```
==============================================================================
CSC 373             Sample Solutions for Week  2 Tutorial          Winter 2019
==============================================================================
```

0. [No sample solution for simple tracing: get in the habit of doing this for yourself and ask questions on Piazza or during office hours if this brings up any issues for you.]

1. Algorithm:

```
    d = [1, 5, 10, 25]  # coin values (aka "denominations")
    k = 3  # start with largest denomination
    C = []  # list of coins used to make change
    while A > 0:
        while A < d[k]:
            # Try the next smaller denomination.
            # Because d[0] = 1, k cannot drop below 0.
            k = k - 1
        # Use one more coin of denomination d[k].
        C = C + [d[k]]
        A = A - d[k]
```

   Proof of correctness:
   Let C_0, C_1, ... be partial solutions generated by the algorithm. Say C_i is promising iff it can be extended to an optimum solution (by adding more coins to the sequence), using only denominations no larger than the smallest coin in C_i.

   NOTE: The exact definition of "extend" is important here, and different from the example in the lecture, because the problem (and algorithm) are different.

   Proof that C_i is promising for all i.
   Base Case:
       C_0 = [] is extended by every optimum solution.
   Ind. Hyp.:
       Suppose C_i is promising, with optimum solution OPT.
   Ind. Step:
       Consider C_{i+1} = C_i + d[k_{i+1}].
       We want to show that OPT - C_i must already contain one coin with value d[k_{i+1}], so that OPT already extends C_{i+1} -- in other words, there is a unique optimum solution.

   NOTE: This is DIFFERENT from what we did in lecture (where the proof contained cases because of the choice made by the algorithm at each step). The difference comes from the difference in the algorithm: the coin-changing algorithm above does not construct its solution in the same manner, so it does not lead to similar cases. It also comes from the difference in the problem: because there is only one possible optimum solution, the algorithm cannot make any choice differently so there are no sub-cases.

       Now, for a contradiction, suppose that OPT - C_i does NOT contain one coin with value d[k_{i+1}].
       Consider cases for d[k_{i+1}]:
         - d[k_{i+1}] = 25
           Since C_{i+1} = C_i + d[k_{i+1}], the amount remaining must exceed 25 (A_i >= 25). If OPT - C_i does not contain any coin with value 25, then it must make up A_i using only coins with

values 1, 5 and 10. But any combination of these coins that adds up to more than 25 can be replaced by a combination that contains 25 and uses fewer coins. (*)

- d[k_{i+1}] = 10
  Similar reasoning shows any combination of 1c and 5c coins that adds up to 10c or more must contain some sub-collection that adds to exactly 10c, which could be replaced by a single coin to get a solution better than OPT.
- d[k_{i+1}] = 5
  As above.
- d[k_{i+1}] = 1
  It is impossible to make up amounts of 1c or more *without* using any coin!

In every case, we get a contradiction. Hence, OPT already extends C_{i+1}.

When the algorithm stops, the final value of C is promising (by the proof above). But at the same time, C contains coins that add up to exactly A. In other words, C is optimum.

(*) Relies on the following fact: every optimum solution contains
- at most two 10c, one 5c, and four 1c pieces,
- not two 10c and one 5c pieces together.

This can be proved with a simple (but tedious) case-by-case analysis.

2. Consider the case d = [1, 10, 25] and A = 30.
   Then the greedy algorithm generates the solution [25, 1, 1, 1, 1, 1].
   But [10, 10, 10] is another solution that uses fewer coins.

*3. [No sample solution for "starred" problems!
   Post on Piazza for discussion, if you're interested.]