

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to [Latex](#).
- In this homework we denote the asymptotic *Big-O* notation by \mathcal{O} and *Small-O* notation is represented as o .
- We recommend using online Latex editor [Overleaf](#). Download the `.tex` file from Canvas and upload it on overleaf to edit.
- You should submit your work through [Gradescope](#) only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the `identikey@colorado.edu` version.
- Gradescope will only accept `.pdf` files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

Piazza threads for hints and further discussion

Piazza Threads

- [Question 1a](#)
- [Question 1b](#)
- [Question 1c](#)
- [Question 1d](#)
- [Question 1e](#)
- [Question 2](#)
- [Question 3](#)

Recommended reading:

Dynamic Programming: Chapter 15 complete

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

1. (65 pts) The sequence L_n of Lucas numbers is defined by the recurrence relation

$$L_n = L_{n-1} + L_{n-2} \quad (1)$$

with seed values $L_0 = 2$ and $L_1 = 1$.

- (a) (14 pts) Consider the recursive top-down implementation of the recurrence (1) for calculating the n -th Lucas number L_n .
 - i. (8 pts) Write down an algorithm for the recursive top-down implementation in pseudocode.

The top down with memoization approach modifies and saves the result of each subproblem (usually in an array or hash table). The procedure now first checks to see whether it has previously solved this subproblem.

prints 2, when n = 0 for the first Lucas number.

```
def f(n):
    if n==0:
        return 2
    if n==1:
        return 1
    else:
        return f(n-1) + f(n-2)
print(f(0))
```

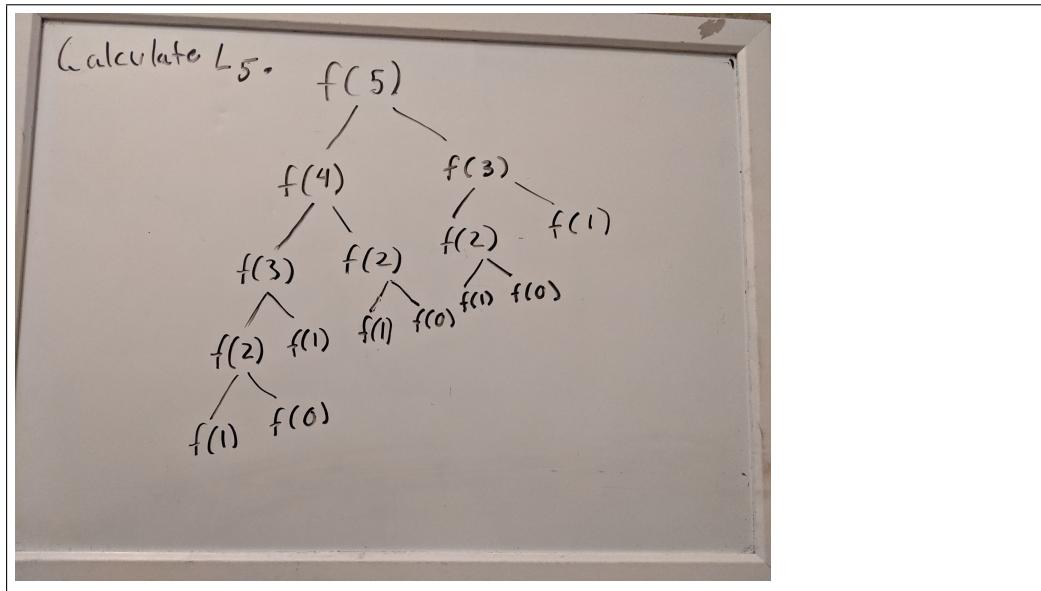
Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- ii. (2 pts) Draw the tree of function calls to calculate L_5 . You can call your function f in this diagram.



- iii. (4 pts) Write down the recurrence relation along with the base case for the running time $T(n)$ of the algorithm.

$$T(n) = T(n - 1) + T(n - 2) + \mathcal{O}(1)$$

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (b) (18 pts) Consider the dynamic programming approach “top-down implementation with memoization” that memoizes the intermediate Lucas numbers by storing them in an array $L[n]$.
- i. (10 pts) Write down an algorithm for the top-down implementation with memoization in pseudocode.

```
memoization =  
def f(n):  
    if n in memoization:  
        return memoization[n]  
    elif n == 0:  
        return 2  
    elif n==1:  
        return 1  
    else:  
        f = f(n-1) + f(n-2)  
        memoization[n] = f  
    return f
```

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- ii. (2 pts) Draw the tree of function calls to calculate L_5 . You can call your function f in this diagram.

- iii. (2 pts) In order to find the value of L_5 , you would fill the array L in a certain order. Provide the order in which you will fill L showing the values.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- iv. (4 pts) Determine and justify briefly the asymptotic running time $T(n)$ of the algorithm.

- (c) (16 pts) Consider the dynamic programming approach “iterative bottom-up implementation” that builds up directly to the final solution by filling the L array in order.

- i. (10 pts) Write down an algorithm for the iterative bottom-up implementation in pseudocode.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- ii. (2 pts) In order to find the value of L_5 , you would fill the array L in a certain order using this approach. Provide the order in which you will fill L showing the values.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- iii. (4 pts) Determine and justify briefly the time and space usage of the algorithm.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (d) (7 pts) If you only want to calculate L_n , you can have an iterative bottom-up implementation with $\Theta(1)$ space usage. Write down an iterative algorithm with $\Theta(1)$ space usage in pseudocode for calculating L_n . There is no requirement for the runtime complexity of your algorithm. Justify your algorithm does have $\Theta(1)$ space usage.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (e) (10 pts) In a table, list each of the four algorithms(as part of (a), (b), (c), (d)) as rows and in separate columns, provide each algorithm's asymptotic time and space requirements. Briefly discuss how these different approaches compare, and where the improvements come from.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

2. (10 pts) Consider the following DP table for the Knapsack problem for the list $A = [(4, 9), (1, 6), (3, 3), (5, 12), (6, 9)]$ of (weight, value) pairs.

The weight threshold $W = 10$.

- Fill in the values of the table.
 - Draw the backward path consisting of backward edges and do not draw (or erase them) the edges that are not part of the optimal backward paths.
- (a) (6 pts) Fill the table with the above requirements (You can also re-create this table in excel/sheet or on a piece of paper and add picture of the same).

Weight	Value	items_considered	0	1	2	3	4	5	6	7	8	9	10
-	-	no items											
4	9	A[0..0]											
1	6	A[0..1]											
3	3	A[0..2]											
5	12	A[0..3]											
6	9	A[0..4]											

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder



- (b) (2 pts) Which cell has the optimal value and what is the optimal value for the given problem?

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (c) (2 pts) List out the optimal subset and provide it's weight and value.

Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

3. (25 pts) Given an array of n size, the task is to find the longest subsequence such that the **absolute difference** between two adjacent values in the sequence is odd and less than or equal to 5. i.e the **absolute difference** between the adjacent elements is one of the values from the set $\{1, 3, 5\}$

For the definition of subsequence click [here](#)

Example 1:

Input: $\{10, 30, 5, 8, 27, 1, 4, 9, 14, 17\}$

output: 6

Explanation: Here the longest sequence satisfying the above condition will be $\{10, 5, 8, 9, 14, 17\}$ having a size of 6

Example 2:

Input: $\{10, 30, 6, 9, 27, 22, 20, 19\}$

output: 4

Explanation: There are several sequences of length 4 one such sequence is $\{30, 27, 22, 19\}$ having a size of 4

- (a) (5 pts) State the base case and recursive relation that can be used to solve the above problem using dynamic programming.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (b) (10 pts) Write down well commented pseudo-code or paste real code to solve the above problem.

Name:

ID:

CSCI 3104, Algorithms
Homework 4 (100 pts)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

- (c) (5 pts) Discuss the space and runtime complexity of the code, providing necessary justification.

- (d) (5 pts) Show how you can modify your pseudo-code or real code to return an optimal subsequence (if the problem has multiple optimal subsequences as part of it's solution, it is sufficient to return any one of those).

4. **Extra Credit (5% of total homework grade)** For this extra credit question, please refer the leetcode link provided below or click [here](#). Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.

Please provide your solution with proper comments which carries points as well.

<https://leetcode.com/problems/regular-expression-matching/>

Replace this text with your source code inside of the .tex document