9th Lecture

INDUCTIVE DEFINITIONS

(familiar from recursive definitions of functions—e.g. in Haskell)

Fibonacci Numbers: $F_0 = 0; F_1 = 1; F_n = F_{n-1} + F_{n-2}$

Obvious recursive algorithm (but if not carefully implemented horribly inefficient)

Exercise: Prove that the number of recursive call to $f(n)$ in the program

$$f(n): \quad \text{if } n < 2$$
$$\quad \quad \quad \text{if } n = 0 \text{ return 0 else return 1}$$
$$\quad \text{else}$$
$$\quad \quad A \leftarrow f(n-1)$$
$$\quad \quad B \leftarrow f(n-2)$$
$$\quad \text{return } A + B$$

if is $2 \cdot F_n + 1$

One can use memoization (google it!)

One can also use a 'bottom up' algorithm, computing $F_3 \cdot F_4 \cdots F_n$ at constant cost for each new number.

FORMULA We look for some $\alpha$ such that $F_k = \alpha^k$ [we know that $F_k$ grows exponentially fast]

Since we don't know the value of $\alpha$, call it $x$, so $F_n = x^n$

We should have $x^n = x^{n-1} + x^{n-2}$ (from the inductive definition)

$x^n - x^{n-1} - x^{n-2} = 0$

$x^{n-2} (x^2 - x - 1) = 0$

So $x$ can be: 0, $\Phi = \frac{1 + \sqrt{5}}{2}$, $\overline{\Phi} = \frac{1 - \sqrt{5}}{2}$

not interesting

Golden Ratio (google it!)

Unfortunately, for $n=0, n=1$

neither $\Phi^n$ nor $\overline{\Phi}^n$ equal $F_n$

Bummer

However
Consider the recurrence \( U(n) = U(n-1) + U(n-2) \) * 

Claim: If the sequences \( h(n), q(n) \) satisfy *, so does \( ah(n) + bq(n) \) for any numbers \( a, b \) (\( a, b \in \mathbb{R} \))

So we can try to find \( a, b \) s.t. \( aq^0 + bq^0 = 0 \) \( aq^1 + bq^1 = 1 \)

Then \( f(n) = aq^n + bq^n \) satisfy the definition of Fibonacci numbers.

It is easy to see that \( a = \frac{1}{\sqrt{5}}, b = \frac{-1}{\sqrt{5}} \) works.

So \( F_n = \frac{q^n}{\sqrt{5}} - \frac{\overline{q}^n}{\sqrt{5}} \) [This can be evaluated in \( \sim \log n \) operations!]

--- * ---

Application to Euclid's gcd algorithm:

Lame's thm: let \( a_n > a_{n-1} > \ldots > a_0 = 0 \) be the sequence of residues in the algorithm (initial values are \( a_n = a, a_{n-1} = b \), we're computing gcd\( (a, b) \) and \( a_{n-2} = a_k \mod a_{k-1} \))

Claim: \( a_i \geq F_i \)

proof: (induction on \( i \)) \( a_0 = 0 = F_0 \)
\( a_1 > a_0 \) so \( a_1 > 0 \) so \( a_1 > 1 = F_1 \)

I.H. is: \( a_i \geq F_i \) for \( i < k \) (and by hypothesis of Euclid \( a_i \geq F_i \))

then \( a_k = qa_{k-1} + a_{k-2} \) (for some \( q \in \mathbb{N}, q > 0 \))
(because \( a_k \equiv a_k \mod a_{k-1} \))

Since \( q \geq 1 \)
\( a_k \geq a_{k-1} + a_{k-2} \)
by IH \( a_{k-1} \geq F_{k-1}, a_{k-2} \geq F_{k-2} \)
so \( a_k \geq F_{k-1} + F_{k-2} = F_k \) \( \square \)
This can be used to prove that EUCLID is efficient.

**Thm** Let \( a > b \) be the inputs to EUCLID and let \( a \) require \( d \) decimal digits to write. Suppose the algorithm requires \( t \) executions of the while loop. Then \( t \leq Cd \) for some constant \( C > 0 \).

**proof** \( a = a_t \times 10^d \)

by Lame, \( a_t \geq F_t \) so \( F_t \leq a_t < 10^d \)

If we could say \( F_i \geq \phi^i \frac{1}{\sqrt{5}} \) we could argue:

\[
10^d \geq F_t \geq \frac{\phi^t}{\sqrt{5}}
\]

Taking logs (base 2)

\[
d \log 10 \geq t \log \phi - \log \sqrt{5}
\]

\[
d \geq t \left[ \frac{\log \phi}{\log 10} \right] - \log \sqrt{5}
\]

\[
d \geq At \quad [\text{why?}]
\]

for some \( A > 0 \).

This is what I sketched in class. Unfortunately only 1 student challenged me,

asking 'Isn't \( F_i = \frac{\phi^i}{\sqrt{5}} - \frac{\bar{\phi}^i}{\sqrt{5}} \)? What about the subtraction?!

It takes a bit of trickery to show that all we need is a smaller constant.

**Claim** \( |\bar{\phi}| \leq 2\phi \)

\[
\frac{1 - \sqrt{5}}{2} \geq \frac{\phi^2 - 1}{(\phi^2 + 1)^2} = \frac{4}{5 + 2\sqrt{5}} \geq \frac{1}{2}
\]

so \( \phi^i - \bar{\phi}^i \geq \phi^i \left( \frac{1}{2} \phi \right)^i \geq \frac{1}{2} \phi^i \) and \( F_t \geq \frac{1}{2} \phi^t \Rightarrow F_t > B \phi^t \)

Again, taking logs we get \( d \geq C t \) for some \( C \)

**Exercise:** Compute (a good bound on) \( C \).

**Corollary** The number of executions of the loop in EUCLID is bounded by a constant times \( d \).

**Corollary** EUCLID runs in time polynomial in the length of its input.