Design and Analysis of Algorithm for Computing GCD of two numbers using Euclid's Algorithm

Nairit Banerjee
IIT2016505

Kamal Nayan Chaubey
IIT2016510

Ashutosh Vishwakarma
IIT2016017

Mandeep Chakma
IIT2016012

March 1, 2018

Abstract

This document is a final report of our study on finding the **GCD**(greatest common divisor) of two positive numbers m,n where $n \leq m$ by implementing Euclid's algorithm. This algorithm is an efficient method for computing the **GCD** of two positive numbers,the largest positive number that divides both of them without leaving a remainder.

Keywords: *GCD,Remainder,Time Complexity, Space Complexity, Algorithm Design, Graphical Analysis,*

1 Introduction And Literature Survey

In mathematics and computer science **Euclid** algorithm is an efficient method for computing the **GCD** of two positive numbers,the largest positive number that divides both of them without leaving a remainder.
The algorithm is based on the below facts:

(1)If we subtract smaller number from larger (we reduce larger number), **GCD** doesn't change. So if we keep subtracting repeatedly the larger of two, we end up with **GCD**.
For example, 21 is the **GCD** of 252 and 105 (as $252 = 21 \times 12$ and $105 = 21 \times 5$), and the same number 21 is also the **GCD** of 105 and $252 - 105 = 147$.
Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the **GCD** of the original two numbers.
The version of the Euclidean algorithm described above (and by Euclid) can take many subtraction steps to find the **GCD** when one of the given numbers is much bigger than the other.
A more efficient version of the algorithm is given below,

(2)Now instead of subtraction, if we divide smaller number by larger number, the algorithm stops when we find remainder 0.With this improvement, the algorithm never requires more steps than five times the number of digits (base 10) of the smaller integer. For example,take a = 210 and b = 45.
If we perform the above step we have:
Divide 210 by 45, and get the result 4 with remainder 30, so 210=4·45+30.
Divide 45 by 30, and get the result 1 with remainder 15, so 45=1·30+15.
Divide 30 by 15, and get the result 2 with remainder 0, so 30=2·15+0.
Thus, in this way we can compute the **GCD** of two positive numbers efficiently.

2 Algorithm Design

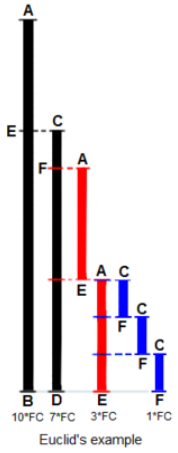

Figure 1: Euclid Algorithm Illustration.

The algorithm for finding the gcd of two positive numbers is quite simple. It can be easily done by using either recursion or loop. We simply find the gcd of $mandm$ **modulo** n recursively untill the remainder is not equal to zero. As the remainder comes out to be zero then at that time the value m will be the gcd of both the numbers and then we simply return the gcd.Following figure illustrate the description:

2.1 Pseudo Code for the Euclid's algorithm

Algorithm 1 Euclidean Algorithm

```
1: procedure GCD(m, n)
2:     if n == 0 then
3:         return m
4:     end if
5:     return GCD(n, m mod n)
6: end procedure
```

3 Analysis and Discussions

3.1 Time Complexity

The time complexity for the above stated algorithm can be computed by adding units of time taken by each statement multiplied by the number of times the statement is executed. Hence,

$T = \Sigma(cost(i) * no.of.times(i))$,for all statements

The running time will be dependent on the number of recursive calls made to the method. There can be three types of input which would lead to different number of recursive calls. If either of the inputs is itself the GCD, there would be only one recursive call made. Hence, this is the best case.
If the GCD of the inputs is anything other than 1 or either of the inputs, there would at least 2 and at most log(max(n,m))-1 recursive calls made. Hence this is the average case.
If the input numbers are co-prime or relatively prime to one another, thier GCD will be 1 and log(max(n,m)) recursive calls will be made. This is the worst case.

$$T_{best} \propto c_1$$

$$T_{average} \propto c_1(\log(\max(n,m)) - c_2)$$

$$T_{worst} \propto c_1(\log(\max(n,m)))$$

Hence the order of growth for best case, average case and worst case are

$\Omega(1), \Theta(log(n)), O(log(n))$ respectively.

We get the following running times for the following pair of numbers as input:

Table 1: Running time data

Input	Running Time	No.of Recursions	Type
1,2	5	1	Best
9,81	5	1	Best
4,26	5	1	Average
87,123	8	2	Average
9,11	17	5	Worst
6,35	11	3	Worst

We will get the following graph if we plot running time against number of recursive calls

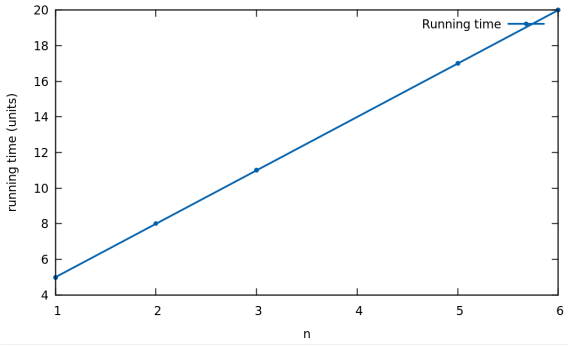

Figure 2: Running time graph.

3.2 Space Complexity

We do not use any extra memory for the computation. Hence :

$$Space(n) \propto 1$$

4 Experimental Study

We may assume the following examples to study the various running times possible,

Best Case: The best case will be when m is a multiple of n that is gcd is being the same smaller number in the input. For example suppose we are given the input as n=4 and m=16,then here in this case the gcd is calculated by our algorithm in only first recursive call as we get m modulo n == 0 and hence return answer from there. Here the gcd will be n itself that is equal to 4.

Average Case: In this case the algorithm instead of taking O(log(max(n,m))) time, takes computing time less than this and geater than O(1) time (as it took in the best case).For example lets say n=4 and m=26 then it takes only O(2) time to compute gcd instead of log(26).If both the numbers will not be coprime then it will be the average case.

Worst Case:In this case the algorithm take whole of time to compute the gcd. That is it takes whole of time O(log(max(n,m))) to compute. This will only be possible in the case when both the numbers given in the input are co-prime to each other that is their gcd is equal to 1. For example n=3 and m=7 then it takes log(7) unit of time to compute gcd = 1.

5 Conclusion

In this paper we proposed an algorithm called Euclid's algorithm for computing the GCD(greatest common divisor) of two positive numbers.

The Euclidean algorithm has many theoretical and practical applications. It is used for reducing fractions to their simplest form and for performing division in modular arithmetic. Computations using this algorithm form part of the cryptographic protocols that are used to secure internet communications, and in methods for breaking these cryptosystems by factoring large composite numbers.

The Euclidean algorithm may be used to solve Diophantine equations, such as finding numbers that satisfy multiple congruences according to the Chinese remainder theorem, to construct continued fractions, and to find accurate rational approximations to real numbers.

Finally, it can be used as a basic tool for proving theorems in number theory such as Lagrange's four-square theorem and the uniqueness of prime factorizations.

References

[1] Introduction To Algorithms (Second Edition) by Thomas H.Cormen, Charles E.Leiserson , R.L.Rivest and Clifford Stein.

[2] How to Solve it by Computer by R.G.Dromey.

[3] The C Programming Language by Brian Kernighan and Dennis Ritchie.

[4] https://www.geeksforgeeks.org/

[5] https://www.stackoverflow.com/