CS 440 Fall 2018 Machine Problem Assignment 1 Due date: 09/23/2018 11:55pm

Guidelines – Read Carefully! Please check each problem for problem-specific instructions. You are provided a zip file containing a single folder named xyz007 with the following folders and files:

xyz007
xyz007/mp440.py
xyz007/main.py

You should first rename the folder name xyz007 to be your NETID. If you are working in a team, the folder should be named with both group members' NETIDs, in the format of NETID1_NETID2. The folder name letters can be either upper or lower case. When you are ready to submit, remove any extra files (e.g., some python interpreter will create .pyc files) that are not required and zip the entire folder. The zip file should also be named with your NETID as NETID.zip (or NETID1_NETID2.zip for groups with two members). For this MP, main.py may be removed at the time of submission.

You are required to write your program adhering to Python 2.7 standards. In particular, DO NOT use Python 3.x. Beside the default libraries supplied in the standard Python distribution, you may use ONLY numpy and matplotlib libraries.

You should only work on mp440.py and do not create additional python files. The file main.py is for you to test your implementations. Note that during grading, your implementation will be called in different ways than what are in main.py, which serve only as minimal sanity checks. You should not use any variable or functions from main.py in your code.

As mentioned in class, you may form groups of up to two students. Only a single student should submit the solution.

Problem 1 [25 points]. Implementation of a standard *greatest common divisor* algorithm. You should implement the function

```
gcd(a, b)
```

where you may assume that a and b are positive integers. You are to compute the greatest common divisor of a and b, and return it at the end of the function call.

Problem 2 [25 points]. Count the number of rectangles on a Rubik's cube. Beside the standard $3 \times 3 \times 3$ Rubik's cube, there are other cubes where each side is divided into n pieces ($n = 3$ for the standard Rubik's cube). Ignoring the colors, the small squares on each surface of a Rubik's cube may be combined to form larger rectangular shapes. For an $n \times n \times n$ Rubik's cube, compute the total number of different rectangles (including squares) that can be obtained this way. You are to implement the function

```
rubiks(n)
```

that returns the number of rectangles. For the $2 \times 2 \times 2$ cube, the answer should be 54.

Problem 3 [25 points]. Guessing a number. You are asked to guess a random number x between 1 and n (inclusive). To test whether your guess is correct, you are provide a function

```
is_this_it(a)
```

that returns `True` if and only if $a == x$. Implement the function

```
guess_unlimited(n, is_this_it)
```

that returns your guess of x. You may call `is_this_it` as many times as you want.

Problem 4 [25 points]. **Guessing a number, with busting**. You are asked to guess a random number x between 1 and n (inclusive). You are provided a function

```
is_this_smaller(a)
```

that returns `True` if and only if $a < x$. Implement the function

```
guess_limited(n, is_this_smaller)
```

that returns your guess of x. You may call `is_this_smaller` as many times as you want as long as it returns `True`. However, you fail if `is_this_smaller` returns `False` three times. You are to minimize the average number of calls to `is_this_smaller` that you make. For grading, you will get 15 points if you have a basic working implementation that guesses x correctly without having `is_this_smaller` returns `False` three times. You get the full score if your number of calls to `is_this_smaller` is no more than two plus the optimal number. That is, for a given n, the optimal number of calls to `is_this_smaller` may be k times. You get the full score if you make no more than $k + 2$ guesses on average.

Problem 5 [20 bonus points]. **Guessing number competition**. In this task you are given the same `is_this_smaller` function as in Problem 4. You can make any number of guesses and there is no limit on how `is_this_smaller` may be called. You are to implement your guessing routine in

```
guess_limited_plus(n, is_this_smaller).
```

Under the condition that your guesses are correct, you want to minimize the total number of calls to `is_this_smaller` plus the number of times `is_this_smaller` returns `False`. For this task, you will be competing with each other. The top 20 teams will get bonus points in 1 point decrements, i.e., the top team will get 20 points, the next 19 points, and so on. This problem is optional and if you choose not to do it, leave guess_limited_plus unchanged.