K. Hohmeier

CSC 380

Homework 1 Written Report

Due September 1, 2020, 11:59 PM

Problem 1 – Comparing GCD Algorithm Efficiency and Runtime

For this problem, three algorithms for computing the greatest common divisor (GCD) were coded and implemented in Python: Euclid's algorithm, a recursive implementation of Euclid's algorithm, and the consecutive integer checking algorithm. All of these algorithms were written from pseudocode provided in the textbook *Introduction to the Design and Analysis of Algorithms*. In addition, multiple timing experiments were performed on all three algorithms to compare their efficiencies and runtimes. Timing was performed via a function called time_this that was written to calculate the runtime of each algorithm. The inputs for each function were obtained by utilizing the random package in Python and creating pseudorandom numbers via a helper function called digits. From these experiments, the Big-O of each algorithm was surmised based on graphs of size of problem versus time.

Due to a large number of fluctuations and "bumps" in the graph, the experiments for Euclid's algorithm and for the recursive algorithm were performed 20 times through each iteration and then averaged. While the graphs are still not completely smooth, this averaging process did improve the outcomes depicted in the graphs. The results of these experiments are shown in the following graphs.

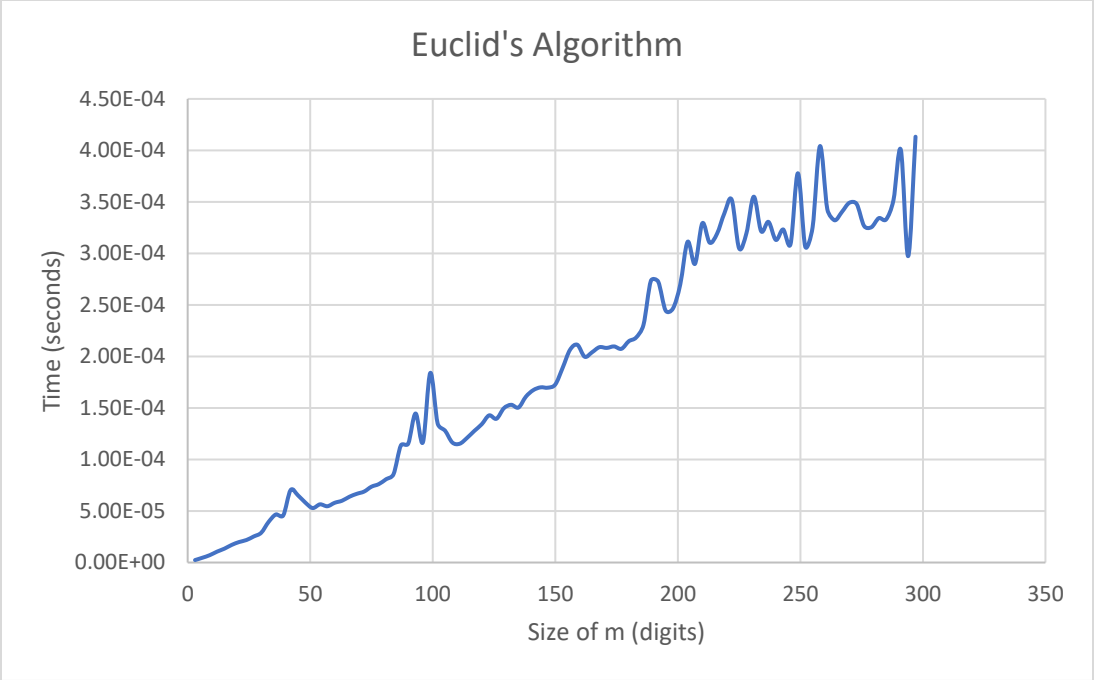

Figure 1 Graph of experiments performed with Euclid's algorithm

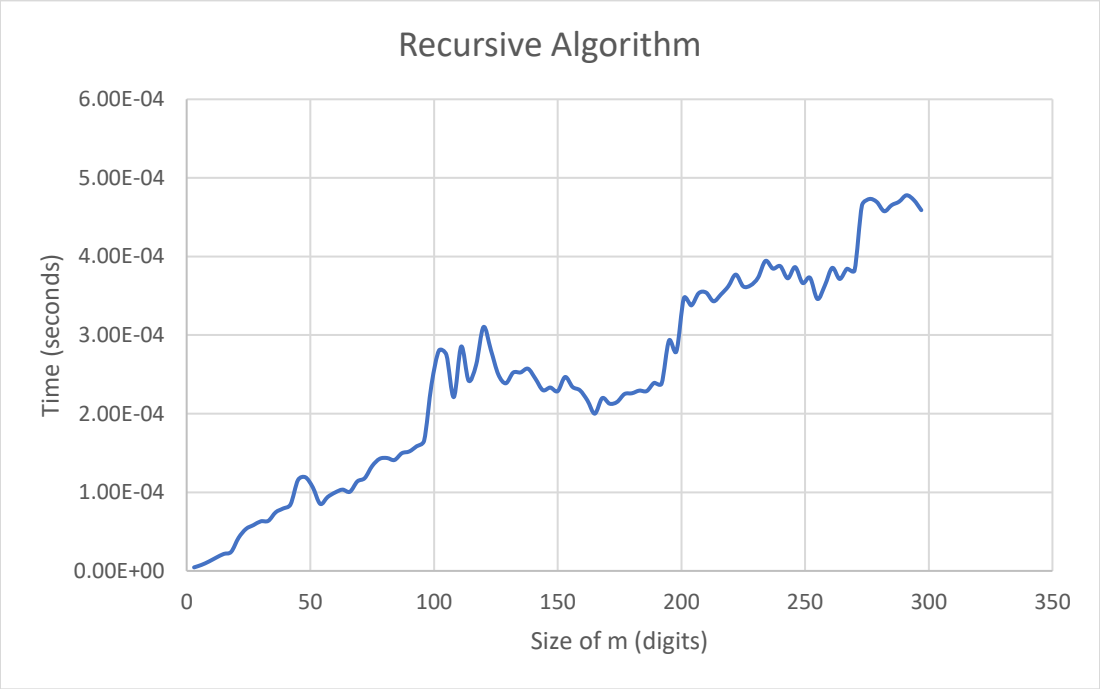

Figure 2 Graph of experiments performed with recursive algorithm

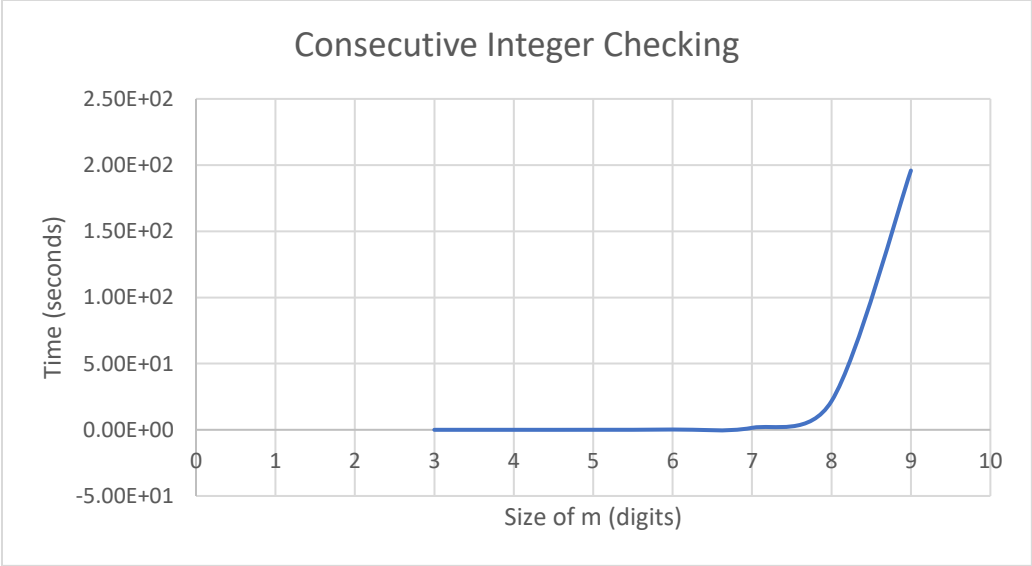

Figure 3 Graphs of experiments performed with the consecutive integer checking algorithm

Based on the results of these experiments, Euclid's algorithm (Figure 1), when graphed against as number of digits of m versus time in seconds, appears to O(n), linear time. The same is true for the recursive implementation of Euclid's algorithm (Figure 2). Although Euclid's algorithm is O(log m), logarithmic time (and the same would be true of the recursive algorithm), the O(m) result appearing on the graph is likely due to the choice of x -axis – graphing number of digits of m – versus time in seconds. Both of these functions were tested for large values of m (up to 297 digits), and they seemed to run very efficiently, with the algorithms running at fractions of a second.

The consecutive integer checking algorithm (Figure 3), on the other hand, appears to be $O(10^m)$. From these results, Euclid's algorithm and the recursive algorithm are clearly more efficient than the consecutive integer checking algorithm. Of these two algorithms, based on the time experiments it is difficult to make a definitive conclusion as to which algorithm – recursive or non-recursive Euclid's algorithm – is more efficient, but since both are log(m), they seem to be. Regardless, the consecutive integer checking algorithm is extremely inefficient. In fact, it is so

inefficient even for small values of m that running experiments for a size of m greater than 9 digits was very difficult, and so the experiments had to ended at m of size 9 digits. The amount of time was too prohibitive to run further experiments for larger values of m.

Problem 2 – Big-O Analysis of the Sieve of Eratosthenes

For this problem, an algorithm for the Sieve of Eratosthenes, a process for finding all the prime numbers less than or equal to an input number n, was implemented in Python. This algorithm was written from pseudocode provided in the textbook *Introduction to the Design and Analysis of Algorithms*. As in problem 1 with the GCD algorithms, experiments were performed to ascertain the Big-O of the algorithm. Due to a large number of fluctuations and "bumps" in the graph, the experiment for each size were performed 20 times through each iteration and then averaged. This produced a smoother-looking graph, as shown in Figure 4.

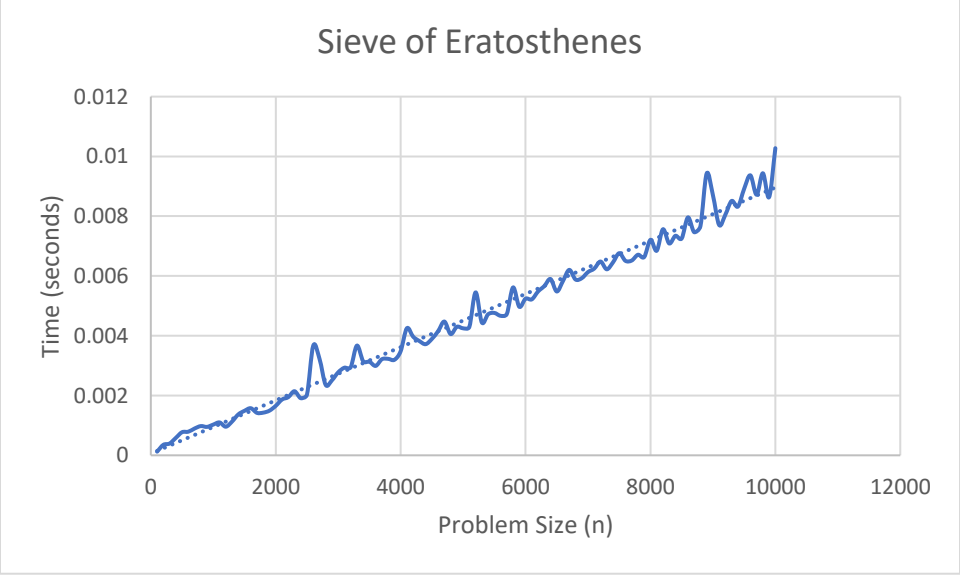

Figure 4 Graph of the experiments perform with the Sieve of Eratosthenes

The results of the experiments (Figure 4) indicate that the Sieve of Eratosthenes, as implemented from the textbook pseudocode, appears to be O(n), linear time. The time was measured in seconds, and the problem size was determined based on the value of n used in the experiments for the algorithms (values from 100 to 10,000). This linear result is interesting because the code for the implementation of the Sieve of Eratosthenes contains nested loops, which from a cursory inspection of the code would have suggested $O(n^2)$, rather than O(n). The choice of x-axis as n likely impacted the appearance of the graph obtained from these timing experiments.