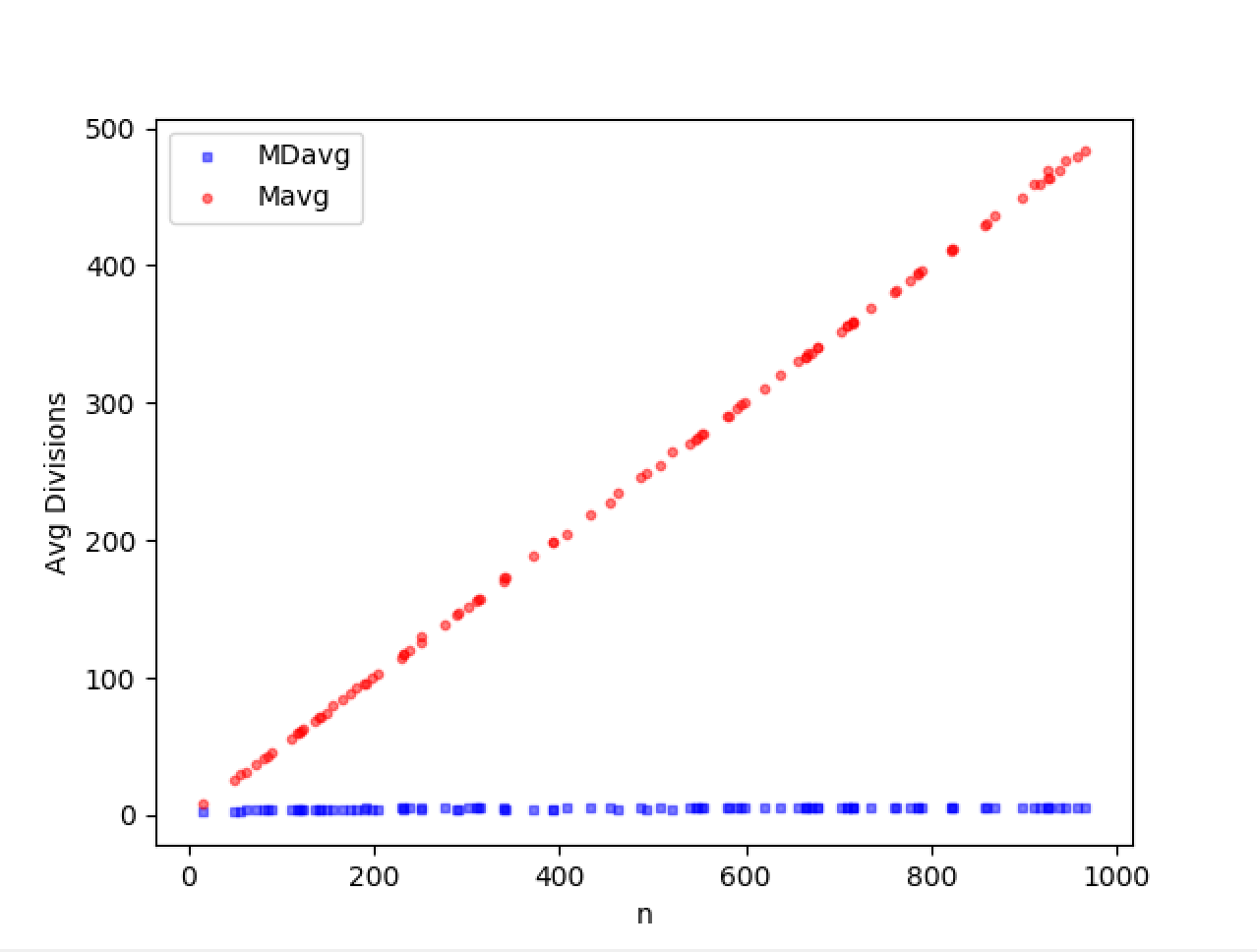
Michael Carr, Bryant Pinto

CS415

Project 1

September 29, 2019

Task 1.

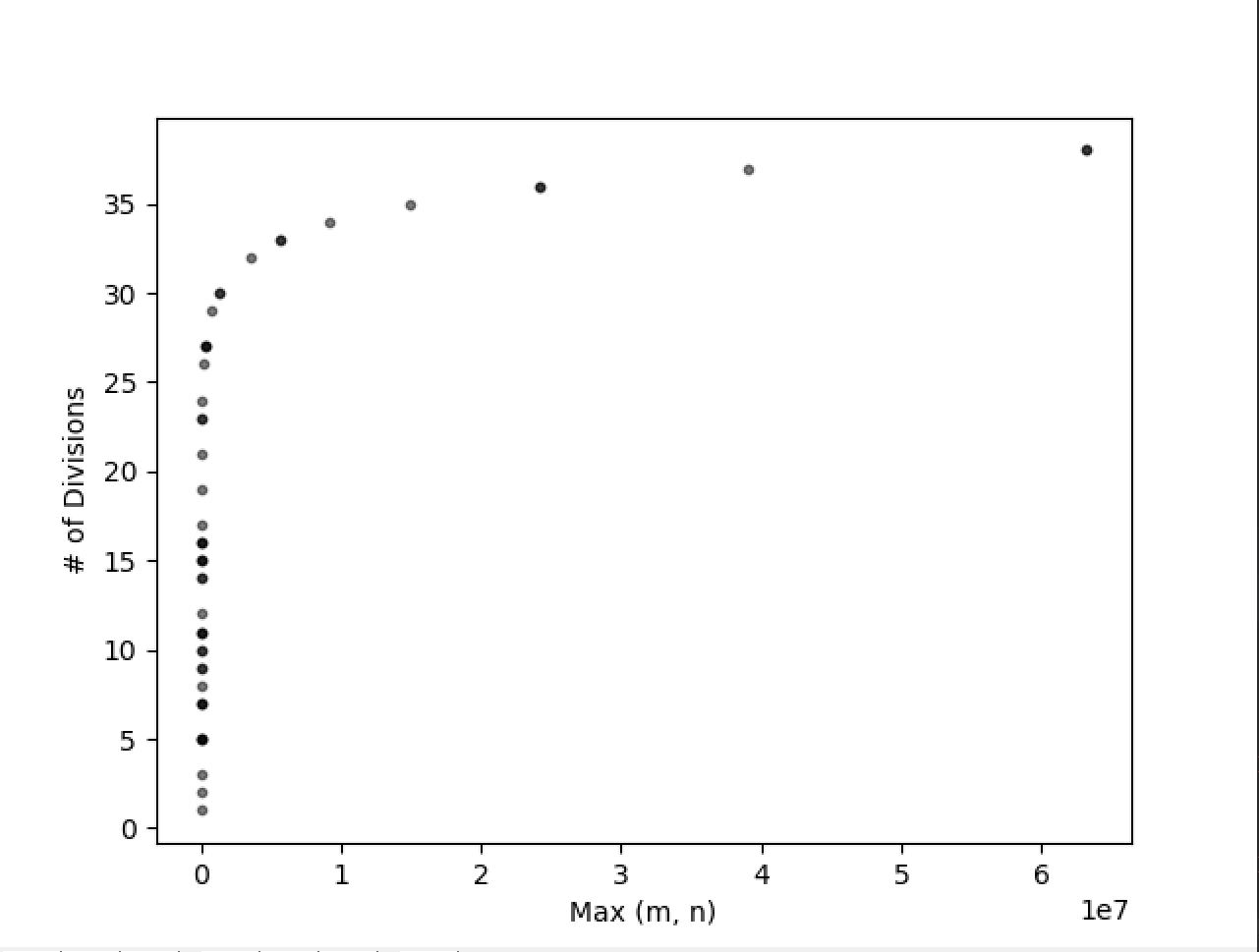


With the first task, the number n is 100 random values from 1 to 1000. In Euclid’s algorithm, if m is greater than or equal to than n/2, then the next iteration n’ will equal m and m’ will be less than n/2 since the number of modulo operations will remove m or more from n. If m is less than n/2, then the next iteration n’ will equal m and m’ will be less than n/2 since the number of modulo operations will return at most n -1. So for the recursive step, one element will be cut, at most, in half. The time complexity of Euclid’s algorithm is *O(log(n))* where n is the max initial value of m or n.

In consecutive integer checking, m and n are first determined which number is the lowest of the two. Once that is found that value is stored int. In the first step, m is divided by t and if it does, it decrements t by 1 and then goes back to the first step. If the m divides t and it is equal to zero, the second step checks to see if n divided by t is 0 and if so returns t. The third and final step just decrements t by 1 and starts over at the first step. This algorithm has at most 3 comparisons which will result in the running time to be in *O(n) in the average-case*.

Small values of m and n yield the GCD faster than larger values on both algorithms, but only a consecutive integer checking algorithm yields more basic operations beyond that of Euclid’s algorithm.

Task 2.



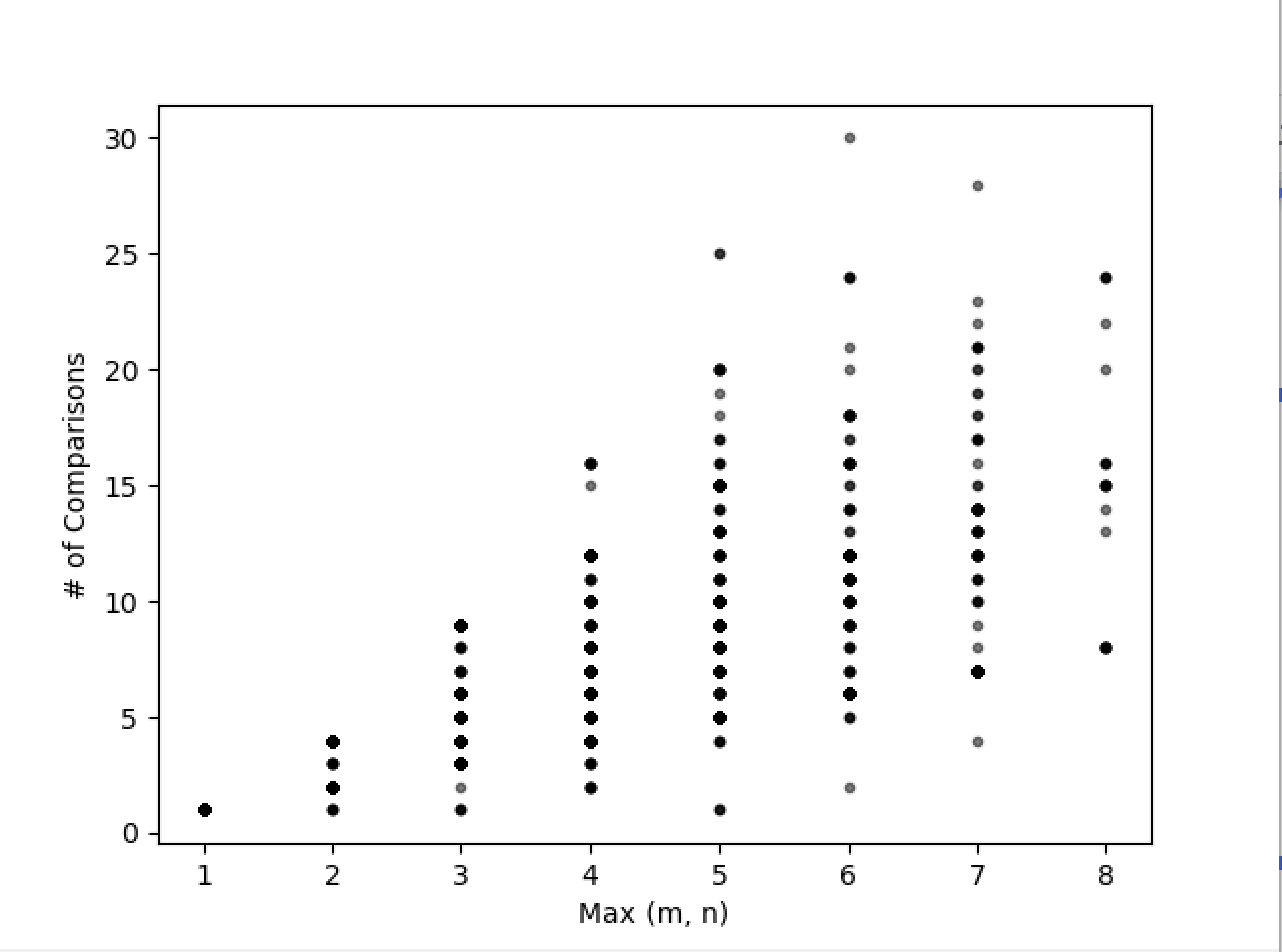
With the Fibonacci sequence using it as the input for Euclid’s algorithm will result in the worst possible time complexity. To calculate an easy Fibonacci sequence, start with m = 1 and n = 1 and add them together to get the next element in the sequence. Then take n and that next element to get the next element ie. 1, 1, 2, 3, 5, 8, 13. . . This can be achieved in reverse as well by taking two elements m and n and modulo the two together to get the next element. For example, 13 modulo 8 will equal 5 which is the above sequence in reverse.

In this task, we took the lowest value of m or n and compared it to the modulo divisions necessary to compute the GCD of m and n in the worst case, which is the two numbers in the Fibonacci sequence. In order to find m, a k value was selected at random between 2 and 50. The reason why 50 was chosen is from experimenting with many different values. If you go too big the graph gets out of hand, go to small and there is not enough data, 50 was just right.

The graph itself looks logarithmic. This would suggest that the number of divisions is greater with a greater value of m. Using values from the Fibonacci sequence is a more time-consuming process for computing their GCD as compared to using random value.

The upper limit of k = which is proven in the worst case. As shown with the Fibonacci’s efficiency class versus the consecutive integer checking algorithm and the Euclidean algorithm’s efficiency classes, they prove that Fibonacci is the worst in the case of computing values for GCD.

Task 3.



With the third task, the middle school approach to getting the GCD of m and n was done. The input used was 5,000 random integers from the range of 2 and 500. A list of prime numbers was generated for each m and n (whose product resulted in each value of m or n), then the total number of comparisons made by both lists were graphed with the correlation of the list with the largest size.

This graph shows a conical correlation between the maximum value of either m or n on the x-axis and the number of total comparisons between the two lists on the y-axis. Looking at the dotted columns going up there is indeed an almost linear line going from the first Max(m, n) to the 8th. Thinking about the way one would calculate the GCD using the middle school approach, a bigger m or n value would result in more basic operations thus requiring more time.