Ankur Gupta
CS 2223 Project 1

In this experiment, we implemented and compared 3 different Greatest Common Divisor (GCD) algorithms: Euclid's method, Consecutive Integer Checking Algorithm, and Middle School procedure. Through a theoretical analysis of how the algorithms are run, as well as experiential data to back up the hypothesis, the space and time efficiencies for each implementation were determined, as described below.

Euclid:

- **Time Efficiency:** $\boldsymbol{O(log_{\phi} \min\{m, n\}), \phi = golden\ ratio}$. The worst case for Euclid's method is two consecutive numbers of the Fibonacci sequence. Assuming that F_n denotes the n^{th} number in the Fibonacci sequence, $F_{n+1}\ mod\ F_n = F_{n-1}$. This behavior causes the algorithm to go down the chain of Fibonacci numbers *n* times until the GCD = 1. Therefore, the efficiency becomes $O(\alpha), where\ \min\{m, n\}\ is\ the\ \alpha^{th}\ Fibonacci\ number$. However, as this isn't a continuous function, we need to need to take into account the asymptotic behavior of the sequence, which is $\lim_{n\to\infty} \frac{F_{n+1}}{F_n} \to \phi$. Because the numbers are being divided by this value in each iteration, the asymptotic time efficiency for the function becomes $O(\log_{\phi} \min\{m, n\})$
- **Time Efficiency:** $\boldsymbol{\Omega(1)}$**.** The best case for Euclid's method is if the two numbers are either equal or if one number is the GCD of the batch. In that situation, the loop will only run once, and terminate because the GCD has been found.
- **Space Efficiency:** $\boldsymbol{O(1)}$ **and** $\boldsymbol{\Omega(1)}$**.** As the algorithm is in-place (no new variables are created), the space efficiency is always constant.

Consecutive Integer Checking Algorithm (CICA):

- **Time Efficiency:** $\boldsymbol{O(\min\{m, n\})}$**.** The worst case for the CICA is when $GCD(m, n) = 1$. This is because the algorithm will loop from the smallest number to 1, thus resulting in a worst case of $O(\min\{m, n\})$.
- **Time Efficiency:** $\boldsymbol{\Omega(1)}$**.** The best case, similar to Euclid, is when one of the two numbers is the GCD. That means that the algorithm will terminate after one iteration, yielding a constant time.
- **Space Efficiency:** $\boldsymbol{O(1)}$ **and** $\boldsymbol{\Omega(1)}$**.** As the algorithm is in place, the space efficiency is always constant.

Middle School (MS):

- **Time Efficiency:** $\boldsymbol{O(m + n)}$. In my implementation of the MS, there are 2 major operations: finding the prime factors for *n* and *m*, and combining the common factors to determine the GCD. The worst case for MS is when both inputs are primes. That means that determining the prime factorizations for the numbers is in $O(\beta)$ time, where β is the number that is being factored. However, since there are only 2 factors in for each prime, combining the common factors is always $O(4)$ time. Therefore, the complexity for the worst case is $O(m) + O(n) + O(4) = O(m + n)$ time.
- **Time Efficiency:** $\boldsymbol{\Omega([\log_2 m] * [\log_2 n])}$**.** The best case for MS is when both of the numbers can be expressed as 2^k. This is when the prime factorizations quickest, each taking *k* iterations. That also means that the number of prime factors for both inputs are *k*, meaning that combination of the common factors is in $O(k_1 k_2)$ time. As $k = \log_2 \gamma$, that means that the complexity for the best case is $\Omega(\log_2 m) + \Omega(\log_2 n) + \Omega([\log_2 n] * [\log_2 m]) = \Omega([\log_2 n] * [\log_2 m])$ time.
- **Space Efficiency:** $\boldsymbol{O(\log_2 mn)}$**.** The worst case for space is when the numbers have the maximum number of factors. It is a known fact that these types of numbers are in the form 2^k. As there are 2 arrays being created to store the values of the factors, the space efficiency becomes $O([\log_2 m] + [\log_2 n]) = O(\log_2 mn)$.
- **Space Efficiency:** $\boldsymbol{\Omega(1)}$**.** The best case for space is when both of the numbers are prime. That means that they both have only 2 factors, and the array creation is always constant space.

As the basics of the algorithms have been summarized in this paragraph, it is obvious that the most optimal algorithm is Euclid's, as the space efficiency is better than MS's, and the time efficiency is better than CICA's. To prove this, following are a list of trials that were conducted on each of the algorithms. The trials were conducted with the focus of ensuring that the theory behind the complexity analysis is what matches in practice. The main lessons learned from this experiment were that theoretical concepts such as the golden ratio can end up playing major roles within real world applications, and that picking the right best and worst case for an algorithm can make a very major difference when trying to determine the upper and lower bounds.

Ankur Gupta
CS 2223 Project 1

m	n	Euclid Runtime	Integer Checking Runtime	Middle School Runtime	GCD	Comments
31415	14142	1.27500E-06	2.38665E-03	5.47150E-04	1	Required for project
8	13	7.75000E-07	1.90000E-06	6.70000E-06	1	Sequencial Fibonnaci numbers: worst case for Euclid
89	144	1.20000E-06	1.31000E-05	1.72250E-05	1	Larger seq Fibonnaci numbers: worst case for Euclid and bad case for Integer checking. Euclid should double and Int checking should increase by a factor of 10
2048	1024	2.50000E-07	5.00000E-07	2.15750E-05	1024	Best case for Euclid, Integer checking, and Middle school
12341	12340	3.50000E-07	2.15213E-03	1.43425E-04	1	Worst case for Integer Checking
12341	6170	3.75000E-07	9.89075E-04	1.43625E-04	1	Int checking should halve time
50001	12501	5.00000E-07	2.23415E-03	6.25750E-04	3	Large number baseline
50001	25001	4.75000E-07	4.26573E-03	7.98875E-04	1	Lower number is roughly doubled: Integer checking should double
94421	96451	8.50000E-07	1.70750E-02	3.10506E-02	1	Baseline for middle school -> Worst case, and both m and n are primes
29	190843	9.75000E-07	6.07500E-06	3.17193E-02	1	Both m and n are prime, so middle school time should be the same

It should be noted that all runtimes are in seconds. Additionally, the table below describes the different efficiencies of the program in a concise manner.

	Euclid	CICA	MS
Time Complexity	$O(log_{\phi} \min\{m, n\}), \phi = golden\ ratio$ $\Omega(1)$	$O(\min\{m, n\})$ $\Omega(1)$	$O(m + n)$ $\Omega([\log_2 m] * [\log_2 n])$
Space Complexity	$O(1)$ $\Omega(1)$	$O(1)$ $\Omega(1)$	$O(\log_2 mn)$ $\Omega(1)$