

Instructions: Please note that handwritten assignments **will not be graded**. Use the provided L^AT_EX template to complete your homework. Please do not alter the order or spacing of questions (keep each question on its own page). When you submit to Gradescope, you must mark which page(s) correspond to each question. **You may not receive credit for unmarked questions.**

When including graphical figures, we encourage the use of tools such as [graphviz](#) or packages like [tikz](#) for simple and complex figures. However, these may be handwritten only if they are neat and legible (as defined by the grader).

List any collaborators (besides TAs or professors) here:

1. (5 points) [W9, ★] For the following questions, select whether the statement is true or false. **No explanation is necessary for these problems.**
 - (a) Dynamic programming is a technique to optimize certain types of recursive algorithms.
☐ True ☐ False
 - (b) Dynamic programming is a technique which increases time complexity in order to decrease space complexity.
☐ True ☐ False
 - (c) Dynamic programming can be applied to problems where there are multiple ways to arrive at the same subproblem instance via many branches of a recursion tree.
☐ True ☐ False
 - (d) When computing values in a dynamic programming table, we always start in the upper-left corner of the table.
☐ True ☐ False

2. (10 points) [W9, ★★] DP Terminology. Consider the Fibonacci problem that was discussed in class. Match the following statements with the following terms.

- | | |
|-------------------------------|---------------------|
| (A) Subproblem Instance | (F) Top-Down DP |
| (B) Base Instance / Base Case | (G) Bottom-Up DP |
| (C) Root Instance | (H) Recursion Tree |
| (D) Recurrence Relation | (I) Computation DAG |
| (E) Recursive Algorithm | |

- (a) ANSWER HERE — The following function:

```

DP = {}
def F(n):
    if n in DP: return DP[n]
    if n == 0: return 0
    if n == 1: return 1
    DP[n] = F(n-1) + F(n-2)
    return DP[n]

```

- (b) ANSWER HERE — Fibonacci of n

- (c) ANSWER HERE — $F(n) = F(n-1) + F(n-2)$

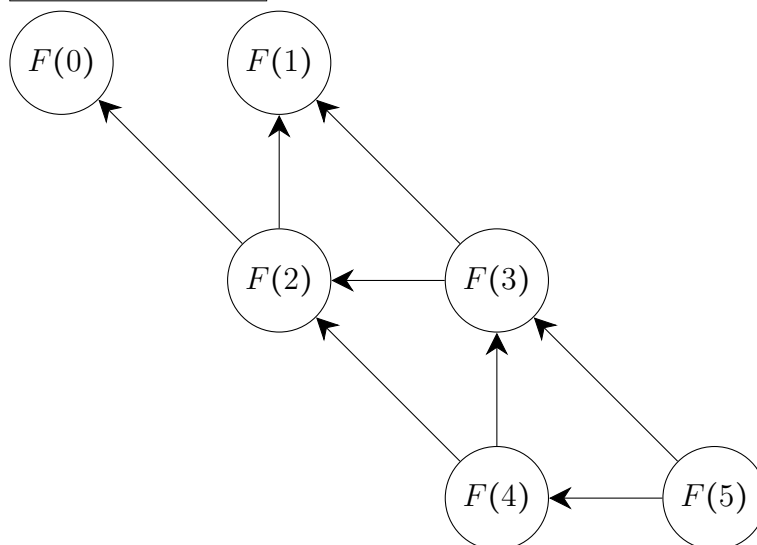
- (d) ANSWER HERE — The following function:

```

def F(n):
    if n == 0: return 0
    if n == 1: return 1
    return F(n-1) + F(n-2)

```

- (e) ANSWER HERE — The following graph:



3. (5 points) [W9, ★★★] Binomial Coefficients. Select the true statement. **No explanation is necessary for this question.**

In terms of n and k , how many calls to the recursive algorithm are required to evaluate $\binom{n}{k}$?

- ☐ k^n
- ☐ n^k
- ☐ $\binom{n}{k}$
- ☐ $2\binom{n}{k} - 1$
- ☐ $(n + k)k$

4. (40 points) [W9, ★★★★★] Stirling Numbers. This question is related to Wednesday's worksheet.

Recall the definition of *Stirling numbers of the second kind*:

Base Cases

$$S_{n,0} = 0 \quad \forall n > 0 \quad (1)$$

$$S_{n,k} = 1 \quad n = k \geq 0 \quad (2)$$

$$S_{n,1} = 1 \quad \forall n > 1 \quad (3)$$

$$S_{n,k} = 0 \quad n < k \quad (4)$$

General Case

$$S_{n,k} = k \times S_{n-1,k} + S_{n-1,k-1} \quad (5)$$

- (a) (15 points) Fill in the following **table** (starting with the base-cases).

n/k	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

- (b) (10 points) Write an unoptimized **recursive algorithm** (no loops) to solve this problem in the programming language of your choice. Assume all necessary libraries/headers are present (only write the function). **Present your code using the minted \LaTeX library. You shouldn't need more than 10 lines.**

YOUR_CODE_HERE

- (c) (5 points) Write a “**top-down**” **DP solution** that augments the recursive algorithm above with memoization. Store intermediate results in a dictionary/array and check to see if a value has already been determined before computing it again. If the value has already been computed, then return it instead of making the recursive call(s).

YOUR_CODE_HERE

- (d) (10 points) Write a “**bottom-up**”, **iterative DP solution** that uses a **for** loop and an array to compute the solution starting from the base cases. Assume n and k have been previously provided to your program.

YOUR_CODE_HERE

5. (40 points) [W9, ★★★★★] Please present your code using the `minted` L^AT_EX library.

You want to construct a rod of integer length n using smaller rods. The smaller rods are available in three lengths: 1, 2, and 3 ft respectively. There are an unlimited number of these smaller rods available. Your plan is to glue several of the smaller rods end-to-end to create a longer rod.

1	1	1	1
1	1	2	
2		1	1
1	2		1
2		2	
1	3		
3			1

Table 1: Each of the 7 rows above gives us a different way to construct a 4 ft rod.

Question: In how many ways can you glue the three types of smaller rods together to create a rod of length $n > 0$? The table above enumerates all of the possibilities when $n = 4$ and giving an answer of 7.

- (5 points) What is the input to the **subproblem instance**?
- (5 points) Which subproblem instances are the **base cases**? What are their values?
- (5 points) Develop notation and describe a **recurrence relation** between the subproblem instances.
- (10 points) Write an unoptimized **recursive algorithm** (no loops) to solve this problem in the programming language of your choice. Assume all necessary libraries/headers are present (only write the function). **Present your code using the `minted` L^AT_EX library. You shouldn't need more than 10 lines.**

YOUR_CODE_HERE

- (5 points) Draw the first three levels (the root is the first level) of the **recursion tree** for $n = 12$ and highlight the repeated computations, that is, identical recursive calls made more than once or **will** be made more than once if the tree was extended to further levels.
- (10 points) Write a “**bottom-up**”, **iterative DP solution** that uses a **for** loop and an array to compute the solution starting from the base cases. Assume n has been previously provided to your program.

YOUR_CODE_HERE