Question 1 and 2 are worth 30% each, and Question 3 is worth 40%. Due Monday, May 4, 6pm.

Question 1

Given a rod of integer length n and an array of prices $P[1, \dots, n]$, where $P[i]$ is the price of selling a piece of rod of length i, the following procedure returns the maximum revenue possible from cutting and selling the rod.

```
cut-rod(P, n)
    if n = 0 then
        return 0
    q ← −∞
    for i = 1 to n do
        q ← max(q, p[i] + cut-rod(P, n − i))
    return q
```

1.1 What is the time complexity of this procedure?

1.2 Modify this procedure to use memoization, producing a `memoized-cut-rod`(P, n) procedure. Answer in pseudo-code.

1.3 What is the time complexity of `memoized-cut-rod`(P, n)?

1.4 Prove formally that a call to `memoized-cut-rod`(P, n) returns the same result as `cut-rod`(P, n) for any array P of length $n \in \mathbb{N}$.

Question 2

You are given a non-empty sequence of coloured balls $S[1, \ldots, n]$, with their colours encoded by capitalized strings. We call such a sequence of balls *symmetric* if the sequence of colours read from the front are mirrored from the back. For example, $[R, G, B, G, R]$ is a symmetric sequence. Of course, any sequence consisting of one ball is symmetric. We can also see that a sequence can be constructed of many smaller symmetric sequences,

$$[R, B, R, G, O, O, G, T, I, I, T] = [R, B, R] + [G, O, O, G] + [T, I, I, T]$$

The example sequence above is constructed from 3 symmetric sequences. The aim is to design an algorithm `min-sim`(S) to compute the minimum number of symmetric sequences that make up a given sequence (i.e. we want to compute k so that S can be written as $s_1 s_2 \ldots s_k$ with each s_i being a symmetric sequence).

2.1 Compute k for the following sequences by filling the table.

Sequence	k
$[R, B, R, G, O, O, G, T, I, I, T]$	3
$[R, G, B, G, R]$	1
$[C, C, C, C, A, X, B, X]$	
$[Z, X, Y, Z, X]$	
$[A, BD, C, BD, A]$	
$[A, O, A, B, X, Z, Y, X]$	
$[A, B, B, A, C, A, B]$	

2.2 We want to build a dynamic programming algorithm to compute k given a non-empty sequence. Define the subproblems, the base cases, and the recurrence relation between the subproblems.

2.3 What would be the worst-case time complexity of an algorithm based on your approach? Justify your answer.

Question 3

The objective of this exercise is to create an efficient algorithm finding the number of solutions to a linear equation, involving only natural numbers. When only integer solutions are sought, such equations are called *Linear Diophantine equations*.

For instance, consider the equation $x_0 + 3x_1 + 5x_2 + 7x_3 = 8$. It has exactly 6 solutions in natural numbers which are as follows:

x_0	x_1	x_2	x_3
1	0	0	1
0	1	1	0
2	2	0	0
3	0	1	0
5	1	0	0
8	0	0	0

Indeed, one can check, for instance line 2, that $1 \times 0 + 3 \times 1 + 5 \times 1 + 7 \times 0 = 8$.

In this exercise, the equation is represented as pair (C, s) corresponding to an array of coefficients and the target sum (($C = [1,\ 3,\ 5,\ 7]$, $s = 8$) in the example above). We will assume that all coefficients in C are positive natural numbers and so is the sum s. We may also assume that the coefficients in C are listed in increasing order.

3.1 Consider the equation $x_0 + 2x_1 + 3x_2 = 4$ (e.g., ([1, 2, 3], 4)). List all the natural number solutions in the table below.

x_0	x_1	x_2

3.2 We first look for a naive and inefficient brute-force approach to counting the number of solutions. Describe such an algorithm in pseudo-code.

3.3 Let m be the smallest value in C, let n be the length of C, and let s be the input target number. Provide an upper asymptotic bound on the runtime of your brute-force algorithm in terms of m, n, and s. In other words, what is a big O complexity of your answer to the above question? The bound need not be tight.

3.4 Describe in English how one can build a Dynamic Programming algorithm finding the number of natural number solutions to a linear equation.