Write your answers in the separate answer booklet.

You have 120 minutes (after you get the answer booklet) to answer five questions.

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

1. **Short answers:**

   (a) Solve the following recurrences:

   - $A(n) = 5A(n/3) + O(n)$
   - $B(n) = 9B(n/3) + O(n^2)$
   - $C(n) = C(n/2) + C(n/3) + O(n)$

   (b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and state the running time of the resulting iterative algorithm to compute $Bar(1, n, n)$.

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

2. A sequence of *pairs* of integers $\langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$ is called ***semi-increasing*** if for each $i \in \{1, \ldots, k-1\}$, it is the case that $a_{i+1} > a_i$ or $b_{i+1} > b_i$ *or both*. The ***shininess*** of the semi-increasing sequence is its length *plus* the *number of indices* $i \in \{1, \ldots, k-1\}$ such that both $a_{i+1} > a_i$ **and** $b_{i+1} > b_i$.

   Describe an algorithm that, given a sequence of n pairs of integers $P[1..n]$, computes the maximum shininess of any semi-increasing subsequence of P. You can use $P[i].a$ to denote the first component of pair $P[i]$ and $P[i].b$ to denote the second component of $P[i]$. You may assume all integers in the input are distinct.

   For example, given the sequence

$$\langle (20,50), (2,3), (5,4), (-3,-2), (6,9), (10,-20), (11,-5), (8,-6), (0,1) \rangle,$$

   your algorithm should return at least 9 due to the semi-increasing subsequence $\langle (2,3), (\underline{5},\underline{4}), (\underline{6},\underline{9}), (\underline{10},-20), (\underline{11},\underline{-5}), (0,\underline{1}) \rangle$.

3. Not satisfied with the current game design, the creative leads behind *Murder the Mountain* decide to add Sprite Bottles that automatically heal the player character (PC) when they would otherwise die. Like before, each map can be modeled as a directed acyclic graph (dag) $G = (V, E)$ with a start vertex $s \in V$ and a boss vertex $b \in V$. The player character (PC) needs to follow a path from s to b. The PC starts with some variable even integer amount of health $k \geq 1$. If the PC enters a vertex v, their health will decrease by $Damage(v) \geq 0$. If the PC's health drops to 0 or less, then they will ***pop open*** a Sprite Bottle automatically, restoring them to exactly $k/2$ health (they will take no further damage at space v). You may assume $Damage(s) = 0$.

   There are no longer relics; the only goal now is *survival*. Describe an algorithm that, given an integer k and the graph and *Damage* function as described above, computes the minimum number of times the PC must pop open a Sprite Bottle while following some path from s to b.

4. You have a server with n RAM slots, but **two** of them are broken, and you don't know which two. The only way to test which slots are broken is to insert a DIMM (a RAM stick) into one or more slots and power the server on. If there's a DIMM in at least one broken slot, the computer won't boot; otherwise, it will. Unfortunately, waiting for the server to go through its boot process takes a long time. Your goal is to find the broken slots without booting the server too many times.

   The RAM slots have IDs given in an array $R[1..n]$. Describe an algorithm to find the IDs of the two broken slots. You may assume you have access to a subroutine TEST($X[1..k]$) that takes an array of k RAM slot IDs and returns TRUE if X contains the ID of *at least one* broken slot and FALSE otherwise. (In other words, TEST is just shorthand for booting the server after populating the slots X.) **Your running time should be the number of times you call TEST (boot the server).** All other operations such as creating an array to give to TEST as input are "free".

5. You somehow find yourself transported to a fantastical village built upon n small islands rising out of a picturesque sea. There are m bridges connecting some (but not all) pairs of islands. Unfortunately, as these things often go, each bridge is guarded by a terrifying troll who demands a payment of gold to cross. Fortunately, as these things seldom go, the trolls have established a loyalty program whereby paying to cross a bridge means the next two bridge crossings you make are free. The bridges have different positive costs for non-free crossings. You're allowed to make two consecutive crossings along the same bridge (in opposite directions). You *cannot* choose to pay for one of your free crossings (so you must go pay, free, free, pay, free, free, etc.)

   You wake up one morning hungry for waffles and decide to journey to the market for ingredients. You have a map of the village given as a connected undirected graph $G = (V, E)$ where V is the set of islands and E is the set of bridges. The map also describes a function $c : E \to \mathbb{R}^+$ where $c(e)$ is the *cost* (in gold coins) of crossing bridge e. Describe an algorithm to compute the minimum cost for traveling from your home on island $s \in V$ to the market on island $t \in V$. For example, given the following graph with each edge labeled by its cost, your algorithm should return 22 for the walk $s \xrightarrow{\$} a \to s \to b \xrightarrow{\$} d \to e \to f \xrightarrow{\$} g \to h \to t$ where $u \xrightarrow{\$} v$ indicates *paying* to cross from island u to island v.

