Investigating GCD in Euclidean Domains

Rohil Prasad
under mentorship of Tanya Khovanova

PRIMES 2013

May 18, 2013

What is GCD?

The *greatest common divisor*, or GCD, of two integers is the largest integer that divides both of them.

- Many algorithms use GCD calculation, one of the more famous being the RSA encryption algorithm.
- Several algorithms have been devised to efficiently calculate GCD for integers.

The Euclidean Algorithm

```
def euclid_gcd(a,b):
    if b > a:
        a,b = b,a
    while(!(b divides a)):
        q = a/b
        a,b = b,a-q*b

    return b
```

$$(1038, 243)$$

The Euclidean Algorithm

```
def euclid_gcd(a,b):
    if b > a:
        a,b = b,a
    while(!(b divides a)):
        q = a/b
        a,b = b,a-q*b

    return b
```

(1038, 243)

(243, 66)

The Euclidean Algorithm

```
def euclid_gcd(a,b):
    if b > a:
        a,b = b,a
    while(!(b divides a)):
        q = a/b
        a,b = b,a-q*b

    return b
```

(1038, 243)

(243, 66)

(66, 45)

The Euclidean Algorithm

```
def euclid_gcd(a,b):
    if b > a:
        a,b = b,a
    while(!(b divides a)):
        q = a/b
        a,b = b,a-q*b

    return b
```

(1038, 243)

(243, 66)

(66, 45)

(45,21)

The Euclidean Algorithm

```
def euclid_gcd(a,b):
    if b > a:
        a,b = b,a
    while(!(b divides a)):
        q = a/b
        a,b = b,a-q*b

    return b
```

(1038, 243)

(243, 66)

(66, 45)

(45,21)

(21, 3)

THE BINARY GCD ALGORITHM

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

$$(1038, 243)$$

THE BINARY GCD ALGORITHM

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

$$(1038, 243)$$

$$(519, 243)$$

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

(1038, 243)

(519, 243)

(243, 138)

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

$$(1038, 243)$$

$$(519, 243)$$

$$(243, 138)$$

$$(243, 69)$$

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

$(1038, 243)$

$(519, 243)$

$(243, 138)$

$(243, 69)$

$(87,69)$

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

(1038, 243)

(519, 243)

(243, 138)

(243, 69)

(87,69)

(69, 9)

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

(1038, 243)

(519, 243)

(243, 138)

(243, 69)

(87,69)

(69, 9)

(30, 9)

THE BINARY GCD ALGORITHM

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

(1038, 243)

(519, 243)

(243, 138)

(243, 69)

(87,69)

(69, 9)

(30, 9)

(9, 3)

The Binary GCD Algorithm

```
def binary_gcd(a,b):
    r = 0
    while(a != b):
        while (a,b) are even:
            a,b = a/2, b/2
            r = r+1
        while a is even:
            a = a/2
        while b is even:
            b = b/2
        a,b = min(a,b), abs(a-b)/2

return a*(2^r)
```

(1038, 243)

(519, 243)

(243, 138)

(243, 69)

(87,69)

(69, 9)

(30, 9)

(9, 3)

(3, 3)

Euclidean Domains

We can extend our definition of GCD to arbitrary Euclidean domains.

- A *Euclidean domain* E is a principal ideal domain with a function f such that for any nonzero a and b in E, there exists q and r in E with $a = bq + r$ and $f(r) < f(b)$. This function is called a norm, and q is called the *quotient* of a and b.
- The integers are an example of a Euclidean domain with norm $f(a) = |a|$.
- We work in $\mathbb{Z}[\sqrt{2}]$ and $\mathbb{Z}[\sqrt{3}]$.

Finding GCD in Euclidean Domains

What are some ways of efficiently calculating GCD in Euclidean domains?

- Option 1: Calculate a quotient

Finding GCD in Euclidean Domains

What are some ways of efficiently calculating GCD in Euclidean domains?

- Option 1: Calculate a quotient
- Option 2: Divide out by a small prime

Finding GCD in Euclidean Domains

What are some ways of efficiently calculating GCD in Euclidean domains?

- Option 1: Calculate a quotient
- Option 2: Divide out by a small prime
- Option 3: Approximate division

Option 1: Calculating the quotient

- The quotient of elements in $\mathbb{Z}[\sqrt{2}]$ is calculated as follows:

$$\frac{a+b\sqrt{2}}{c+d\sqrt{2}} = \frac{(a+b\sqrt{2})(c-d\sqrt{2})}{c^2-2d^2} = \frac{ac-2bd}{c^2-2d^2} + \frac{(bc-ad)\sqrt{2}}{c^2-2d^2}$$

Option 1: Calculating the quotient

- The quotient of elements in $\mathbb{Z}[\sqrt{2}]$ is calculated as follows:

$$\frac{a+b\sqrt{2}}{c+d\sqrt{2}} = \frac{(a+b\sqrt{2})(c-d\sqrt{2})}{c^2-2d^2} = \frac{ac-2bd}{c^2-2d^2} + \frac{(bc-ad)\sqrt{2}}{c^2-2d^2}$$

- Rounding each component to the nearest integer gives the quotient.

Option 1: Calculating the quotient

- The quotient of elements in $\mathbb{Z}[\sqrt{2}]$ is calculated as follows:

$$\frac{a+b\sqrt{2}}{c+d\sqrt{2}} = \frac{(a+b\sqrt{2})(c-d\sqrt{2})}{c^2-2d^2} = \frac{ac-2bd}{c^2-2d^2} + \frac{(bc-ad)\sqrt{2}}{c^2-2d^2}$$

- Rounding each component to the nearest integer gives the quotient.
- Quotient calculation is identical in $\mathbb{Z}[\sqrt{3}]$.

Option 2: Division by a small prime

- We use primes of norm 2 because it is easiest to check for divisibility.

Option 2: Division by a small prime

- We use primes of norm 2 because it is easiest to check for divisibility.
- Primes with small components have the fastest implemented division.

Option 2: Division by a small prime

- We use primes of norm 2 because it is easiest to check for divisibility.
- Primes with small components have the fastest implemented division.
- We use $1+\sqrt{3}$ for $\mathbb{Z}[\sqrt{3}]$ and $2 \pm \sqrt{2}$ for $\mathbb{Z}[\sqrt{2}]$.

Option 3: Approximate division

- Calcuation of the quotient limits the Euclidean algorithm's runtime.

Option 3: Approximate division

- Calcuation of the quotient limits the Euclidean algorithm's runtime.
- A possible solution to this is to *approximate* the quotient quickly.

Option 3: Approximate division

- Calcuation of the quotient limits the Euclidean algorithm's runtime.
- A possible solution to this is to *approximate* the quotient quickly.
- Current implementations for $\mathbb{Z}[\sqrt{2}]$ and $\mathbb{Z}[\sqrt{3}]$ involves bitshifting the components of each number by about half their bitsize and approximating the quotient with these.

Comparing algorithms

Component Bitsize	Euclidean	Binary	Approx.
100	1.45	2.70	1.46
200	2.88	5.37	2.90
300	4.36	8.62	4.78
400	6.48	12.57	6.64
500	8.21	15.96	8.65

Future Research

There are many directions in which this research can be taken:

- Extend ideas to $\mathbb{Z}[\sqrt{d}]$ for squarefree d.

Future Research

There are many directions in which this research can be taken:

- Extend ideas to $\mathbb{Z}[\sqrt{d}]$ for squarefree d.
- Improve performance of the new ‘binary’ and ‘approximate division’ algorithms.

Future Research

There are many directions in which this research can be taken:

- Extend ideas to $\mathbb{Z}[\sqrt{d}]$ for squarefree d.
- Improve performance of the new 'binary' and 'approximate division' algorithms.
- Find worst cases for the Euclidean algorithm in $\mathbb{Z}[\sqrt{d}]$.

Acknowledgments

I would like to acknowledge the following:

- Stefan Wehmeier and Ben Hinkle of Mathworks.
- My mentor Tanya Khovanova.
- The MIT PRIMES Program.
- My family.