Announcements

- PS 1 due Mon 9/17 (1 week from today)

- Reading for today's lecture
  CLRS 2.1
  31.1, 31.2
Program Correctness

Recall our array Insertion Sort Algorithm:

**Insertion Sort** \( (A) \)

\[
\text{for } j = 2 \text{ to } n \\
\text{do } \text{Key} \leftarrow A[j] \\
\text{i} \leftarrow j - 1 \\
\text{while } i > 0 \text{ and } A[i] > \text{Key} \\
\text{do } A[i+1] \leftarrow A[i] \\
\text{i} \leftarrow i - 1 \\
A[i+1] \leftarrow \text{Key}
\]

How do we know it works? We prove it!

**Loop Invariant** \( \Rightarrow \) "Induction hypothesis" (property that is true at start of each iteration of loop)

at start of "for loop", \( A[1..j-1] \)

consists of elements originally in \( A[1..j-1] \) but in sorted order
Must show 3 things:

- **Base case**
  - Initialization: true prior to 1st iteration

- **Induction step**
  - Maintenance: If true before an iteration, then remains true before next iteration
  - Termination: When terminates, invariant gives a useful property (hopefully one that tells us the program is correct)

For **Insertion sort**: (just a sketch on this example)

- **Initialization**: Show loop invariant holds before 1st loop iteration
  - When j=2
    - Trivially sorted & element originally in $A[1]$ ✓

- **Maintenance**: (informal) Assume true for $j-1$
  - Show $A[j-1], A[j-2], ...$ right until "proper" position for $A[j]$ is found
    - (formally would analyze loop invariant for "while" loop)
Termination:

terminates after $j=n$ loop, i.e. when $j=n+1$

loop invariant $\Rightarrow A[1..n] (= \text{whole array } A)$

is in sorted order
Let's do another example:

What does this code do?

<table>
<thead>
<tr>
<th>S(m,n)</th>
<th>m,n are positive integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := m</td>
<td>[ A ]</td>
</tr>
<tr>
<td>y := n</td>
<td>[ B ]</td>
</tr>
<tr>
<td>while</td>
<td>x ≠ y</td>
</tr>
<tr>
<td>do</td>
<td>if x &lt; y</td>
</tr>
<tr>
<td></td>
<td>then y := y - x</td>
</tr>
<tr>
<td></td>
<td>else x := x - y</td>
</tr>
<tr>
<td>return</td>
<td>x</td>
</tr>
</tbody>
</table>

Does it even terminate?

Let's "break it up":

Part A: x := m, y := n

Part B: while loop & contents

What is it doing?

Surprise: final output \( x = \gcd(m,n) \)

so what is a good loop invariant?
good loop invariant: predicate based on \(xy, m, n\) 

1) true after "A" "Initialization"
2) remains true after each while loop execution "Maintenance"
3) useful: i.e., implies output = \(\gcd(m, n)\) "Termination"

How do we guess?

• dumb luck, inspiration, understand the program...
• requirements above imply:
  1) true after A \(\forall m, n\) ints
  2) \((x < y) \rightarrow true\) for \((m, n, x, y)\)
     implies true for \((m, n, x, y-x)\)

and
\((x \geq y) \rightarrow true\) for \((m, n, x, y)\)
implies true for \((m, n, x-y, y)\)

3) true for \((m, n, x)\)
implies \(x = \gcd(m, n)\)

missing link:
\[\gcd(x, y) = \gcd(x, y-x) = \gcd(x-y, y)\] (*)

So \(\gcd(x, y)\) remains the same after every iteration of the loop!

So how about following loop invariant:
\[x, y\] are positive ints \(\rightarrow\) \(\gcd(x, y) = \gcd(m, n)\)
Correctness Argument:

**Initialization:**

\( \exists m,n \) positive ints \( \Rightarrow \) after executing "A",

1. \( x,y \) positive ints
2. \( \gcd (x,y) = \gcd (m,n) \)

1st "Initialization requirement satisfied!"

**Maintenance:**

Assume \( \{ \)

1. \( x,y \) positive ints
2. \( \gcd (x,y) = \gcd (m,n) \)

\( \} \) after iteration \( i \)

Then after one more iteration:

( place subscripts to denote iteration \( i \) )

if \( x_i < y_i \)

then

\[ y_{i+1} = y_i - x_i \]  
\[ x_{i+1} = x_i \]  

\( \gcd (x_{i+1}, y_{i+1}) = \gcd (x_i, y_i - x_i) = \gcd (x_i, y_i - x_i) = \gcd (x_i, y_i) \)  
by \( \ast \)  
by definition of \( x_{i+1}, y_{i+1} \)

if \( x_i > y_i \)

similar to above

if \( x_i = y_i \)

nothing gets done in the loop

**Termination:** (of loop)

terminates when \( x_i = y_i \)

loop invariant \( \Rightarrow \) \( \gcd (x_i, y_i) = \gcd (m,n) \) \( \Rightarrow \) right answer!
Why does the program terminate?

(Note, different from "termination" condition of loop invariant) only says what happens if we terminate, but doesn't promise that we terminate.

$x, y$ positive ints

$x, y$ gets smaller at each iteration

$: \# \text{iterations} \leq m+n$

Run time is $O(m+n)$

↑

is that good or bad?

$b$ bit number is in $[2^{b-1}, 2^b - 1]$

$\Rightarrow$ (description) size of $n, m$ is $\log m + \log n$

E.g. if $n, m$ are 500 bits, need $O(2^{500})$ runtime

BAD
One more example

What does this do?

\[ S(m,n); \quad m, n \text{ are positive integers} \]

\[
\begin{align*}
\text{x} &\leq m \\
\text{y} &\geq n \\
\text{while} &\quad \text{y} \neq 0 \\
\text{do} &\quad r \leftarrow \text{x} \mod y \\
\text{x} &\leftarrow \text{y} \\
\text{y} &\leftarrow r \\
\text{return} &\quad \text{x}
\end{align*}
\]

\[ \left\{ \begin{array}{l}
\text{new (x, y)} \\
\text{is (y, x \mod y)}
\end{array} \right. \]

Same thing! Euclid's Algorithm

Loop invariant:
positive
\[ x, y \text{ ints} \quad + \gcd(x, y) = \gcd(m, n) \]

Initialization: \checkmark

Maintenance: Assume true after round \(i\)
\[ x_{i+1}, y_{i+1} \text{ positive ints} \checkmark \]
\[ \gcd(x_{i+1}, y_{i+1}) = \gcd(y_{i}, x_{i} \mod y_{i}) = \gcd(x_{i}, y_{i}) \]

Termination of loop
\[ \text{x=gcd}(x,0) = \gcd(m, n) \]

nice \ 6.042 \\
\text{fact} \\
"induction hypothesis"
Why does this program terminate?

- $y$ is non-negative
- $y$ strictly decreases at each iteration

Running time is $O(\max(m,n))$!

But wait...

Can do better:

Claim: $x > y \geq 1$

Algorithms performs at least $k$ loop iterations

$\Rightarrow x \geq F_{k+2} \quad y \geq F_{k+1}$

Why? If perform 1 loop iteration:

- $y \geq 1$ (else stop right away)
- $y = F_2$
- $x \geq y \geq 2 = F_3$
if true for \( k-1 \) loop iterations; assume \((x,y) \) needs \( i \) item

After 1st time through loop \( (y, x \mod y) \) must be

such that \( y \leq F_{k+1} \)

\( x \mod y \geq F_k \) from induction hypothesis

So \( x \geq y + x \mod y \) happens to be true since \( x \mod y = x - c \cdot y \) for some \( c > 0 \) for \( y \leq F_k \)

\[ \begin{align*}
F_k & \geq F_{k+1} + F_k \\
& = F_{k+2}
\end{align*} \]

Why is this interesting?

Corollary (Lamb Thm) if \( x > y \geq 1 \)

\[ \begin{align*}
y & \leq F_{k+1} \\
\text{then fewer than } k \text{ loops executed}
\end{align*} \]

Since \( F_k \sim \frac{(1 + \sqrt{5})^k}{2} \)

+ \# loops \leq 1 + \log_\phi y \text{ i.e. } \Theta(\log(y)) \text{ (exercise 31.2-5)}

we get \( \Theta(\log(\text{xy})) \) time!

\[ \text{exponentially better!} \]