```
public int Mystery(int m, int n){
      if(m<n){
            int temp = m;
            m=n;
            n=temp;
      }
      while(n != 0) {
            int rem = m % n;
            m=n;
            n=rem;
      }
      return m;
}
```

Finding the inverse…

gcd(26,7) = gcd(7, 26%7) = gcd(7,5)
= gcd(5, 7%5) = gcd(5,2)
= gcd(2, 5%2) = gcd(2, 1)
= gcd(1, 2%1) = gcd(1,0)= 1.

$26 = 3 \cdot 7 + 5$; $5 = 26 - 3 \cdot 7$

$7 = 5 \cdot 1 + 2$; $2 = 7 - 5 \cdot 1$

$5 = 2 \cdot 2 + 1$; $1 = 5 - 2 \cdot 2$

An application of all of this modular arithmetic

Amazon chooses random 512-bit (or 1024-bit) prime numbers p, q and an exponent e (often about 60,000).

Amazon calculates $\mathrm{n} = pq$. They tell your computer (n, e) (not p, q)

You want to send Amazon your credit card number a.

You compute $C = a^e \% n$ and send Amazon C.

Amazon computes d, the multiplicative inverse of e $(mod\ [p-1][q-1])$

Amazon finds $C^d \% n$

Fact: $a = C^d \% n$ as long as $0 < a < n$ and $p \nmid a$ and $q \nmid a$

Fast Exponentiation Algorithm

What if e isn't exactly a power of 2?

Step 1: Write e in binary.

Step 2: Find $a^c \% n$ for c every power of 2 up to e.

Step 3: calculate a^e by multiplying a^c for all c where binary expansion of e had a 1.