CSCI 3104 Quiz 5

Jonathan Phouminh

TOTAL POINTS

8 / 16

QUESTION 1

1 1.5 / 2

+ **2 pts** Correct

✓ **+ 1 pts** **Provided only two LCS**

✓ **+ 0.5 pts** **Provided length**

+ **0 pts** Incorrect

+ **0.5 pts** Provided only one LCS

+ **1.5 pts** Listed LCS, but not length

- **0.5 pts** You provided a sequence of length 3. The length of a LCS is 2.

These are ordered sequences. Note that sets are unordered. You should use parentheses to denote a sequence, not curly braces (which denote a set). For example, use (0, 1) to denote a sequence and not {0, 1}.

You are also missing (0, 2).

QUESTION 2

5 pts

2.1 1.5 / 2

+ **2 pts** Correct

✓ **+ 1.5 pts** **Don't forget to divide by 2**

+ **0 pts** Provide an algorithm

+ **0.5 pts** This is not a bottom up algorithm. See solution.

+ **0 pts** No or incorrect solution

+ **1 pts** You need to calculate and store all 10 grades. Your solution overwrites previous grades.

+ **1.5 pts** Need to handle the base cases

+ **1.5 pts** Minor discrepancies compared to bottom up solution

+ **1 pts** Don't need to loop and recurse

+ **0.5 pts** Significant errors

+ **1.5 pts** Need to append values to the array as you calculate them.

2.2 0.5 / 1

+ **1 pts** Correct

✓ **+ 0.5 pts** **Need to divide by 2**

+ **0 pts** No answer

+ **0 pts** Give explicit formulas

+ **0.5 pts** Gave correct formula for either G[4] or G[5], but not both

+ **0.5 pts** Give explicit formulas for G[4] and G[5].

+ **0.5 pts** Order of operations issue

2.3 2 / 2

✓ **+ 2 pts** **Correct**

+ **0.5 pts** Top-down only

+ **0.5 pts** Memoizes only

+ **0.5 pts** Handles the base cases G[0] and G[1] correctly.

+ **0 pts** No or incorrect solution

- **0.5 pts** Minor errors

+ **1.5 pts** Top down and memoizes, but does not take advantage of the memoization

+ **1.5 pts** Need to handle the base cases. Did you memoize those first?

+ **1 pts** Need to recurse if element does not exist in the lookup table

+ **0.5 pts** Some description, but lacks significant detail and is very un-clear

+ **1.5 pts** Incorrect formula in recursion

QUESTION 3

5 pts

3.1 0 / 3

- **0 pts** Correct.

- **1.5 pts** The code works towards the problem but can not work properly.

✓ - **3 pts** **Not correct.**

- **3 pts** No answer.

- **0 pts** Your solution cannot work for other cases.

3.2 2 / 2

✓ - **0 pts** **Correct**

- **2 pts** wrong

- **2 pts** No answer

QUESTION 4

4 0.5 / 4

+ **1.5 pts** Correctly notes true if T[k-1, sum]

+ **1.5 pts** Correctly notes true if T[k-1, sum-s[k]]

✓ + **0.5 pts** **Correct Base Case of T[0,0] = True (or T[k >= 0, 0] = True)**

+ **0.5 pts** Correct Base Case of T[0, sum >0] = False (this base case also follows from explicitly stating that out of bound rows are False)

+ **0 pts** Incorrect - see solution

Name: Jonathan Pharmah

ID: 106005461(1

CSCI 3104, Algorithms
Quiz 5 : 16 points total

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

Instructions: This quiz is open book and open note, but an individual effort. Electronic devices are not allowed on your person (including in your pocket). Possession of such electronics is grounds to receive a 0 on this quiz. Proofs should be written in **complete sentences. Show all work to receive full credit.**

Please provide these:
Left neighbor name : ZASD
Right neighbor name :

We provide the Master Theorem for your reference.

Master Theorem: Suppose $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$.

(a) If there exists $c < \log_b(a)$ such that $f(n) \in \Theta(n^c)$, then $T(n) \in \Theta(n^{\log_b(a)})$.

(b) If $f(n) \in \Theta(n^{\log_b(a)})$, then $T(n) \in \Theta(n^{\log_b(a)} \log(n))$.

(c) If $f(n) \in \Theta(n^c)$, where $c > \log_b(a)$, then $T(n) \in \Theta(f(n))$.

Name:

ID:

CSCI 3104, Algorithms
Quiz 5 : 16 points total

Profs. Hoenigman & Agrawal
Fall 2019, CU-Boulder

Answer the following questions. Justify each of your answers with a ***short*** *explanation. You won't receive any credit on the solutions that lack any explanation. You don't have to formally prove it.*

1. (2 pts) Let $\omega = (0, 1, 2)$ and $\tau = (1, 0, 2, 1, 3)$. List all longest common subsequences of ω and τ and determine the length of the longest common subsequence.
(You can solve this manually without any algorithm)

   Solution.

   (0,1,2)

   subsequence$_1$ = {1, 2}

   subsequence$_2$ = {0, 1}

   length of longest common subsequence = 2

2. (5 pts) Professor X has noticed that the class average on assignment grades improved as the semester progressed, and he developed the following recurrence to describe how students grades on an assignment were related to their previous two assignments:

$$G[i] = \frac{1.05 * G[i-1] + 1.10 * G[i-2]}{2}, 2 \leq i \leq 10$$

$$G[0] = 50$$

$$G[1] = 55$$

Name: _____

ID: _____

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Quiz 5 : 16 points total **Fall 2019, CU-Boulder**

(a) (2 pts) Write a bottom-up algorithm to calculate the expected grades for the first 10 assignments.

Solution.

G[i] <=> memo[i]

```
def expected_grades():
    memo=[]
    memo.append(50)
    memo.append(55)
    for i in range(2,10):        #Assuming range is inclusive with 10
        memo.append(1.05 * memo[i-1] + 1.10 * memo[i-2])

#at this point the first 10 assignments
will be calculated in memo array
```

(b) (1 pts) Show the recursive equations for the values of $G[i]$, for $i = 4, i = 5$. You don't need to solve the equations.

Solution.

$i=4$, $G(4) = 1.05 * G(3) + 1.10 * G(2)$

$i=5$, $G(5) = 1.05 * G(4) + 1.10 * G(3)$

(c) (2 pts) Write a top-down algorithm with memoization to calculate the expected grades for the first 10 assignments.

Solution.

```
#Recursive solution#

# going to assume memo is passed through main
def expected_grades(n,memo):
    if memo[n] != None:
        return memo[n]
    if n=0:
        memo[n] = 50
        return memo[n]
    if n=1:
        memo[n] = 55
        return memo[n]
    else:
        memo[n] = (1.05 * expected_grades(n-1)+1.10*expected_grades(n-2)

    return memo[n]
```

Name: _____

ID: _____

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Quiz 5 : 16 points total **Fall 2019, CU-Boulder**

3. (5 pts) In a previous assignment, you developed a dynamic programming algorithm to maximize points on assignments, given that no two consecutive assignments could be completed. For example, in the following list of assignments, the maximum point value is 12.

   Input: [2,7,9,3,1]
   Output: 12
   Explanation: Maximum points available = 2 + 9 + 1 = 12.

   (a) (3 pts) Assume you have the following list of assignments and dynamic programming table that shows the optimal point totals for each of the 1...j subproblems.
   Input: [7,1,4,10,2,5]

Assignment Index	0	1	2	3	4	5
Max Subset Value	7	7	11	17	17	22

   Write an algorithm, either recursive or iterative, to extract which assignments to complete.
   Solution.

```
# assumes never non-empty
                                                    array
def extract_assignments(assignment_array, optimal_array, memo, index)
    if index = 1 or index = 0
        memo.append(max(assignment_array[0], [1])
        return
    else
        memo.append(max(optimal_array[index-1], assignment[index+1]))
    return memo
```

(More space for Q3.a)

(b) (2 pts) What are the values of the assignments in the final solution?
Solution.

Assignments: {7, 10, 5}

Name: _____

ID: _____

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Quiz 5 : 16 points total **Fall 2019, CU-Boulder**

4. (4 pts) Suppose we are given a set of non-negative distinct numbers s and a target t, and we want to find if there exists a subset of s that sums up to **exactly** t. [**Note:** This is different than Knapsack, where the elements in our subset can add up to be less than t.]
For example -

Input: s = {2,1,5,7}, t = 4
Output: False
Explanation: No subset sums to 4.

Input: s = {2,1,5,7}, t = 6
Output: True
Explanation: Subset {1, 5} sums up to the target $t = 6$.

Any sub-problem can be represented by `k` and `sum`, where, like Knapsack, `k` represents the first `k` numbers of the given set `s`, and `sum` represents any target value from $0 \leq$ `sum` $\leq$ `t`.

Write the recurrence relation (or the recursive solution) for an instance of (`k`, `sum`) in terms of smaller relevant sub-problems. Do not forget to complete your recurrence relation with the base case.
You do not have to solve this for any particular example. (There is a detailed example worked out on next page that shows the dynamic programming table that would be developed in a solution to this problem.)

F(i,w) = { if

Table[i,w] = { elseif k=0 and sum = 0 : Table[i,w] = TRUE
if sum[i] != k : Table[i,w] = Table[i,w-1]
if sum[i] = k : Table[i,w] = TRUE

Here are two worked out examples (corresponding to the above examples) with the DP table included were `T` corresponds to `True` and `F` corresponds to `False` which is filled using the recursive relation/solution.
The top example corresponds to `s = {2, 1, 5, 7}, and t = 4 (answer is False)` and the bottom example corresponds to `s = {2, 1, 5, 7}, and t = 6 (answer is True)`. If you look at the highlighted cell in the top example, it corresponds to the answer with the 1^{st} *three* elements of `s` and a target of `sum = 3` and the answer is `True`

s	k \ sum	0	1	2	3	4
	0	T	F	F	F	F
2	1	T	F	T	F	F
1	2	T	T	T	T	F
5	3	T	T	T	T	F
7	4	T	T	T	T	F

If k=0, False
If k=5, True

s	k \ sum	0	1	2	3	4	5	6
	0	T	F	F	F	F	F	F
2	1	T	F	T	F	F	F	F
1	2	T	T	T	T	F	F	F
5	3	T	T	T	T	F	T	T
7	4	T	T	T	T	F	T	T

Solution.

If k=0
Table[i,w] = True

If k != 0
Table[i,w] = (sum[i] == k)

Bool(i,j) = { True, False,