Computer Laboratory 3
CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development
September 24–25, 2019

0. Introduction.

This assignment asks you to write a Python function that sorts a tuple of integers using the techniques of functional programming. This is the last Python lab for this course. We're not done with Python yet: there is still a Python project yet to be assigned, there will be questions about Python on the first midterm, and one question about Python on the final.

1. Theory.

We say that a sequence of integers is *sorted* if it has elements e_0, e_1 ..., e_{j-1} in that order, and also $e_0 \le e_1 \ldots \le e_{j-1}$. We can sort such a sequence S using the following algorithm.

1. If S has no elements, then it is already sorted, so return S.
2. Choose the largest element m from S. If m appears more than once in S, then choose any appearance of m.
3. Make a new sequence that is like S, except that the first appearance of m is removed.
4. Sort the new sequence by applying this algorithm recursively.
5. Return the result of adding m to the end of the sorted sequence.

For example, suppose that the algorithm is implemented by the Python function `Sort`, which takes a tuple of integers as its argument. We can trace the algorithm like this, where `+` is Python's concatenation operator, and ⇒ means *returns*.

```
Sort((1, 2, 3, 1)) ⇒ Sort((1, 2, 1)) + (3,)
                   ⇒ Sort((1, 1)) + (2,) + (3,)
                   ⇒ Sort((1,)) + (1,) + (2,) + (3,)
                   ⇒ Sort(()) + (1,) + (1,) + (2,) + (3,)
                   ⇒ () + (1,) + (1,) + (2,) + (3,)
                   ⇒ (1, 1, 2, 3)
```

The algorithm does not change the values of variables after they are assigned, and does not use mutable data structures. As a result, it can be implemented using the techniques of functional programming that were discussed in the lectures.

2. Implementation.

You must implement the sorting algorithm described in the previous section by defining the following Python functions. You must use the same names for the functions as are shown here, but you may use different parameter names if you like. Throughout, assume that `T` is a tuple of integers. The tuple may contain negative integers, zeroes, positive integers, or some mixture of these.

`Sort(T)`

Return a new tuple that is like `T`, except that its elements are sorted (see above). It works by calling `Maximum` and `Remove` somehow.

`Maximum(T)`

Assume that `T` has at least one element. Return the largest element from `T`, an integer. Hint: use a "helper" function to do most of the work.

`Remove(T, E)`

Return a new tuple that is like `T`, except that the first appearance of its element `E` is removed. If `E` does not appear in `T`, then `Remove` must return a tuple that is like `T`.

Some of these functions may be written easily using higher-order functions such as `lambda`, `filter`, `map`, and `reduce`. However, I won't tell you which ones can be written this way, or which higher-order functions you should use. You need not use higher-order functions if you don't want to.

All these functions *must* be written in a functional, recursive style, and your code must demonstrate that you know how to write in such a style. As a result, you will receive **ZERO POINTS** for this lab if your functions do any of the following things.

- Use one or more global variables.
- Use a loop, such as `for` or `while`.
- Assign a value to a variable more than once.
- Use mutable data structures, such as lists or dictionaries.

Also, you must not use operators or predefined functions that do things you are asked to write yourself—so you must not call a predefined `sort` function from the Python library. If you write additional "helper" functions, then they must not do these things either.

3. Tests.

The file **`tests3.py`** on Canvas contains a series of tests. Each test calls a function from your program, then prints what the function returns. Each test also has a comment that says what it should print, and how many points it is worth.

To grade your work, the TA's will run the tests using your functions. If a test prints exactly what it should, without error, then you will receive all the points for that test. However, if a test does anything else, then you will receive no points for that test. Your score for this lab is the sum of points you receive for all the tests.

4. Deliverables.

Run the tests, then submit the Python code for your functions along with the tests. The lab TA's will tell you how and where to turn them in. Your work must be submitted by **11:55 PM** on **Wednesday, October 2, 2019.**