- See previous practice problem sets for instructions.

1. For $n \in \mathbb{N}$ the n^{th} Fibonacci number F_n is defined by: $F_0 = F_1 = 1, F_i = F_{i-1} + F_{i-2}\ ;\ i \geq 2$.

   (a) Translate the following pseudocode for computing F_n into Python 3 code, and *time* it[1] on inputs $n = 35, n = 40, n = 45, n = 50$.

   SimpleFib(n)

   ```
   if n ≤ 1
         return 1
   F1 ← SimpleFib(n − 1)
   F2 ← SimpleFib(n − 2)
   return F1 + F2
   ```

   (b) Write the pseudocode for a *memoized* version of SimpleFib. Translate this pseudocode into Python 3 code, and time it on inputs $n = 35, n = 40, n = 45, n = 50$.

   Do you notice a significant difference? If you would like to *really* understand what is going on here: Try adding print statements to inspect the arguments passed in to the recursive calls in your memoized version.

   (c) Derive a good asymptotic upper bound in terms of n, on the running time of your pseudocode from part (b).

2. For non-negative integers $n, k\ ;\ k \leq n$ the binomial coefficient $\binom{n}{k}$ represents the number of ways to choose k items from n items without regard to order. It satisfies the recurrence:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k},$$

   with base cases $\binom{n}{0} = \binom{n}{n} = 1$.

   (a) Write the pseudocode for a *recursive* function Choose(n, k) which implements the above logic to compute and return $\binom{n}{k}$.

   (b) Translate your pseudocode of part (a) into Python 3 code, and time it on inputs $(n, k) = (20, 10), (30, 15), (40, 20), (50, 25)$.

   (c) Write the pseudocode for a *memoized* version of the recursive function from part (a). Implement this memoized version in Python 3, and time it on the same inputs as above. What improvements do you observe?

   (d) Derive a good asymptotic upper bound in terms of n, on the running time of your pseudocode from part (c).

3. Consider the problem of finding the minimum *number* of coins required to make change for a given integer amount $n \geq 0$, using coins of positive integral denominations $d_1, d_2, \ldots, d_k$.

[1] That is: find the clock-time taken by the code.

(a) Derive a recurrence relation for this problem. That is: let MinCoins(n) denote the minimum *number* of coins that add up to exactly the value n, where each coin takes one of the values ("denominations") $d_1, d_2, \ldots, d_k$, and there can be zero or more coins with the same denomination. Make sure that you capture all the relevant base cases of your recurrence.

Explain why your recurrence correctly computes MinCoins(n).

(b) Write the pseudocode for a *recursive* function MinCoins(n) which implements the above logic to compute and return MinCoins(n).

(c) Translate your pseudocode of part (b) into Python 3 code, and time it on the following inputs:

- The denominations are $d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25$ for all the values.
- The values are: $n = 20, n = 50, n = 100, n = 200$
- (Thus there are four instances, one for each value of n.)

(d) Write the pseudocode for a *memoized* version of the recursive function from part (b). Implement this memoized version in Python 3, and time it on the same inputs as above. What improvements do you observe?

(e) Derive a good asymptotic upper bound in terms of n, on the running time of your pseudocode from part (d).