Today: Euclid’s GCD algorithm

• Why look at it?
  – Clever.
  – Historically important.

• Lessons to learn:
  – Proofs of correctness
  – Order notation
  – Worst-case analysis
  – Definition of basic computational “steps”.

Algorithm Description

Intuition: how do we find a GCD of two numbers?

1. Divide the larger number by the smaller.
2. If it divides evenly, we’re done.
3. Otherwise, repeat (1) with the smaller number and the remainder.

Pseudocode:

```plaintext
gcd(x, y): x, y : Integer, 0 < x <= y
   while x > 0
      a <- y mod x
      y <- x
      x <- a
   end
   return y
```
Correctness proof (does it work)

**Theorem:** \( \gcd(x, y) \) computes the GCD

**Lemma:** Let \((x_t, y_t) = \text{values of } (x, y) \) after \( t \) loops. Then it’s easy to see that

1. \((x_0, y_0) = (x, y)\)
2. \(x_{t+1} = y_t \mod x_t\)
3. \(y_{t+1} = x_t\)

We want to show that \( \gcd(x_t, y_t) = \gcd(x, y) \).

**Proof (by induction):**

- Base case \((t = 0): \gcd(x_0, y_0) = \gcd(x, y)\)

- Inductive case: Assume \( \gcd(x_{t-1}, y_{t-1}) = \gcd(x, y) \).

  We want \( \gcd(x_t, y_t) = \gcd(x, y) \). We’ll show this by showing that \( d \) is a common divisor of \((x_t, y_t)\) if and only if it is a common divisor of \((x_{t-1}, y_{t-1})\).

  First, assume \( d \mid x_{t-1} \) and \( d \mid y_{t-1} \) (\( d \) divides both evenly). Then by (3), \( d \mid (y_t = x_{t-1}) \). Also, by (2) above, \( \exists q \) such that \( y_{t-1} = qx_{t-1} + x_t \). Therefore \( d \mid (x_t = y_{t-1} - qx_{t-1}) \), so \( d \) is a common divisor of \( x_t \) and \( y_t \).

  Now assume \( d \mid x_t \) and \( d \mid y_t \). Then by (3), \( d \mid (x_{t-1} = y_t) \). Also, by (2), \( y_{t-1} = qx_{t-1} + x_t \), so \( d \mid (y_{t-1} = qx_{t-1} + x_t) \), and \( d \) is a common divisor of \( x_{t-1} \) and \( y_{t-1} \). Done!

Thus, the set of common divisors of \( x_t, y_t \) is the same as the set of common divisors of \( x_{t-1}, y_{t-1} \). In particular, the greatest elements of these sets are the same. So \( \gcd(x_t, y_t) = \gcd(x_{t-1}, y_{t-1}) \). Since by the induction assumption \( \gcd(x_{t-1}, y_{t-1}) = \gcd(x, y) \), we have \( \gcd(x_t, y_t) = \gcd(x, y) \), as needed.

To prove that the algorithm’s correct, we have to show not only that the invariants hold, but that the algorithm terminates. In this case, it’s easy: \( x \) and \( y \) are positive, and decrease on each iteration. When it terminates, \( x_t = 0 \) and since \( y_t = \gcd(0, y_t) = \gcd(x_t, y_t) = \gcd(x, y) \) by the invariant, the algorithm outputs \( y_t = \gcd(x, y) \).

Complexity analysis (how fast does it work?)

Intuitively, it seems to be “pretty fast,” but how can we prove this? We’ll show that the variables decrease quickly. Then we’ll sue this to bound the number of iterations.

**Lemma:** \( y_{t+2} \leq y_t / 2 \).

**Proof:** If \( x_t > y_t / 2 \),

\[
y_{t+2} = x_{t+1} = y_t \mod x_t = y_t - x_t \leq y_t / 2
\]

2
Otherwise $x_t < y_t/2$, so

$$y_{t+2} \leq y_{t+1} = x_t \leq y_t/2$$

Therefore the binary length of $y_t$ decreases by 1 every 2 iterations, so there are at most $2 \log y$ iterations.

Note: this is a worst-case complexity bound ($O(n)$), not a tight one ($\Theta(n)$).

Note: the cost of “basic operations” is maybe not realistic – we’re assuming division takes constant time for arbitrarily large integers. In the next class, we’ll reconsider this assumption.