Name: Natalie Ammerman
Due September 16, 2024

COMP 550
Homework 2 (10 points)

Directions: You may use any resources, collaborate in any manner, call any algorithms from the course notes, and assume that they are correct. Every student should make an individual submission that exactly follows the `.tex` template provided on Canvas, and every submission will be individually graded.

Each homework submission should be a single PDF file generated from the `.tex` file using LaTeX. Embedded images (e.g., a hand-drawn graph with labeled vertices) are acceptable, but any substantial amounts of text (e.g., an algorithm, explanation, or analysis) must be typed out in LaTeX. Algorithms should be written in pseudocode at a level similar to that found in the course notes and Problem Sets.

Every homework problem is worth 5 points, and your score will always be an integer. For each problem, if your submission does not follow the `.tex` template, you will receive at most 1 point. For each problem, if you receive at least 2 points, your score for that problem will be rounded up to 5. (This only applies to homework, not quizzes or exams.)

Advice: Try to solve each problem without consulting any resources or people for at least 15 minutes. Many problems (on homework and exams) have solutions that are similar to, or inspired by, solutions in the lectures/notes. Encountering difficulties is normal; I encourage you to attend office hours if it happens.

A relatively simple way to write pseudocode is the following:

```
ALG(A):
  t = 0
  for i = 1, ..., n:
    t += A[i]
  return t
```

Problem 1. The input is (G, T, e) where G is a connected, undirected graph with distinct edge weights, T is the MST of G, and e is an edge in T. Suppose $w(e)$, the weight of e, increases. (Assume that all edge weights are still distinct.) Describe an $O(m)$-time algorithm that returns the MST of this new graph (i.e., G with one edge weight increased), explain why it's correct, and analyze its running time.

Solution.

```
Algorithm:
    update w(e) in G
    remove e from T
    split T into separate components
    iterate through each edge in G that would connect the components to see which
    e' has the least weight
    add e' to T
Correctness: The algorithm removes e because its weight could violate the MST, and T is
now in two separate parts. The lightest edge connecting the two parts of T e' is added
and T retains its MST properties. Via the Cut Property, the lighest edge crossing any
subset of vertices will always be in the MST. Given this, adding the lightest edge
guarentees that the MST is valid. The alg completes when the new edge is found and
returns an MST with n-1 edges, confirming that the MST is vaild.
Running Time: Removing and adding an edge would be constant time O(1). Iterating through
every edge (m edges in G) to see which would connect T with the least weight would take
O(m). So the running time of this would be O(m).
```

Problem 2. Suppose there are an unlimited number of dimes (10 cents each), nickels (5 cents), and pennies (1 cent). The input is a positive integer t, and we want to select the fewest number of coins such that their total value is t cents. The greedy algorithm is to pick as many dimes as possible (i.e., without exceeding t), then as many nickels as possible, then as many pennies as possible.

Describe an $O(1)$-time implementation of the greedy algorithm, explain why the greedy algorithm is correct, and analyze the running time of your implementation. The algorithm should return a tuple (a, b, c), where a is the number of dimes, b is the number of nickels, and c is the number of pennies chosen by the greedy algorithm.

Hint: For the correctness explanation, start by using an exchange argument to prove that ALG and OPT pick the same number of dimes. Bonus (0 points): Explain why your proof is not correct if the coins had values $(4, 3, 1)$ instead of $(10, 5, 1)$.

Solution.

```
Algorithm:
def coins(t):
    dimes = t // 10
    t = t - dimes * 10
    nickels = t // 5
    t = t - nickels * 5
    pennies = t
    return (dimes, nickels, pennies)
Correctness: Given input t cents, the algorithm first divides t by 10 to see how many
dimes are possible, then it divides the remaining cents by 5 to see how many nickels
are possible, then finally decides the remainder is how many pennies are needed. This
will be correct because it is looking to use the least amount of coins possible, and
by starting with the largest coin, the alg is minimizing number of smaller coins that
could be used to represent the same amount. ALG and OPT would pick the same number of
dimes given t = 55, that number being 5 dimes. If the coins had values (4, 3, 1), they
would not be multiples of each other, and therefore would not function the same way as
when they stack upon each other.
Running Time: The running time is O(1) because all arithmetic operations are O(1) and
there is no dependence on the input value t for run time.
```