

**CS/ECE 374 A ✦ Spring 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.)

$$A(n) = 4A(n/9) + O(\sqrt{n}) \quad B(n) = 2B(n/3) + B(n/6) + O(n) \quad C(n) = 9C(n/3) + O(n^2)$$

$$O(n^{\log_3 2})$$

$$O(n)$$

$$O(n^2 \log n)$$

**Solution:** Level  $\ell$  of the recursion tree for  $A(n)$  has  $4^\ell$  nodes, each with value  $\sqrt{n}/3^\ell$ , so the total value at level  $\ell$  is  $(4/3)^\ell \sqrt{n}$ . The level sums form an increasing geometric series, so only the number of leaves matters. The depth of the tree is  $\log_9 n$ , so  $A(n) = O(4^{\log_9 n}) = O(n^{\log_9 4}) = O(n^{\log_3 2})$ .

The recursion tree for  $B(n)$  is a ternary tree with unbalanced problem sizes. The root has value  $n$ . The root has three children, two with value  $n/3$  and one with value  $n/6$ , so the total value of the children is  $5n/6$ . More generally, level  $\ell$  of the recursion tree for  $B(n)$  has total value  $(5/6)^\ell n$ . 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  $\ell$  of the recursion tree for  $C(n)$  has  $9^\ell$  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  $C(n) = O(n^2 \log n)$ . ■

**Rubric:**

- 2.5 points each for  $A(n)$  and  $B(n)$ .
- 2 points for  $C(n)$ .
- 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 computing  $BestFlobble(1, n)$  using the following recurrence, and state the running time of the resulting iterative algorithm.

$$BestFlobble(i, k) = \begin{cases} 0 & \text{if } i \geq k \\ \min \left\{ \begin{array}{l} BestFlobble(i, j-1) \\ + (j-i) \cdot A[j] \\ + BestFlobble(j, k) \end{array} \middle| i < j \leq k \right\} & \text{otherwise} \end{cases}$$

**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  $BestFlobble[i, k]$  requires a for loop over  $j$ .) ■

**Rubric:** 3 points = 1 for correct structure + 1 for correct evaluation order + 1 for running time.

**CS/ECE 374 A ✦ Spring 2026**  
**Midterm 2 Problem 2 Solution**

Consider the following context-free grammar (CFG)  $G$ :

$$S \rightarrow \emptyset S 1 \mid 1 S \emptyset \mid \varepsilon$$

(See the questions handout for a refresher on the strings generated by  $G$ .)

Describe an algorithm that given a string  $w[1..n] \in \{0, 1\}^*$  computes the length of the longest subsequence of  $w$  that is generated by  $G$ .

**Solution (dynamic programming):** Let  $LGS(i, j)$  denote the length of the longest subsequence of  $w[i..j]$  that is generated by  $G$ . We need to compute  $LGS(1, n)$ . The function obeys the following recurrence:

$$LGS(i, j) = \begin{cases} 0 & \text{if } i \geq j \\ \max \left\{ \begin{array}{l} LGS(i+1, j), \\ LGS(i, j-1) \end{array} \right\} & \text{if } i < j \text{ and } w[i] = w[j] \\ \max \left\{ \begin{array}{l} 2 + LGS(i+1, j-1), \\ LGS(i+1, j), \\ LGS(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a 2D array evaluated in decreasing order of  $i$  and increasing order of  $j$  with the loops nested either way. The resulting iterative algorithm takes  $O(n^2)$  time total. ■

**Rubric:** 10 points: standard dynamic programming rubric.

**Solution (graph reduction):** We define an edge-weighted directed acyclic graph  $H = (V, E)$  as follows:

- $V := \{t\} \cup \{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$

Vertex  $(i, j)$  means we seek a longest subsequence of  $w[i..j]$  generated by  $G$ .

- $E := \{((i, j), (i + 1, j)) \mid 1 \leq i < j \leq n\} \cup$   
 $\{((i, j), (i, j - 1)) \mid 1 \leq i < j \leq n\} \cup$   
 $\{((i, j), (i + 1, j - 1)) \mid 1 \leq i < j \leq n \text{ and } w[i] \neq w[j]\} \cup$   
 $\{((i, j), t) \mid 1 \leq j \leq i \leq n\}$

- Each edge  $((i, j), (i + 1, j - 1))$  gets a weight of 2.
- All other edges get a weight of 0.

The length of the longest subsequence of  $w$  generated by  $G$  is the length of the longest path from  $(1, n)$  to  $t$  in  $H$ . Accordingly, we call the algorithm for longest path in a dag seen in the book from  $(1, n)$  to  $t$ . Alternatively, we negate the edge weights and call DAGSSSP on  $(1, n)$ .

The resulting algorithm takes  $O(|V| + |E|) = O(n^2)$  time. ■

**Rubric:** 10 points: standard graph reduction rubric.

**CS/ECE 374 A ✦ Spring 2026**  
**Midterm 2 Problem 3 Solution**

Describe an algorithm to compute the minimum cost to get from your friend's house at vertex  $s$  to the Transit Authority at vertex  $t$ .

**Solution:** We reduce to shortest paths in a new directed graph  $G' = (V', E')$  with positive edge lengths  $\ell' : E' \rightarrow \mathbb{R}^+$  as follows.

- $V' := V \times \{A, B, C\}$ .

Vertex  $(v, a)$  means we're at station  $v$  while appearing to be a member of System  $a$ .

- $E' := \{((u, a), (v, a)) \mid (u, v) \in E, a \in \{A, B, C\}\} \cup \{((v, A), (v, B)) \mid v \in V\} \cup \{((v, B), (v, C)) \mid v \in V\}$ .
- $\ell'(((u, a), (v, a))) := \$(u, v)/2$  if  $system((u, v)) = a$ .
- $\ell'(((u, a), (v, a))) := \$(u, v)$  if  $system((u, v)) \neq a$ .
- $\ell'(((v, a), (v, b))) := 0$  for all edges  $((v, a), (v, b)) \in E'$ .

The minimum cost to get from  $s$  to  $t$  in  $G$  is equal to the distance from  $(s, A)$  to  $(t, C)$  in  $G'$ . Therefore, we run Dijkstra's algorithm from  $(s, A)$ .

Our algorithm runs in  $O(|E'| \log |V'|) = O(m \log n)$  time. ■

**Rubric:** 10 points: standard graph reduction rubric.

**CS/ECE 374 A ♦ Spring 2026**  
**Midterm 2 Problem 4 Solution**

Suppose you are given a sorted array of  $n$  distinct numbers that has been rotated *forward*  $k$  steps for some **unknown** integer  $k$  between 1 and  $n - 1$ . In other words, you are given an array  $A[1..n]$  such that some prefix  $A[1..k]$  is sorted in increasing order, the complementary suffix  $A[k + 1..n]$  is sorted in increasing order, and  $A[n] < A[1]$ .

Describe an algorithm to determine if the given array  $A$  contains a given number  $x$ .

**Solution (explicitly recursive binary search):** The following procedure solves the problem as stated, implying  $1 \leq k < n$ .

```
ROTHasX(A[1..n], x):
  if n = 2
    return A[1] = x ∨ A[2] = x
  m ← ⌈n/2⌉
  if A[1] ≤ A[m] << 1 < m ≤ k >>
    if A[1] ≤ x and x ≤ A[m]
      return result of standard binary search for x over A[1..m]
    else
      return ROTHasX(A[m..n])
  else << k < m < n >>
    if A[m] ≤ x and x ≤ A[n]
      return result of standard binary search for x over A[m..n]
    else
      return ROTHasX(A[1..m])
```

We claim the algorithm return TRUE if and only if  $x$  is contained in the array  $A[1..n]$ . Assume the algorithm is correct for all strictly shorter arrays that meet the specifications of the problem. Suppose  $n = 2$ . The algorithm correctly determines if  $x$  is in  $A$  via brute force. Suppose from here on that  $n > 2$ .

If  $A[1] \leq A[m]$ , then  $1 < m \leq k$ . If, in addition,  $A[1] \leq x$  and  $x \leq A[m]$ , then  $x$  appears in the increasing array  $A[1..m]$  if it appears at all, and we can find it via standard binary search. Otherwise, either  $x < A[1]$  or  $x > A[m]$ , implying it appears in  $A[m..n]$  if it appears at all. The latter subarray follows the specifications of the problem and has fewer than  $n$  elements, so we can recursively search it by the induction hypothesis.

If  $A[1] > A[m]$ , then  $k < m < n$ . If, in addition,  $A[m] \leq x$  and  $x \leq A[n]$ , then  $x$  appears in the increasing array  $A[m..n]$  if it appears at all, and we can find it via standard binary search. Otherwise, either  $x < A[m]$  or  $x > A[n]$ , implying it appears in  $A[1..m]$  if it appears at all. The latter subarray follows the specifications of the problem and has fewer than  $n$  elements, so we can recursively search it by the induction hypothesis.

The running time of ROTHasX satisfies the binary search recurrence of  $T(n) = T(n/2) + O(1)$  until it hits a single base case call that takes time  $O(\log n)$ . We conclude that the algorithm runs in  **$O(\log n)$  time**. ■

**Rubric:** 10 points: standard recursive algorithm rubric. The proofs in gray are not required for full credit. A correct  $O(n)$  time algorithm is worth 3 points.

**Solution (iterative binary search):** The following procedure solves the problem as stated, implying  $1 \leq k < n$ .

```

ROTHasX(A[1..n], x):
  lo ← 1
  hi ← n
  repeat
    if hi - lo = 1
      return A[lo] = x ∨ A[hi] = x
    mid ← ⌊(hi + lo)/2⌋
    if A[lo] ≤ A[mid]
      if A[lo] ≤ x and x ≤ A[mid]
        return result of standard binary search for x over A[lo..mid]
      else
        lo ← mid
    else
      if A[mid] ≤ x and x ≤ A[hi]
        return result of standard binary search for x over A[mid..hi]
      else
        hi ← mid

```

See the previous solution for a proof of correctness.

The algorithm is a binary search that turns into a different binary search exactly once, so it runs in  $O(\log n)$  time. ■

**Rubric:** 10 points: standard recursive algorithm rubric. This and the previous solution are the same algorithm.

**Solution:** We compute  $k$  using the algorithm shown in the Conflict Midterm 2 solutions. If  $x \geq A[1]$ , then we do a standard binary search in  $A[1..k]$ . Otherwise, we do a standard binary search in  $A[k+1..n]$ . *The algorithm takes  $O(\log n)$  time.* ■

**Rubric:** 10 points: standard recursive algorithm rubric. You do have to actually describe the algorithm to find  $k$  and not simply refer to something you shouldn't know exists.

**CS/ECE 374 A ✦ Spring 2026**  
**Midterm 2 Problem 5 Solution**

Describe an algorithm to compute the maximum score you can achieve over the course of the solitaire card game described in the questions handout. You may assume the input to your algorithm is an array  $Cards[1..n]$  where  $Cards[i]$  is the number written on the  $i$ th card from the left of those initially on the table.

**Solution (dynamic programming):** Let  $MaxScore(i, j)$  denote the following: if  $i = 0$ , then it is the maximum score we can obtain playing with cards  $j$  through  $n$  (numbered left to right in the input) assuming we *are not* holding a card when we begin. Otherwise, it is the maximum score we can obtain playing with cards  $j$  through  $n$  assuming we are holding the  $i$ th card from the left in the original input. We need to compute  $MaxScore(0, 1)$ . The function obeys the following recurrence:

$$MaxScore(i, j) = \begin{cases} 0 & \text{if } j > n \text{ and } i = 0 \\ 10 \cdot Cards[i] & \text{if } j > n \text{ and } i \geq 1 \\ \max \left\{ \begin{array}{l} MaxScore(0, j + 1), \\ MaxScore(j, j + 1) \end{array} \right\} & \text{if } j \leq n \text{ and } i = 0 \\ \max \left\{ \begin{array}{l} MaxScore(i, j + 1), \\ Cards[i] \cdot Cards[j] + MaxScore(0, j + 1) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a 2D array evaluated in decreasing order of  $j$  in the outer loop and arbitrary order of  $i$  in the inner loop. The resulting iterative algorithm takes  $O(n^2)$  time total. ■

**Rubric:** 10 points: standard dynamic programming rubric.

**Solution (graph reduction):** We define an edge-weighted directed acyclic graph  $G = (V, E)$  as follows:

- $V := \{t\} \cup \{(i, j) \mid 0 \leq i < j \leq n + 1\}$

Vertex  $(i, j)$  means we reach card  $j$  while holding nothing (when  $i = 0$ ) or card  $i$  (when  $i \geq 1$ ).

- $E := \{((i, j), (0, j + 1)) \mid 0 \leq i < j \leq n\} \cup$   
 $\{((i, j), (i, j + 1)) \mid 1 \leq i < j \leq n\} \cup$   
 $\{((0, j), (j, j + 1)) \mid 1 \leq j \leq n\} \cup$   
 $\{((i, n + 1), t) \mid 0 \leq i \leq n\}$

- Each edge  $((i, j), (0, j + 1))$  with  $i \geq 1$  gets a weight of  $Cards[i] \cdot Cards[j]$ .
- Each edge  $((i, n + 1), t)$  with  $i \geq 1$  gets a weight of  $10 \cdot Cards[i]$ .
- All other edges get a weight of 0.

The maximum score we can achieve is the length of the longest path from  $(0, 1)$  to  $t$  in  $G$ . Accordingly, we call the algorithm for longest path in a dag seen in the book from  $(0, 1)$  to  $t$ . Alternatively, we negate the edge weights and call DAGSSSP on  $(0, 1)$ .

The resulting algorithm takes  $O(|V| + |E|) = O(n^2)$  time. ■

**Rubric:** 10 points: standard graph reduction rubric.