Instructions: Please note that handwritten assignments **will not be graded**. Use the provided LaTeX 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
(B) Base Instance / Base Case
(C) Root Instance
(D) Recurrence Relation
(E) Recursive Algorithm
(F) Top-Down DP
(G) Bottom-Up DP
(H) Recursion Tree
(I) Computation DAG

(a) Top-Down DP — The following function:

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

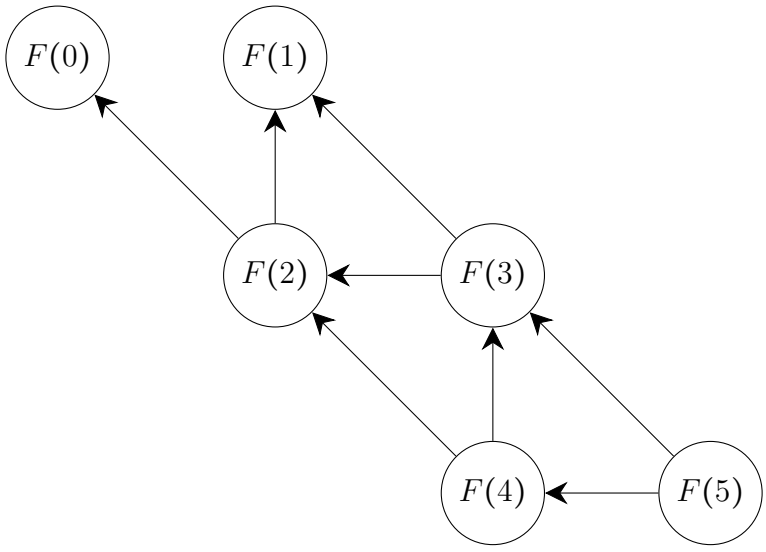
(b) Root Instance — Fibonacci of n

(c) Recurrence Relation — $F(n) = F(n-1) + F(n-2)$

(d) Recursive Algorithm — The following function:

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

(e) Computation DAG — 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

$$S_{n,0} = 0 \qquad \forall n > 0 \tag{1}$$
$$S_{n,k} = 1 \qquad n = k \geq 0 \tag{2}$$
$$S_{n,1} = 1 \qquad \forall n > 1 \tag{3}$$
$$S_{n,k} = 0 \qquad n < k \tag{4}$$

   General Case

$$S_{n,k} = k \times S_{n-1,k} + S_{n-1,k-1} \tag{5}$$

   (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	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0
3	0	1	3	1	0	0	0	0
4	0	1	7	6	1	0	0	0
5	0	1	15	25	10	1	0	0
6	0	1	31	90	65	15	1	0
7	0	1	63	301	350	140	21	1

   (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.**

```
def stirling(n, k):
    # Base case 1 & 4
    if (k == 0 and n > 0) or (n < k):
        return 0
    # Base case 2 & 3
    elif (n == k and k >= 0) or (k == 1 and n > 1):
        return 1
    # General Case
    else:
        return (k * stirling(n-1, k)) + stirling(n-1, k-1)
```

   (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).

```
def sterling_top_down(n, k):
    # Create a dictionary to store the results
    S = {}
    # Base case 1 & 4
    if (k == 0 and n > 0) or (n < k):
        return 0
    # Base case 2 & 3
    elif (n == k and k >= 0) or (k == 1 and n > 1):
        return 1
    # General Case
    else:
        if (n, k) not in S:
            S[(n, k)] = (k * sterling_top_down(n-1, k)) + \
                sterling_top_down(n-1, k-1)
        return S[(n, k)]
```

(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.

```
def stirling_bottom_up(n, k):
    # Create a 2D array to store the results
    S = [[0 for _ in range(k+1)] for _ in range(n+1)]

    for row in range(n+1):
        # Base cases
        if k >= 1 and row > 1:
            S[row][1] = 1
        if k >= row:
            S[row][row] = 1

        # General Case
        if row >= 3 and k >= 2:
            for col in range(2, min(row, k)+1):
                S[row][col] = (col * S[row-1][col]) + S[row-1][col-1]

    return S[n][k]
```

5. (40 points) [W9, ★★★★★] Please present your code using the `minted` LaTeX 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.

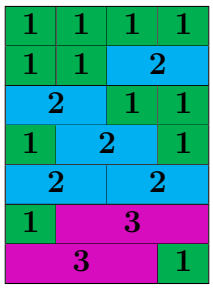


Table 1: Each of the 7 rows above gives us a different way to construct a 4 ft rod.

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.

(a) (5 points) What is the input to the **subproblem instance**?

The input to the subproblem instance is the length of the rod, n.

(b) (5 points) Which subproblem instances are the **base cases**? What are their values?

The base cases are when the length of the rod is 1, 2, or 3. Their values are 1, 2, and 4, respectively.

(c) (5 points) Develop notation and describe a **recurrence relation** between the subproblem instances.

Let $W(n)$ be the number of ways to construct a rod of length n. Then, the recurrence relation is:

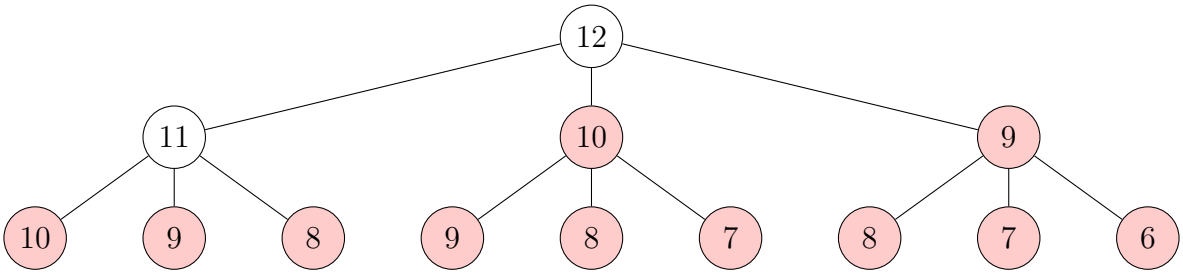
$$W(n) = W(n-1) + W(n-2) + W(n-3)$$

(d) (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.**

```
def rod(n):
    # Base cases
    if n == 1:
        return 1
    if n == 2:
        return 2
    if n == 3:
```

```
        return 4
    # General Case
    return rod(n-1) + rod(n-2) + rod(n-3)
```

(e) (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.



(f) (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.

```
def rod_bottom_up(n):
    # Create an array to store the results
    W = [0] * (max(n+1,4))
    # Base cases
    W[1] = 1
    W[2] = 2
    W[3] = 4
    # General Case
    for i in range(4, n+1):
        W[i] = W[i-1] + W[i-2] + W[i-3]
    return W[n]
```