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:

   - $A(n) = A(5n/11) + O(\sqrt{n})$
   - $B(n) = 8B(n/2) + O(n^2)$
   - $C(n) = C(n/2) + C(n/3) + C(n/6) + O(n)$

   Big-Oh notation make them clearly poly in n

   (b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrences, and state the running time of the resulting iterative algorithm to compute the requested function value.

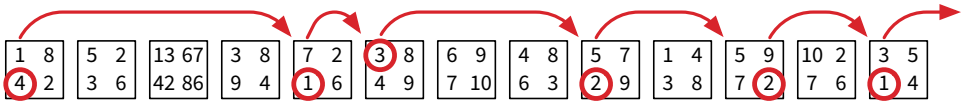
   - Compute $Foo(1, n)$ where

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

   - Compute $Bar(n, 1)$ where

$$Bar(i,s) = \begin{cases} \infty & \text{if } i < 0 \text{ or } s > n \\ 0 & \text{if } i = 0 \\ \min\left\{ \begin{array}{c} Bar(i, 2s), \\ X[i] \cdot s + Bar(i-s, s) \end{array} \right\} & \text{otherwise} \end{cases}$$

2. *Quadhopper* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A quadhopper puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

Counterexample

   (a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every quadhopper puzzle.

   (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given quadhopper puzzle.

DP or graphs

graphs!

3. After moving to a new city, you decide to walk from your home to your new office. To get a good daily workout, you want to reach the *highest* possible altitude during your walk (to maximize exercise), while keeping the total length of your walk below some threshold (to get to your office on time). Describe and analyze an algorithm to compute the best possible walking route.

   Your input consists of an undirected graph G, where each vertex v has a height $h(v)$ and each edge e has a positive length $\ell(e)$, along with a start vertex s, a target vertex t, and a maximum length L. Your algorithm should return the *maximum height reachable* by a walk from s to t in G, whose total length is at most L.

DP? But that may be slow..

4. Suppose you are given a string of symbols, representing a message in some foreign language that you do not understand, in an array $T[1..n]$. You have access to a black-box subroutine IsWord that can decide whether an arbitrary string w is a word in $O(|w|)$ time.

   You eagerly implement and run the text-splitting algorithm we saw in class, only to discover that the given string *cannot* be split into words! Apparently, as a crude form of cryptography, the message has been corrupted by adding extra symbols between words.

   So you decide instead to look for as many non-overlapping words in T as possible. A ***verbal subsequence*** of T is a sequence of non-overlapping substrings of T, each of which is a word. The *length* of a verbal subsequence is the number of words it contains. Describe and analyze an algorithm to find the length of the longest verbal subsequence of a given string T.

   For example, suppose IsWord(w) returns True if and only if w is a common English word. Then (STUDY, AM, ICE, TRAP, RAMBLE) and (DYNAMIC, EXTRA, PROGRAM) are verbal subsequences of the string STUDYNAMICEXTRAPROGRAMBLE:

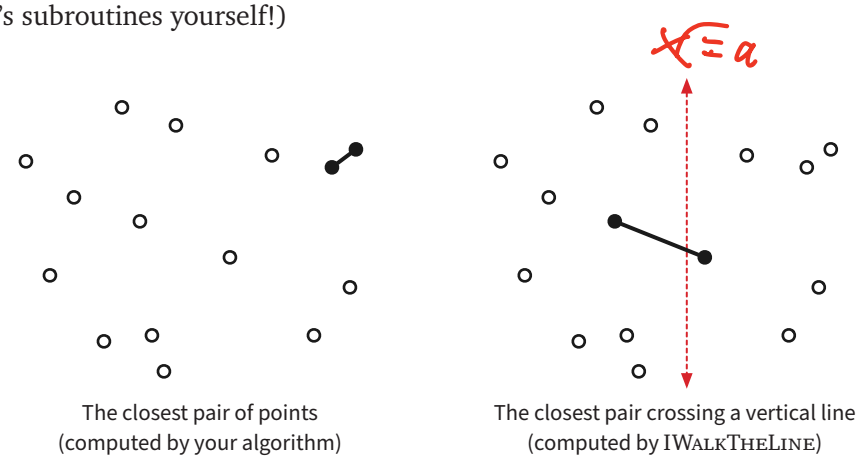
   Thus, given the input string STUDYNAMICEXTRAPROGRAMBLE, the output of your algorithm should be *at least* 5, which is the length of the subsequence (STUDY, AM, ICE, TRAP, RAMBLE). (This string may contain longer verbal subsequences.)

DP!

5. This problem asks you to design an algorithm to find the closest pair of points in a given set P of n points in the plane. Points in P are represented by an array $P[1..n]$ of coordinate pairs; the coordinates of the ith point are $(P[i].x, P[i].y)$. The points are sorted by their x-coordinates, and all x- and y-coordinates are distinct. In particular, we have $P[i].x < P[i+1].x$ for every index i. Timothy has helpfully implemented a pair of subroutines for you.

   - IWALKTHELINE(P, a) computes the closest pair of points in a given n-point set P that cross the vertical line $x = a$, in $O(n)$ time. Specifically, this function returns the indices i and j of the closest pair of points such that $P[i].x \le a < P[j].x$. See the figure below for an example.
   - DISTANCE(p, q) returns the Euclidean distance between two points p and q, in $O(1)$ time.

   Describe and analyze an algorithm to compute the Euclidean distance between the closest pair of points in P, using Timothy's subroutines as black boxes. (Do *not* try to implement Timothy's subroutines yourself!)



The closest pair of points (computed by your algorithm)

The closest pair crossing a vertical line (computed by IWALKTHELINE)

divide-and-conquer?

Midterm 2 Mon Nov 9th 7pm - 9pm
Conflict Midterm 2 Tue Nov 10 (morning)
ACM Review Siebel 0216 Sat Nov 7th ~12pm-4pm
Also Friday labs & sunday party!

CS/ECE 374 A ✧ Fall 2026

Midterm 2 Practice 1

November 5, 2026

Name:	Emily Fox
NetID:	ekfox

- ***Don't panic!***
- You have 120 minutes to answer five questions. **The questions are described in more detail in a separate handout.**
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, and your university ID, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- Please clearly print your name and your NetID in the boxes above.
- Please also print your name at the top of every page of the answer booklet, except this cover page. We want to make sure that if a staple falls out, we can reassemble your answer booklet. (It doesn't happen often, but it does happen.)
- Greedy algorithms require formal proofs of correctness to receive credit. Otherwise, proofs or other justifications beyond those listed in the standard grading rubrics are required if and only if we explicitly ask for them, using the word ***prove*** or ***justify*** in bold italics.

- **Do not write outside the black boxes on each page**. These indicate the area of the page that our scanners will actually scan. If the scanner can't see your work, we can't grade it.
- If you run out of space for an answer, feel free to use the overflow/scratch pages at the back of the answer booklet, but **clearly indicate where we should look**. If we can't find your work, we can't grade it.
- **Only work that is written into the stapled answer booklet will be graded**. We **do not** recommend detaching pages from the answer booklet, because any work on those detached pages will not be graded (and it disrupts our process for uploading solutions to Gradescope.) Please let us know if you detach a page either on purpose or accidentally so we can reattach it.
- We will provide additional blank scratch paper on request, but we will not grade anything written on it.
- **Please return all paper to Emily or a TA** including your **answer booklet**, your question sheet, your cheat sheet, and all scratch paper. There will be two piles of collected paper, the first for answer booklets and the second for everything else. We **will not** consider the paper in the second pile when grading your exam.

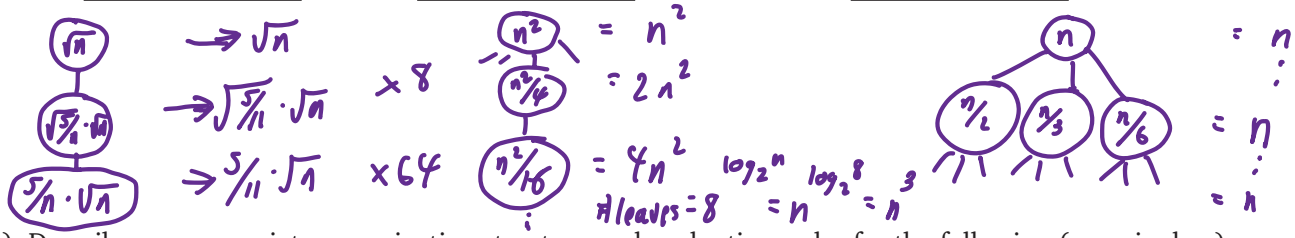
This sentence contains four and three fourths percent a's, five and one half percent c's, three percent d's, eighteen and one fourth percent e's, four and one half percent f's, three fourths percent g's, five percent h's, two and one half percent i's, two percent l's, twelve and one half percent n's, six percent o's, five percent p's, eight percent r's, seven percent s's, nine and three fourths percent t's, two and one fourth percent u's, one and one half percent v's, one and one fourth percent w's and one half percent x's.

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 1 Problem 1	Name: Emily Fox

(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.)

$A(n) = A(5n/11) + O(\sqrt{n})$ $B(n) = 8B(n/2) + O(n^2)$ $C(n) = C(n/2) + C(n/3) + C(n/6) + O(n)$

$O(\sqrt{n})$ $O(n^3)$ $O(n \log n)$



(b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrences, and state the running time of the resulting iterative algorithm to compute the requested function value.

- Compute $Foo(1, n)$ where

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

Use a 2D array $Foo[1..n-1, 2..n]$



-or- for i decreasing
for k increasing

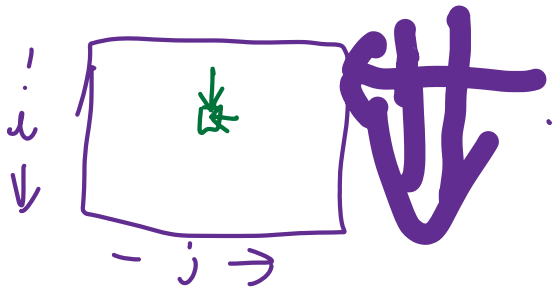
Time: $O(n) \cdot O(n^2) = O(n^3)$

- Compute $Bar(n, 1)$ where

$$Bar(i,s) = \begin{cases} \infty & \text{if } i < 0 \text{ or } s > n \\ 0 & \text{if } i = 0 \\ \min\left\{ \begin{array}{c} Bar(i, 2s), \\ X[i] \cdot s + Bar(i-s, s) \end{array} \right\} & \text{otherwise} \end{cases}$$

Very sneaky worth EC?

(sneaky): Store in 2D array $Bar[0..n, 0..\lceil \lg n \rceil]$ where $Bar[i,j] = Bar(i, 2^j)$



-or- for j increasing
for j decreasing

Time: $O(1) \cdot O(n \log n) = O(n \log n)$

Would be $O(n^2)$ if s weren't doubling.

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 1 Problem 2	Name: Emily Fox

Quadhopper is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.

(a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every quadhopper puzzle.

(b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given quadhopper puzzle.

(a) Consider the instance:



Greedy goes 1 → 100 →

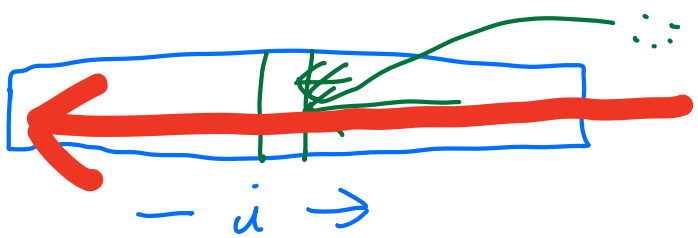
Better is 2 → 1 → 1 →

(b) Number the squares 1 to n from left to right. Let MaxMoves(i) be max # moves playing game on squares i through n. Want to compute MaxMoves(1).

$$MaxMove(i) = \begin{cases} 0 & \text{if } i > n \\ 1 + \max\{MaxMove(i+x) \mid x \text{ is on square } i\} & \text{o.w.} \end{cases}$$

Store in MaxMove[1..n] (& use 0 for all other subproblems)

-or- solve in decreasing i order. Time: O(n)



CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 1 Problem 3	Name: Emily Fox

After moving to a new city, you decide to walk from your home to your new office. To get a good daily workout, you want to reach the *highest* possible altitude during your walk (to maximize exercise), while keeping the total length of your walk below some threshold (to get to your office on time). Describe and analyze an algorithm to compute the best possible walking route.

Your input consists of an undirected graph G, where each vertex v has a height $h(v)$ and each edge e has a positive length $\ell(e)$, along with a start vertex s, a target vertex t, and a maximum length L. Your algorithm should return the *maximum height reachable* by a walk from s to t in G, whose total length is at most L.

vertex of height $\leq h$ & length $\leq L$.

Challenge: Find min height h s.t. can use only

How to check if we can reach some $v \in V$ & get to work on time? Is $dist(s,v) + dist(v,t) \leq L$?

$\Leftrightarrow dist(s,v) + dist(t,v) \leq L$?

We can walk to any $v \in V$ from s & get to t with length $\leq L$ iff distance from s to v + distance from t to $v \leq L$.

So, want max $h(v)$ of all such v.

Alg: Run Dijkstra(s) on G with edge weights ℓ & store distances as values $dist(s,v)$.

Run Dijkstra(t) on G with weights ℓ & store distances as values $dist(t,v)$.

Return $\max\{h(v) \mid dist(s,v) + dist(t,v) \leq L\}$.

Let $G=(V,E)$.

Runs in $O(|E| \log |V|) = 2 \cdot O(|E| \log |V|) + O(|V|)$

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 1 Problem 4	Name: Emily Fox

A ***verbal subsequence*** of a string T is a sequence of non-overlapping substrings of T, each of which is a word. The *length* of a verbal subsequence is the number of words it contains. Describe and analyze an algorithm to find the length of the longest verbal subsequence of a given string $T[1..n]$. You have access to a black-box subroutine IsWord that can decide whether an arbitrary string w is a word in $O(|w|)$ time.

---

do we keep first symbol of suffix?

1 . . . . . | [?]

Let LVS(i) be the length of the longest verbal subsequence of $T[i..n]$.

We want to compute LVS(1).

$$LVS(i) = \begin{cases} 0 & i > n \\ \max\begin{cases} LVS(i+1), & \leftarrow \text{skip } T[i] \\ \max\{1 + LVS(j+1) \mid i \le j \le n \text{ and } IsWord(T[i..j])\} & \uparrow \text{start with some } T[i..j] \end{cases} & \text{o.w.} \end{cases}$$

where inner max over $\emptyset$ is $-\infty$.

Store in 1D array $LVS[1..n+1]$.



-or-

for i descending

Time: $O(n^2) \cdot O(n) = O(n^3)$

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 1 Problem 5	Name: Emily Fox

Describe and analyze an algorithm to compute the Euclidean distance between the closest pair of points in a given set P of n points in the plane, using subroutines IWalkTheLine and Distance as black boxes.

(See the question handout for definitions of the black-box subroutines.)

Procedure MinPair(P[1..n]) returns min pairwise distance for points in P[1..n]. (∞ if no pairs)

```
MinPair(P[1..n]):
  if n < 2                    (or use n=2 & n=3 as base cases)
    return ∞
  else
    m ← ⌊n/2⌋
    (i, j) ← IWalkTheLine(P[1..n], P[m].x)
    across ← Distance(P[i], P[j])
    left ← MinPair(P[1..m])
    right ← MinPair(P[m+1...n])
    return min{across, left, right}
```

Time: $T(n) = 2T(n/2) + n$

$= O(n \log n)$

(scratch paper)

(scratch paper)

(scratch paper)

(scratch paper)

(scratch paper)