Testing Primality

Tom James

May 4, 2012
## Contents

I Introduction 2

1 Prerequisite Skills 2
   1.1 Math material prerequisites ........................................ 2
   1.2 Calculator programming prerequisites .............................. 2

II Student Version 3

2 Introduction 3

3 Testing Primality 3
   3.1 Assignment .......................................................... 3
   3.2 Exercises ............................................................ 3

III Teacher Version 5

4 Trial division algorithm 5

5 Sample solution 5
   5.1 Sample executions .................................................... 5
   5.2 Error-proofing ........................................................ 5

6 Anticipated trouble spots 6

7 Extensions 6
   7.1 Listing primes ........................................................ 6
   7.2 Improving efficiency ............................................... 6
Part I

Introduction

This module will facilitate student understanding of how prime numbers are identified by guiding students through the process of implementing a simple primality testing algorithm on their graphing calculators using the programming language TI-BASIC.

Gaining firsthand experience in figuring out what is involved in deciding whether a number is prime or not lays the groundwork for students to begin to understand the increasingly relevant problems posed by cryptography.

1 Prerequisite Skills

This module would be appropriate for students in an algebra class at any year and could be adapted to suit the needs of younger or older students.

1.1 Math material prerequisites

Primes and composite numbers Students have heard of the difference between prime and composite numbers. They are able to given examples of each and justify their examples.

Prime factorization Students understand that every natural number can be written as a product of powers of primes. Given a natural number, students are able to determine its unique prime factorization.

1.2 Calculator programming prerequisites

Basic syntax Students recognize the need to close any structure that is opened, whether it is an open quote, open parenthesis, or beginning of a conditional statement.

Use of variables Students are able to assign, reference, and copy the value of a variable. They are able to create a counter variable that is incremented by 1 after each pass through a loop.

Conditional statements Students are able to construct an “If – Else If – Else” conditional statement.
Part II
Student Version

2 Introduction

This module is designed for you to explore a simple question: how can we tell whether a given number $n$ is prime or composite? Our goal will be to build a program that answers this question.

3 Testing Primality

Let’s begin with an example. Is the number 13 prime? How did you know?

Here is the basic algorithm we’ll use:

1. Suppose $n$ is the number we’d like to test.
2. Let $a = 2$ be our “tester” variable we’ll try to divide $n$ by to see if it’s a factor.
3. If $\frac{n}{a}$ is a whole number, then $a$ went into $n$ evenly!
4. Otherwise, $a$ doesn’t evenly divide $n$. Increase $a$ by 1 and try the previous step again.
5. At some point, if we haven’t found an even divisor, then we can safely declare $n$ to be prime.

3.1 Assignment

Write a program that takes as input a natural number $n$, and outputs either “Prime” or “Composite” and the factor that evenly divides $n$. How many numbers do we have to try to divide $n$ by before we know it must be prime?

Here are a couple commands you will find useful:

**Fractional part** If you want to know what the fractional part of a number is, use $\text{MATH NUM 4: fPart}$. So if $x = 3.5$, then $\text{fPart}(x) = 0.5$.

**Stop** If the program ought to finish its execution and not continue anymore, use $\text{PRGM CTL F: Stop}$

3.2 Exercises

1. Using your program, determine whether the following numbers are prime or composite:
<table>
<thead>
<tr>
<th>$n$</th>
<th>prime or composite?</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>377</td>
<td></td>
</tr>
<tr>
<td>1759</td>
<td></td>
</tr>
<tr>
<td>13081</td>
<td></td>
</tr>
</tbody>
</table>

2. Find two large prime numbers $p$ and $q$. See if your program can identify those factors given an input of $N = p \cdot q$. 
4 Trial division algorithm

This algorithm is the simplest and also most practical for handheld computing devices like graphing calculators without much RAM.

We only need to try to divide $n$ by

5 Sample solution

Precondition: $x \in \mathbb{N}$

```
1 Input '"x="',X
2 2->A
3 While A<sqrt(X)
4 If fPart (X/A)=0
5 Then
6 Disp '"Composite "'
7 Disp '"Factor: "',A
8 Stop
9 End
10 A+1->A
11 End
12 Disp '"Prime "'
```

5.1 Sample executions

```
1 x=31201
2 Composite
3 Factor:
4  41
5 Done

1 x=1777
2 Prime
3 Done
```

5.2 Error-proofing

As an extension, students may take it upon themselves to write in a few conditional checks to guarantee that the precondition that the input is a natural number is met.

For example, this code snippet, inserted after the program accepts input, checks to see if the input is both greater than 1 and an integer:
6 Anticipated trouble spots

- Students may need a reminder on how to insert a blank line of code in the middle of their program using `2nd DEL ENTER`
- Students may need help with figuring out how to incorporate the fPart operator into their primality tester. They may need to see examples to realize formally that the fractional part of a quotient is 0 if and only if the divisor goes into the dividend evenly.
- Students may need examples to see that it’s only necessary to check the first \( \sqrt{n} \) factors of \( n \). We can show that if \( p \) is a factor of \( n \), then there’s a number \( q \) such that \( n = pq \). But if \( q \) were smaller than \( p \), then we’d have detected \( q \) as a factor of \( n \) before we got to \( p \). And since \( n = \sqrt{n} \cdot \sqrt{n} = pq \), then neither \( p \) nor \( q \) needs to exceed \( \sqrt{n} \).

7 Extensions

7.1 Listing primes

Students familiar with loops could easily adapt this program to iterate through the first \( n \) natural numbers, displaying which of them are prime.

7.2 Improving efficiency

We saw that the naive approach was to try to divide \( n \) by \( 2 \ldots n - 1 \). We were able to improve this by realizing that we only had to test \( 2 \ldots \lceil \sqrt{n} \rceil \).

Another improvement we could make would be to eliminate our duplicated effort in dividing by composite numbers like 4, 6, 8, \ldots by choosing only to try dividing by prime numbers. Trying to divide \( n \) by an even number greater than 2, for example, is not necessary because if 2 didn’t divide \( n \), then any other even number certainly wouldn’t either.