CS/ECE 374 A ✧ Fall 2026
Midterm 2 Problem 1 Solution

(a) Write the solution to each of the following recurrences in the box immediately below it. (Use the space below the boxes for scratch work.)

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

$A(n)$	$B(n)$	$C(n)$
$O(n^2 \log n)$	$O(n)$	$O(n^{\log_3 5})$

Solution: Level ℓ of the recursion tree for $A(n)$ has 9^ℓ nodes, each with value $n^2/9^\ell$, so the total value is n^2. Because every level has total value at most n^2, and the depth of the tree is $O(\log n)$, we have $A(n) = O(n^2 \log n)$.

The recursion tree for $B(n)$ is a ternary tree with unbalanced problem sizes. The root has value n. The root has two children, one with value $n/2$ and one with value $n/3$, so the total value of the children is $5n/6$. More generally, level ℓ of the recursion tree for $B(n)$ has total value at most $(5/6)^\ell n$. The level sums form a decreasing geometric series, so only the root value n matters. (Fans of the Master Theorem should notice that it cannot be used to solve this recurrence. If you've never heard of the Master Theorem, you're not missing much.)

Level ℓ of the recursion tree for $C(n)$ has 5^ℓ nodes, each with value $n/3^\ell$, so the total value at level ℓ is $(5/3)^\ell n$. The level sums form an increasing geometric series, so only the number of leaves matters. The depth of the tree is $\log_3 n$, so $C(n) = O(5^{\log_3 n}) = O(n^{\log_3 5})$. ■

Rubric:

2 points each for $A(n)$.

2.5 points each for $B(n)$ and $C(n)$.

Explanations and/or recursion tree drawings are not required for full credit, only the final answers in the boxes.

(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 $Foo(1, n)$.

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

Solution: Two-dimensional array, filled in *decreasing* order of i and *increasing* order of k with the loops nested either way, in $O(n^3)$ time. (Evaluating each entry $Foo[i,k]$ requires a for loop over j; note that we can update $\sum_{\ell=i}^{j} \left((-1)^{\ell-i} \cdot A[i,j] \right)$ in constant time per j by just adding the newest term to the value of the previous sum.) ■

Rubric: 3 points = 1 for correct structure + 1 for correct evaluation order + 1 for running time. 1/2 point for quoting the running time as $O(n^4)$, which corresponds to naïvely recomputing the summation from scratch for each value of j.

CS/ECE 374 A ✧ Fall 2026

Midterm 2 Problem 2 Solution

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.

Solution: Add a sentinel value $A[0] = -\infty$ to A. Let $LKIS(i, j, \ell)$ be the length of the longest subsequence X of $A[j..n]$ such that $A[i] \cdot X$ is ℓ-drop increasing. We need to compute $LKIS(0, 1, k)$. The function obeys the following recurrence:

$$LKIS(i,j,\ell) = \begin{cases} 0 & \text{if } j > n \\ LKIS(i, j+1, 0) & \text{if } j \le n, A[j] \le A[i], \text{ and } \ell = 0 \\ \max\left\{ \begin{array}{c} LKIS(i, j+1, \ell), \\ 1 + LKIS(j, j+1, \ell-1) \end{array} \right\} & \text{if } j \le n, A[j] \le A[i], \text{ and } \ell > 0 \\ \max\left\{ \begin{array}{c} LKIS(i, j+1, \ell), \\ 1 + LKIS(j, j+1, \ell) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this recurrence using a 3D array $LKIS[0..n, 1..n+1, 0..k]$. We fill it using nested loops over j decreasing, i in any order, and ℓ in any order from outside in. The running time is $O(kn^2)$. ■

Rubric: 10 points: standard dynamic programming rubric.

CS/ECE 374 A ✧ Fall 2026
Midterm 2 Problem 3 Solution

(a) Describe an algorithm that, given an integer r with $0 \le r \le |V|$ 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.

Solution (dynamic programming): Let $MinDamage(u, t)$ be the minimum amount of damage the PC can take while following a path from u to b that acquires at least t relics (but only after leaving u). We need to compute $MinDamage(s, r)$. The function obeys the following recurrence:

$$MinDamage(u,t) = \begin{cases} 0 & \text{if } u = b \text{ and } t = 0 \\ \infty & \text{if } u = b \text{ and } t > 0 \\ \min\limits_{u \to v} \left(\begin{array}{l} Damage(v) + \\ MinDamage\left(\begin{array}{l} v, \\ \max\{0, t - [Relic(v)]\} \end{array} \right) \end{array} \right) & \text{otherwise} \end{cases}$$

We can memoize this function by giving each vertex u an array $u.MinDamage[0..r]$. We can evaluate the subproblems using an outer loop over t increasing and postorder over the vertices. The running time is $O(r(|V| + |E|))$.

■

Rubric: 6 points: standard dynamic programming rubric (scaled).

Solution (graph reduction): We reduce the problem to shortest paths in a dag. We build a directed graph $G' = (V', E')$ with edge weights $w' : E \to \mathbb{R}$ where

- $V' := V \times \{0, \ldots, r\}$. Vertex (v, t) represents reaching v from s having collected t relics (up to a maximum of r).
- $E' := \{(u, t) \to (v, \min\{r, t + [Relic(v)]\}) \mid u \to v \in E, t \in \{0, \ldots, r\}\}$.
- $w'((u, t) \to (v, t')) := Damage(v)$.

The minimum amount of damage the PC can take going from s to b while collecting at least r relics is the distance from $(s, 0)$ to (b, r). Graph G' is a dag, so we run DAGSSSP$((s, 0))$ and return $(b, r).dist$. The running time is $O(|V'| + |E'|) = O(r(|V| + |E|))$.

■

Rubric: 6 points: standard graph reduction rubric (scaled).

(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 $|E|$ **and not** K.

Solution: The procedure MaxRelics($rmin, rmax$) given below returns the maximum number of relics the PC can acquire along a path from s to b while taking at most K damage *assuming that maximum is at least rmin and at most rmax*. As a subroutine, it uses MinDamage(r) which is the algorithm from part (a) that returns the minimum damage taken while acquiring at least r relics. We return $-\infty$ if MinDamage$(0) > K$. Otherwise, we return the result of MaxRelics$(0, |V|)$.

```
MaxRelics(rmin, rmax):
  if rmin = rmax
      return rmin
  r ← ⌈(rmin+rmax)/2⌉
  if MinDamage(r) ≤ K
      return MaxRelics(r, rmax)
  else
      return MaxRelics(rmin, r)
```

The algorithm is a binary search where $rmax - rmin + 1$ is approximately halved every recursive call. The work done outside of recursive calls is $O(r(|V| + |E|)) \leq O(|V|(|V|+|E|))$, so the total running time is $O(|V|(|V|+|E|)\log|V|)$ or $O(|E||V|\log|V|)$ assuming the graph is connected. ■

Rubric: 4 points: standard recursion rubric (scaled).

CS/ECE 374 A ✧ Fall 2026

Midterm 2 Problem 4 Solution

A collection of n RAM slots have IDs given in an array $R[1..n]$. Exactly one RAM slot is broken. You have access to a subroutine $\textsc{Test}(X[1..k])$ that takes an array of k RAM slot IDs and returns $\textsc{True}$ if X contains the ID of the broken slot and $\textsc{False}$ otherwise. Describe an algorithm to find the ID of the broken slot. **Your running time should be the number of times you call $\textsc{Test}$ (boot the server).**

Solution: The procedure $\textsc{FindBroken}(R[1..n])$ takes an array of RAM slot IDs where exactly one slot is broken. It returns the ID of the broken slot.

```
FindBroken(R[1..n]):
  if n = 1
      return R[1]
  m ← ⌈n/2⌉
  if Test(R[1..m])
      return FindBroken(R[1..m])
  else
      return FindBroken(R[m+1..n])
```

The number of calls to $\textsc{Test}$ follows the standard binary search recurrence of $T(n) = T(n/2) + O(1)$, so there are $O(\log n)$ calls. ■

Rubric: 10 points: standard recursion rubric.

CS/ECE 374 A ✧ Fall 2026
Midterm 2 Problem 5 Solution

You have a map of your 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* 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 $t \in V$.

Solution: We build a *directed* graph $G' = (V', E')$ with non-negative edge weights $w' : E \to \mathbb{R}^+$ as follows:

- $V := V \times \{0, 1, 2\}$. Vertex (v, x) means you're on island v and can take the next x bridge crossings for free.
- $E := \{(u, x) \to (v, 2) \mid uv \in E, x \in \{0, 1, 2\}\}$
  $\cup \{(u, x) \to (v, x-1) \mid uv \in E, x \in \{1, 2\}\}$
- $w'((u, x) \to (v, 2)) := c(uv)$ for all $uv \in E$ and $x \in \{0, 1, 2\}$.
- $w'((u, x) \to (v, x-1)) := 0$ for all $uv \in E$ and $x \in \{1, 2\}$.

The minimum cost for traveling from island s to island t is the minimum distance from $(s, 0)$ to any (t, x) in G'. Therefore, we run DIJKSTRA$((s, 0))$ and return $\min_x (t, x).dist$. The running time is $O(|E'| \log |V'|) = O(|E| \log |V|)$. ■

Rubric: 10 points: standard graph reduction rubric.