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:

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

   (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.

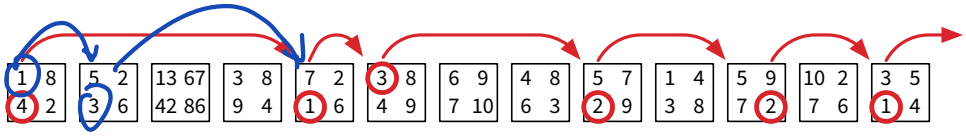
   - Compute $Foo(1, n)$ where

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

   - Compute $Bar(n, 1)$ where

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

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.)

   (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.

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.

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.)

5. Recall that an *arithmetic progression* is any sequence of real numbers $x_1, x_2, \ldots, x_n$ such that $x_{i+1} - x_i = x_i - x_{i-1}$ for every index $2 \le i \le n-1$.

   Suppose we are given a sorted array $X[1..n]$ that contains an arithmetic sequence ***with one element deleted***. Describe and analyze an algorithm to find the deleted element as quickly as possible. If there are multiple correct answers, your algorithm can return any one of them. To avoid annoying boundary cases, you can assume $n \ge 4$.

   For example, given the input array $X = [2, 4, 8, 10, 12]$, your algorithm should return 6, and given the array $X = [21, 18, 15, 12]$, your algorithm should return either 9 or 24.

CS/ECE 374 A ✧ Fall 2025

Midterm 2 Practice 3

November 9, 2025

Name:	Jeff E
NetID:	jeffe

- ***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 any credit, even if they are correct. Otherwise, proofs or other justifications are required for full credit if and only if we explicitly ask for them, using the word ***prove*** or ***justify*** in bold italics.

- **Please do not write outside the black boxes on each page.** These indicate the area of the page that our scanners can actually scan. If the scanner can't see your work, we can't grade it.
- If you run out of space for an answer, please use the overflow/scratch pages at the back of the answer booklet, but **please 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**. In particular, you are welcome to detach scratch pages from the answer booklet, but any work on those detached pages will not be graded. We will provide additional scratch paper on request, but any work on that scratch paper will not be graded.

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 2025 Midterm 2 Practice 3 Problem 1	Name: Jeff

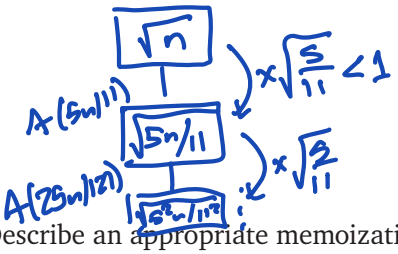
(a) Write the solution to each of the following recurrences in the box immediately below it. (Use the space below the boxes for scratch work.)

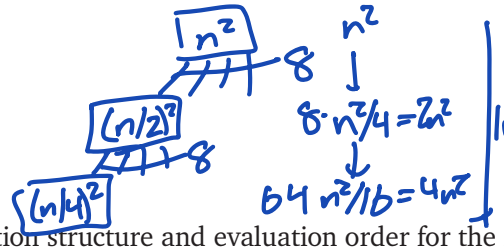
$$A(n) = A(5n/11) + O(\sqrt{n}) \quad B(n) = 8B(n/2) + O(n^2) \quad C(n) = C(n/2) + C(n/3) + C(n/6) + O(n)$$

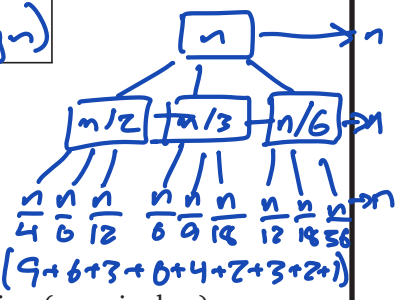
$O(\sqrt{n})$

$O(n^3)$

$O(n \log n)$

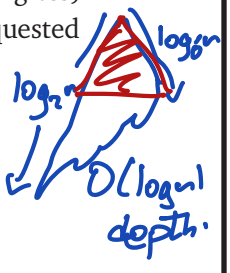






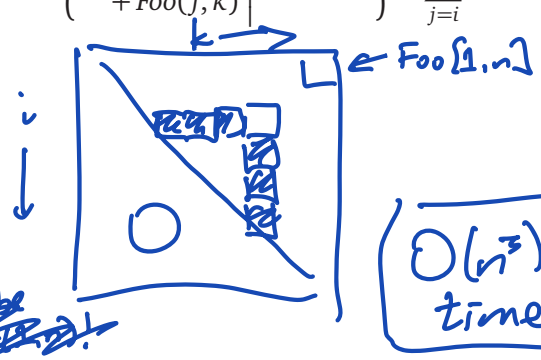
(b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrences, and state the running time of the resulting iterative algorithm to compute the requested function value.

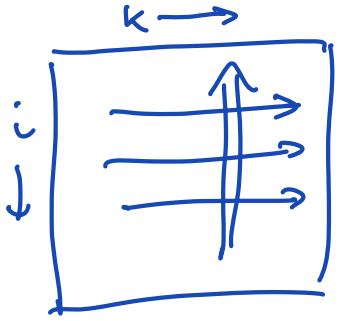
$8^{\log_2 n} := n^{\log_2 8} = n^3$



- Compute $Foo(1, n)$ where

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



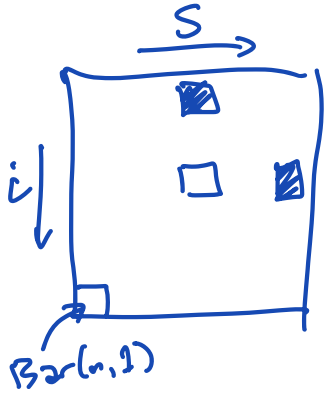


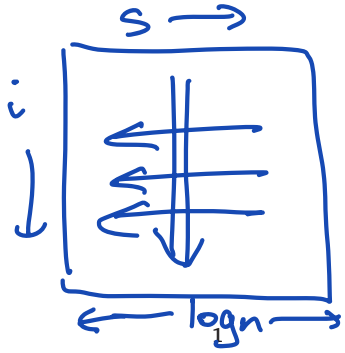
wrong should be $Foo(1,n)$!

- Compute $Bar(n, 1)$ where

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

s is always a power of 2!





~~$O(n^3)$ time~~

$O(n \log n)$ time

CS/ECE 374 A ✧ Fall 2025 Midterm 2 Practice 3 Problem 2	Name:

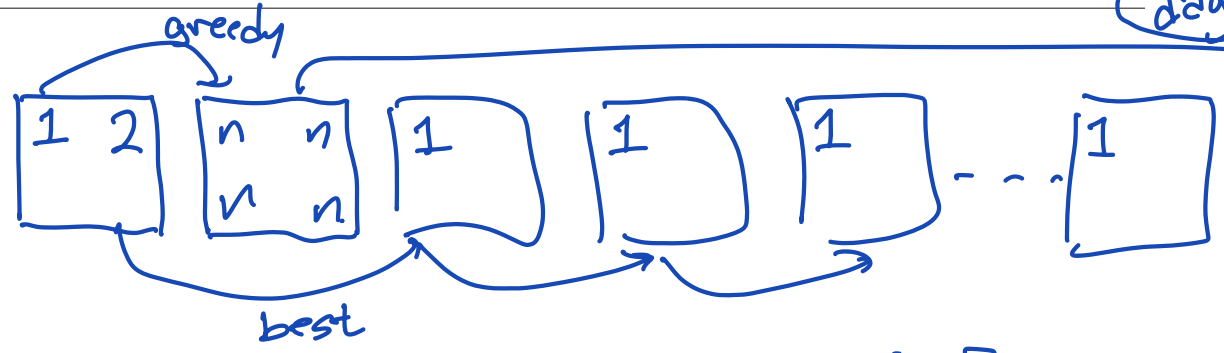
Quadhopper is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to 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.

DP? dag?

(a)



Greedy: jump to 2nd box. — done in 2 moves

Better: First jump 2, then 1,1,1.. → n−2 moves

n − 1 > 2 □

(b) Input: $Q[1..n][1..4]$

Build dag $G=(V,E)$

dag because $i \to j \Rightarrow i < j$

graph

$V = \{1..n, \infty\}$

$E = \{ i \to (i + Q[i,j]) \mid 1 \le i \le n,\ 1 \le j \le 4,\ i + Q[i,j] \le n \}$ ← not last moves

$\cup \{ i \to \infty \mid 1 \le i \le n,\ i + Q[i,j] > n \text{ for some } j \}$ ← Final moves

Problem: $n+1$ verts, $\le 4n$ edges

we need longest path in G from 1 to ∞

algo: Use dag longest path algo from class $O(V+E) \neq O(n)$ time

time

CS/ECE 374 A ✧ Fall 2025 Midterm 2 Practice 3 Problem 3	Name:

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.

This was right!

shortest paths?

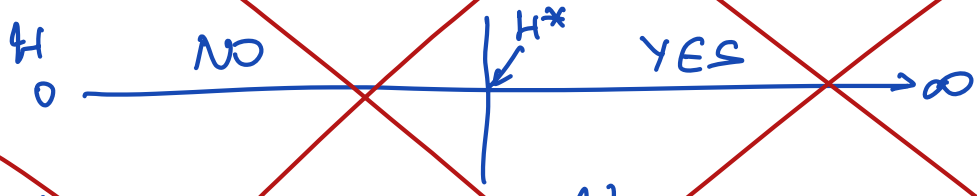
This is where I messed up!

Ignore this page! See page 7!!

Decision: Is there path with max height $\leq H$ length at most $\leq L$

Solve by: delete vertices height $> H$
Dijkstra → distance from s to t
if dist $\leq L$ → YES
dist $> L$ → NO

$O(E \log V)$



To solve optimization problem:
binary search over possible H
$= \{h(v) \mid v \in V\}$
Sort array $H[1..V]$ of heights

$O(V \log V)$

lo ← 1
hi ← V
while hi − lo ≥ 5
  mid ← ⌊(lo + hi)/2⌋
  if Decision(H[mid]) is YES
    hi ← mid
  else
    lo ← mid

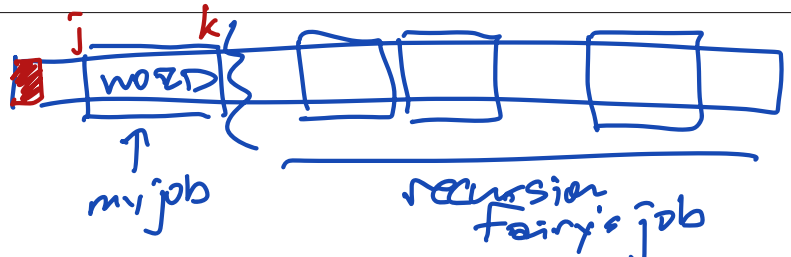
$O(E \log^2 V)$ time

Run Decision[H[i]] for all lo ≤ i ≤ hi
return smallest YES

CS/ECE 374 A ✧ Fall 2025 Midterm 2 Practice 3 Problem 4	Name:

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.

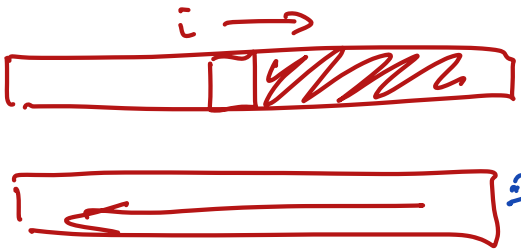
DP



$LVS(i)$ = length of longest verbal subsequence of suffix $T[i..n]$

we need $LVS(1)$

$$LVS(i) = \begin{cases} 0 & \text{if } i > n \\ \max\left\{ \begin{array}{l} LVS(i+1) \\ \max\left\{ \begin{array}{l} [\text{IsWord}(T[i..j])] \\ +LVS(j+1) \end{array} \middle| \; i \le j \le n \right\} \end{array} \right. & \text{O/w} \end{cases}$$



For i ← n down to 1
  For j ← i to n
    IsWord

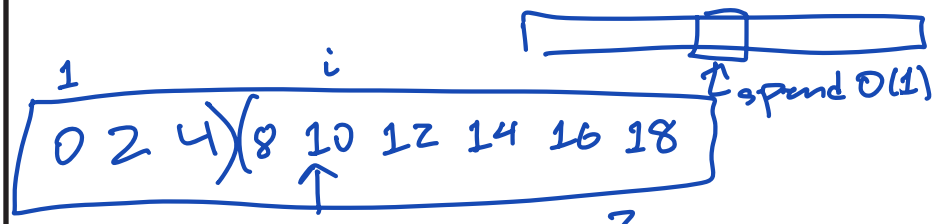
$O(n^3)$ time

DONE

CS/ECE 374 A ✧ Fall 2025 Midterm 2 Practice 3 Problem 5	Name:

Suppose we are given a sorted array $X[1..n]$ that contains an arithmetic sequence ***with one element deleted***. Describe and analyze an algorithm to find the deleted element as quickly as possible. If there are multiple correct answers, your algorithm can return any one of them. To avoid annoying boundary cases, you can assume $n \geq 4$.

div + conq?



spend O(1) time, discard half of input array

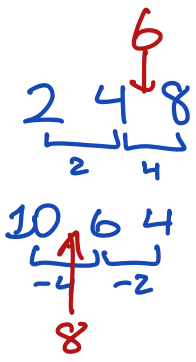
1 1 1 1 1

- Figure out step size?
  Look at X[2]-X[1] and X[3]-X[2]

Special case: if X[1] = X[2] return X[1]

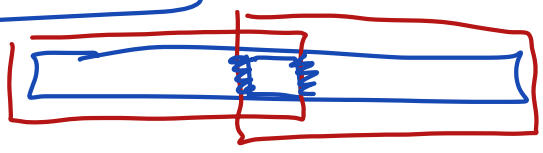
assume X[1] ≠ X[2]

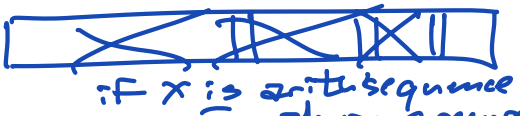
if X[2]-X[1] = X[3]-X[2]
  Δ ← X[2]-X[1]
else if (X[2]-X[1]) = ½(X[3]-X[2])
  ~~Δ ← X[2]-1~~
  return (X[3]-X[2])/2
else
  return (X[2]+X[1])/2



Main binary search:

lo ← 1
hi ← n
while hi - lo > 2
  mid ← ⌊(lo+hi)/2⌋
  if X[mid] = (mid-1)·Δ + X[1]
    lo ← mid
  else
    hi ← mid
compare X[lo] X[lo+1] X[lo+2]





O(log n) time

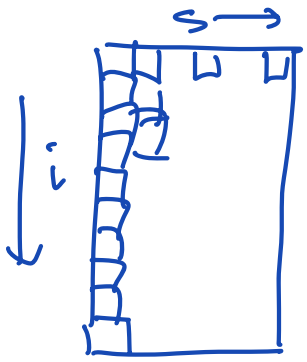
check if all arith seq
return X[1]+Δ
otherwise find gap
by brute force
return (X[lo]+X[lo+1])/2

See page 6 for rest

(scratch paper)

$$Bar(i,s) = \begin{cases} \\ \max\{Bar(i,2s), \\ \quad Bar(i-s,s)\} \end{cases}$$

we want $Bar(n,1)$

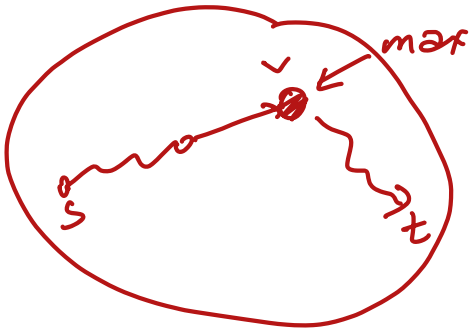


$\Theta(n \log n)$ entries

#3

(scratch paper)

Max height reachable by path $\leq L$



If there is a path $s \to v \to t$ of length $\leq L$

then

shortest(s,v) + shortest(v,t) $\leq L$

1. Run Dijkstra: dist(s,v) for all v — $O(E \log V)$
   als
2. Run Dijkstra ~~on reversed graph~~ — $O(E \log V)$
   at : dist(v,t) = dist(t,v) for all v
3. maxh $\leftarrow -\infty$
   For all $v \in V$
   if dist(s,v) + dist(v,t) $\leq L$ — $O(V)$
   and h(v) > maxh
   maxh $\leftarrow$ h(v)
   return maxh

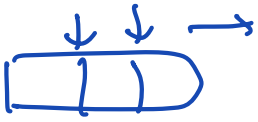
$O(E \log V)$ time

(scratch paper)

At end of algo, left with small base case.

$lo < hi \leq lo+2$


```
if hi = lo+1
    if x[lo] + Δ = x[hi]
        x is arith prog so
        return x[n] + Δ
    else
        return (x[lo] + x[hi])/2
if hi = lo+2
    if x[lo] + Δ ≠ x[lo+1]
        return (x[lo] + x[lo+1])/2
    else if x[lo+1] + Δ ≠ x[lo+2]
        return (x[lo+1] + x[lo+2])/2
    else // arith prog.
        return x[n] + Δ
```



(scratch paper)

(scratch paper)