Math 127, Wed Feb 03

- Use a laptop or desktop with a large screen so you can read these words clearly.
- In general, please turn off your camera and mute yourself.
- Exception: When we do groupwork, please turn both your camera and mic on. (Groupwork will not be recorded.)
- Please always have the chat window open to ask questions.
- Reading for today: 2.5–2.6.
- Reading for Mon: 3.1.
- PS01 outline due today, full version due Mon Feb 08.
- First real problem session Fri Feb 05, 10:00–noon on Zoom.

When you submit outlines on Gradescope:

1. Definitions: Match this "problem" to all pages on which definitions appear.

2. Problem outlines: Match this "problem" to all of the pages on which problems plan appear.

3. Completion: You can match this to whatever you like. Basically you 4 of the 10 points just for doing the outline.

Correction from last time

Certainly not true that two random numbers a and b have gcd 1; e.g., 50% chance that each of a and b is even, so 25% chance that 2 is a common divisor of a and b.
In fact, probability that $\gcd(a, b) = 1$ is:

$$\begin{aligned}
&(\text{not both even})(\text{not both div by } 3)(\text{not both div by } 5)\cdots \\
&= (1 - \frac{1}{4})(1 - \frac{1}{9})(1 - \frac{1}{25})\cdots \\
&= \frac{6}{\pi^2} \approx 61\%
\end{aligned}$$

$\frac{1}{\zeta(2)}$

But if you avoid common factors that you can see in the digits of a and b, like 2, 3, and 5, probability goes up to around 95%.

The Euclidean Algorithm, written out in a table

gcd(a,b)

a b

$$r_{-1} = q_1 r_0 + r_1 \qquad (0 \le r_1 < r_0)$$

$$r_0 = q_2 r_1 + r_2 \qquad (0 \le r_2 < r_1)$$

$$r_1 = q_3 r_2 + r_3 \qquad (0 \le r_3 < r_2)$$

$$\vdots$$

$$r_{N-4} = q_{N-2} r_{N-3} + r_{N-2} \qquad (0 \le r_{N-2} < r_{N-3})$$

$$r_{N-3} = q_{N-1} r_{N-2} + r_{N-1} \qquad (0 \le r_{N-1} < r_{N-2})$$

$$r_{N-2} = q_N r_{N-1}$$

$= \gcd(a,b)$

Precise statement of results

Thm: It is a fact that:

1. The Euclidean Algorithm terminates after finitely many steps, with some final nonzero remainder r_{N-1}.
2. Any common divisor of a and b divides r_{N-1}. (So r_{N-1} is at least as big as any common divisor of a and b.)
3. The last nonzero remainder r_{N-1} divides both a and b. (So r_{N-1} is, in fact, a common divisor of a and b, which means that r_{N-1} is the **greatest** common divisor of a and b.)

Super-non-obvious consequence: Every common divisor of a and b divides $\gcd(a, b)$.

$\gcd(416, 127) = 1$

Proof that every common divisor divides r_{N-1}

Suppose d is a common divisor of a and b, i.e., d divides both a and b.

Rewrite each remainder in terms of previous remainders:

$$r_1 = r_{-1} - q_1 r_0$$

$$r_2 = r_0 - q_2 r_1$$

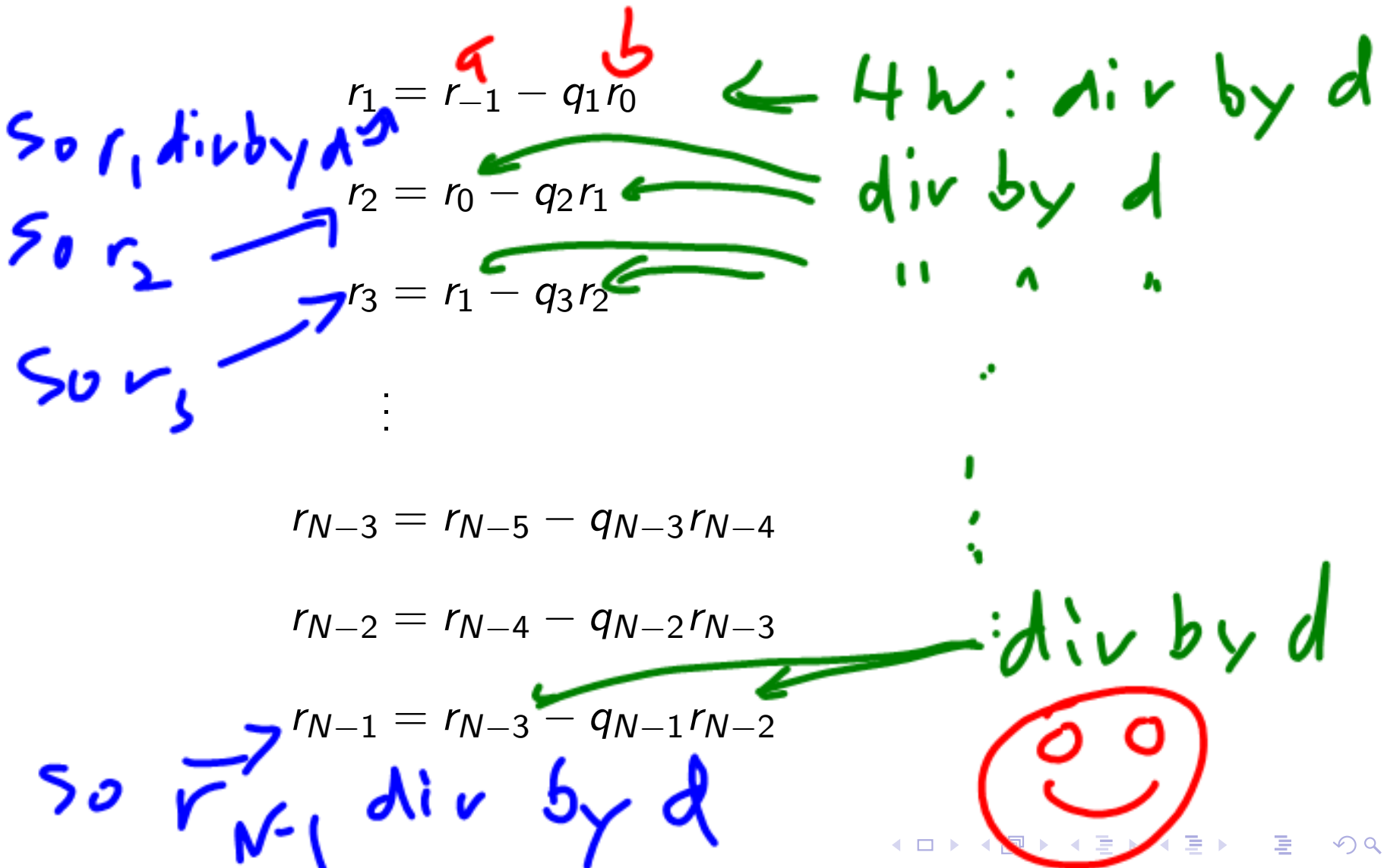
$$r_3 = r_1 - q_3 r_2$$

$$\vdots$$

$$r_{N-3} = r_{N-5} - q_{N-3} r_{N-4}$$

$$r_{N-2} = r_{N-4} - q_{N-2} r_{N-3}$$

$$r_{N-1} = r_{N-3} - q_{N-1} r_{N-2}$$



The Signed Euclidean Algorithm

Same as Euclidean Algorithm, except that you use the Signed Division Theorem (i.e., you allow negative remainders so you can make the remainders as small as possible).

Example: gcd(416, 127).

$416 = 4(127) - 92$

$416 = 3(127) + 35$

$127 = 4(35) - 13$

$35 = 3(13) - 4$

$13 = 3(4) + 1$

$-4 = -1(4)$

$16 = 2(8)$

gcd

Another look at the Euclidean algorithm

An **integer linear combination of** a **and** b is an expression of the form $ra + sb$, where $r, s \in \mathbf{Z}$.

$$r_1 = r_{-1} - q_1 r_0$$

So r_1 is an integer lin comb of a and b.

$$r_2 = r_0 - q_2 r_1$$

So r_2 is an integer lin comb of a and b.

$$r_3 = r_1 - q_3 r_2$$

So r_3 is an integer lin comb of a and b.

$$\vdots$$

$$r_{N-3} = r_{N-5} - q_{N-3} r_{N-4}$$

$$r_{N-2} = r_{N-4} - q_{N-2} r_{N-3}$$

$$r_{N-1} = r_{N-3} - q_{N-1} r_{N-2}$$

So gcd(a,b) is an int lin comb of a and b.

Bezout's identity

It follows that:

Ex $867509x + 8334527y = 1$

Theorem (Bezout's Identity)

Let a and b be nonzero integers. The equation

$$ax + by = \gcd(a, b)$$

has a solution $x, y \in \mathbf{Z}$.

The algorithm used to solve the above equation is called **Euclidean Rewriting**, since the point is that we can rewrite all of the remainders r_i as integer linear combinations of a and b.
We also note, for later:

Corollary

Let a and b be nonzero integers. For $c \in \mathbf{Z}$, the equation

$$ax + by = c$$

has a solution $x, y \in \mathbf{Z}$ if and only if $\gcd(a, b)$ divides c.

An example of Bezout

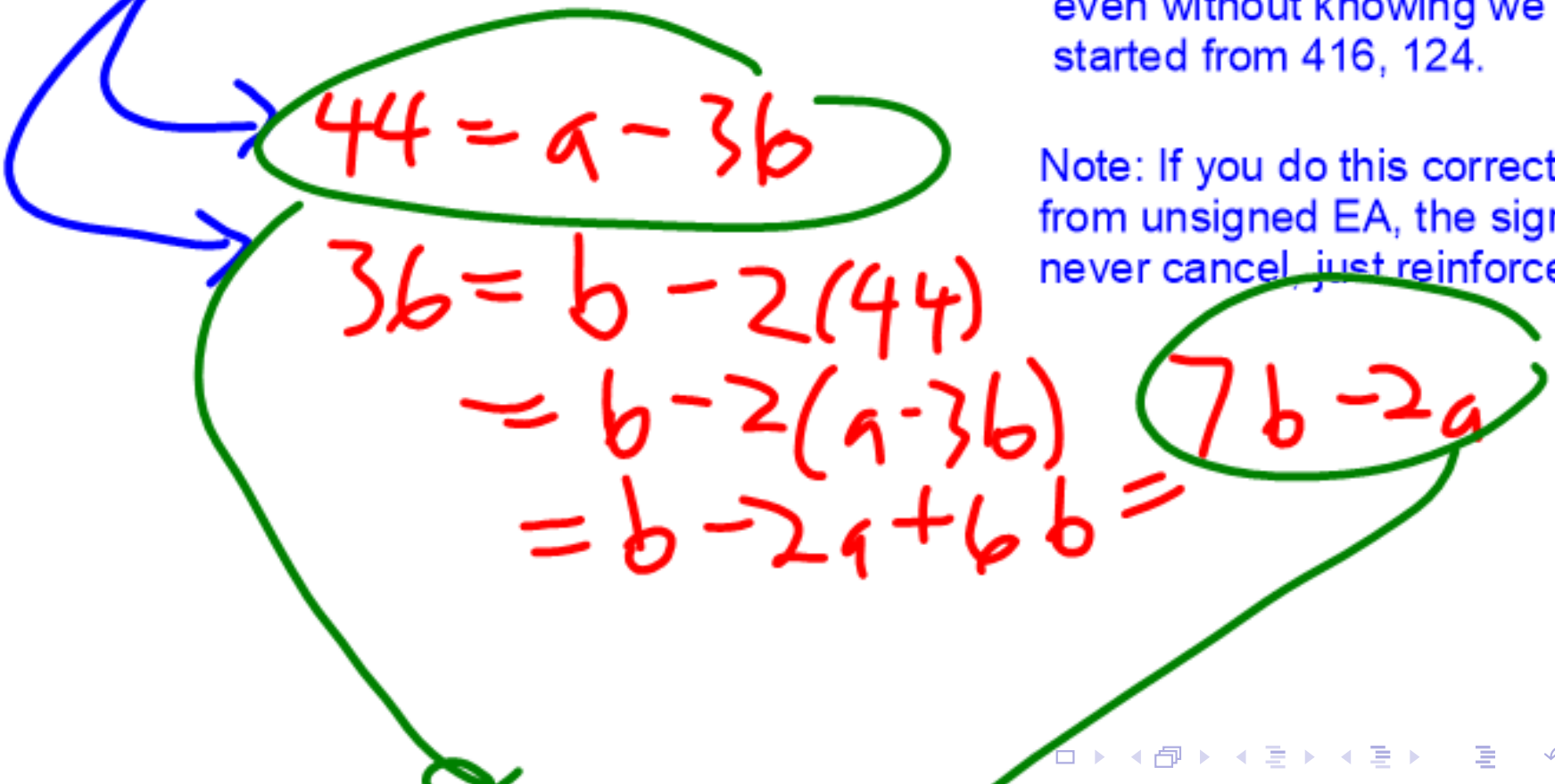
Solve $416x + 124y = \gcd(416, 124)$:

$$416 = 3(124) + 44$$
$$124 = 2(44) + 36$$
$$44 = 1(36) + 8$$
$$36 = 4(8) + 4$$

If d divides both 416 and 124, d must divide 4.

Not obvious from defn: If gcd=4, maybe 3 is also a common divisor. But Thm tells us it can't be a c.d., even without knowing we started from 416, 124.

Note: If you do this correctly from unsigned EA, the signs never cancel, just reinforce.



$$8 = 44 - 36$$

$$= (9 - 3b) - (7b - 2a)$$

$$= 9 - 3b - 7b + 2a$$

$$= 3a - 10b$$

$$4 = 36 - 4(8)$$

$$= 76 - 2a - 4(3a - 10b)$$

$$= 47b - 14a$$

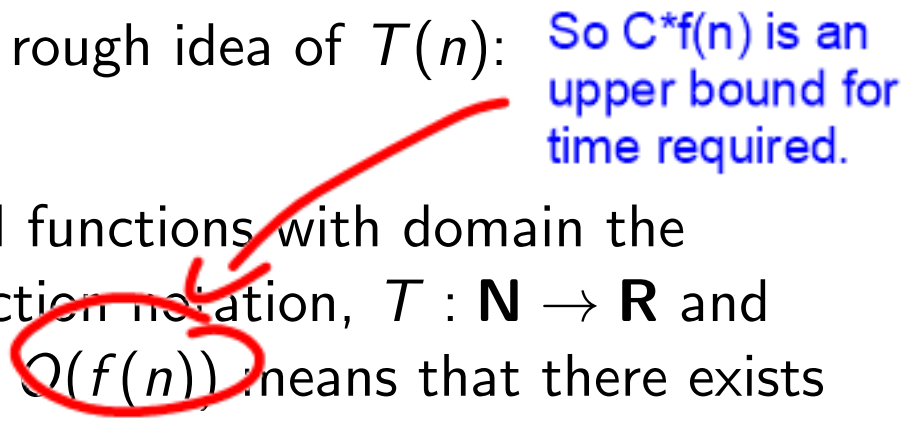
Check: 47(124) - 14(416) = 4

A crash course in complexity

Definition

The **complexity** of an algorithm is the (estimated) time or space that an algorithm needs to finish, given an input of size n. Often, this description comes in the form of a **worst-case time estimate** $T(n)$, that is, a function $T(n)$ such that, given an input of size n, the algorithm is guaranteed to finish within $T(n)$ steps (though possibly sooner).

We use **big-O** notation to give a rough idea of $T(n)$:



Definition

Let $T(n)$ and $f(n)$ be real-valued functions with domain the natural numbers $\mathbf{N}$ (in fancy function notation, $T : \mathbf{N} \to \mathbf{R}$ and $f : \mathbf{N} \to \mathbf{R}$). To say that $T(n) = O(f(n))$ means that there exists some constant C such that $T(n) \leq Cf(n)$ for all $n \in \mathbf{N}$. The notation $O(f(n))$ is known as **big** O **notation**.

Examples of complexities we've already seen

- Naive algorithm for $\gcd(a, b)$ has worst-case time estimate $T(n) = 2n$, or in big-O notation, $O(n)$.
- Improved naive algorithm on PS01: $T(n) = O(\sqrt{n})$.

Recall from calculus/precalc:

$$\log(n) << \sqrt{n} << n$$

Meaning $\lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$ and $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$. So $O(\sqrt{n})$ is way faster than $O(n)$, but $O(\log n)$ leaves both of those in the dust.

The Euclidean Algorithm is exponentially faster

Theorem

Let a, b, and n be nonzero integers with $|a| \geq |b|$ and $|b| \leq n$. Using the Signed Euclidean Algorithm to compute $\gcd(a, b)$ finishes in $O(\log n)$ time, or more precisely, requires $O(\log n)$ division-with-remainder steps to finish.

Idea of proof: Look back at the Signed Euclidean Algorithm.

Another example of a complexity estimate

The traditional Christmas carol "The 12 Days of Christmas" has the following structure: On day 1, the singer gets one gift of type 1 (a partridge in a pear tree) from their true love; on day 2, the singer gets two gifts of type 2 and one gift of type 1 (two turtledoves and a partridge in a pear tree); and so on. Suppose this song can be extended to any arbitrary number of days.

- Give a big-O estimate of the *number* of gifts the singer receives on day n.
- Give a big-O estimate of the *total number* of gifts the singer receives over the entire song, going from day 1 through day n.