Programming Problem 2: Primality testing

FEB 1 2013

Our second programming problem

Primality Testing

Given a positive integer (> 1), determine whether it is a prime number or not.

Examples:

Input	Output
31	prime
2001	composite
987654321	composite

Why do we care?

- Our digital life depends on the *security* of information that we send over the internet.
- Security of information is made possible by *encryption* methods.
- One of the most well known encryption methods is the *RSA algorithm* (R = Ron Rivest, S = Adi Shamir, and A = Leonard Adleman).
- The first step of this algorithm is to find two *large* primes p and q and compute their product n = p*q.
- “Large” here could mean 300 digits or so.

Algorithmic Idea

- Generate all “candidate” factors of n, namely

  2, 3, …, n-1
- For each generated “candidate” factor, check if n is evenly divisible by the factor (i.e., the remainder is 0).
- If a “candidate” factor is found to be a real factor, then n is composite.
- If no “candidate” factor is found to be a real factor, then n is a prime.

Algorithm in pseudocode

```
Input n
For each factor = 2, 3, …, n-1 do the following
    if n is evenly divisible by factor then
        remember that n is a composite
If we have detected that  n is a composite
    output that n is a composite
Otherwise output that n is a prime
```

Python code (Version 1)

```
number = int(raw_input("Enter a positive integer: "))

factor = 2
isPrime = True
while(factor <= number  - 1):
        if(number % factor == 0):
                isPrime = False
        factor = factor + 1

if(isPrime):
        print number, "is prime“
else:
        print number, “is composite”
```