CS 361 – Lab #2 – Growing Squares

The purpose of this lab is to study the asymptotic time complexity of an algorithm. Instead of counting individual operations, the difference today is that we will now use actual elapsed time, which is more practical in the long run.

Step #4 of this lab will require you to wait for a program to finish running. While you are waiting, let's review big-O notation and its friends. Place a check mark in each box indicating whether the indicated relationship holds between functions f and g. Assume that all logs are to the base 2.

f(n)	g(n)	f is ω(g)	f is Ω(g)	f is θ(g)	f is O(g)	f is o(g)
n^2	n					
$n \log n$	$n^{1.01}$					
$8n^2 + 99n$	n^2					
2^n	3^n					
$\sqrt{n}$	2^{100}					
$n!$	4^n					
$\log(n^2)$	$\log(n)$					

In your account, create a folder called lab02, and do all your work there. From the lab02 folder on the class Web site, download the file `square.py` into your lab02 folder.

Here are the steps you need to complete for the lab today:

1. If desired, change the way that the program `square.py` obtains input from the user. I have assumed that the program will be run from the command line, and the number of squares would be entered as a command-line argument. If you prefer to ask the user interactively for the number of squares, please make the necessary modification to the code.

2. Look at the code in `square.py` to find where the variables `begin` and `end` are initialized. What part of the program is being timed? Should the input routine mentioned in step #1 be included in the code to time? Why or why not?

3. If the user tells the program to create 40 squares, then according to the code, how large will each square be, initially?

4. Now, we are ready for the time trials! Run your program several times, using the values in the first column of the following table as input. It is necessary to run the program several times for a given input value because of the random placement of the squares. In other words, with the same input, the execution time will differ somewhat. Let's do 2 trials of each input size, and average each result. I recommend that you run multiple instances of your program at the same time to take advantage of your machine's multicore processor. This will allow you to run multiple time trials at once. But don't get greedy, or else too much CPU processing will artificially inflate the times.

Number of squares	Time of trial #1	Time of trial #2	Average time
100			
200			
300			
400			
500			
1000			
More? _____			

5. Based on the figures in the last column of the table, what can you conclude about the asymptotic time complexity of the program? Based on the code, is this a reasonable or a surprising conclusion?

6. Finally, let's modify the algorithm. Create a second copy of square.py and call it `square2.py`. In this second version, let's change the way that the squares grow. Instead of growing at a linear rate, have them grow at an exponential rate. Find the place in the code where we specify how the side length should change. Change this line of code so that the side length increases in size by 1% of its current value, rather than 1% of its original value.

7. Re-run the time trials, and fill in the results in the following table.

Number of squares	Time of trial #1	Time of trial #2	Average time
100			
200			
300			
400			
500			
1000			
More? _____			

8. Based on the figures in the last column, what can you conclude about the new algorithm's complexity? How does it compare to the complexity of the original algorithm?