Write your answers in the separate answer booklet.

You have 180 minutes (after you get the answer booklet) to answer seven questions.

1. For each statement below, there are two boxes in the answer booklet labeled "Yes" and "No". Check "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* ≠ *NP*.** If there is any other ambiguity or uncertainty about an answer, check "No". For example:

   - $x + y = 5$

     Yes [✓ No] Suppose $x = 3$ and $y = 4$.

   - 3SAT can be solved in polynomial time.

     Yes [✓ No] 3SAT is NP-hard.

   - If P = NP then Jeff is the Queen of England.

     [✓ Yes] No The hypothesis is false, so the implication is true.

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

   (a) Which of the following statements are true for ***every*** language $L \subseteq \{0,1\}^*$ and every ***regular*** language $R \subseteq \{0,1\}^*$?

   - If $L \cup R$ is regular, then L is regular.
   - If $L \cup R$ is not regular, then L is not regular.
   - If $L \cap R$ is regular, then L is regular.
   - If there is a reduction from L to R, then $L \in P$.
   - If $L \cap R$ is not decidable, then L is not decidable.

   (b) Which of the following statements are true for ***every*** language $L \subseteq \{0,1\}^*$?

   - L^* is regular.
   - L^* contains the empty string ε.
   - If L is NP-hard, then L^* is NP-hard.
   - If L is a fooling set for an NP-hard language, then L is infinite.
   - If L is the intersection of two NP-hard languages, then L is NP-hard.

Problems 2–7 appear on the next three pages.

2. **More of the same:** For each statement below, there are two boxes in the answer booklet labeled "Yes" and "No". Check "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, check "No".

   (a) Which of the following languages can be proved undecidable ***using Rice's Theorem***?

   - $\{\langle M\rangle \mid M$ has at most 374 states$\}$
   - $\{\langle M\rangle \mid M$ accepts a finite number of strings$\}$
   - $\{\langle M\rangle \mid M$ accepts `KALM`, rejects `PANIK`, and hangs on `STONKS`$\}$
   - $\{\langle M\rangle \mid$ There are exactly 374 palindromes that M does *not* accept$\}$
   - $\{\langle M\rangle \mid \langle M\rangle$ is a palindrome$\}$

   (b) Suppose we want to prove that the following language is undecidable.

   $$\text{Hunter} := \{\langle M\rangle \mid M \text{ accepts } \texttt{RAMEN} \text{ and } \texttt{FANS} \text{ but does not accept } \texttt{DEMONS}\}$$

   Your mentor Celine suggests a reduction from the standard halting language

   $$\text{Halt} := \{(\langle M\rangle, w) \mid M \text{ halts on input } w\}.$$

   Specifically, suppose there is a machine Golden that decides Hunter. Celine claims that the following algorithm decides Halt.

```
DecideHalt($\langle M\rangle, w$):
  Write code for the following algorithm:
      Couch($x$):
        if $x$ = DEMONS
            return False
        else
            ⟨⟨run $M$ on $w$ and return result⟩⟩
            return $M(w)$
  return Golden(⟨Couch⟩)
```

   Which of the following statements must be true for ***all*** inputs $(\langle M\rangle, w)$?

   - If M accepts w, then Couch rejects `DEMONS`.
   - If M accepts w, then Golden accepts $\langle$Couch$\rangle$.
   - If M diverges on w, then Golden rejects $\langle$Couch$\rangle$.
   - DecideHalt decides the language Halt. (That is, Celine's reduction is correct.)
   - We could instead prove Hunter is undecidable using Rice's theorem.

Problems 3–7 appear on the next two pages.

3. Exactly one of the following languages is regular. Which one?

   (a) $\{\mathtt{0}^a\mathtt{1}^b\mathtt{0}^c \mid a+b=c\}$

   (b) $\{\mathtt{0}^a\mathtt{1}^b\mathtt{0}^c \mid a+b\equiv c \pmod 2\}$

   Indicate which one of these two languages is regular. Describe a DFA or NFA that accepts the regular language, and ***prove*** that the other language is not regular. (You do not need to prove that your DFA or NFA is correct.)

4. Suppose we are given two unsorted arrays $A[1..n]$ and $B[1..n]$. Elements of A and B come from some fixed totally ordered set, like integers or letters of the alphabet, that can be compared in $O(1)$ time. An ***upside-down pair*** for A and B is an ordered pair (i, j) of indices such that $i < j$ and $A[i] > B[j]$.

   Describe and analyze an efficient algorithm that either finds one upside-down pair for two given arrays A and B, or correctly reports that no such pair exists. If there is more than one upside-down pair, your algorithm can arbitrarily choose which one to return. To receive full credit, your algorithm's running time must be strictly better than $O(n^2)$.

   For example, given the arrays $A = [\mathtt{D}, \mathtt{I}, \mathtt{V}, \mathtt{I}, \mathtt{D}, \mathtt{E}]$ and $B = [\mathtt{C}, \mathtt{O}, \mathtt{N}, \mathtt{Q}, \mathtt{E}, \mathtt{R}]$ as input, your algorithm should return either $(3,4)$ or $(3,6)$, because $A[3] = \mathtt{V}$ is later in the alphabet than both $B[4] = \mathtt{Q}$ and $B[6] = \mathtt{R}$. On the other hand, given the arrays $A = [\mathtt{D}, \mathtt{E}, \mathtt{N}, \mathtt{I}, \mathtt{E}, \mathtt{D}]$ and $B = [\mathtt{C}, \mathtt{O}, \mathtt{N}, \mathtt{Q}, \mathtt{U}, \mathtt{R}]$, your algorithm should report that there are no upside-down pairs.

5. Submit a solution to ***exactly one*** of the following problems (either (a) or (b)).

   (a) An *ultra-Hamiltonian tour* in a graph G is a closed walk W that visits every vertex in G exactly once, except for *at most* one vertex that W visits more than once.

      i. Give an example of a graph that contains a ultra-Hamiltonian tour, but does not contain a Hamiltonian cycle (which visits every vertex exactly once).

      ii. ***Prove*** that it is NP-hard to determine whether a given graph contains a ultra-Hamiltonian tour.

   (b) ***Prove*** that the following problem is NP-hard: Given an undirected graph $G = (V, E)$ and a subset of vertices $L \subset V$, does G have a spanning tree T such that every leaf of T is in L?

   In fact, both of these problems are NP-hard, but we only want a proof for one of them. Don't forget to tell us which problem you've chosen! There is a list of (mostly) standard NP-hard problems at the end of the answer booklet.

Problems 6 and 7 appear on the next page.

6. Your company has an interesting stock-purchasing program. Each business day, you must perform one of three possible actions:

   - Buy exactly one share of stock at that day's price.
   - Sell exactly one share of stock at that day's price (if you own at least one share)
   - Hold your stock, neither buying nor selling.

   As a one-time holiday bonus, your company is allowing you to participate in this program *retroactively*, starting on your first day of employment. They give you a list $\mathit{Price}[1\,..\,n]$ of historical stock prices, where $\mathit{Price}[i]$ was the price on your ith day of employment, and n is the number of days that you have been employed. All stock prices are positive.

   Describe and analyze an algorithm to compute the maximum total profit you can earn for those n days, given the *Price* array as input. You must start with zero shares, but you have enough money to buy as many shares as you like. Any shares that you own after the nth day do *not* count toward your profit.

   For example, given the input array $\mathit{Price} = [4, 6, 3, 5, 7, 2, 1]$, your algorithm should return 6, because the best strategy (buy, sell, buy, hold, sell, hold, hold) earns a total profit of $-4 + 6 - 3 + 7 = 6$ dollars. Similarly, given the input array $\mathit{Price} = [1, 2, 3, 4, 5, 6]$, your algorithm should return 9 (buy, buy, buy, sell, sell, sell), and given the input array $\mathit{Price} = [6, 5, 4, 3, 2, 1]$, your algorithm should return 0 (hold, hold, hold, hold, hold, hold).

7. Suppose you are given a directed graph $G = (V, E)$, where each edge has a positive weight, two vertices s and t, and an integer k. Describe an algorithm that computes the length of the shortest walk in G from s to t that traverses ***at least*** k edges. Analyze your algorithm in terms of V, E, and k.