1. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, write "NO". For example:

   Read each statement *very* carefully; some of these are deliberately subtle!

Rubric: For each part: 1 point = ½ for correct answer + ½ for explanation. These are not the only correct explanations.

Which of the following statements are true?

(a) The solution to the recurrence $T(n) = 4T(n/2) + O(n^2)$ is $T(n) = O(n^2)$.

Solution: NO — The level sums of the recursion tree are equal. The correct solution is $T(n) = O(n^2 \log n)$. ■

(b) The solution to the recurrence $T(n) = 2T(n/4) + O(n^2)$ is $T(n) = O(n^2)$.

Solution: YES — The level sums of the recursion tree form a decreasing geometric series. ■

(c) Every directed acyclic graph contains at least one sink.

Solution: YES — Start at any vertex and follow edges until you can't any more. If you aren't done after $V + 1$ steps, you've repeated a vertex, which means the graph has a cycle. ■

(d) Given *any* undirected graph G, we can compute a spanning tree of G in $O(V + E)$ time using whatever-first search.

Solution: NO — If G is disconnected, it doesn't have a spanning tree. ■

(e) Suppose we want to iteratively evaluate the following recurrence:

$$What(i,j) = \begin{cases} 0 & \text{if } i > n \text{ or } j < 0 \\ \max \begin{Bmatrix} What(i, j-1) \\ What(i+1, j) \\ A[i] \cdot A[j] + What(i+1, j-1) \end{Bmatrix} & \text{otherwise} \end{cases}$$

We can fill the array $What[0..n, 0..n]$ in $O(n^2)$ time, by decreasing i in the outer loop and decreasing j in the inner loop.

Solution: NO — j should both be increasing. ■

Which of the following statements are true for ***at least one*** language $L \subseteq \{0,1\}^*$?

(f) $L^* = (L^*)^*$

Solution: YES — Concatenation is associative. Concatenating several concatenations of strings in L is the same as concatenating several strings in L. In particular, when $L = \emptyset$ or $L = \{\varepsilon\}$, we have $L^* = (L^*)^* = \{\varepsilon\}$. ■

(g) L is decidable, but L^* is undecidable.

Solution: NO — We can decide L^* by dynamic programming. Specifically, we can use the dynamic programming algorithm for text segmentation described in class, with the decision algorithm for L as the subroutine IsWord. ■

(h) L is neither regular nor NP-hard.

Solution: YES — $\{0^n 1^n \mid n \geq 0\}$. This language can be recognized in linear time. ■

(i) L is in P, and L has an infinite fooling set.

Solution: YES — $\{0^n 1^n \mid n \geq 0\}$. This language can be recognized in linear time! ■

(j) The language $\{\langle M \rangle \mid M \text{ accepts } L\}$ is undecidable.

Solution: YES — Rice's theorem! Specifically, when L is any *acceptable* language, Rice's theorem implies that this language is undecidable. But if L is *not* an acceptable language (for example, $L =$ SelfDiverge), then this language is empty and therefore *trivially* decidable! ■

2. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, write "NO".

   Read each statement *very* carefully; some of these are deliberately tricky!

   (Please remember to start your answers to this problem on a new page. Yes, this is really just a continuation of problem 1; we split it into two problems to make grading easier.)

Consider the following pair of languages:

- ACYCLIC := $\{\text{undirected graph } G \mid G \text{ contains no cycles}\}$
- HALFIND := $\{\text{undirected graph } G = (V, E) \mid G \text{ has an independent set of size } |V|/2\}$

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) The language HALFIND is actually NP-hard; **you do *not* need to prove that fact**.

Which of the following statements are true, assuming P $\neq$ NP?

(a) ACYCLIC is NP-hard.

> **Solution: NO** — WFS. We can decide whether a given graph has a cycle in $O(V + E)$ time using whatever-first search, so ACYCLIC $\in$ P. ■

(b) HALFIND $\setminus$ ACYCLIC $\in$ P
(Recall that $X \setminus Y$ is the subset of elements of X that are not in Y.)

> **Solution: NO** — It's still NP-hard. HALFIND $\cap$ ACYCLIC is the set of all acyclic graphs (also known as *forests*) with an even number of vertices, which is in P. Thus, we can reduce HALFIND to HALFIND $\setminus$ ACYCLIC in polynomial time as follows: If the input graph is acyclic, check if the number of vertices is even; if not, call the HALFIND $\setminus$ ACYCLIC subroutine. ■

(c) HALFIND is decidable.

> **Solution: YES** — Brute force. For every subset of $V/2$ vertices, check whether that subset is independent. The resulting algorithm runs in $O(2^V E)$ time, which is exponential, but that's fine. ■

(d) A polynomial-time reduction from HALFIND to ACYCLIC would imply P=NP.

> **Solution: YES** — HALFIND is NP-hard, but ACYCLIC is in P. ■

(e) A polynomial-time reduction from ACYCLIC to HALFIND would imply P=NP.

> **Solution: NO** — The reduction is in the wrong direction. ■

Suppose there is a ***polynomial-time*** reduction from some language A over the alphabet $\{0, 1\}$ to some other language B over the alphabet $\{0, 1\}$. Which of the following statements are true, assuming P $\neq$ NP?

(f) A is a subset of B.

Solution: NO — Reductions have nothing to do with subsets. In particular, there is a trivial polynomial-time reduction from $A = \Sigma^*$ to $B = \emptyset$. ■

(g) If $B \in P$, then $A \in P$.

Solution: YES — To solve A, run the reduction and then the algorithm for B. ■

(h) If B is NP-hard, then A is NP-hard.

Solution: NO — The reduction is in the wrong direction. ■

(i) If B is decidable, then A is decidable.

Solution: YES — To solve A, run the reduction and then the algorithm for B. ■

(j) If B is regular, then A is decidable.

Solution: YES — If B is regular, then B is decidable; see the previous statement. ■

3. Suppose you are asked to tile a $2 \times n$ grid of squares with dominos (1×2 rectangles). Each domino must cover exactly two grid squares, either horizontally or vertically, and each grid square must be covered by exactly one domino.

   Each grid square is worth some number of points, which could be positive, negative, or zero. The ***value*** of a domino tiling is the sum of the points in squares covered by vertical dominos, *minus* the sum of the points in squares covered by horizontal dominos.

   Describe and analyze an efficient algorithm to compute the largest possible value of a domino tiling of a given $2 \times n$ grid. Your input is an array $Points[1..2, 1..n]$ of point values.

Solution: For any index i, let $MaxValue(i)$ denote the largest value of a domino tiling of the first i columns of the given grid. We need to compute $MaxValue(n)$.

This function obeys the following recurrence:

$$MaxValue(i) = \begin{cases} 0 & \text{if } i = 0 \\ Points[1,1] + Points[2,1] & \text{if } i = 1 \\ \max \left\{ \begin{array}{c} MaxValue(i-1) + Points[1,i] + Points[2,i], \\ MaxValue(i-2) - Points[1,i] - Points[2,i] \\ -Points[1,i-1] - Points[2,i-1] \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a one-dimensional array $MaxValue[1..n]$. Finally, we can fill the array from left to right (increasing i) in ***$O(n)$ time***. ■

Rubric: 10 points: Standard dynamic programming rubric. This is not the only correct solution.

4. Submit a solution to *exactly one* of the following problems. Don't forget to tell us which problem you've chosen!

   (a) Let Φ be a boolean formula in conjunctive normal form, with exactly three literals per clause (or in other words, an instance of 3SAT). ***Prove*** that it is NP-hard to decide whether Φ has a satisfying assignment in which *exactly half* of the variables are TRUE.

Solution: We reduce from 3SAT.

Let Φ be an arbitrary 3CNF formula with n variables and m clauses. We create a new 3CNF formula Φ' by making a clone of Φ (with n new variables, but the same clause pattern), and then negating every literal. For example, if Φ contains the clause $(x_3 \vee x_7 \vee \overline{x}_4)$, then Φ' contains the corresponding clause $(\overline{y}_3 \vee \overline{y}_7 \vee y_4)$. The combined 3CNF formula $\Phi \wedge \Phi'$ has $2n$ variables and $2m$ clauses.

We claim that Φ has a satisfying assignment if and only if $\Phi \wedge \Phi'$ has a satisfying assignment with exactly half of the variables set to TRUE. (To save space, I'll call this a *balanced* satisfying assignment.)

$\Longrightarrow$ Suppose $(x_1, x_2, \ldots, x_n)$ is a satisfying assignment for Φ. We extend this assignment by setting $y_i = \overline{x}_i$ for each index i. Each clause in Φ is satisfied by the original variables x_i; it follows that the corresponding clause in Φ' is satisfied by the negated variables y_i. The combined assignment $(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ is a balanced satisfying assignment for $\Phi \wedge \Phi'$.

$\Longleftarrow$ Suppose $(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ is a balanced satisfying assignment for $\Phi \wedge \Phi'$. The clauses in Φ use only the variables x_i, so the restricted assignment $(x_1, x_2, \ldots, x_n)$ satisfies Φ.

Given Φ, we can easily construct Φ' in polynomial time. ■

Rubric: 10 points: Standard NP-hardness rubric. This is not the only correct solution.

(b) Let $G = (V, E)$ be an arbitrary undirected graph. Recall that a *proper 3-coloring* of G assigns each vertex of G one of three colors—red, blue, or green—so that every edge in G has endpoints with different colors. ***Prove*** that it is NP-hard to decide whether G has a proper 3-coloring in which *exactly half* of the vertices are red.

Solution: We reduce from the standard 3Color problem.

Let $G = (V, E)$ be an arbitrary graph, and let $n = |V|$. We construct a new graph G' by ading n isolated vertices to G. More formally, $G' = (V', E')$, where

- $V' = V \cup W$, where W is a set of n new vertices.
- $E' = E$.

We claim that G has a proper 3-coloring if and only if G' has a proper 3-coloring in which half the vertices are red.

$\Longrightarrow$ Suppose G has a proper 3-coloring with r red vertices, b blue vertices, and g green vertices. We can extend this to a proper 3-coloring of G' by coloring any $n - r$ vertices in W red, and the remaining r vertices in W arbitrarily blue or green. In the resulting 3-coloring of G', exactly half of the vertices are red.

$\Longleftarrow$ Suppose G' has a proper 3-coloring with exactly n red vertices. The restriction of this coloring to G is a proper 3-coloring of G.

Given G, we can easily construct G' in polynomial time. ■

Rubric: 10 points: Standard NP-hardness rubric. This is not the only correct solution.

5. Suppose you are given a height map of a mountain, in the form of an $n \times n$ grid of evenly spaced points, each labeled with an elevation value. You can safely hike directly from any point to any neighbor immediately north, south, east, or west, but only if the elevations of those two points differ by at most Δ. (The value of Δ depends on your hiking experience and your physical condition.)

Describe and analyze an algorithm to determine the longest hike from some point s to some other point t, where the hike consists of an uphill climb (where elevations must increase at each step) followed by a downhill climb (where elevations must decrease at each step). Your input consists of an array $Elevation[1..n, 1..n]$ of elevation values, the starting point s, the target point t, and the parameter Δ.

Solution: We construct a directed graph $G = (V, E)$ as follows. (To save space, I'll write $Elev[i, j]$ everywhere instead of $Elevation[i, j]$.)

- $V = \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\} \times \{\uparrow, \downarrow\}$. Each vertex $(i, j, \uparrow)$ denotes being located at row i and column j during the uphill portion of the hike; each vertex $(i, j, \downarrow)$ denotes being located at row i and column j during the down portion of the hike.
- $E = E_\uparrow + E_\wedge + E_\downarrow$, where edges in $E_\uparrow$ represent steps allowed during the uphill potion of the hike; edges in $E_\downarrow$ represent steps allowed during the downhill potion of the hike; and edges in $E_\wedge$ represent switching from uphill to downhill at some vertex. Finally, we have

$$E_\uparrow = \left\{ (i,j,\uparrow) \to (i',j',\uparrow) \;\middle|\; \begin{array}{c} |i-i'| + |j-j'| = 1 \text{ and} \\ Elev[i,j] < Elev[i',j'] \le Elev[i,j] + \Delta \end{array} \right\}$$

$$E_\wedge = \left\{ (i,j,\uparrow) \to (i,j,\downarrow) \;\middle|\; (i,j,\uparrow) \in V \right\}$$

$$E_\downarrow = \left\{ (i,j,\downarrow) \to (i',j',\downarrow) \;\middle|\; \begin{array}{c} |i-i'| + |j-j'| = 1 \text{ and} \\ Elev[i,j] > Elev[i',j'] \ge Elev[i,j] - \Delta \end{array} \right\}$$

Altogether G has $2n^2$ vertices and at most $9n^2$ edges.

G is a directed *acyclic* graph. Every edge in $E_\uparrow$ increases elevation; every edge in $E_\downarrow$ decreases elevation; and every path contains at most one edge in $E_\wedge$.

We nee to compute the longest path from $(s, \uparrow)$ to $(t, \downarrow)$. Because G is a dag, we can compute this path in $O(V + E) = \mathbf{O(n^2)}$ ***time*** using the depth-first search algorithm (or equivalently, the topological sort and dynamic programming algorithm) described in class. ■

Rubric: 10 points: Standard graph reduction rubric.

6. Recall that a ***run*** in a string $w \in \{0,1\}^*$ is a maximal substring of w whose characters are all equal.

   (a) Let L_a denote the set of all strings in $\{0,1\}^*$ where every 0 is followed immediately by at least one 1.

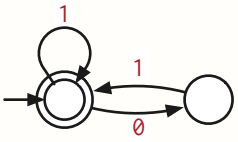
   For example, L_a contains the strings 010111 and 1111 and the empty string ε, but does not contain either 001100 or 1111110.

   - Describe a DFA or NFA that accepts L_a ***and***
   - Give a regular expression that describes L_a.

   (You do not need to prove that your answers are correct.)

Solution:

- The following NFA accepts L_a. This figure also describes a DFA, but only if we state that all unspecified transitions go to a dump state.



- L_a is described by the regular expression $1^*(011^*)^*$.

■

Rubric: 5 points = 2½ for DFA/NFA (standard rubric) + 2½ for regular expression (standard rubric). These are not the only correct solutions.

(b) Let L_b denote the set of all strings in $\{0,1\}^*$ whose run lengths are increasing; that is, every run except the last is followed immediately by a *longer* run.

For example, L_b contains the strings 0110001111 and 1100000 and 000 and the empty string ε, but does not contain either 000111 or 100011.

Prove that L_b is not a regular language.

Solution: Let F be the infinite language 00^*.
Let x and y be arbitrary strings in F.
Then $x = 0^i$ and $y = 0^j$ for some positive integers $i \neq j$.
Without loss of generality, we can assume that $i < j$. (Otherwise, swap x and y.)
Let $z = 1^j$.

- Then $xz = 0^i 1^j \in L_b$ because $0 < i < j$.
- But $yz = 0^j 1^j \notin L_b$ because both runs in yz have the same length.

We conclude that F is a fooling set for L_b.
Because F is infinite, L_b cannot be regular. ■

Solution: Let F be the infinite language 0^*.
Let x and y be arbitrary strings in F.
Then $x = 0^i$ and $y = 0^j$ for some positive integers $i \neq j$.
Without loss of generality, we can assume that $i < j$. (Otherwise, swap x and y.)
Let $z = 1^j$.

- Then $xz = 0^i 1^j \in L_b$. If $i = 0$, then xz consists of exactly one run (because $j > i$), so trivially $xz \in L_b$. If $i > 0$, then xz consists of a run of 0s followed by a longer run of 1s (because $j > i$).
- But $yz = 0^j 1^j \notin L_b$. The string yz consists of two runs (because $j > i \geq 0$) of equal length.

We conclude that F is a fooling set for L_b.
Because F is infinite, L_b cannot be regular. ■

Rubric: 5 points: Standard fooling set rubric. These are not the only correct solutions.