Homework 1

By Anthony DeDominic <dedominica@my.easternct.edu>

Consecutive Integer Check GCD (Bruteforce) Algorithm

	$GCD(int, int)$	Iterations	GCD
worst	$GCD(17711, 19035)$	17711	1
best	$GCD(1033, 16906)$	1033	1
average	??	7283	??

Initial Analysis

The worst case time complexity of the bruteforce algorithm is linearly proportional to the $min(N, M)$; where N is an integer and M is another integer as defined here. The worst case being when $GCD(N, M) = 1$. This is because the algorithm goes from the smallest of the two numbers and recursively counts downwards, by a factor of 1, from the starting value of $min(N, M)$ until it reaches the base case of 1 which will always divide N and M completely with no remainders. In general, this algorithm will perform faster when the $min(N, M)$ is small, $max(N, M)$ will not change anything assuming GCD will equal 1 for $GCD(N, M)$.

Time Complexity

Like explained above, this algorithm's best and worst number pairs appear to be directly influenced by the smallest of the two numbers. This is a linear relationship since the decreasing factor is only 1.

$O(min(N, M))$

```
// GCD using a bruteforce method - recursive
// large   -> larger of the two numbers
// small   -> smaller of the two numbers
// gcd     -> possible gcd, decrements by 1
// count   -> number of times this algo ran
// RETURNS -> a struct with the gcd and the count
gcd_count gcd_brute(int large, int small, int gcd, int count) {

    if (large % gcd == 0 && small % gcd == 0) {

        return (gcd_count) {gcd, count+1};
    }
```

```
    return gcd_brute(large,small,gcd-1,count+1);
}
```

Euclidean GCD Algorithm

	$GCD(int, int)$	Iterations	GCD
worst	$GCD(9955, 12581)$	15	1
best	$GCD(1331, 9331)$	4	1
average	??	8	??

Initial Analysis

This algorithm uses significantly less operations to complete. This is due in fact that the decreasing factor is $\frac{max(N,M)}{max(N,M) \bmod min(N,M)}$. This value is much larger than ~1, which is the decreasing factor for the Consecutive Integer Check algorithm. Unlike the other above algorithm, it will complete faster on average when the difference between $max()$ and $min()$ is either very far apart or very close; in the bruteforce method, closeness of said difference is irrelevant. This will allow the algorithm to fall nearer to the base case of 1 faster.

NOTE that this assumes $GCD(N, M) = 1$ which would be the worst case

Time Complexity

Where the bruteforce method has a linear time complexity, this algorithm appears to have a logarithmic complexity. Doing $log_{10}(min(N, M)) \times C$ gets me numbers very close to the amount of iterations these gcd algorithms had to run through.

$O(log_{10}(min(N, M))$

```
// GCD using euclid' algo - recursive
// large   -> larger of the two numbers
// small   -> smaller of the two numbers
// count   -> number of times this algo ran
// RETURNS -> a struct with the gcd and the count
gcd_count gcd_eculid(int large, int small, int count) {

    if (small == 0) {

        return (gcd_count) {large, count+1};
    }
```

```
    return gcd_eculid(small, (large % small), count+1);
}
```