

1 Short Answer I (8 questions) - 16 points

For each of the problems circle *true* if the statement is *always* true, circle *false* otherwise. There is no partial credit for these questions.

- (a) If A is a NP-Complete language and B is a NP-Hard language, then $A \leq_p B$.

Solution: ☒ True ☐ False



- (b) If A is a NP-Complete language then $A \leq_p SAT$ and $SAT \leq_p A$.

Solution: ☒ True ☐ False



- (c) If $A \leq_p B$ and B is NP-Complete, then A is NP-Complete.

Solution: ☒ True ☐ False



- (d) If $A \leq_p B$ and A is NP-Complete, then B is NP-Complete.

Solution: ☒ True ☐ False



- (e) Every decidable language is in NP.

Solution: ☒ True ☐ False



- (f) All NP problems are recursively enumerable.

Solution: ☒ True ☐ False



- (g) Every regular language is in P

Solution: ☒ True ☐ False



- (h) If A and B are both in NP, then $A \leq_p B$

Solution: ☒ True ☐ False




2 Short Answer II (3 questions) - 9 points

For each of the problems circle all the answers that apply. There is no partial credit for these questions. Points are not necessarily divided evenly among all possible choices.


- (a) Assume X is the problem that finds the *hamiltonian cycle of minimum length* given a directed, weighted graph G . Circle the complexity classes this problem belongs to:

Solution:	P	NP	<input checked="" type="radio"/> NP-hard	<input type="radio"/> NP-complete
	<input checked="" type="radio"/> decidable	<input type="radio"/> undecidable		




- (b) The Tautology problem is the problem of determining if a 3SAT evaluates to true under every possible assignment to its variables. Tautology belongs to:

Solution:	P	NP	<input type="radio"/> NP-hard	<input type="radio"/> NP-complete
	<input checked="" type="radio"/> decidable	<input type="radio"/> undecidable		



- (c) Recall the primality problem is problem of determining if a number (n) is prime (has factors $< n$). Primality belongs to:

Solution:	P	<input checked="" type="radio"/> NP	<input type="radio"/> NP-hard	<input type="radio"/> NP-complete
	<input checked="" type="radio"/> decidable	<input type="radio"/> undecidable		



3 Classification I (P/NP) - 15 points

Is the following problem in P, NP, or some combinations of complexity classes? For each of the following problems, circle all the complexity classes that problem belongs to. Whatever class it is in, prove it!

The 374 path problem (374P) asks given an undirected graph G , does G contain a path that visits atleast 374 vertices.

- INPUT: A graph G .
- OUTPUT: TRUE if there exists a path that is at least 374 vertices long. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle **all** that apply:

Solution: ☒ P ☒ NP ☐ NP-hard ☐ NP-complete

To prove that the 374 Path problem is in P we need to come up with an polynomial-time algorithm that solves the problem. There are many polynomial solutions and one of them is listed below.

The first thing to realize is that if there exists a path that is > 374 vertices long, then there is a path that is 374 vertices long. Hence, to solve this problem we simply need to know if there exists a path with 374 vertices. The number of possible paths with 374 vertices is n^{374} (you choose one of n vertices for every spot in the path).

Hence, to solve this problem you simply check each of the n^{374} paths to make sure if there is a edge between every one of the adjacent vertices. If there is, it means that path exists in the graph. This takes $O(n)$ time for each path. Therefore the brute force approach takes $O(n * n^{374}) = O(n^{375})$ which is polynomial time.

As the 374P is solvable in polynomial time, thereby classifying it within P, we state that it inherently qualifies as a member of the NP class as well.

Additionally, we can prove that a problem is in NP if we can prove that the solution or a YES instance of the problem can be verified in polynomial time. For 374P, given a specific path of at, we can easily check in polynomial time whether this path is valid solution. Imagine a 'certifier' that takes a proposed path as a 'certificate'. The certifier then checks if each consecutive pair of vertices in the path is connected by an edge in the graph and checks if the count of vertices in the path is > 374 . This process would take linear time relative to the length of the path, which is a polynomial-time operation.

Note: Most of you believed this to be NP-complete because it looks similar to the longest path problem. The reason this doesn't work is that you have to show the reduction $\text{LONGESTPATH} \leq_p 374P$. In other words, for the \leq direction you have to determine if a graph has a path of weight $> k$ using a machine that only tells you if the graph has a path with more than 374 vertices. I don't think that is possible (definitely not in polynomial time).

Additionally, there's a common misconception regarding the classification of problems as NP, that if a problem can be reduced from an NP-complete problem, it automatically belongs to NP. This is incorrect. While such a reduction does indicate the hardness of the problem, it doesn't confirm its membership in NP. To categorically place a problem in NP, we need to

establish the existence of a polynomial-time verifier or certifier as given above.



4 Classification II (P/NP) - 15 points

Is the following problem in P, NP, or some combinations of complexity classes? For each of the following problems, circle all the complexity classes that problem belongs to. Whatever class it is in, prove it!

The multi-solution SAT (**MultiSAT**) problem asks whether a SAT problem has multiple satisfiable truth assignments.

- INPUT: A SAT formula ϕ .
- OUTPUT: TRUE if there exists atleast two distinct variable assignments that satisfy this formula. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle **all** that apply:

Solution: P

NP

NP-hard

NP-complete

To show NP-hard, we need to reduce a known NP-hard problem to MULTISAT. Given that this is a SAT variant, the obvious reduction would be $\text{SAT} \leq_P \text{MULTISAT}$. So we need to transform an instance of SAT (ϕ) into an instance of MULTISAT (ϕ_M) so that if ϕ is satisfiable, then ϕ_M is satisfiable and if ϕ is not satisfiable, then ϕ_M is also not satisfiable.

Basically we just got to double the number of satisfying assignments in ϕ . An easy way to do this is by adding variable x to ϕ_M and saying $\phi_M = \phi \wedge (x \vee \bar{x})$. Therefore, whatever satisfying assignment ϕ has, ϕ_M also has with x equal to 0 or 1.

To prove the problem is in NP you have to show that a YES instance is checkable in polynomial time. In this case we can simply define a certificate that is a variable assignment and a certifier which traverses ϕ_M and makes sure all the clauses have atleast one literal which is true.

Common mistake: Notice that we want that when ϕ has one satisfying assignment, ϕ_M has at least 2. Not that when ϕ_M has a satisfying assignment, ϕ has at least 2. Because if we do that, we are reducing from MULTISAT to SAT. A common modification attempted like this is $\phi_M = \phi \wedge (\hat{\phi})$ which doesn't work for multiple reasons. ■

5 Classification III (P/NP) - 15 points

Is the following problem in P, NP, or some combinations of complexity classes? For each of the following problems, circle all the complexity classes that problem belongs to. Whatever class it is in, prove it!

$HALT_{TM}$ you are given a turing machine $\langle M \rangle$ and must determine if it halts on a empty input.

- INPUT: A TM $\langle M \rangle$.
- OUTPUT: TRUE if the will halt on an empty input. FALSE otherwise.

Which of the following complexity classes does this problem belong to? Circle **all** that apply: You must justify (prove) your answer!

Solution: P NP NP-hard NP-complete

Right off the bat, we know HALT is undecidable so it is probably not in P or NP or NP-complete.

Now we just got to prove it is NP-hard. We can do this the same way we prove any other problem is NP-hard, reduction from a known NP-hard problem.

Probably the easiest way to do this is to show a $SAT \leq_p HALT$. We can construct a TM that does the following:

$M'(x)$: Loop trying every variable assignment on ϕ if ϕ is satisfied return TRUE Loop forever

M' only halts if ϕ is satisfiable. Hence, we can use the oracle for HALT to determine if ϕ is satisfiable or not. The reduction only requires us to encode the TM which takes constant time and hence, HALT is Np-hard. ■

6 Classification I (Decidability) - 15 points

Are the following languages decidable? For each of the following languages,

- Circle one of "decidable" or "undecidable" to indicate your choice.
- If you choose "decidable", prove your choice correct by describing an algorithm that decides that language. If you choose "undecidable", prove your choice correct by giving a reduction proving its correctness.
- Regardless of your choice, explain *briefly* (i.e., in 3 sentences maximum, diagrams, *clear* pseudo-code) why the proof of the choice you gave is valid.

$\text{REACHQ}_{TM} = \{ \langle M, w, q \rangle \mid M \text{ is a TM } (= \langle Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle) \text{ and will enter state } q \in Q, \text{ on input } w \}$

$\Sigma = \{0, 1\}$

Solution: decidable undecidable

This is similar to a lab and lecture question and we do a reduction from the accept language:

$$A_{TM} \Rightarrow \text{REACHQ}_{TM}$$

The reduction is as follows. We can simply find every accepting state in M (input to A_{TM}) and run the oracle for REACHQ_{TM} to see if it reaches that accept state. Doing this for every accept state in M will tell you if M accepts w (i.e the thing A_{TM} is looking for).

$DEC_{ATM}(M, w)$:

$\forall q \text{ in } q_{acc}$

if $ORAC_{\text{REACHQ}_{TM}}(< M, w, q >)$

return TRUE

return FALSE

Clearly this reduction is valid and therefore REACHQ is atleast as hard as A_{TM} . Since A_{TM} is undecidable, REACHQ must also be undecidable. ■

7 Classification II (Decidability) - 15 points

Are the following languages decidable? For each of the following languages,

- Circle one of "decidable" or "undecidable" to indicate your choice.
- If you choose "decidable", prove your choice correct by describing an algorithm that decides that language. If you choose "undecidable", prove your choice correct by giving a reduction proving its correctness.
- Regardless of your choice, explain *briefly* (i.e., in 3 sentences maximum, diagrams, *clear* pseudo-code) why the proof of the choice you gave is valid.

$$\text{ACCEPT}_{374_{TM}} = \{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and accepts } w \text{ in 374 steps.} \}$$

$$\Sigma = \{0, 1\}$$

Solution: decidable undecidable

Pretty much exactly a lab/lecture problem. Since there is a upperbound to the number of steps, you can simply run the Turing machine for 374 steps. If it accepts, then accept, otherwise reject. There is no possibility to loop forever and therefore, the problem(language) is decidable. ■