Profiling the Eratosthenes' Prime Sieve

Out: 8/23
Due: 9/5 by 11:59 AM

Learning Objectives

- ⊙ Setting up your Programming Environment and Tools
- ⊙ Review of the Use of Control Structures
- ⊙ Empirical Analysis of The Prime Sieve of Eratosthenes Algorithm
- ⊙ Familiarization with the Course Programming Conventions

The purpose of the first programming exercise is to familiarize you with the programming standards and coding conventions that are required for this course. All students taking this course are expected to have a certain level of proficiency which includes the ability to read and understand code written by others. Students are also expected to write code that efficiently implements and tests various algorithms using programming constructs studied in your programming classes and methods whose syntaxes and usage information you can find in the online Java API documentation.

In this exercise and all subsequent programming projects, you will be expected to properly document your code using the coding conventions for this course, program in multiple files so that there is a wall between the public interface and implementation of your algorithms as well as the application that uses the implementation. This assignment will test your ability to follow this approach to programming no matter what IDE (integrated development environment) that you use to write your code. Your code will be tested using Java™Platform, Standard Edition 8. Your program will be considered acceptable for grading only if it compiles without any syntax errors. You will generally submit only your source code, the .java files, in a zip archive file via a drop box on Moodle for grading.

Definition 1. **Empirical algorithmics** is a computer science area of specialization that involves the use of experimental and statistical, rather than theoretical, methods to analyze the behavior of algorithms.

The goals of empirical algorithmics are characterization of an algorithm based on its performance, enhancing the performance of an algorithm using empirical techniques and the comparative analysis of the various algorithms that solve the same problem within a given context.

Definition 2. A **prime number** (or prime integer, often simply called a"prime" for short) is a positive integer $p > 1$ that has no positive integer divisors other than 1 and p itself. More concisely, a prime number p is a positive integer having exactly one positive divisor other than 1, meaning it is a number that cannot be factored.

Definition 3. The **Sieve of Eratosthenes** is an ancient algorithm for generating a sequence of primes. First, integers from 2 to the highest number n you wish to include written in the list. You then remove all numbers greater than 2 which are divisible by 2. Next, find the smallest number greater than 2. The number is 3 so you cross out all numbers greater than 3 which are divisible by 3. You then find the smallest number greater than 3. The number is 5. You cross out all numbers greater than 5 which are divisible by 5. This process continues until all numbers greater than or equal to $\lfloor\sqrt{n}\rfloor$ are removed from the list. The numbers that are removed are composite numbers and those remaining a primes. The sieve of Eratosthenes can be used to compute the prime counting function.

In theory $t(n) \propto n\log(\log(n))$, where $t(n)$ denotes that time that it takes to generate primes in $[2, n]$. Therefore, $t(n) \approxeq kn\log(\log(n))$, for some real number k.

Definition 4. The **prime counting function** is the function denoted $\pi(x)$ that gives the number of primes less than or equal to the number x. For example, there are no primes ≤ 1, so $\pi(1) = \pi(-3) = 0$. More generally, $pi(x) = 0$, where $x \leq 1$. The only prime ≤ 2 is 2, so $\pi(2) = 1$. There are four one-digit primes, 2, 3, 5 and 7, so $pi(9) = 4$. One consequence of the Prime Number Theorem is that $\pi(n)$ is approximately $\frac{n}{\ln n}$.

Here is the pseudocode for a primality testing algorithm:

Algorithm: $ERATOSTHENES - SIEVE(n)$
Data: Input: n - an integer
Output: primes in [2,n]
if $n < 2$ **then**
  primes := {} ;
else
  isPrime[0:n] := true ;
  primes := {} ;
  for $i := 2{:}\lfloor\sqrt{n}\rfloor$ *stepsize := 1* **do**
    if *isPrime[i] = true* **then**
      for $j := i^2{:}n$ *stepsize* $:= i$ **do**
        isPrime[j] := false ;
      end
    end
  end
  for *i := 2:n stepsize := 1* **do**
    if *isPrime[i] = true* **then**
      primes := append(primes,i) ;
    end
  end
end
ERATOSTHENES-SIEVE(n) = primes

Algorithm 1: Generates a List of Primes in $[2, n]$

The EratosthenesUtil Class

Complete the implementation of *EratosthenesUtil.java*. The class in this file contains two public static methods. Do not implement any additional methods in this file. Write code only where indicated in this file.

The EratosthenesProfiler Program

The EratosthenesProfiler.java file will consist of only one method, the *main*. The main method will perform the following tasks:

1. It prompts the user to enter an integer (long), n. It invokes the relevant methods from the *EratosthenesUtil* class to generate and display the list of all primes in $[2, n]$. It also displays the time, in microseconds, taken to generate the list.

2. For n = $\{10000, 20000, 30000, \cdots, 150000\}$, your program will generate primes in $[2, n]$ and determine the time is μs used to generate the primes, display the exact value of $\pi(n)$, the approximate value of $\pi(n)$ denoted $\widetilde{\pi(n)}$, equal to $\frac{n}{\ln(n)}$, and the percentage error in the approximation. To determine the percentage error in the approximation of $\pi(n)$, use the expression $\%\Delta(\pi(n)) = \frac{\widetilde{\pi(n)} - \pi(n)}{\pi(n)} \times 100$. Display these results as shown in the table in the sample run.

.

Additional Requirements

Test the program to ensure that it works correctly and generates its output in the same format as shown in the sample run. Each file should have the following javadocs:

```
/**
* EXPLAIN THE PURPOSE OF THIS FIlE
* CSC 3102 Programming Project # 0
* @author YOUR NAME
* @see FILES REFERENCED (only in EratosthenesProfiler)
* <pre>
* Date: LAST DATE MODIFIED
* Course: CSC 3102 Section 1
* Instructor: Dr. Duncan
* Project: 0
* </pre>
*/
```

Also, submit an excel spreadsheet, *eratosthenestimes.xls*, containing the data generated by your program and two line graphs. The first line graph contains the recorded execution times and the theorectical times ($kn\log(\log(n))$) to generate each prime sequence in μs (on the Y-axis) versus n (X-axis).

For the first line graphs, enter a value for k for which the actual time recorded and the theorectical time overlap as much as possible. Finding the best fit for k will require some trial and error. The second line graph contains $\pi(n)$ and $\widetilde{\pi(n)}$ (Y-axis) versus n (X-axis). Enter data only in the yellow coded cells. Locate the source files, *EratosthenesProfiler.java* and *EratosthenesUtil.java* and enclose them along with the spreadsheet in a zip file. Name the zip file *YOURPAWSID_proj0.zip*, and submit your project for grading using the digital drop box on Moodle.

Sample Run:

```
Enter an integer n to generate primes in [2,n] -> 25
P(25) = {2, 3, 5, 7, 11, 13, 17, 19, 23}
Time to Generate the Primes: 988 microseconds
pi(25) = 9
```

n	Time(us)	pi(n)	n/ln(n)	%Error in pi(n)
10000	:	:	1085.7	:
20000	:	:	:	:
30000	:	:	:	:
40000	:	:	:	:
50000	:	:	:	:
60000	:	:	:	:
70000	:	:	:	:
80000	:	:	7086.1	:
90000	:	:	:	:
100000	:	:	:	:
110000	:	:	:	:
120000	:	:	:	:
130000	:	:	:	:
140000	:	:	:	:
150000	:	:	:	: