machine w' where the original start state in w is replaced by a new state s_R, which just transitions to the original start state of w. Every transition in w which would halt is replaced with a transition to state s_R. Since these are the only transitions to state s_R, the machine w' will only re-enter s_R if the machine represented by w would halt. Thus, the output of $REENTERS(w')$ is exactly the necessary output of $HALTS$. Hence, with M_R we could implement a machine M_{HALTS} that computes halts but just doing the input transformation described above and running M_R on the reslting w'.

Complexity Classes

3. For some function $F : \{0,1\}^* \to \{0,1\}$, given that each of the following is True, identify the first class that F is *guaranteed* to belong to. We list the classes so each is a subset of the class to its right, so circle the leftmost class which F must belong to. If it is not guaranteed to belong to any class, don't circle anything. Briefly justify your choice.

(a) There is an algorithm that computes F with running time in $\Theta(n^{5.5})$.

$$F \in \mathsf{TIME}_{\mathsf{TM}}(O(n^5 \log n)) \qquad \underline{F \in \mathsf{P}} \qquad F \in \mathsf{NP}$$

Justification (≤ 3 sentences):

We know $F \notin \mathsf{TIME}_{\mathsf{TM}}(O(n^5 \log n))$ since $n^{5.5}$ grows faster than $n^5 \log n$ which means there is no function in $\Theta(n^{5.5}$ which is also in $O(n^5 \log n)$.

We know $F \in \mathsf{P}$ since all functions in $\Theta(n^{5.5})$ are polynomials, so the running time of F must be in P.

(b) There is a function $R \in \mathsf{TIME}_{\mathsf{TM}}(O(n^{3102}))$ such that, for all $x \in \{0,1\}^n$ that is a well-formed input to 3-SAT, 3-$SAT(x) = F(R(x))$.

$$F \in \mathsf{NP\text{-}Complete} \qquad \underline{F \in \mathsf{NP\text{-}Hard}} \qquad F \in \mathsf{Computable}$$

Justification (≤ 4 sentences):

We don't know if $F \in \mathsf{NP\text{-}Complete}$ since that requires $F \in \mathsf{NP}$ and there is nothing here to allow us to conclude that.

We do know $F \in \mathsf{NP\text{-}Hard}$ since there is a polynomial-time reduction that allows us to use F to compute 3-SAT, which is known to be in NP-Hard.

Always, Sometimes, Never

4. For a function $f : \{0,1\}^{3102} \rightarrow \{0,1\}$ that can be implemented by a NAND circuit with s gates, which of the statements that follow would be *Always True, Possibly True* (meaning there are some functions f for which the statement is true and others for which is it false), or *Never True* (circle one option). Give a brief statement to justify your answer.

(a) f is computable

Always True *Possibly True* *Never True*

Justification ($\leq$ 5 words):

It can be implemented by a NAND circuit with s gates.

(b) $f \in \mathsf{NP}$

Always True ***Possibly True*** ***Never True***

Justification ($\leq$ 5 words):

Any answer could be justified here.

Since f is a finite function, it is not in NP which is a subset of the $\{0,1\}^* \rightarrow \{0,1\}$ functions.

We also accepted other answers that assumed a relaxed definition of NP to include partial functions. In this cases, $f \in \mathsf{NP}$ since it is a finite function which can be computed in a fixed (not growing with any input size) number of steps.

(c) There is some function $g : \{0,1\}^{3102} \rightarrow \{0,1\}$ that can be implemented using $s + 10$ NAND gates, but cannot be implemented using s NAND gates.

Always True ***Possibly True*** *Never True*

Justification ($\leq$ 3 sentences):

Because of the size hierarchy theory, we know this is true for some values of s. But, we also know that it is not true for other values of s. For example, if s is large enough to compute all $\{0,1\}^{3102} \rightarrow \{0,1\}$ functions, then there is no function g that cannot be computed with s gates but can be computed with more gates.

Constant Time

5. Show whether $O(1) = \Theta(1)$. (It is left up to you to determine if the two sets are equivalent, and provide a convincing proof to support your answer.)

One important thing this question was testing is your understanding that $O(1)$ and $\Theta(1)$ are both *sets of functions*. We often use them in computing to talk about running times and other resources like memory required to compute functions, but the notations themselves (and we provided the definitions on the second page) are just defining sets of functions. The 1 in these notation is the constant function that maps every natural number input to 1.

So, $O(1)$ is the set of functions that grow no faster than the constant function and $\Theta(1)$ is the set of functions that grow as fast as the constant function. Lots of functions grow slower than the constant function — this just means that as the input increases, the output decreases. An example is $f(n) = 1/n$. Hence, $O(1) \neq \Theta(1)$.

To show this more formally, we show (1) $1/n \in O(1)$ and (2) $1/n \notin \Theta(1)$.

(1) $1/n \in O(n)$: Choose $c = 1$ and $n_0 = 1$. Then, $\forall n > n_0, 1/n \leq 1$. (In the given definition of O, this is using $g(n) = 1$ and $f(n) = 1/n$.

(2) $1/n \notin \Theta(n)$: Since $\Theta(n) = O(n) \cap \Omega(n)$, we need to show $1/n \notin \Omega(n)$. This means that for any choice of c and n_0, we can always find an n that invalidates $f(n) \geq cg(n)$, which means finding an n such that $f(n) < cg(n)$. Substituting $f(n) = 1/n$ and $g(n) = 1$, this is $1/n < c$. For any choice of c, we can choose $n = \lceil 1/c \rceil + 1$. (Since $n \in \mathbb{N}$, we need the ceiling operator to round up to a natural number, and the $+1$ to make sure the denominator exceeds $1/c$ to ensure the inequality holds.)

3-TAUT

We know that 3-SAT is NP-Complete, where 3-SAT requires determining whether there exists at least one way to assign Boolean values to each variable in a 3-CNF formula so that the formula evaluates to True.

For this question we will show 3-TAUT is NP-Hard. This problem requires determining whether *every* assignment causes a 3-DNF formula (see definitions page) to evaluate to True (i.e., no assignments will cause the formula to evaluate to False).

6. Show 3-TAUT is NP-Hard.

To show a problem is NP-Hard, we need to show that an implementation of that problem could be used to solve every problem in NP. This can be done by showing that an implementation could solve any one problem in NP-Complete. As suggested by the problem, a straightforward reduction is to use 3-TAUT to implement 3-SAT.

For 3-TAUT, the output is **1** iff *all* assignments of the variables satisfying the 3-DNF formula:

$$\forall x_0 \in \{0,1\}, x_1 \in \{0,1\}, \ldots, x_n \in \{0,1\}. P_1(X) \vee P_2(X) \vee \cdots P_k(X)$$

where each P_i is one of the clauses in the formula, $P_i(X) = x_{i1} \wedge x_{i2} \wedge x_{i3}$ (the terms can also be negated).

For 3-SAT, the output is **1** iff there exists an assignment of variables that satisfies the 3-CNF formula:

$$\exists x_0 \in \{0,1\}, x_1 \in \{0,1\}, \ldots, x_n \in \{0,1\}. S_1(X) \wedge S_2(X) \wedge \cdots S_k(X)$$

where each S_i is a CNF clause, $S_i(X) = x_{i1} \vee x_{i2} \vee x_{i3}$ (the terms can also be negated).

We can use De Morgan's laws to turn CNF into DNF by negating the clauses. Then, as long as the DNF is not a tautology, there is some satisfying assignment. So the output of 3-SAT is the negation of the output of 3-TAUT to the negated formula.

For example, the CNF formula $(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3})$ is satisfied iff this DNF formula is not a tautology: $(\bar{x_1} \wedge x_2 \wedge \bar{x_3}) \vee (x_1 \wedge \bar{x_2} \wedge x_3)$.

7. 3-TAUT is not known to belong to NP. Give an intuitive reason why it is difficult to show that 3-TAUT belongs to NP.

It seems difficult to show that 3-TAUT belongs to NP since the output **1** means there is no assignment of variables that makes the formula false. There is no obvious witness for this (unlike for 3SAT where the satisfying assignment is a witness). It is not known if there is a polynomial time witness for 3-TAUT, and finding one would solve a longstanding open problem.

