1. ***True, false, or unknown.*** For each of the subproblem below, read the statement ***very carefully***, and decide if the statement is *true*, *false*, or *unknown*. A statement is ***unknown*** if its true/false value is currently open. *You don't need to justify your answers*.

   Let M be a standard Turing machine (with a single tape and one head). Also, we always assume the *Church-Turing thesis* that everything computable can be implemented by a Turing machine. We *do not* assume that $P \neq NP$, unless specified otherwise.

   (1) Any Turing machine can be simulated by another Turing machine that, when moving right, *never* moves its head to the right three times in a row.

   Solution: True. Implement the head movement of a standard TM one step to the right by moving two steps to the right then one step to the left. ■

   (2) Any Turing machine can be simulated by another Turing machine that, when moving right, *always* moves its head to the right three times in a row.

   Solution: True. Implement the head movement of a standard TM one step to the right by moving three steps to the right then two steps to the left. ■

   (3) If a physical system can simulate the Turing machine, then theoretically it can run Doom (or Zoom).

   Solution: True. This is a consequence of the Church-Turing thesis. ■

   (4) A nondeterministic Turing machine can be implemented in real life.

   Solution: True. With an exponential slowdown, one can simulate an NTM using standard TM by trying all the possible guesses made by the NTM. False. Finite memory in reality. Unknown. New physic law? ■

   (5) $\{\langle M\rangle : M$ accepts $\langle M\rangle$ in time at most $|\langle M\rangle|\}$ is decidable.

   Solution: True. Use a universal TM to simulate M on reading $\langle M\rangle$ for exactly $|\langle M\rangle|$ many steps; if it ever accepts then accept, otherwise reject. ■

   (6) $\{\langle M\rangle : \langle M\rangle$ is a valid description of some Turing machine$\}$ is undecidable.

   Solution: False. Checking if $\langle M\rangle$ is a proper encoding of a TM can be done for any reasonable encoding (for example, the TM encoding we described in class). ■

   (7) $\{\langle M\rangle : M$ accepts any valid description of some Turing machine$\}$ is undecidable.

   Solution: True. This is equivalent in asking whether the language of M is

   $$\left\{\left\langle M'\right\rangle : \left\langle M'\right\rangle \text{ is a valid description of some Turing machine}\right\};$$

   checking any nontrivial property of $L(M)$ is undecidable. ■

   (8) The 3Color problem is decidable.

   Solution: True. 3Color can be decided by an NTM (or, by DTM in exponential time). ■

   (9) If language L is in NP, then any instance w has a proof that $w \in L$ of polynomial length.

   Solution: False. Only the yes-instances of L has a polynomial-size proof. ■

   (10) Cook-Levin theorem implies that if we can solve the Clique problem in polynomial time, then we can solve the 3Sat problem in polynomial time.

   Solution: True. Cook-Levin theorem implies that all NP-complete problems are equivalent; in particular, because Clique is NP-complete (and thus NP-hard), by definition solving Clique in polynomial time will solve all NP problems in polynomial time, including 3Sat. ■

(11) A nondeterministic Turing machine can be emulated by a deterministic Turing machine with polynomial slowdown.

Solution: Unknown. The statement is equivalent to whether P = NP. ■

(12) If a language L is in NP, then the complement language $\overline{L} := \Sigma^* \setminus L$ is also in NP.

Solution: Unknown. The statement is equivalent to whether we can verify the UNSAT problem in polynomial time, or equivalently, if coNP = NP. ■

(13) NP-hard problems are the hardest problems in NP.

Solution: False. NP-hard problems might not even be in NP. ■

(14) To show that CLIQUE is NP-hard, it is sufficient to reduce CLIQUE to some other NP-hard problem like INDEPENDENTSET.

Solution: False. Wrong direction; also the reduction has to run in polynomial time. ■

(15) There is a polynomial-time reduction from the shortest path problem to the Hamiltonian path problem.

Solution: True. Shortest path problem can be solved in polynomial time, and thus lies in P which is a subset of NP. By Cook-Levin theorem all problems in NP are polynomial-time reducible to any NP-hard problem. ■

(16) There is a polynomial-time reduction from the Hamiltonian path problem to the shortest path problem.

Solution: Unknown. If there is such a reduction then P = NP, but we don't have a proof that this won't happen. ■

(17) The exponential-time hypothesis says that no NP-complete problem can be solved in sub-exponential time (that is, $2^{o(n)}$).

Solution: False. Many NP-hard problems like planar 3-coloring can be solved in $2^{O(\sqrt{n})}$ time. ■

(18) The strong exponential-time hypothesis implies 3SAT cannot be solved in 1.99^n time.

Solution: False. SETH only concerns about CNFSAT; in fact it is known that 3SAT can be solved in 1.31^n time. ■

(19) Assuming ETH, P is not equal to NP.

Solution: True. ETH is a stronger hypothesis than P $\neq$ NP. ■

(20) If we can find a clique of size 1000 in n^{999} time, then SETH is false.

Solution: Unknown. This is basically asking whether solving k-CLIQUE in $O(n^{k-\varepsilon})$ time is SETH-hard, which is open. We do know how to solve 1000-CLIQUE problem in n^{792} time, which will break SETH if the above statement is true. ■

Rubric: Everyone receives full points for the following subproblems: (4) is debatable, and (20) is technically not in scope (extra credit in Homework 6 Problem 2b).

Each correct answer gets 1 point; each incorrect answer gets −0.5 points; leaving the answer blank gets 0 points. The lowest total score is zero.

2. Consider the following problem:

   39Color
   - ***Input:*** *An n-vertex undirected graph G.*
   - ***Output:*** *Does G have a proper coloring (no two vertices of the same color share an edge) using at most 39 colors?*

   Prove that 39Color is NP-complete. Can you solve 39Color in $2^{o(n)}$ time assuming ETH?

   Solution: To prove that 39Color is NP-complete, we show that 39Color is both NP-hard and in NP. To certify that 39Color is in NP, the prover can provide a 39-coloring of the input graph G; the verifier first check if there are indeed at most 39 colors, then for each edge in G check if the colors on the two endpoints are indeed different. The whole check can be done in time linear to the number of vertices and edges in G.

   We will prove that 39Color is NP-hard by reducing from the NP-hard problem 3Color. Given an undirected graph H as an instance to 3Color, we construct graph G as the instance for 39Color as follows:

   - Insert a 36-clique C to H and connect all vertices between C and H.

   Constructing graph G takes time linear to the size of H, and thus the reduction can be performed in polynomial time.

   Now we prove that H has a 3-coloring if and only if G has a 39-coloring.

   - For the only if direction, let $f(H)$ be a 3-coloring of H. Color the 36-clique C with the rest of the 36 colors that weren't presented in the 3-coloring f. This implies G is 39-colorable.
   - For the if direction, let $g(G)$ be a 39-coloring of G. Since C is a subgraph of G as a 36-clique, g must spend 36 different colors on C. Because every vertex in H is adjacent to every single vertex in C, g must have used at most 3 colors on H. This means that H has a 3-coloring.

   This concludes the reduction from 3Color to 39Color and thus proving that 39Color is NP-hard. Finally, as the constructed graph G is at most linear in size to H, the ETH-hardness result for 3Color implies that 39Color cannot be solved in $2^{o(n)}$ time assuming ETH. ■

   Rubric: 10 points in total.
   - NPC = NP + NP-hard: 1 point
   - Proving 39Color is in NP: 2 points
   - Proving 39Color is NP-hard: 5 points, standard scale; maximum 1 point if the direction of reduction is incorrect; maximum 2 points if coloring is part of the input.
   - Arguing 39Color cannot be solved in $2^{o(n)}$ time: 2 points; 0 point if no explanation given; maximum 1 point if the size of the 39Color instance is not mentioned.

3. Consider the following problem:

   LongestPath
   - ***Input:*** *An n-vertex undirected graph G.*
   - ***Output:*** *Compute a longest path in G.*

   (a) Design an algorithm to solve LongestPath in polynomial time assuming we are given a subroutine that solves the decision version of LongestPath ("Given a graph G and an integer k, is there a path of length at least k?") in polynomial time.

Solution: Let ORACLE(G, k) be the oracle deciding if there is a path in G of length at least k in polynomial time. Consider the following algorithm solving LONGESTPATH on an n-vertex graph G:

```
LONGESTPATH($G$):
  let $k$ be the largest integer such that ORACLE$(G,k) = yes$
  while $G$ is not a path of length $k$:
      for each edge $e$ in $G$:
          if ORACLE$(G - e, k) = yes$:
              $G \leftarrow G - e$
              break from inner-loop
  output $G$ ⟨⟨as a path of length $k$⟩⟩
```

The LONGESTPATH algorithm is correct because the oracle queries certify that every graph G we have examined must contain a path of length k. The outer while-loop only terminates when G itself is a path of length k.

The algorithm runs in polynomial time because at most n^2 inner-loops will be executed, each taking at most polynomial time because of the oracle query. ■

(b) Either solve LONGESTPATH in polynomial time, or prove that LONGESTPATH is NP-hard.

Solution: We prove that LONGESTPATH is NP-hard by describing a polynomial-time reduction from the famous NP-hard problem HAMILTONIANPATH to the decision version of LONGESTPATH. The decision version of LONGESTPATH reduces to computing the actual longest path in polynomial time because we can simply check if the computed path has length at least the parameter k; this implies that LONGESTPATH is NP-hard as well.

Let G be an undirected graph as an instance to HAMILTONIANPATH. Set the parameter k to be $n-1$ (if we count the path length by the number of edges within). This can be easily done in time linear to G.

Now we prove that G has a Hamiltonian path if and only if the longest path in G is at most $k = n-1$.

- For the only if direction, any Hamiltonian path in G must have length $n-1$.
- For the if direction, any simple path of length $n-1$ in G must visit every vertex, and thus a Hamiltonian path by definition.

This concludes the reduction from HAMILTONIANPATH to LONGESTPATH and thus proving that LONGESTPATH is NP-hard. ■

Rubric: Standard 5-point grading scale for each subproblem. Maximum 1 point if the direction of reduction is incorrect. Maximum 2 points if one assumes G has a Hamiltonian path.

4. Let Stack be the class of languages accepted by DFAs with a single *stack*.

   - The stack has infinite-depth, supporting the *push* and *pop* operations. The stack always follows the first-in-last-out rule.

   Let Queue be the class of languages accepted by DFAs with a single *queue*.

   - The queue has infinite-length, supporting the *push* and *pop* operations. The queue always follows the first-in-first-out rule.

   Compare the two classes Stack and Queue. Which machine model is more powerful?

   [Be careful that neither machine models here have any extra memory, not even one single-bit. Everything has to be stored inside the stack or the queue.]

Solution: We will show how to emulate a stack-machine using a queue-machine, by mimicking the push and pop operations on a stack with queue operations. Intuitively, the opening of the stack is matched with the pop-end of the queue; and we will keep a separate "end" symbol in the queue to represent the bottom of the stack.

- Before start, we push an "end" symbol not in the input alphabet into the queue.
- Whenever the stack pops, we pop the queue as well and obtain the element.
- Whenever the stack pushes, we push the new element into the queue, follow by cycling through the queue and popping each element and pushing it back from the other end of the queue, until the "end" symbol is reached, which we also popped and push it back.

The only modification to the DFA is the pushing of the extra "end" symbol at the start, and the replacement of each stack push with the cycling process described above. Therefore we have successful emulate a stack-machine using a queue-machine. ■

Rubric: There are multiple ways to emulate the stack using a queue. In particular, the elements in the stack can be kept in the opposite direction inside the queue. In that case the stack-push operation can be easily implemented by pushing into the queue. However the stack-pop requires a single-element buffer, recording the last symbol popped out from the queue, so that when the "end" symbol is popped out we know the element in the buffer is the one to be popped from the stack. Similar buffer is needed if one tries to emulate a TM using the queue-machine.

Standard 5-point grading scale, scaled to 10 points. It is sufficient to show that the queue-machine is at least *as powerful as* the stack-machine. It is also sufficient to prove that the queue-machine is equivalent to a Turing machine. If the emulation method requires a buffer:

- Maximum 8 points if the buffer is mentioned by the implementation is not explained.
- Maximum 6 points if no buffers are mentioned.