Algorithm Design: GCD

- Problem solution through refinement
  - GCD Example of use of loops
  - Arguing the complexity of an algorithm
  - Greek mathematics achievement: Euclid’s Algorithm
How to find the GCD of 2 ints?

• Greatest common divisor
  – Given two numbers: small, large
  – Write a class method to find their GCD

• Example of refining a solution procedure for a problem until you get it RIGHT! (cheap and elegant)

• First approach: try largest possible guess and try it; if doesn’t work, decrement guess and try again. repeat.
GCD - Algorithm 1

```java
public static int GCD(int small, int large) {
    int div;
    for (div = large; div > 0; div--) {
        if (((large % div) == 0) &&
            ((small % div) == 0)) return div;
    }
}
```

How many checks do we do in the loop in the worst case? at most large if checks

But largest common factor can’t be larger than small!
GCD - Algorithm 2

public static int GCD(int small, int large) {
    int div;
    for (div = small; div > 0; div--) {
        if ( ((large % div) == 0) &&
            ((small % div) == 0) ) return div;
    }
}

Do in worst case small if checks.
Is there a better way to do this?
How GCD works?

• Take 15 and 18 in algorithm 1
  – when div is 15 - 15%15==0? yes, 18%15==0? no, 14 (no,no), 13 - (no,no)
  – 12, 11, 10, 9, 8, 7, 6, 5, 4 all fail to produce (yes,yes)
  – 3 succeeds

• Should only test the following (divisor,quotient) pairs: (1, 15), (2, 7.5), (3, 5)
  – Then (4, 3.75) will have been already tested
  – So when div > quotient means you have already checked this divisor, quotient pair if they are both integers
GCD - Algorithm 3

public static int GCD(int small, int large) {
    int div = 0, best = 1, quo;
    loopLabel: while (true) {
        div++;
        quo = small / div;
        if (quo < div) break loopLabel;
        if ( ((large % quo) == 0) && ((small % quo) == 0) ) return quo;
        if ( ((large % div) == 0) && ((small % div) == 0) ) best = div;
    }
    return best;
}
**Algorithm 3 - trace**

small = 15, large = 18

<table>
<thead>
<tr>
<th>div</th>
<th>quo</th>
<th>%quo</th>
<th>%div</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>false</td>
<td>true</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>false</td>
<td>false</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>false</td>
<td>true</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

loop exits and returns 3.
Performance of Algm 3

• How many if checks? $2*\sqrt{\text{small}}$

• Can we use another form of loop for this code?
  – Want no redundant statements or messy control flow
Algorithm 3 - with While

int div=1, best=1, quo=small/div;

loopLabel: while (div<quo){
    if ( ((large%quo) == 0) &&
         ((small%quo) == 0) ) return quo;
    if ( ((large%dv) == 0) &&
         ((small%dv) == 0) ) best = div;
    div++;
    quo = small/div;
}

return best;
Algorithm 3 -with For

```
int best = 1, quo;
f1: for (int div = 1; true ; div++){
    quo = small/div;
    if (quo < div) break f1;
    ....
}
```

seems to add no code, but stopping condition harder to understand with check being true
Algorithm 3 - with Do-while

```c
int div=1, quo=small/div;
dol: do{
    if ( ((large%quo...
        ...
        div++;
        quo = small/div;
    }
    while (div < quo);
```
Which loop to use?

- Most straightforward to understand code
- Can make them all work, but why?
- Redundant code or obscured control flow are not desirable
- Generalized loop construct is what was used in first code for algorithm
General Loop Structure

public static int GCD(int small, int large) {
    int div = 0, best = 1, quo;
    loopLabel: while (true) {
        div++;
        quo = small / div;
        if (quo < div) break loopLabel;
        if ( ((large % quo) == 0) &&
             ((small % quo) == 0) ) return quo;
        if ( ((large % div) == 0) &&
             ((small % div) == 0) ) best = div;
    }
    return best;
}
Bishop’s Method for GCD

• If large and small are both multiples of $k$, then large - small is a multiple of $k$
  – Note: large-small is smaller than large, so we have reduced the problem to one easier to solve
  – Need greatest multiple of large - small and small.

• Why? $lg = k \times n$; $sm = k \times m$; $So
g - sm = k \times (n-m) = k \times diff$
Algorithm 4 - Bishop

public int static GCD (int small, int large) {
    int smsave = small, lgsave = large;
    while (small != large) {
        if (large > small) large = large - small;
        else {
            int tmp = small; // swap large and small
            small = large;  // small
            large = tmp;
        }
    }
    return sm;
}  // save of original arguments on entry is not necessary
How does this work?

• *large* is reduced by successive subtraction (i.e., division) until it is smaller than *small*
• *small* and *large* are then swapped
• Continues until *large* == *small*
  – then *large* is the GCD (it can be 1)
• Bishop’s program is a variant of Algorithm 4
Algorithm 4 Trace

small - 6, large - 21

large   small
21      6
15      6
9       6
3       6
6       3
3       3

3 is returned

How long does it take?

at least large/small steps

Which is better,

large/small or sqrt(small)?

Neither.

large - 2000, small - 2
large - 2000, small - 1000
Euclid’s Method

• Recognize that repeated subtraction is division

• Exchange small and large when large is replaced by large%small

• Greek mathematician Euclid discovered this algorithm around 300 B.C.
  – Father of Geometry

• Moral: in mathematics we can’t ignore the past
public static int GCD(int small, int large){
    while ((large%small) != 0){
        int tmp = large%small;
        large = small;
        small = tmp;
    }
    return small;
}

small = 6, large = 28

    large    small    tmp
28        6        4
6        4        2
4        2

2 is returned
Algm 6: Another Formulation

in class A define:

```java
public static int GCD(int small, int large) {
    int rem = large % small;
    if (rem == 0) return small;
    return A.GCD(rem, small);
}
```

small-6, large-28

Bert: What is GCD of 28, 6?  ask Ernie 6,4
Ernie: What is GCD of 6,4?  ask Elmo 4,2
Elmo: What is GCD of 4, 2?  It’s 2!!
Algorithm 6

- Problem decomposition in terms of function calls
- Particular usage called *recursion*
- Succinct statement of solution of one problem in terms of another reduced problem