Grace Seiche

Professor Mello-Stark

CS 2223/B Term 2017

8 November 2017

Project 1

Executive Summary Report

To conduct my experiment, I created a program that would run Euclid's Algorithm, the Continuous Integer Checking Algorithm, and the Middle School Algorithm on the same set of inputted integers m and n. The program then prints out the greatest common denominator and the time (in CPU ticks) that it took to run each of the three algorithms on the given set of numbers. This allowed me to collect data, as shown in the chart below about the efficiency of each algorithm for a variety of different inputted numbers.

From the data in the chart, I can see that Euclid's Algorithm is always the fastest, most time efficient algorithm because it always has the fastest runtime. When looking at the algorithm itself, I determine that in worst case, Euclid runs in O(log(n)) because it runs through the while loop less than n times, but the number of times that it loops is still dependent on the inputted values (not O(1)). Each time, it at least cuts in half so therefore, the worst case would be O(log(n)). In terms of the slowest, least time efficient algorithm, the data I collected is indecisive because in some instances, the continuous integer checking is slower, but in other cases, the middle school method is slower because sometimes the continuous integer checking has a slower runtime, but sometimes the middle school method has a slower runtime. This may be due to the way that I implemented the algorithms or the m and n values that I chose to test. When analyzing the algorithm for consecutive integer checking, in worst case scenario (GCD=1) the algorithm would iterate through the loop n times (with t being all integers n to 1) where n where n is the minimum of the inputs m and n, so the worst case time efficiency would be O(n). Finally, in the middle school method algorithm contains three non-nested loops that can iterate a maximum of n times. This means that the worst case for time efficiency would be around O(3n), but when using big O notation the coefficients are dropped, so the worst case time efficiency for the middle school method is also O(n).

For space efficiency, I have to examine the algorithms that I wrote. Euclid and consecutive integer check both have the same space efficiency of O(1) because in both algorithms the only data used in addition to the integers m and n that are passed to the function there is only one integer created and the amount of storage used does not change as the inputs change. Therefore, Euclid and consecutive integer check are the most space efficient. The middle school method is the least space efficient because it requires making two additional lists of integers that vary in size depending on the number of prime factors of each input. As a rough maximum, an integer can have log(n) prime factors (the worst case being that the number is a power of two). Since this is the length of the lists created, the space efficiency of the middle school method is O(log(n)).

See next page for tables of test times and worst case efficiencies.

	M	N	Euclid Runtime	Integer Checking Runtime	Middle School Runtime	GCD
Test 1	31415	14142	2.370372e-06	0.001257	0.000426	1
Test 2	8373	5654	2.370372e-06	0.000569	0.000438	1
Test 3	7262	8495	2.370372e-06	0.000597	0.000836	1
Test 4	1521	4783	2.765434e-06	0.000143	0.000488	1
Test 5	2324	178	2.370372e-06	1.738273e-05	3.950620e-05	2
Test 6	7796	307	2.370372e-06	2.923459e-05	0.000391	1
Test 7	6729	6165	2.370372e-06	0.000599	0.000420	3
Test 8	4704	9300	2.370372e-06	0.000422	2.330866e-05	12
Test 9	5200	2205	2.765434e-06	0.000219	2.133335e-05	5
Test 10	9639	5257	2.370372e-06	0.000578	0.000149	7

	Euclid	Integer Checking	Middle School
Worst Case Time Efficiency	O(log(n))	O(n) Where n is the minimum of n and m	O(n)
Worst Case Space Efficiency	O(1)	O(1)	O(log(n))