You have 180 minutes to answer six numbered questions.

Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

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 $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, check "No". For example:

   - $x + y = 5$

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

   - 3SAT can be solved in polynomial time.

     [Yes] [X No] 3SAT is NP-hard.

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

     [X 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?

   - The solution to the recurrence $T(n) = 2T(n/2) + O(\sqrt{n})$ is $T(n) = O(\sqrt{n}\log n)$.
   - The solution to the recurrence $T(n) = 2T(n/4) + T(n/3) + O(n)$ is $T(n) = O(n \log n)$.
   - There is a forest with 374 vertices and 374 edges. (Recall that a *forest* is an undirected graph with no cycles.)
   - Given any directed graph G whose edges have positive weights, we can compute shortest paths from one vertex s to every other vertex of G in $O(VE)$ time using Bellman-Ford.
   - Suppose $A[1..n]$ is an array of integers. Consider the following recursive function:

$$Oops(i,k) = \begin{cases} 0 & \text{if } i > k \\ 1 & \text{if } i = k \\ \max\left\{A[i]\cdot A[j]\cdot A[k] + Oops(i,j) + Oops(j,k) \;\middle|\; i \le j \le k\right\} & \text{otherwise} \end{cases}$$

   We can compute $Oops(1,n)$ by memoizing this function into a two-dimensional array $Oops[1..n, 1..n]$, which we fill by decreasing i in the outer loop and increasing k in the inner loop, in $O(n^2)$ time.

Problem 1 continues onto the next page.

1. *[continued]*

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

- Either L is regular or L is infinite.
- L^* is regular.
- L contains arbitrarily long strings.
- If L is decidable, then its complement $\overline{L}$ is also decidable.
- If L is undecidable then L^* is undecidable.

(c) Consider the following pair of languages:

- 3Color $= \{G \mid G$ is an undirected graph with a proper 3-coloring$\}$
- Forest $= \{G \mid G$ is an undirected graph with no cycles$\}$

(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?

- Forest $\in P$
- Forest $\cap$ 3Color $\in P$
- 3Color is undecidable.
- 3Color is regular.
- A polynomial-time reduction from 3Color to Forest would imply P = NP.

(d) Suppose there is a ***polynomial-time*** reduction R from some language $A \in \{0,1\}^*$ to some other language $B \in \{0,1\}^*$. Which if the following statements are ***always*** true, assuming P $\neq$ NP?

- The reduction transforms every string in A into a string in B.
- The reduction transforms every string in B into a string in A.
- If A is infinite, then B is infinite.
- If B is undecidable, then A is undecidable.
- If B is undecidable, then A is NP-hard.

Problem 2 appears on the next page.

2. Several years after graduating from Sham-Poobanana University, you decide to open a one-day pop-up art gallery selling NFTs, using the following dynamic pricing strategy.

   All NFTs art your gallery have the same advertised price, which you set at the start of the day, but which you can *decrease* later. Customers visit your gallery one at a time. If a customer is willing to pay your current advertised price, they buy one NFT at that price. On the other hand, if your advertised price is too high, the customer will suggest a lower price that they are willing to pay. If you refuse to lower your advertised price, the customer will leave without buying anything. If you agree to lower your advertised price to match their offer, the customer will buy one NFT at the new lower price. Whenever you lower your advertised price, your new lower price stays in effect until you lower it again, or until the end of the day. *You can never increase your advertised price.*

   You know your customers extremely well, so you can accurately predict both when each customer will come to the gallery, and how much each customer is willing to pay for one of your NFTs.

   Describe and analyze an algorithm that computes the maximum amount of money you can earn using this dynamic pricing strategy. Your input consists of an array $Value[1..n]$, where $Value[i]$ is the amount that the ith customer (in chronological order) is willing to pay for one NFT.

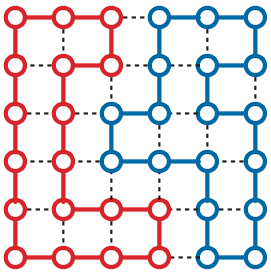
   For example, if the input array is $[5,3,1,4,2]$, your algorithm should output 13, because you can earn $5+3+0+3+2=13$ dollars using the prices $[5,3,3,3,2]$, and this is optimal.

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

   (a) A *mostly-3-coloring* of a graph $G=(V,E)$ is any function $C : V \to \{red, yellow, blue, none\}$ such that $C(v) = none$ for less than half the vertices in V. A mostly-3-coloring C is *proper* if, for every edge $uv \in E$, either $C(u) \neq C(v)$ or $C(u) = C(v) = none$.

   Prove that it is NP-hard to determine whether a given graph G has a proper mostly-3-coloring.

   (b) A *Hamiltonian bicycle* in a graph G is a pair of simple cycles in G, with identical lengths, such that every vertex of G lies on exactly one of the two cycles.



A Hamiltonian bicycle in the 6×6 grid graph.

   Prove that it is NP-hard to determine whether a given graph G has a Hamiltonian bicycle.

(In fact, both of these problems are NP-hard, but we only want a proof for one of them.)

4. (a) Let L_a denote the set of all strings in $\{0,1,2\}^*$ that do not contain any symbol twice in a row. For example, this language includes the strings `0101201202`, `1010101`, `2`, and the empty string ε, but it does not include the strings `01210220` or `11`.
   - 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.)

   (b) Let L_b denote the set of all strings $w \in \{0,1,2\}^*$ such that $\#(0,w)+\#(1,w) = \#(2,w)$. For example, this language includes the strings `01012222` and `20221020` and the empty string ε, but it does not include the string `01212` or `2120210`.

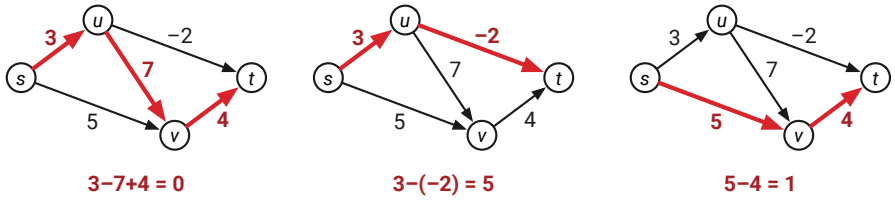
   Prove that L_b is not a regular language.

5. Let G be a directed **acyclic** graph, in which every edge $e \in E$ has a weight $w(e)$, which could be positive, negative, or zero. We define the ***alternating length*** of any path in G to be the weight of the first edge, ***minus*** the weight of the second edge, ***plus*** the weight of the third edge, and so on. More formally, for any path $P = v_0 \to v_1 \to v_2 \to \cdots \to v_\ell$ in G, we define

$$AltLen(P) = \sum_{i=0}^{\ell-1} (-1)^i \cdot w(v_i \to v_{i+1}).$$

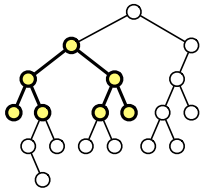
   Describe an algorithm to find a path from s to t with the largest alternating length, given the graph G, the edge weights $w(e)$, and vertices s and t as input.

   For example, given the graph shown below, your algorithm should return 5, which is the alternating length of the path $s \to u \to t$.



6. Recall that the *depth* of a vertex v in a binary tree T is the length of the unique path in T from v to the root of T. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. An *internal subtree* of a binary tree T is any connected subgraph of T.

   Describe and analyze a recursive algorithm to compute the *largest complete internal subtree* of a given binary tree. Your algorithm should return both the root and the depth of this internal subtree.



The largest complete internal subtree of this binary tree has depth 2.