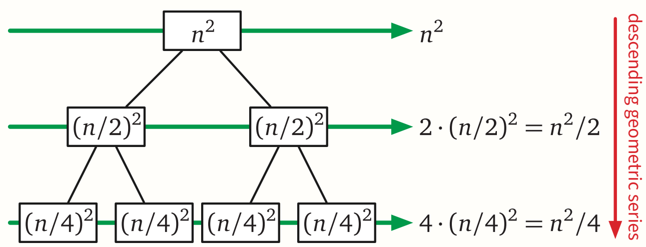
CS/ECE 374 A ✧ Fall 2025
Midterm 2 Problem 1 Solution

(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.)

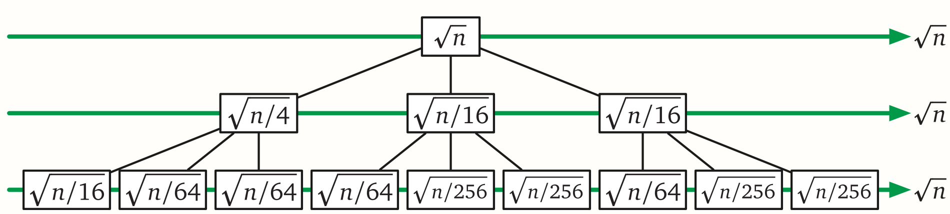
$A(n) = 2 \cdot A(n/2) + O(n^2)$	$B(n) = B(n/4) + 2 \cdot B(n/16) + O(\sqrt{n})$	$C(n) = 4 \cdot C(n/3) + O(n)$
$O(n^2)$	$O(\sqrt{n}\log n)$	$O(n^{\log_3 4})$

Solution: The recursion tree for $A(n)$ is a binary tree. Each level ℓ has 2^ℓ nodes, each with value $(n/2^\ell)^2$, so the total value at level ℓ is $n^2/2^\ell$. The level sums form a descending geometric series, so only the root value n^2 matters.

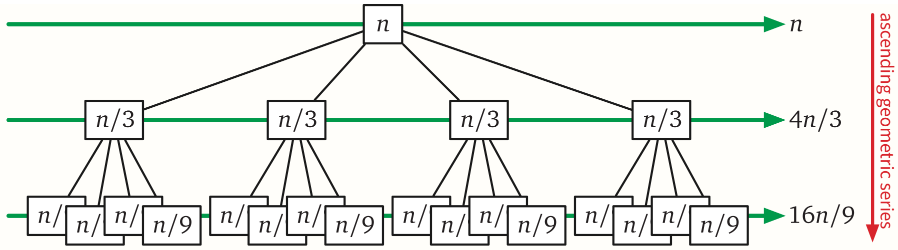


The recursion tree for $B(n)$ is a ternary tree with unbalanced problem sizes. The root has value $\sqrt{n}$. The root has three children, one with value $\sqrt{n/4} = \sqrt{n}/2$ and two with value $\sqrt{n/16} = \sqrt{n}/4$, so the total value of the children is also $\sqrt{n}$. In fact every level has total value at most $\sqrt{n}$, and every leaf has depth $O(\log n)$, so $B(n) = O(\sqrt{n}\log n)$.

(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 anything.)



Level ℓ of the recursion tree for $C(n)$ has 4^ℓ nodes, each with value $n/3^\ell$, so the total value at level ℓ is $(4/3)^\ell n$. 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 $C(n) = O(4^{\log_3 n}) = O(n^{\log_3 4})$.



■

Rubric: 2½ points each. −½ for $C(n) = O(4^{\log_3 n})$. Explanations and/or recursion tree drawings are not required for full credit, only the final answers in the boxes. 1 point for a correct recursion tree with an incorrect conclusion.

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

$$Foo(i) = \max\left\{ \left(\sum_{k=1}^{j} A[j,k]\right)^2 + Foo(j) \;\middle|\; 1 \le j < i \right\} \qquad (\max\varnothing = 0)$$

Solution: One-dimensional array, filled in increasing index order, in $O(n^3)$ time. (Evaluating each entry $Foo[i]$ requires two nested for loops over j and k.) ■

Rubric: 2½ points = ½ for correct structure + 1 for correct evaluation order + 1 for running time

CS/ECE 374 A ✧ Fall 2025
Midterm 2 Problem 2 Solution

Suppose you are given a directed acyclic graph $G = (V, E)$, where every vertex represents a required class and each edge $u \to v$ indicates that class u must be completed before you can begin class v.

(a) Describe an algorithm that computes a class schedule that uses the minimum number of terms. For each class, the output of your algorithm should specify the term when you will take that class.

Solution: Let $Earliest(v)$ denote the earliest term when we can take class v. We need to compute this function for every vertex v. This function satisfies the recurrence

$$Earliest(v) = \begin{cases} 1 & \text{if } v \text{ has no incoming edges} \\ 1 + \max\{Earliest(u) \mid u \to v \in E\} & \text{otherwise} \end{cases}$$

(The base case is actually redundant if we assume $\max \varnothing = 0$.)

We can memoize this recurrence into a new field $v.Earliest$ in each vertex, and we can fill these fields in topological order, in ***$O(V + E)$ time***.

Alternatively, after topologically sorting G in $O(V + E)$ time, we can memoize the recurrence into an array $Earliest[1 .. V]$ indexed by the vertices in topological order, and can fill this array from left to right in ***$O(V + E)$ time***. ■

Solution: Suppose we add a new source vertex s to the graph, with edges $s \to v$ for every class v. Then the earliest term that we can take class v is exactly the length of the longest path from s to v. We can compute all these longest-path lengths using the LongestPath dynamic programming algorithm in the textbook in $O(V + E)$ time. ■

Rubric: 5 points: graph-reduction rubric. These are the same algorithm.

(b) Describe an algorithm that computes the minimum number of terms required to take ***at least 75%*** of the classes represented in G, while still obeying all prerequisite constraints.

Solution: After computing $\mathit{Earliest}(v)$ for all v using the algorithm from part (a), select the $(3V/4)$th earliest course v^* in $O(V)$ time using linear-time selection, and finally return $\mathit{Earliest}(v^*)$. The overall algorithm runs in $\boldsymbol{O(V+E)}$ ***time***. ■

Solution: The following algorithm runs in $\boldsymbol{O(V+E)}$ ***time***.

```
SpeedRun¾(V, E):
  compute v.earliest for all v ∈ V using part (a)
  for term ← 1 to V
      count[term] ← 0
  for all v ∈ V
      count[v.earliest] ← count[v.earliest] + 1
  total ← 0
  term ← 0
  while total < 3V/4
      term ← term + 1
      total ← total + count[term]
  return term
```

The first two for-loops are most of the *counting sort* algorithm, which sorts all vertices v by $v.\mathit{earliest}$. Each array entry $\mathit{count}[i]$ stores the number of classes whose earliest term is i. The algorithm relies on the fact that sorting classes by $v.\mathit{earliest}$ is a topological order; however, not every topological order sorts classes by their earliest terms. ■

Rubric: 5 points

CS/ECE 374 A ✧ Fall 2025
Midterm 2 Problem 3 Solution

Describe and analyze an algorithm to compute a non-crossing matching of two given n-point sets R and B, using subroutines HamSammy and Above? as black boxes.

Solution:

```
NoCrossMatching(R[1..n], B[1..n]):
  if n = 0
      return ∅
  if n = 1
      return {(R[1], B[i])}
  ℓ ← HamSammy(R, B)
  R⁺ ← all points in R that are Above ℓ
  B⁺ ← all points in B that are Above ℓ
  M⁺ ← NoCrossMatching(R⁺, B⁺)
  R⁻ ← R \ R⁺
  B⁻ ← B \ B⁺
  M⁻ ← NoCrossMatching(R⁻, B⁻)
  return M⁺ ∪ M⁻
```

We can argue by induction that this algorithm returns a non-crossing matching of R and B as follows. The claim is trivial when $n \leq 1$, so assume otherwise. The induction hypothesis implies that M^+ is a non-crossing matching of R^+ and B^+ and that M^- is a non-crossing matching of R^- and B^-. Thus, no two segments in M^+ intersect, and no two segments in M^- intersect. The definition of ham-sandwich cut implies that all segments in M^+ are above ℓ and all segments in M^- are below ℓ, so no segment in M^+ intersects any segment in M^-. We conclude that no two segments in $M^+ \cup M^-$ intersect, which completes the proof.

Timothy helpfully computes the ham-sandwich run ℓ in $O(n)$ time. We can extract the subsets R^+, B^+, R^-, and B^- from R and B using $O(n)$ calls to Above? in $O(n)$ time. The definition of ham-sandwich cut implies $|R^+| = |B^+| = \lfloor n/2 \rfloor$, and therefore $|R^-| = |B^-| = \lceil n/2 \rceil$. Thus:

The running time of NoCrossMatching satisfies the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$. We conclude that the algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***. ■

Rubric: 10 points. The proofs in gray are not required for full credit.

CS/ECE 374 A ✧ Fall 2025
Midterm 2 Problem 4 Solution

Describe and analyze an algorithm that correctly determines whether a given directed graph G, each of whose vertices is labeled "badger", "mushroom", or "snake", contains an orthodox closed walk.

Solution (fewer layers, many searches): We construct a layered graph $G' = (V', E')$ as follows:

- $V' = V \times \{1, 2, \dots, 14\}$
- E' is the union of two subsets

$$\big\{(u,i)\to(v,i+1) \;\big|\; u\to v \in E \text{ and } 1 \le i \le 11 \text{ and } u \text{ is a badger}\big\} \cup$$
$$\big\{(u,i)\to(v,i+1) \;\big|\; u\to v \in E \text{ and } 12 \le i \le 13 \text{ and } u \text{ is a mushroom}\big\}$$

G' has $14V$ vertices and (crudely) at most $13E$ edges. Every orthodox closed walk in G corresponds to a path in G' from $(v,1)$ to $(v,14)$ for some badger vertex v. We can check whether G' contains such a path by running whatever-first search at each badger vertex $(v.0)$ in $O(V'(V'+E')) = \boldsymbol{O(V(V+E))}$ ***time***. ■

Solution (more layers, one search): We construct a layered graph $G' = (V', E')$ as follows:

- $V' = \{s\} \cup (B \times V \times \{0, 2, \dots, 13\})$, where $B \subseteq V$ is the subset of badger vertices in G.

  Each vertex (b, v, t) means that we started at badger vertex b and we are now at vertex v after traversing exactly t edges.
- E' is the union of *three* subsets

$$\big\{s\to(b,b,0) \;\big|\; b \text{ is a badger}\big\}$$
$$\cup \left\{(b,u,t)\to(b,v,t+1) \;\middle|\; \begin{array}{c} b \text{ is a badger, } v \text{ is a badger,} \\ u\to v \in E, \text{ and } 0 \le t \le 10 \end{array}\right\}$$
$$\cup \left\{(b,u,t)\to(b,v,t+1) \;\middle|\; \begin{array}{c} b \text{ is a badger, } v \text{ is a mushroom,} \\ u\to v \in E, \text{ and } 11 \le t \le 12 \end{array}\right\}$$

G' has $O(V^2)$ vertices and $O(VE)$ edges. Every orthodox closed walk in G corresponds to a path in G' from s to $(b, b, 13)$ for some badger vertex b. We can check whether G' contains such a path by running whatever-first search at s, in $O(V'+E') = \boldsymbol{O(V^2+VE)}$ ***time***. ■

Rubric: 10 points: standard graph reduction rubric. These are not the only correct solutions! Solving this problem in $o(VE)$ time would be a ***major*** breakthrough.

Non-solution: We construct a layered graph $G' = (V', E')$ as follows:

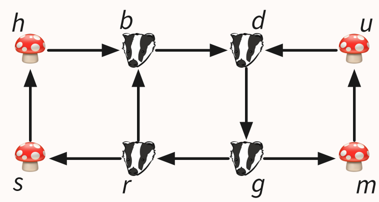
- $V' = V \times \{0, 1, \ldots, 12\}$
- E' is the union of two subsets

$$\Big\{(u,i){\to}(v,i+1) \;\Big|\; u{\to}v \in E \text{ and } 0 \le i \le 10 \text{ and } u \text{ is a badger}\Big\} \cup$$
$$\Big\{(u,i){\to}(v,i+1 \textbf{ mod } \mathbf{13}) \;\Big|\; u{\to}v \in E \text{ and } 11 \le i \le 12 \text{ and } u \text{ is a mushroom}\Big\}$$

G' has $O(V)$ vertices and $O(E)$ edges. Every orthodox walk in G corresponds to a cycle in G', so we need to determine whether G' contains any cycles. We can test whether G' is acyclic using any topological sorting algorithm in $\boldsymbol{O(V+E)}$ ***time***. ♣

While it is true that every orthodox walk in G corresponds to a cycle in G', not every cycle in G' corresponds to an orthodox walk in G. Only cycles ***with length* 13** correspond to orthodox walks. It is possible for a cycle in G' to "wrap around" the layers more than once.

Consider the following example.



This graph G does not contain an orthodox walk, but it does contain a "double-orthodox" walk of length 26:

$$b{\to}d{\to}g{\to}r{\to}b{\to}d{\to}g{\to}r{\to}b{\to}d{\to}g{\to}m{\to}u{\to}d{\to}g{\to}r{\to}b{\to}d{\to}g{\to}r{\to}b{\to}d{\to}g{\to}r{\to}s{\to}h{\to}b$$

The corresponding layered graph G' is not a dag, but its shortest cycle has length 26.

We can fix this reduction by looking for the *shortest* cycle in G', instead of an *arbitrary* cycle, but the fastest algorithms known to find shortest cycles run in $O(VE)$ time, just like our earlier solutions.

CS/ECE 374 A ✧ Fall 2025
Midterm 2 Problem 5 Solution

Suppose you are given two arrays $NY[1..n]$ and $SF[1..n]$ containing the profits you can earn in either New York or San Francisco for each of the next n weeks. Flying between the two cities costs \$1000. You want to design a travel schedule that yields the maximum *total* profit.

(a) ***Prove*** that the obvious greedy strategy—each week, work in the city with higher profit—does ***not*** always yield the maximum total profit.

Solution: Let $NY = [1, 2]$ and $SF = [2, 1]$. We can earn a total profit of \$3 by staying in either city for both weeks, but the greedy algorithm actually *loses* \$996. ■

(b) Describe and analyze an algorithm to compute the maximum total profit you can earn.

Solution (dynamic programming): We define two functions:

- $MaxProfitSF(i)$ is the total profit we can make starting with week i, assuming we spend week $i-1$ in San Francisco.
- $MaxProfitNY(i)$ is the total profit we can make starting with week i, assuming we spend week $i-1$ in New York.

We need to compute $\max\{MaxProfitSF(1), MaxProfitNY(1)\}$. (It doesn't matter where we spend week 0!) These functions obey the following mutual recurrences:

$$MaxProfitSF(i) = \begin{cases} 0 & \text{if } i > n \\ \max\left\{ \begin{array}{c} SF[i] + MaxProfitSF(i+1) \\ NY[i] - 1000 + MaxProfitNY(i+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

$$MaxProfitNY(i) = \begin{cases} 0 & \text{if } i > n \\ \max\left\{ \begin{array}{c} SF[i] - 1000 + MaxProfitSF(i+1) \\ NY[i] + MaxProfitNY(i+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize these functions into two one-dimensional arrays, which we can fill in parallel from right to left in ***O(n) time***. ■

Rubric: 8 points: standard dynamic programming rubric. This is not the only correct dynamic programming algorithm.

Solution (graph reduction): We define a directed acyclic graph $G = (V, E)$ as follows:

- $V = \{s, t\} \cup \{1, 2, \ldots, n\} \times \{\mathsf{SF}, \mathsf{NY}\}$ — Each vertex (i, c) means we are spending week i in city c.
- E contains three types of edges:
  - start edges $\{s{\to}(1, \mathsf{NY}),\ s{\to}(1, \mathsf{SF})\}$
  - regular edges

$$\left\{ \begin{matrix} (i,\mathsf{SF}){\to}(i+1,\mathsf{SF}), & (i,\mathsf{SF}){\to}(i+1,\mathsf{NY}), \\ (i,\mathsf{NY}){\to}(i+1,\mathsf{SF}), & (i,\mathsf{NY}){\to}(i+1,\mathsf{NY}) \end{matrix} \;\middle|\; 1 \le i \le n-1 \right\}$$

  - end edges $\{(n, \mathsf{SF}){\to}t,\ (n, \mathsf{NY}){\to}t\}$
- Edges are weighted as follows:
  - Start edge $s \to (1, \mathsf{SF})$ has weight $SF[1]$
  - Start edge $s \to (1, \mathsf{NY})$ has weight $NY[1]$
  - Each regular edge $(i, \mathsf{SF}){\to}(i+1, \mathsf{SF})$ has weight $SF[i+1]$
  - Each regular edge $(i, \mathsf{NY}){\to}(i+1, \mathsf{NY})$ has weight $NY[i+1]$
  - Each regular edge $(i, \mathsf{SF}){\to}(i+1, \mathsf{NY})$ has weight $NY[i+1] - 1000$
  - Each regular edge $(i, \mathsf{NY}){\to}(i+1, \mathsf{SF})$ has weight $SF[i+1] - 1000$
  - End edges have weight 0

Altogether G has $2n+2$ vertices and $4n$ edges. G is a dag because every edge either leaves s, increments the week, or enters t. We need to compute the length of the longest path from s to t. We can do that using the LongestPath algorithm presented in class in $O(V+E) = \boldsymbol{O(n)}$ ***time***. ■

Rubric: 8 points: standard graph reduction rubric. This is not the only correct reduction to longest path. These two solutions are arguably identical.