Lab 5: Dynamic Programming with "AlgoWare Defender"

You are working for a new cybersecurity company “AlgoWare Defender” on a decoding tool. The decoding tool is meant to help reverse a nefarious ransomware attack of a one way encoding scheme of letters to digits.

The encoding scheme coverts letter strings into numeric strings by converting each letter to a digit based on its place in the alphabet.

Here is the encoding table:

A	1
B	2
...	...
Y	25
Z	26

For example, the string ’ZAB’ maps to 2612.

The nefarious part of this scheme is that it is not reversible! If you are given 2612, this could be decoded in multiple ways:

- 26-1-2 = ZAB
- 2-6-12 = BFL
- 2-6-1-2 = BFAB
- 26-12 = ZL

AlgoWare Defender’s decoding tool will have a few components. You will be working on the first piece, determining how many decodings are possible. For example, there are 4 decodings for 2612.

You must write a function: **findNumDecodings**, which when given a string of digits, returns the number of possible decodings.

You must solve this problem using dynamic programming.

Deliverable Explanation

You must efficiently solve this problem in either Python or Java, using one of the attached starter code files. Several unit tests are provided to help you test your code.

Having the optimal runtime will grant you extra credit. On the other hand, you may also lose points if your code is substantially inefficient. You must also provide an explanation of the running time of your code in the provided placeholder in the starter code.

Your code will be evaluated against a set of visible and hidden test cases (the visible ones are also provided in the starter code for your convenience) to test correctness. The hidden test cases are intended to verify that you have a robust solution that considers possible edge cases and various inputs.

If you are using Java, your file's package must be set to "student".

Submission Instructions

Submit the following file on Gradescope:
AlgoWareDefender.java to Lab5 (Java)
OR
algoware_defender.py to Lab5 (Python)

Grading Methodology

The lab is worth 100 points and is broken down into the following categories:

- Test Cases Passing (80 points)
- Big O explanation (15 points)
- Overall code performance and style (5 points)
- Having Optimal Running Time (5 points of Extra Credit)