CSCI 5451 Fall 2015

Week 2 Notes

Professor Ellen Gethner

August 30, 2015

Fibonacci Numbers, continued

Topic: Three Algorithms to compute F_i

Algorithm 1

- The first algorithm is recursive and the easiest to design and verify correctness.

Algorithm 1

- The first algorithm is recursive and the easiest to design and verify correctness.

- **Fib(n)**

  If $n \leq 1$ return n

  else return Fib($n-1$) $+$ Fib($n-2$)

Algorithm 1

- The first algorithm is recursive and the easiest to design and verify correctness.
- **Fib**(n)

  If $n \leq 1$ return n

  else return Fib$(n-1)$ + Fib$(n-2)$
- This may be the one and only algorithm that we'll see this semester that is clearly correct because it mimics word for word the definition of F_n.

Algorithm 1

- The first algorithm is recursive and the easiest to design and verify correctness.
- **Fib(n)**

  If $n \leq 1$ return n

  else return Fib($n-1$) + Fib($n-2$)
- This may be the one and only algorithm that we'll see this semester that is clearly correct because it mimics word for word the definition of F_n.
- The big question now is: what is the runtime of Fib(n)?

Runtime of Fib(n)

- We'll begin by finding an upper bound on the runtime.

Runtime of Fib(n)

- We'll begin by finding an upper bound on the runtime.
- To this end, let $f(n) = F_n$ and suppose $h(n)$ is the number of operations used to compute $f(n)$.

Runtime of Fib(n)

- We'll begin by finding an upper bound on the runtime.
- To this end, let $f(n) = F_n$ and suppose $h(n)$ is the number of operations used to compute $f(n)$.
- What is an upper bound for $h(n)$?

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.

- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.
- $\Rightarrow h(n) \leq 2h(n-1)$

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.
- $\Rightarrow h(n) \leq 2h(n-1)$
- $\leq 2 \times 2h(n-2)$

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.
- $\Rightarrow h(n) \leq 2h(n-1)$
- $\leq 2 \times 2h(n-2)$
- $\leq 2 \times 2 \times 2h(n-3)$

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.
- $\Rightarrow h(n) \leq 2h(n-1)$
- $\leq 2 \times 2h(n-2)$
- $\leq 2 \times 2 \times 2h(n-3)$
- $\vdots$

Upper bound on $h(n)$

- For $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-1) \geq h(n-2)$.
- $\Rightarrow h(n) \leq 2h(n-1)$
- $\leq 2 \times 2h(n-2)$
- $\leq 2 \times 2 \times 2h(n-3)$
- $\vdots$
- $\leq 2^i h(n-i)$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$
- $\vdots$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$
- $\vdots$
- $\leq 2^{n-1} h(n-(n-1))$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n - i)$
- $\vdots$
- $\leq 2^{n-1} h(n - (n - 1))$
- $= 2^{n-1} h(1)$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$
- $\vdots$
- $\leq 2^{n-1} h(n-(n-1))$
- $= 2^{n-1} h(1)$
- $= 2^{n-1}$

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$
- $\vdots$
- $\leq 2^{n-1} h(n-(n-1))$
- $= 2^{n-1} h(1)$
- $= 2^{n-1}$
- Conclusion so far: the runtime of Fib(n), at the worst, is 2^n.

Upper bound on $h(n)$, continued

- $\leq 2^i h(n-i)$
- $\vdots$
- $\leq 2^{n-1} h(n-(n-1))$
- $= 2^{n-1} h(1)$
- $= 2^{n-1}$
- Conclusion so far: the runtime of Fib(n), at the worst, is 2^n.
- In that case, the runtime of Fib(n) is at most exponential in the size of the input.

Lower bound on $h(n)$

- As before, for $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.

Lower bound on $h(n)$

- As before, for $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-2) \leq h(n-1)$.

Lower bound on $h(n)$

- As before, for $n > 1$, $h(n) = h(n-1) + h(n-2) + c$, where c is a constant describing the work done during initialization.
- Thus for large enough n, we have that $h(n-2) \leq h(n-1)$.
- Since $h(n) = h(n-1) + h(n-2) + c$, it follows that

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$

[1] Note that $h(0) = h(1) = 1$

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$
- $\geq 2 \times 2h(n-4)$

[1] Note that $h(0) = h(1) = 1$

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$
- $\geq 2 \times 2h(n-4)$
- $\vdots$

[1] Note that $h(0) = h(1) = 1$

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$
- $\geq 2 \times 2h(n-4)$
- $\vdots$
- $\geq 2^i h(n-2i)$

[1] Note that $h(0) = h(1) = 1$

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$
- $\geq 2 \times 2h(n-4)$
- $\vdots$
- $\geq 2^i h(n-2i)$
- $\vdots$

[1] Note that $h(0) = h(1) = 1$

Lower bound on $h(n)$, continued[1]

- $h(n) \geq 2h(n-2)$
- $\geq 2 \times 2h(n-4)$
- $\vdots$
- $\geq 2^i h(n-2i)$
- $\vdots$
- $= \begin{cases} 2^{\frac{n}{2}} h(0) & \text{if } n \text{ is even} \\ 2^{\lfloor \frac{n}{2} \rfloor} h(1) & \text{if } n \text{ is odd} \end{cases}$

[1] Note that $h(0) = h(1) = 1$

Runtime for algorithm Fib

- **Conclusion.** Fib(n) has exponential runtime because

Runtime for algorithm Fib

- **Conclusion.** Fib(n) has exponential runtime because
- $2^{\lfloor \frac{n}{2} \rfloor} \leq Fib(n) \leq 2^{n-1}$

Second Algorithm to Compute F_n

- **fibit**(n)

Second Algorithm to Compute F_n

- **fibit**(n)

Second Algorithm to Compute F_n

- **fibit**(n)
- $\quad i = 1; j = 0$

Second Algorithm to Compute F_n

- **fibit**(n)
- $i = 1$; $j = 0$
- for $k = 1$ to n do

Second Algorithm to Compute F_n

- **fibit(n)**
- $i = 1; j = 0$
- for $k = 1$ to n do
- $j = i + j$

Second Algorithm to Compute F_n

- **fibit(n)**
- $i = 1; j = 0$
- for $k = 1$ to n do
- $j = i + j$
- $i = j - i$

Second Algorithm to Compute F_n

- **fibit(n)**
- $i = 1; j = 0$
- for $k = 1$ to n do
- $j = i + j$
- $i = j - i$
- return j

Algorithm fibit(n)

- Is our (alleged) algorithm correct?

Algorithm fibit(n)

- Is our (alleged) algorithm correct?
- If so, what is the run time?

Algorithm fibit(n)

- Is our (alleged) algorithm correct?
- If so, what is the run time?
- We'll start by proving correctness.

Proof of correctness of fibit(n) by induction

Recall that F_n stands for the nth Fibonacci number.

- There is a lot going on in this algorithm, none of which is particularly intuitive.

Proof of correctness of fibit(n) by induction

Recall that F_n stands for the nth Fibonacci number.

- There is a lot going on in this algorithm, none of which is particularly intuitive.

- **Claim.** After the loop finishes iterating, we have $j = F_n$ and $i = F_{n-1}$. (This is our statement $P(n)$.)

Proof of correctness of fibit(n) by induction

Recall that F_n stands for the nth Fibonacci number.

- There is a lot going on in this algorithm, none of which is particularly intuitive.
- **Claim.** After the loop finishes iterating, we have $j = F_n$ and $i = F_{n-1}$. (This is our statement $P(n)$.)
- **Base Cases.**
  - $n = 1$ and $k = 1 \Rightarrow j = 1 = F_1$ and $i = 0 = F_0$
  - Thus fibit(1)=j= $1 = F_1$ and $i = 0 = F_0$ ✓

Bases cases, continued

-
  - $n = 2$ and $k = 1 \Rightarrow j = 1 = F_1$ and $i = 0 = F_0$
  - $n = 2$ and $k = 2 \Rightarrow j = 1$ and $i = 1$.

Bases cases, continued

-
  - $n = 2$ and $k = 1 \Rightarrow j = 1 = F_1$ and $i = 0 = F_0$
  - $n = 2$ and $k = 2 \Rightarrow j = 1$ and $i = 1$.
- Thus $\text{fibit}(2) = j = 1 = F_2$ and $i = 1 = F_1$. $\checkmark$

Bases cases, continued

- Let's do one more base case.

Bases cases, continued

- Let's do one more base case.
- Since the first two iterations of the loop are already computed in the previous base case, we can start at $n = 3$ and $k = 3$.

Bases cases, continued

- Let's do one more base case.
- Since the first two iterations of the loop are already computed in the previous base case, we can start at $n = 3$ and $k = 3$.
- That is, $n = 3$ and $k = 3 \Rightarrow j = 2$ and $i = 1$

Bases cases, continued

- Let's do one more base case.
- Since the first two iterations of the loop are already computed in the previous base case, we can start at $n = 3$ and $k = 3$.
- That is, $n = 3$ and $k = 3 \Rightarrow j = 2$ and $i = 1$
- $\Rightarrow fibit(3) = 2 = F_3$ and $i = 1 = fibit(2)$. $\checkmark$

Induction Hypothesis

- For a fixed integer $N \geq 1$ we have fibit(N)$= F_N$

Induction Hypothesis

- For a fixed integer $N \geq 1$ we have fibit(N)$= F_N$
- and at the last iteration of the loop we have $j = F_k$ and $i = F_{k-1}$ for all $k \leq N$.

Induction Hypothesis

- For a fixed integer $N \geq 1$ we have fibit(N)$= F_N$
- and at the last iteration of the loop we have $j = F_k$ and $i = F_{k-1}$ for all $k \leq N$.
- **Question.** What is fibit($N + 1$)?

Induction Hypothesis

- For a fixed integer $N \geq 1$ we have fibit(N)$= F_N$
- and at the last iteration of the loop we have $j = F_k$ and $i = F_{k-1}$ for all $k \leq N$.
- **Question.** What is fibit($N + 1$)?
- Of course we hope the answer is F_{N+1}.

Inductive Step

- We need only examine the last iteration of the loop, namely

Inductive Step

- We need only examine the last iteration of the loop, namely
- for the case $k = N + 1$. Why?

Inductive Step

- We need only examine the last iteration of the loop, namely
- for the case $k = N + 1$. Why?
- In particular, by the induction hypothesis, at the Nth iteration of the loop,

Inductive Step

- We need only examine the last iteration of the loop, namely
- for the case $k = N + 1$. Why?
- In particular, by the induction hypothesis, at the Nth iteration of the loop,
- the value of j is F_N and the value of i is F_{N-1}.

Inductive step so far: at $k = N$, we have $j = F_N$ and $i = F_{N-1}$

- So at $k = N + 1$ we have

Inductive step so far: at $k = N$, we have $j = F_N$ and $i = F_{N-1}$

- So at $k = N + 1$ we have
- $j = F_N + F_{N-1} = F_N$, and

Inductive step so far: at $k = N$, we have $j = F_N$ and $i = F_{N-1}$

- So at $k = N + 1$ we have
- $j = F_N + F_{N-1} = F_N$, and
- $i = F_{N+1} - F_{N-1} = F_N + F_{N-1} - F_{N-1} = F_N$. ✓

Inductive step so far: at $k = N$, we have $j = F_N$ and $i = F_{N-1}$

- So at $k = N + 1$ we have
- $j = F_N + F_{N-1} = F_N$, and
- $i = F_{N+1} - F_{N-1} = F_N + F_{N-1} - F_{N-1} = F_N$. ✓
- **Conclusion.** By the second principle of mathematical induction, we have shown that $\text{fibit}(n) = F_n \ \ \forall n \in \mathbb{Z}^+$.

Inductive step so far: at $k = N$, we have $j = F_N$ and $i = F_{N-1}$

- So at $k = N + 1$ we have
- $j = F_N + F_{N-1} = F_N$, and
- $i = F_{N+1} - F_{N-1} = F_N + F_{N-1} - F_{N-1} = F_N$. ✓
- **Conclusion.** By the second principle of mathematical induction, we have shown that $\text{fibit}(n) = F_n \;\; \forall n \in \mathbb{Z}^+$.
- **QED**.

What about the run time of fibit(n)?

- The loop iterates n times with two additions at each iteration.

What about the run time of fibit(n)?

- The loop iterates n times with two additions at each iteration.
- The initialization takes two operations.

What about the run time of fibit(n)?

- The loop iterates n times with two additions at each iteration.
- The initialization takes two operations.
- In total, computing fibit(n) takes $2n + 2$ operations, which means

What about the run time of fibit(n)?

- The loop iterates n times with two additions at each iteration.
- The initialization takes two operations.
- In total, computing fibit(n) takes $2n + 2$ operations, which means
- the run time is linear in n.

What about the run time of fibit(n)?

- The loop iterates n times with two additions at each iteration.
- The initialization takes two operations.
- In total, computing fibit(n) takes $2n + 2$ operations, which means
- the run time is linear in n.
- Whew, much better than exponential!!

Are we finished computing the Fibinacci numbers?

- No way.

Are we finished computing the Fibinacci numbers?

- No way.

- **Question.** Is there a better-than-linear-time algorithm to compute F_n????

Third Algorithm to Compute F_n

- Let's work on a new algorithm **fibel(n)**, which stands for "Elegant Fibonacci"

Third Algorithm to Compute F_n

- Let's work on a new algorithm **fibel(n)**, which stands for "Elegant Fibonacci"
- **Here goes.** Let $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, which is a 2×2 matrix.

Third Algorithm to Compute F_n

- Let's work on a new algorithm **fibel(n)**, which stands for "Elegant Fibonacci"
- **Here goes.** Let $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, which is a 2×2 matrix.
- **Claim.** The lower left entry of F^n is F_n and the lower right entry is F_{n+1}.

Third Algorithm to Compute F_n

- Let's work on a new algorithm **fibel(n)**, which stands for "Elegant Fibonacci"
- **Here goes.** Let $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, which is a 2×2 matrix.
- **Claim.** The lower left entry of F^n is F_n and the lower right entry is F_{n+1}.
- In other words, after raising matrix F to the nth power, the resulting 2×2 matrix has the property that

Third Algorithm to Compute F_n

- Let's work on a new algorithm **fibel(n)**, which stands for "Elegant Fibonacci"

- **Here goes.** Let $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, which is a 2×2 matrix.

- **Claim.** The lower left entry of F^n is F_n and the lower right entry is F_{n+1}.

- In other words, after raising matrix F to the nth power, the resulting 2×2 matrix has the property that

- the lower left entry of F^n is F_n, the nth Fibonacci number, and the lower right entry is the F_{n+1}th Fibonacci number.

Proof of claim by induction

- **Base Cases.**

Proof of claim by induction

- **Base Cases.**
- For $n = 1$ note that $F^1 = F = \begin{bmatrix} 0 & 1 \\ \color{red}{1} & \color{blue}{1} \end{bmatrix}$.

Proof of claim by induction

- **Base Cases.**
- For $n = 1$ note that $F^1 = F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$.
- Thus the lower left corner of F^1 is 1, which is F_1, and the lower right entry is 1, which is F_2, as hoped.

Proof of claim by induction

- **Base Cases.**
- For $n = 1$ note that $F^1 = F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$.
- Thus the lower left corner of F^1 is 1, which is F_1, and the lower right entry is 1, which is F_2, as hoped.
- For $n = 2$ note that $F^2 = F \times F = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ and we see that the lower left entry is $1 = F_2$ and the lower right entry is $2 = F_3$.

Base Cases, continued

- $n = 3$. $F^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$

Base Cases, continued

- $n = 3$. $F^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$
- and note that the lower left entry of F^3 is $2 = F_3$, and

Base Cases, continued

- $n = 3$. $F^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$
- and note that the lower left entry of F^3 is $2 = F_3$, and
- and the lower right entry of F^3 is $3 = F_4$.

Base Cases, continued

- $n = 3$. $F^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$
- and note that the lower left entry of F^3 is $2 = F_3$, and
- and the lower right entry of F^3 is $3 = F_4$.
- Similarly, when $n = 4$ we have $F^4 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$, with $F_4 = 3$ and $F_5 = 5$, as desired.

Induction Hypothesis and Inductive Step

- **Induction Hypothesis.** For some fixed integer $k \geq 1$ we have that

Induction Hypothesis and Inductive Step

- **Induction Hypothesis.** For some fixed integer $k \geq 1$ we have that
- the lower left entry of F^k is F_k and the lower right entry of F^k is F_{k+1}.

Induction Hypothesis and Inductive Step

- **Induction Hypothesis.** For some fixed integer $k \geq 1$ we have that
- the lower left entry of F^k is F_k and the lower right entry of F^k is F_{k+1}.
- **Inductive Step.** We must prove that the lower left entry of F^{k+1} is F_{k+1} and

Induction Hypothesis and Inductive Step

- **Induction Hypothesis.** For some fixed integer $k \geq 1$ we have that

- the lower left entry of F^k is F_k and the lower right entry of F^k is F_{k+1}.

- **Inductive Step.** We must prove that the lower left entry of F^{k+1} is F_{k+1} and

- the lower right entry of F^{k+1} is F_{k+2}.

Inductive Step

- **Subproof.** $F^{k+1} = F^k F = \begin{bmatrix} * & * \\ F_k & F_{k+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

Inductive Step

- **Subproof.** $F^{k+1} = F^k F = \begin{bmatrix} * & * \\ F_k & F_{k+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
- $= \begin{bmatrix} * & * \\ F_{k+1} & F_k + F_{k+1} \end{bmatrix} = \begin{bmatrix} * & * \\ F_{k+1} & F_{k+2} \end{bmatrix} \checkmark$

Inductive Step

- **Subproof.** $F^{k+1} = F^k F = \begin{bmatrix} * & * \\ F_k & F_{k+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

- $= \begin{bmatrix} * & * \\ F_{k+1} & F_k + F_{k+1} \end{bmatrix} = \begin{bmatrix} * & * \\ F_{k+1} & F_{k+2} \end{bmatrix} \checkmark$

- **Conclusion.** By induction, the lower left entry of F^n is $F_n \ \ \forall n \in \mathbb{Z}^+$.

Inductive Step

- **Subproof.** $F^{k+1} = F^k F = \begin{bmatrix} * & * \\ F_k & F_{k+1} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
- $= \begin{bmatrix} * & * \\ F_{k+1} & F_k + F_{k+1} \end{bmatrix} = \begin{bmatrix} * & * \\ F_{k+1} & F_{k+2} \end{bmatrix} \checkmark$
- **Conclusion.** By induction, the lower left entry of F^n is $F_n \ \ \forall n \in \mathbb{Z}^+$.
- **QED**

Run Time?

- What about the run time of fibel(n)?

Run Time?

- What about the run time of fibel(n)?
- The next homework assignment will contain a step-by-step algorithm describing fibel(n),

Run Time?

- What about the run time of fibel(n)?
- The next homework assignment will contain a step-by-step algorithm describing fibel(n),
- and your job will be to determine the run time.

New Topic

Next: The Euclidean Algorithm and its relation to the Fibonacci numbers.

Warmup: The Division Algorithm (not really an algorithm)

- **Division Algorithm.** For any $a, b \in \mathbb{Z}^+$ with $a \geq b$, $\exists$ unique integers q and r such that

Warmup: The Division Algorithm (not really an algorithm)

- **Division Algorithm.** For any $a, b \in \mathbb{Z}^+$ with $a \geq b$, $\exists$ unique integers q and r such that

- $a = bq + r$ with $0 \leq r < b$.

Warmup: The Division Algorithm (not really an algorithm)

- **Division Algorithm.** For any $a, b \in \mathbb{Z}^+$ with $a \geq b$, $\exists$ unique integers q and r such that
- $a = bq + r$ with $0 \leq r < b$.
- **Trivia.** q stands for *quotient* and r stands for *remainder*.

Warmup: The Division Algorithm (not really an algorithm)

- **Division Algorithm.** For any $a, b \in \mathbb{Z}^+$ with $a \geq b$, $\exists$ unique integers q and r such that
- $a = bq + r$ with $0 \leq r < b$.
- **Trivia.** q stands for *quotient* and r stands for *remainder*.
- The proof can be found in any Discrete Math textbook (or ask me).

Warmup, continued.

- A direct and easy-to-prove consequence of the Division Algorithm is the following.

Warmup, continued.

- A direct and easy-to-prove consequence of the Division Algorithm is the following.

- **Theorem D.** If $a, b, c, d, r, s \in \mathbb{Z}$ with $d \neq 0$ such that $d|a$ and $d|b$ then $d|(ra + sb)$.

Warmup, continued.

- A direct and easy-to-prove consequence of the Division Algorithm is the following.
- **Theorem D.** If $a, b, c, d, r, s \in \mathbb{Z}$ with $d \neq 0$ such that $d|a$ and $d|b$ then $d|(ra + sb)$.
- It then follows that $d|(a + b)$, $d|(a - b)$ and $d|ra$ (among many other things...).

Warmup, continued.

- A direct and easy-to-prove consequence of the Division Algorithm is the following.
- **Theorem D.** If $a, b, c, d, r, s \in \mathbb{Z}$ with $d \neq 0$ such that $d|a$ and $d|b$ then $d|(ra + sb)$.
- It then follows that $d|(a + b)$, $d|(a - b)$ and $d|ra$ (among many other things...).
- In other words, if d divides both a and b, then d divides any integer linear combination of a and b. The above are just a few special cases.

Warmup, greatest common divisor

- **Definition.** Let $a, b \in \mathbb{Z}^+$. The *greatest common divisor of a and b*, written $gcd(a, b)$, is the largest positive integer d that divides both a and b.

Warmup, greatest common divisor

- **Definition.** Let $a, b \in \mathbb{Z}^+$. The *greatest common divisor of a and b*, written $gcd(a, b)$, is the largest positive integer d that divides both a and b.

- That is, $gcd(a, b) = d$.

Warmup, greatest common divisor

- **Definition.** Let $a, b \in \mathbb{Z}^+$. The *greatest common divisor of a and b*, written $gcd(a, b)$, is the largest positive integer d that divides both a and b.

- That is, $gcd(a, b) = d$.

- **Example 1.** $gcd(4, 6) = 2$, $gcd(3, 7) = 1$, $gcd(100, 4) = 4$, and $gcd(100001, 0) = 100001$, etc.

Warmup, greatest common divisor

- **Definition.** Let $a, b \in \mathbb{Z}^+$. The *greatest common divisor of a and b*, written $gcd(a, b)$, is the largest positive integer d that divides both a and b.

- That is, $gcd(a, b) = d$.

- **Example 1.** $gcd(4, 6) = 2$, $gcd(3, 7) = 1$, $gcd(100, 4) = 4$, and $gcd(100001, 0) = 100001$, etc.

- One place (of many) where *gcd* is needed is in RSA encryption.

Warmup, greatest common divisor

- **Definition.** Let $a, b \in \mathbb{Z}^+$. The *greatest common divisor of a and b*, written $gcd(a, b)$, is the largest positive integer d that divides both a and b.

- That is, $gcd(a, b) = d$.

- **Example 1.** $gcd(4, 6) = 2$, $gcd(3, 7) = 1$, $gcd(100, 4) = 4$, and $gcd(100001, 0) = 100001$, etc.

- One place (of many) where *gcd* is needed is in RSA encryption.

- Thus the actual computation of $gcd(a, b)$ is necessary and important.

Example 2. (Illustration of the Euclidean Algorithm)

Problem. Find $gcd(54, 21)$. That is, compute $d = gcd(54, 21)$ where $a = 54$ and $b = 21$.

- $54 = 2 \times 21 + 12$ and note that
  - $0 \leq 12 < 21$ and by Theorem D that
  - $d|12$ because $d|54$ and $d|21$.

Example 2. (Illustration of the Euclidean Algorithm)

Problem. Find $gcd(54, 21)$. That is, compute $d = gcd(54, 21)$ where $a = 54$ and $b = 21$.

- $54 = 2 \times 21 + 12$ and note that
  - $0 \leq 12 < 21$ and by Theorem D that
  - $d|12$ because $d|54$ and $d|21$.
- For notational purposes, let $r_1 = 12$, and $q_1 = 2$, where r_1 stands for the first remainder and q_1 stands for the first quotient.

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.
- For notational purposes, let $r_2 = 9$, and $q_2 = 1$, where r_2 stands for the second remainder and q_2 stands for the second quotient.

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.
- For notational purposes, let $r_2 = 9$, and $q_2 = 1$, where r_2 stands for the second remainder and q_2 stands for the second quotient.
- $12 = 1 \times 9 + 3 \Rightarrow d|3$ $(q_3 = 1,\ r_3 = 3)$

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.

- For notational purposes, let $r_2 = 9$, and $q_2 = 1$, where r_2 stands for the second remainder and q_2 stands for the second quotient.

- $12 = 1 \times 9 + 3 \Rightarrow d|3$ $(q_3 = 1,\ r_3 = 3)$

- $9 = 3 \times 3 + 0$. $(q_4 = 3$ and $r_4 = 0)$

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.
- For notational purposes, let $r_2 = 9$, and $q_2 = 1$, where r_2 stands for the second remainder and q_2 stands for the second quotient.
- $12 = 1 \times 9 + 3 \Rightarrow d|3$ $(q_3 = 1,\ r_3 = 3)$
- $9 = 3 \times 3 + 0$. $(q_4 = 3$ and $r_4 = 0)$
- Conclusion so far: $d|3$ in which case either $d = 1$ or $d = 3$.

Euclidean algorithm example, continued

- $21 = 1 \times 12 + 9$ and note that
  - $0 \leq 9 < 12$ and by Theorem D, that
  - $d|9$ because $d|12$ and $d|21$.
- For notational purposes, let $r_2 = 9$, and $q_2 = 1$, where r_2 stands for the second remainder and q_2 stands for the second quotient.
- $12 = 1 \times 9 + 3 \Rightarrow d|3$ $(q_3 = 1,\ r_3 = 3)$
- $9 = 3 \times 3 + 0$. $(q_4 = 3$ and $r_4 = 0)$
- Conclusion so far: $d|3$ in which case either $d = 1$ or $d = 3$.
- Which is it?

The Actual Euclidean Algorithm

Recall that $a \pmod b$ is the remainder upon dividing a by b. That is if $a = qb + r$ with $0 \leq r < b$ then $a \pmod b = r$.

- **Algorithm Euclid**
- **Input:** $a, b \in \mathbb{Z}^+ \cup \{0\}$, $a \geq b$, $b \neq 0$.
- **Output:** $gcd(a, b)$
- If $b = 0$
- then return a
- else return Euclid$(b, a \pmod b)$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \le r_1 < b$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \leq r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \leq r_2 < r_1$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \leq r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \leq r_2 < r_1$
- $r_1 = q_3 r_2 + r_3 \qquad 0 \leq r_3 < r_2$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \le r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \le r_2 < r_1$
- $r_1 = q_3 r_2 + r_3 \qquad 0 \le r_3 < r_2$
- $r_2 = q_4 r_3 + r_4 \qquad 0 \le r_4 < r_3$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \leq r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \leq r_2 < r_1$
- $r_1 = q_3 r_2 + r_3 \qquad 0 \leq r_3 < r_2$
- $r_2 = q_4 r_3 + r_4 \qquad 0 \leq r_4 < r_3$
- $\vdots$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \le r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \le r_2 < r_1$
- $r_1 = q_3 r_2 + r_3 \qquad 0 \le r_3 < r_2$
- $r_2 = q_4 r_3 + r_4 \qquad 0 \le r_4 < r_3$
- $\vdots$
- $r_{k-3} = q_{k-1} r_{k-2} + r_{k-1} \qquad 0 \le r_{k-1} < r_{k-2}$

Euclidean Algorithm Written out in Stages

- $a = q_1 b + r_1 \qquad 0 \leq r_1 < b$
- $b = q_2 r_1 + r_2 \qquad 0 \leq r_2 < r_1$
- $r_1 = q_3 r_2 + r_3 \qquad 0 \leq r_3 < r_2$
- $r_2 = q_4 r_3 + r_4 \qquad 0 \leq r_4 < r_3$
- $\vdots$
- $r_{k-3} = q_{k-1} r_{k-2} + r_{k-1} \qquad 0 \leq r_{k-1} < r_{k-2}$
- $r_{k-2} = q_k r_{k-1} + r_k \qquad r_k = 0$ and we halt.

Correctness of the Euclidean Algorithm

- Does the algorithm halt?

Correctness of the Euclidean Algorithm

- Does the algorithm halt?
- If so, does it produce the desired output, namely $gcd(a, b)$?

Correctness of the Euclidean Algorithm

- Does the algorithm halt?
- If so, does it produce the desired output, namely $gcd(a, b)$?
- The following lemma will help answer the latter question.

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

- **Proof.** Let $d = gcd(a, b)$ and $d_0 = gcd(b, r)$.

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

- **Proof.** Let $d = gcd(a, b)$ and $d_0 = gcd(b, r)$.

- Our goal is, therefore, to show that $d_0 = d$.

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

- **Proof.** Let $d = gcd(a, b)$ and $d_0 = gcd(b, r)$.

- Our goal is, therefore, to show that $d_0 = d$.

- To that end, it suffices to show that $d | d_0$ and $d_0 | d$.

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

- **Proof.** Let $d = gcd(a, b)$ and $d_0 = gcd(b, r)$.

- Our goal is, therefore, to show that $d_0 = d$.

- To that end, it suffices to show that $d | d_0$ and $d_0 | d$.

- Note that $r = a - bq$, in which case $d | r$ by Theorem D (because $d | a$ and $d | b$).

Euclidean Algorithm: A helpful lemma

- **Lemma.** Let $a, b \in \mathbb{Z}^+$ satisfy $a = bq + r$ with $0 \leq r < b$ (so $q = \lfloor \frac{a}{b} \rfloor$).

  Then $gcd(a, b) = gcd(b, r)$.

- **Proof.** Let $d = gcd(a, b)$ and $d_0 = gcd(b, r)$.
- Our goal is, therefore, to show that $d_0 = d$.
- To that end, it suffices to show that $d | d_0$ and $d_0 | d$.
- Note that $r = a - bq$, in which case $d | r$ by Theorem D (because $d | a$ and $d | b$).
- So far we have that $d | r$ and $d | b$, which means that $d | gcd(b, r)$ and thus $d | d_0$.

Proof of lemma, continued. So far we have $d | d_0$

- Thus it remains to show that $d_0 | d$.

Proof of lemma, continued. So far we have $d|d_0$

- Thus it remains to show that $d_0|d$.
- **Here goes.** By the definition of *gcd* we have that $d_0|b$ and that $d_0|(a - \lfloor \frac{a}{b} \rfloor b)$

Proof of lemma, continued. So far we have $d|d_0$

- Thus it remains to show that $d_0|d$.
- **Here goes.** By the definition of *gcd* we have that $d_0|b$ and that $d_0|(a - \lfloor \frac{a}{b} \rfloor b)$
- $\Rightarrow d|a$ (by Theorem D)

Proof of lemma, continued. So far we have $d|d_0$

- Thus it remains to show that $d_0|d$.
- **Here goes.** By the definition of gcd we have that $d_0|b$ and that $d_0|(a - \lfloor \frac{a}{b} \rfloor b)$
- $\Rightarrow d|a$ (by Theorem D)
- $\Rightarrow d_0|gcd(a, b)$

Proof of lemma, continued. So far we have $d|d_0$

- Thus it remains to show that $d_0|d$.
- **Here goes.** By the definition of *gcd* we have that $d_0|b$ and that $d_0|(a - \lfloor \frac{a}{b} \rfloor b)$
- $\Rightarrow d|a$ (by Theorem D)
- $\Rightarrow d_0|gcd(a, b)$
- $\Rightarrow d_0|d$.

Proof of lemma, continued. So far we have $d|d_0$

- Thus it remains to show that $d_0|d$.
- **Here goes.** By the definition of *gcd* we have that $d_0|b$ and that $d_0|(a - \lfloor \frac{a}{b} \rfloor b)$
- $\Rightarrow d|a$ (by Theorem D)
- $\Rightarrow d_0|gcd(a, b)$
- $\Rightarrow d_0|d$.
- In total, we have $d|d_0$ and $d_0|d$

Proof of lemma, continued. So far we have $d | d_0$

- Thus it remains to show that $d_0 | d$.
- **Here goes.** By the definition of *gcd* we have that $d_0 | b$ and that $d_0 | (a - \lfloor \frac{a}{b} \rfloor b)$
- $\Rightarrow d | a$ (by Theorem D)
- $\Rightarrow d_0 | gcd(a, b)$
- $\Rightarrow d_0 | d$.
- In total, we have $d | d_0$ and $d_0 | d$
- $\Rightarrow d = d_0$. That is, $gcd(a, b) = gcd(b, r)$. **QED**

Why is the Euclidean Algorithm Correct?

- First, by the lemma, we have $r_{k-1} = gcd(a, b)$ because

Why is the Euclidean Algorithm Correct?

- First, by the lemma, we have $r_{k-1} = gcd(a, b)$ because
- $gcd(a, b) = gcd(b, r_1) = gcd(r_1, r_2) = gcd(r_2, r_3) = \cdots = gcd(r_{k-1}, r_k) = gcd(r_{k-1}, 0) = r_{k-1}$.

Why is the Euclidean Algorithm Correct?

- First, by the lemma, we have $r_{k-1} = gcd(a, b)$ because
- $gcd(a, b) = gcd(b, r_1) = gcd(r_1, r_2) = gcd(r_2, r_3) = \cdots = gcd(r_{k-1}, r_k) = gcd(r_{k-1}, 0) = r_{k-1}$.
- Finally, and importantly, the algorithm halts because $r_1 > r_2 > r_3 > \cdots$ is a strictly decreasing sequence of non-negative integers, which means that, eventually, $r_k = 0$ for some $k \geq 1$.

Why is the Euclidean Algorithm Correct?

- First, by the lemma, we have $r_{k-1} = gcd(a, b)$ because
- $gcd(a, b) = gcd(b, r_1) = gcd(r_1, r_2) = gcd(r_2, r_3) = \cdots = gcd(r_{k-1}, r_k) = gcd(r_{k-1}, 0) = r_{k-1}$.
- Finally, and importantly, the algorithm halts because $r_1 > r_2 > r_3 > \cdots$ is a strictly decreasing sequence of non-negative integers, which means that, eventually, $r_k = 0$ for some $k \geq 1$.
- In particular, the last non-zero remainder, namely r_{k-1}, is exactly $gcd(a, b)$.

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.

- **Proof by induction on n.**

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.

- **Proof by induction on n.**

- The statement $P(n)$ is exactly the statement of the lemma.

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.

- **Proof by induction on n.**

- The statement $P(n)$ is exactly the statement of the lemma.

- **Base Case.** $n = 1$. Since one recursion occurs we have $b > 0$. That is, $b \geq 1 = F_2$.

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.

- **Proof by induction on n.**

- The statement $P(n)$ is exactly the statement of the lemma.

- **Base Case.** $n = 1$. Since one recursion occurs we have $b > 0$. That is, $b \geq 1 = F_2$.

- By assumption $a > b \Rightarrow a \geq 2 = F_3$.

Lemma 31.10 in our text: number of recursive calls

- **Lemma 31.10 in CLR.** If $a, b \geq 1$ are integers and Euclid(a, b) performs $n \geq 1$ recursive calls, then $a \geq F_{n+2}$ and $b \geq F_{n+1}$, where F_i is the ith Fibonacci number.
- **Proof by induction on n.**
- The statement $P(n)$ is exactly the statement of the lemma.
- **Base Case.** $n = 1$. Since one recursion occurs we have $b > 0$. That is, $b \geq 1 = F_2$.
- By assumption $a > b \Rightarrow a \geq 2 = F_3$.
- Also, since $b > a \pmod{b}$ (why?) in each recursive call Euclid$(\hat{a}, \hat{b})$ we have $\hat{a} > \hat{b}$.

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).

- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).

- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).

- To that end, we have $k > 0$ and $b > 0$.

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).
- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).
- To that end, we have $k > 0$ and $b > 0$.
- Assume Euclid(a, b) makes k recursive calls.

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).
- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).
- To that end, we have $k > 0$ and $b > 0$.
- Assume $\text{Euclid}(a, b)$ makes k recursive calls.
- $\text{Euclid}(a, b)$ first returns $\text{Euclid}(b, a \pmod b)$, which makes $k-1$ recursive calls.

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).
- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).
- To that end, we have $k > 0$ and $b > 0$.
- Assume Euclid(a, b) makes k recursive calls.
- Euclid(a, b) first returns Euclid$(b, a \pmod{b})$, which makes $k-1$ recursive calls.
- By the induction hypothesis, we have $b \geq F_{k+1}$ and $a \pmod{b} \geq F_k$.

Proof of lemma 31.10, continued.

- **Induction Hypothesis.** Assume the lemma is true if $k-1$ recursive calls have made (i.e., assume $P(k-1)$ is true).
- **Inductive Step.** We must prove the lemma is true when k recursive calls have been made (i.e., verify $P(k)$ is true).
- To that end, we have $k > 0$ and $b > 0$.
- Assume Euclid(a, b) makes k recursive calls.
- Euclid(a, b) first returns Euclid$(b, a \pmod b)$, which makes $k-1$ recursive calls.
- By the induction hypothesis, we have $b \geq F_{k+1}$ and $a \pmod b \geq F_k$.
- It remains to show that $a \geq F_{k+2}$.

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Note that $b + a \pmod{b}$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Note that $b + a \pmod{b}$
- $= b + r$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Note that $b + a \pmod{b}$
- $= b + r$
- $= b + (a - \lfloor \frac{a}{b} \rfloor b)$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Note that $b + a \pmod{b}$
- $= b + r$
- $= b + (a - \lfloor \frac{a}{b} \rfloor b)$
- $\leq a$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Note that $b + a \pmod{b}$
- $= b + r$
- $= b + (a - \lfloor \frac{a}{b} \rfloor b)$
- $\leq a$
- because $\lfloor \frac{a}{b} \rfloor > 1$.

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Thus we have that $a \geq b + a \pmod{b}$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Thus we have that $a \geq b + a \pmod{b}$
- $\geq F_{k+1} + F_k$

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Thus we have that $a \geq b + a \pmod{b}$
- $\geq F_{k+1} + F_k$
- $= F_{k+2}$, as desired.

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Thus we have that $a \geq b + a \pmod{b}$
- $\geq F_{k+1} + F_k$
- $= F_{k+2}$, as desired.
- **Conclusion.** If Euclid(a, b) makes n recursive calls then $a \geq F_{n+2}$ and $b \geq F_{n+1}$.

Proof of lemma 31.10, continued. Must show $a \geq F_{k+2}$.

- Thus we have that $a \geq b + a \pmod{b}$
- $\geq F_{k+1} + F_k$
- $= F_{k+2}$, as desired.
- **Conclusion.** If Euclid(a, b) makes n recursive calls then $a \geq F_{n+2}$ and $b \geq F_{n+1}$.
- **QED**

Direct Consequence of Lemma 31.10

- **Theorem 31.11**. If $a > b \geq 1$ and $b < F_{n+1}$ then Euclid(a, b) makes at least $n - 1$ recursive calls.

Direct Consequence of Lemma 31.10

- **Theorem 31.11**. If $a > b \geq 1$ and $b < F_{n+1}$ then Euclid(a, b) makes at least $n - 1$ recursive calls.

- This theorem is best possible:

Direct Consequence of Lemma 31.10

- **Theorem 31.11**. If $a > b \geq 1$ and $b < F_{n+1}$ then Euclid(a, b) makes at least $n - 1$ recursive calls.
- This theorem is best possible:
- To see why, let $a = F_{n+2}$ and $b = F_{n+1}$ then Euclid(a, b) will perform as follows:

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.
- Then $gcd(F_{n+1}, F_n) = gcd(F_n, F_{n-1}) = \cdots = gcd(F_1, F_0) = gcd(1, 0) = 1$.

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.
- Then $gcd(F_{n+1}, F_n) = gcd(F_n, F_{n-1}) = \cdots = gcd(F_1, F_0) = gcd(1, 0) = 1$.
- Thus n recursive calls will be made $\Rightarrow$ the theorem is best possible.

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.
- Then $gcd(F_{n+1}, F_n) = gcd(F_n, F_{n-1}) = \cdots = gcd(F_1, F_0) = gcd(1, 0) = 1$.
- Thus n recursive calls will be made $\Rightarrow$ the theorem is best possible.
- Finally, given that $F_n \approx \frac{\phi^n}{5}$, where $\phi = \frac{1+\sqrt{5}}{2}$, we see that

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.
- Then $gcd(F_{n+1}, F_n) = gcd(F_n, F_{n-1}) = \cdots = gcd(F_1, F_0) = gcd(1, 0) = 1$.
- Thus n recursive calls will be made $\Rightarrow$ the theorem is best possible.
- Finally, given that $F_n \approx \frac{\phi^n}{5}$, where $\phi = \frac{1+\sqrt{5}}{2}$, we see that
- Euclid(a, b) makes C log b recursive calls.

Best possible theorem

- Recall that $a = F_{n+2}$ and $b = F_{n+1}$.
- Then $gcd(F_{n+1}, F_n) = gcd(F_n, F_{n-1}) = \cdots = gcd(F_1, F_0) = gcd(1, 0) = 1$.
- Thus n recursive calls will be made $\Rightarrow$ the theorem is best possible.
- Finally, given that $F_n \approx \frac{\phi^n}{5}$, where $\phi = \frac{1+\sqrt{5}}{2}$, we see that
- Euclid(a, b) makes C log b recursive calls.
- Moral of the story: recursion is not always bad!

What's up next week?

Asymptotics and the Master Theorem