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:

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

   (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 $Foo(1, n)$.

$$Foo(i,k) = \begin{cases} 0 & \text{if } i \geq k \\ \max \left\{ \left. \begin{array}{l} Foo(i,j) + Foo(j+1,k) \\ \quad + \displaystyle\sum_{\ell=i}^{j} \left((-1)^{\ell - i} \cdot A[i,j] \right) \end{array} \right| \; i \leq j < k \right\} & \text{otherwise} \end{cases}$$

2. For any non-negative integer k, a sequence of integers X is called ***k-drop increasing*** if it can be partitioned into *at most* $k+1$ contiguous increasing subsequences; in other words, X is increasing except between at most k pairs of consecutive integers where it is allowed to decrease. For example, the sequence $\langle 1, 5, 9, 2, 12, 20, -5, 0, 4 \rangle$ is k-drop increasing for all $k \geq 2$.

   Describe an algorithm that, given an array of integers $A[1..n]$ and a non-negative integer k, computes the length of the longest k-drop increasing subsequence of A. Your running time should be a function of n and k.

3. Good news! Your game *Murder the Mountain* was an amazing success, and your studio has begun production on the sequel *Final Murder the Mountain Quest XII*. 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. If the PC enters a vertex v, they will take $Damage(v) \geq 0$ damage. They will also acquire a single relic if and only if $Relic(v) = \text{TRUE}$. And as before, you may assume $Damage(s) = 0$ and $Relic(s) = \text{FALSE}$. You want to test difficulty again, but the sequel uses some really *really* large damage numbers. Your old algorithm for testing relic acquisition may not be fast enough.

   (a) Describe an algorithm that, given an integer r with $0 \leq r \leq |V|$ along with the graph and functions as described above, computes the **minimum amount of damage** the PC can take while following some path from s to b that acquires **at least** r relics or ∞ if no such path exists. Your running time should be a function of $|V|$, $|E|$, and r.

   (b) *Now*, describe an algorithm that, given a *very large* positive number K along with the graph and functions as described above, computes the maximum number of relics the PC can acquire while following some path from s to b and taking *at most* K damage or $-\infty$ if no such path exists. Your running time should be a function of $|V|$ and $|E|$ **but not** K.

   You may (and probably should) use your algorithm from part (a) as a black box. You can still receive substantial partial credit even if part (a) is incorrect; *if necessary*, use T_a to denote an upper bound on the running time of part (a)'s algorithm).

   [Hint: Your goal is to search for a particular integer r where $0 \leq r \leq |V|$.]

4. You have a server with n RAM slots, but one of them is broken, and you don't know which one. The only way to test which slot is 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 the 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 slot 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 ID of the broken slot. You may assume you have access to a subroutine $\text{TEST}(X[1..k])$ that takes an array of k RAM slot IDs and returns TRUE if X contains the ID of the 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 you're given the option of taking your next zero, one, or two bridge crossings for 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 wake up one morning hungry for pancakes and decide to journey to the market for ingredients. You have a map of the village given as an 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 16 for the walk $s \xrightarrow{\$} a \to s \to b \xrightarrow{\$} d \to e \xrightarrow{\$} f \to g \to h \xrightarrow{\$} t$ where $u \xrightarrow{\$} v$ indicates *paying* to cross from island u to island v.

