Week 6 Lecture 2

Greatest Common Divisor

Additions to Calculator

- Greatest Common Divisor
  - Simple numerical algorithm

Greatest Common Divisor (GCD)

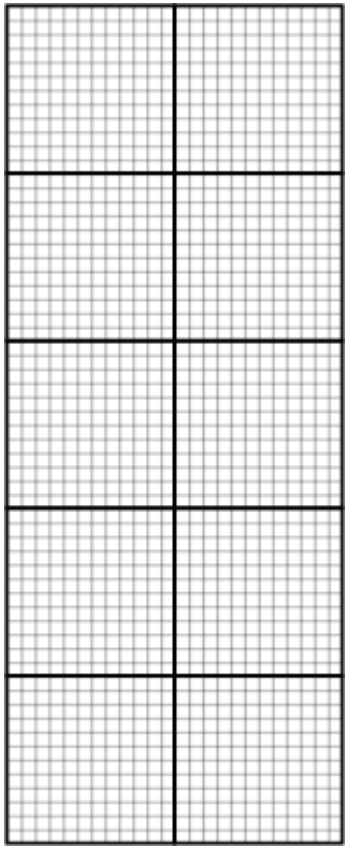
- The *Greatest Common Divisor* (GCD) of two numbers is the largest number that divides both numbers evenly
  - I.e. the division have a remainder of zero.
- If GCD(a, b) = x, then x is the largest number where a mod x = 0 and b mod x = 0.

Examples

- GCD(2, 3) = 1
  - 2 and 3 are co-prime
- GCD(3, 9) = 3
- GCD(6, 9) = 3
- GCD(8, 12) = 4
- GCD(18, 12) = 6

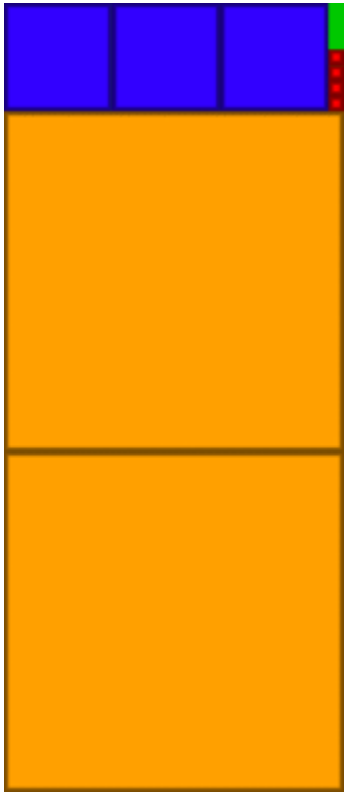
Analysis

- GCD(a, b) can be visualized as the largest square that will fit in an a x b rectangle.
  - Since both sides of the square are the same, the length of the side divides both a and b evenly.



Analysis (Euclid's Algorithm)

- Euclid's Algorithm uses the observation that the what is left over after subtracting the smaller number from the large as many times as possible, has the same GCD as the two original numbers.
  - i.e. GCD(a, b) = GCD(a, a mod b) as long as (a mod b > 0)



Euclid's Algorithm

- GCD(a, b)
  - If (a = 0)
    - Return b
  - Return GCD(a mod b, a)

Implementation

```
int gcdr(int a, int b)
{
  if (a == 0) {
    return b;
  }
  return gcdr (b % a, a);
}
```

```
GCD(10, 12) = 2
GCD(10, 15) = 5
GCD(10, 18) = 2
GCD(10, 21) = 1
GCD(10, 24) = 2
GCD(10, 27) = 1
GCD(10, 30) = 10
GCD(10, 33) = 1
GCD(10, 36) = 2
GCD(10, 39) = 1
GCD(15, 6) = 3
GCD(15, 9) = 3
```

How it works

```
int gcdr(int a, int b)
{
  if (a == 0) {
    return b;
  }
  return gcdr (b % a, a);
}
```

```
Enter integer> 12345
Enter integer> 234567
gcdr(12345, 234567)
gcdr(12, 12345)
gcdr(9, 12)
gcdr(3, 9)
gcdr(0, 3)
gcd(12345, 234567) = 3
```

```
gcdr(12345, 2345678)
gcdr(128, 12345)
gcdr(57, 128)
gcdr(14, 57)
gcdr(1, 14)
gcdr(0, 1)
gcd(12345, 2345678) = 1
```

```
int gcdr(int a, int b)
{
  if (a == 0) {
    return b;
  }
  return gcdr (b % a, a);
}
```

Tail Recursion

- Recursive call is last
  - i.e., function does not use any of the values passed in.
- Tail recursion is easy to make iterative.
  - Iterative function are more space efficient.
    - They do not need to save the parameter values for each call.

Iterative GCD

```
int gcd ( int a, int b )
{
  int c;
  while ( a != 0 ) {
    c = a;
    a = b%a;
    b = c;
  }
  return b;
}
```

- Complexity is swapping
  - (a % b) < a
  - (a % b) < b
- Let a = b % a, and let b = a
  - After the first iteration, the smaller number will be b
    - Insure that the algorithm stops when the smaller number is 0

Euclid's Algorithm is fast

- Log(n) algorithm
  - Finds the GCD in a time that is the logarithm of the size of the number.
  - Designated O(log(n))

```
Enter integer> 12345
Enter integer> 234567
gcdr(12345, 234567)
gcdr(12, 12345)
gcdr(9, 12)
gcdr(3, 9)
gcdr(0, 3)
gcd(12345, 234567) = 3
```

```
gcdr(12345, 2345678)
gcdr(128, 12345)
gcdr(57, 128)
gcdr(14, 57)
gcdr(1, 14)
gcdr(0, 1)
gcd(12345, 2345678) = 1
```

Algorithmic Complexity

- Algorithmic complexity is the time it takes an algorithm to compute a number.
- Computation complexity is usually measured by O(f(n)), pronounce big-O of "f" of "n"
  - O(f(n)) represents the slowest growing function of the input that is higher than the longest time the algorithm takes.

Examples

- Retrieval from an array is O(1)
- Binary search is O(log(n))
- Linear search is O(n)
- Merge Sort is O(nlog(n))
- Selection Sort is $O(n^2)$
- Traveling Salesman is $O(2^n)$
- Euclid's Algorithm is O(log n)

Euclid's Algorith is O(log(n))

- (b % a) < a/2
- Each step input reduced by ½
- # steps grows by $\log_2 n$

```
int gcdr(int a, int b)
{
  if (a == 0) {
    return b;
  }
  return gcdr (b % a, a);
}
```