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 $P \neq NP$.

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

**Rubric:** For each part: 1 point = $\frac{1}{2}$ for correct answer + $\frac{1}{2}$ for explanation. These are not the only correct explanations.

Which of the following statements are true?

(a) 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.

(b) 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 $T(n) = O(n^2 \log n)$.

(c) For every directed graph $G$, if $G$ has at least one source, then $G$ has at least one sink.

**Solution:** NO

(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:

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

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

**Solution:** NO — $i$ and $j$ should both be increasing.
Which of the following statements are true for all languages $L \subseteq \{0,1\}^*$?

(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 $L = \{\epsilon\}$, we have $L^* = (L^*)^* = \{\epsilon\}$. ■

(g) If $L$ is decidable, then $L^*$ is decidable.

**Solution:** YES — 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 either regular or NP-hard.

**Solution:** NO — $\{0^n1^n \mid n \geq 0\}$. This language can be recognized in linear time. ■

(i) If $L$ is undecidable, then $L$ has an infinite fooling set.

**Solution:** YES — If $L$ is undecidable, then $L$ is not regular. ■

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

**Solution:** NO! — When $L$ is undecidable, 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 really just a continuation of problem 1; we split it into two problems to make grading easier.)

**Rubric:** For each part: 1 point = \( \frac{1}{2} \) for correct answer + \( \frac{1}{2} \) for explanation. These are not the only correct explanations.

Consider the following pair of languages:

- \( \text{DirHamPath} := \{ G \mid G \text{ is a directed graph with a Hamiltonian path} \} \)
- \( \text{Acyclic} := \{ G \mid G \text{ is a directed acyclic graph} \} \)

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following statements are true, assuming \( P \neq NP \)?

(a) \( \text{Acyclic} \in NP \)

**Solution:** YES — WFS. We can decide whether a given directed graph has a cycle in \( O(V + E) \) time using whatever-first search. Thus, \( \text{Acyclic} \in P \subset NP \). ■

(b) \( \text{Acyclic} \cap \text{DirHamPath} \in P \)

**Solution:** YES — Dynamic programming / depth-first search. We can compute the longest path in a dag in \( O(V + E) \) time using either topological sort and dynamic programming, or depth-first search. ■

(c) \( \text{DirHamPath} \) is decidable.

**Solution:** YES — Brute force. For every permutation of the vertices, check if there is a path through the vertices in that order. The resulting algorithm runs in \( O(V! \cdot E) \) time, which is exponential, but that’s fine. ■

(d) A polynomial-time reduction from \( \text{DirHamPath} \) to \( \text{Acyclic} \) would imply \( P=NP \).

**Solution:** YES — \( \text{DirHamPath} \) is NP-hard, but \( \text{Acyclic} \) is in \( P \). ■

(e) A polynomial-time reduction from \( \text{Acyclic} \) to \( \text{DirHamPath} \) would imply \( P=NP \).

**Solution:** NO — The reduction is in the wrong direction. ■
Suppose there is a *polynomial-time* reduction from some language $A \subseteq \{0, 1\}$ to some other language $B \subseteq \{0, 1\}$. Which of the following statements are true, assuming $P \neq NP$?

(f) $A \subseteq B$.

**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) There is an algorithm to transform any Python program that solves $B$ in polynomial time into a Python program that solves $A$ in polynomial time.

**Solution:** YES — Combine Python code for the reduction with Python code for $B$.

```python
print(''
from solveB import solveB
from reduceAtoB import reduceAtoB
def solveA(X):
    return reduceAtoB(X, solveB)
''
```

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

**Solution:** YES — By definition of NP-hard.

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

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

(j) If a Turing machine $M$ accepts $B$, the same Turing machine $M$ also accepts $A$.

**Solution:** NO — Each Turing machine has only one accepting language.
3. Aladdin and Badroulbadour are playing a cooperative game. Each player has an array of positive integers, arranged in a row of squares from left to right. Each player has a token, which starts at the leftmost square of their row; their goal is to move both tokens to the rightmost squares.

On each turn, both players move their tokens in the same direction, either left or right. The distance each token travels is equal to the number under that token at the beginning of the turn. For example, if a token starts on a square labeled 5, then it moves either five squares to the right or five squares to the left. If either token moves past either end of its row, then both players immediately lose.

Describe and analyze an algorithm to determine whether Aladdin and Badroulbadour can solve their puzzle, given the input arrays $A[1..n]$ and $B[1..n]$.

**Solution:** We define a directed graph $G = (V, E)$ as follows:

- $V = \{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$. Each vertex $(i, j)$ represents the configuration where A’s token is on $A[i]$ and B’s token is on $B[j]$.
- $E = E_\rightarrow \cup E_\leftarrow$ where
  
  $$E_\rightarrow = \{(i,j)\rightarrow (i+A[i],j+B[j]) \mid i+\text{A}[i] \leq n \text{ and } j+\text{B}[j] \leq n\}$$
  $$E_\leftarrow = \{(i,j)\leftarrow (i-A[i],j-B[j]) \mid i-\text{A}[i] \geq 1 \text{ and } j-\text{B}[j] \geq 1\}$$

  Each edge in $E_\rightarrow$ represents a legal move to the right, and each edge in $E_\leftarrow$ represents a legal move to the left.

We need to determine whether $(n,n)$ is reachable from $(1,1)$ in $G$. We can solve this reachability problem in $O(V + E) = O(n^2)$ time using whatever-first search. ■

**Rubric:** 10 points: Standard graph reduction 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 \( G = (V, E) \) be an arbitrary undirected graph. A subset \( S \subseteq V \) of vertices is mostly independent if less than half the vertices of \( S \) have a neighbor that is also in \( S \). Prove that finding the largest mostly independent set in \( G \) is NP-hard.

**Solution:** We reduce from the standard maximum independent set problem.

Let \( G = (V, E) \) be the given input graph, and let \( n = |V| \). We construct a new graph \( G' = (V', E') \) as by adding a disjoint clique of size \( n \) to \( G \). Specifically:

- \( V' = V \cup K \), where \( K \) is a set of \( n \) new vertices.
- \( E' = E \cup \{xy \mid x, y \in K\} \)

Now we claim that \( G \) has an independent set of size \( k \) if and only if \( G' \) has a mostly independent set of size \( 2k - 1 \). The claim is trivial if \( k \leq 2 \), so assume otherwise.

\[ \Rightarrow \]

Let \( S \) be an independent set of size \( k \) in \( G \). Let \( T \) be any subset of \( k - 1 \) vertices in \( K \). Then \( S \cup T \) is a mostly independent set of size \( 2k - 1 \) in \( G' \).

\[ \Leftarrow \]

Let \( S' \) be a mostly independent set of size \( 2k - 1 \) in \( G' \). Call a vertex of \( S' \) bad if it has a neighbor in \( S' \); there are at most \( k - 1 < n \) bad vertices. If \( S' \) contained every vertex in \( K \), there would be too many bad vertices.

Suppose \( S' \) contains a bad vertex \( v \in V \). Then for any vertex \( x \in K \setminus S' \), the subset \( S' + x - v \) is another mostly independent set of size \( 2k - 1 \) in \( G' \).

Thus, by induction, there is a mostly independent set \( S'' \) of size \( 2k - 1 \) in \( G' \), whose bad vertices are all in \( K \). Then \( S'' \setminus K \) is an independent set in \( G \) of size at least \( k \).

Given \( G \), we can easily construct \( G' \) in polynomial time.

**Rubric:** 10 points: Standard NP-hardness rubric. −1 for assuming without proof that every bad vertex in \( S' \) is in \( K \). This is not the only correct solution.
(b) Let $G = (V,E)$ be an arbitrary directed graph with colored edges. A rainbow Hamiltonian cycle in $G$ is a cycle that visits every vertex of $G$ exactly once, in which no pair of consecutive edges have the same color. Prove that it is NP-hard to decide whether $G$ has a rainbow Hamiltonian cycle.

**Solution:** We reduce from the standard directed Hamiltonian cycle problem.

Let $G$ be the given input graph. Color the edges of $G$ by assigning every edge a different color. Now every Hamiltonian cycle in $G$ is a rainbow Hamiltonian cycle! In particular $G$ (without colors) has a Hamiltonian cycle if and only if $G$ (with colors) has rainbow Hamiltonian cycle.

Given $G$, we can easily color its edges in polynomial time.

---

Rubric: 10 points: Standard NP-hardness rubric. These are not the only correct solutions.

Yes, the first solution is enough for full credit!
5. Suppose we are given an \( n \)-digit integer \( X \). Repeatedly remove one digit from either end of \( X \) (your choice) until no digits are left. The square-depth of \( X \) is the maximum number of perfect squares that you can see during this process.

Describe and analyze an algorithm to compute the square-depth of a given integer \( X \), represented as an array \( X[1..n] \) of \( n \) decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given \( k \)-digit number (represented by an array of digits) is a perfect square in \( O(k^2) \) time.

**Solution:** For any indices \( i \) and \( j \), let \( SqDepth(i, j) \) denote the square-depth of the interval \( X[i..j] \). We need to compute \( SqDepth(1, n) \).

Without loss of generality, suppose the IsSquare returns 1 to indicate a perfect square and 0 otherwise. Then the SquareDepth function obeys the following recurrence:

\[
SqDepth(i, j) = \begin{cases} 
0 & \text{if } i > j \\
\text{IsSquare}(X[i..j]) + \max \left\{ SqDepth(i + 1, j), SqDepth(i, j - 1) \right\} & \text{otherwise}
\end{cases}
\]

We can memoize this function into a two-dimensional array \( SqDepth[1..n, 1..n] \).

We can fill this array using two nested for loops, decreasing \( i \) in one loop and increasing \( j \) in the other; the nesting order doesn’t matter.

For each of the \( O(n^2) \) subproblems, we spend \( O(n^2) \) time calling IsSquare, so the entire algorithm runs in \( O(n^4) \) time.

**Rubric:** 10 points: Standard dynamic programming rubric. This is not the only correct solution.
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 $\{0, 1\}^*$ in which every run of 1s has even length and every run of 0s has odd length.

- 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 $(11)^* (0(00)^*11(11)^*)^* (\varepsilon + 0(00)^*)$.

**Rubric:** 5 points = 2½ for DFA/NFA + 2½ for regular expression. These are not the only correct solutions.
(b) Let \( L_b \) denote the set of all strings in \( \{0, 1\}^* \) in which every run of 0s is immediately followed by a longer run of 1s. **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 \( xz = 0^i1^j \in L_b \) because \( 0 < i < j \).
- But \( yz = 0^j1^j \notin L_b \) 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. ■

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