# Department of Computing

**CS 213: Advanced Programming**

**Class: BSCS 5 AB**

# Lab 5: Sieve of Eratosthenes

**Date: October 19th, 2017**

**Time: Thursday (10:00-12:50 & 14:00 – 16:50)**

# Instructor: Fahad Ahmed Satti

# 

# Lab 5: Sieve of Eratosthenes

## Introduction

In this lab the students have to design, develop and test a Scala based application, using functional programming paradigm, to calculate prime numbers between 1 to a given large n, using the Sieve of Eratosthenes, method.

## Objectives

After completion of this lab, the students will get better familiarization with Functional Programming and Scala.

## Tools/Software Requirement

* Solutions should be made using Scala.
* You can take help from internet but remember **no plagiarism.**

**Description**

A prime number is a natural number that has exactly two distinct natural number divisors: 1 and itself.

In mathematics, the sieve of Eratosthenes, is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2. The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

* Create a list of consecutive integers from 2 through n: (2, 3, 4, ..., n).
* Initially, let p equal 2, the smallest prime number.
* Enumerate the multiples of p by counting to n from 2p in increments of p, and mark them in the list (these will be 2p, 3p, 4p, ...; the p itself should not be marked).
* Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.

When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.

The main idea here is that every value given to p will be prime, because we have already marked all the multiples of the numbers less than p. Note that some of the numbers being marked may have already been marked earlier (e.g., 15 will be marked both for 3 and 5).

As a refinement, it is sufficient to mark the numbers in step 3 starting from p2, as all the smaller multiples of p will have already been marked at that point. This means that the algorithm is allowed to terminate in step 4 when p2 is greater than n.

### Example

To find all the prime numbers less than or equal to 30, proceed as follows.

First generate a list of integers from 2 to 30:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30.

The first number in the list is 2; cross out every 2nd number in the list after 2 by counting up from 2 in increments of 2 (these will be all the multiples of 2 in the list):

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30.

The next number in the list after 2 is 3; cross out every 3rd number in the list after 3 by counting up from 3 in increments of 3 (these will be all the multiples of 3 in the list):

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30.

The next number not yet crossed out in the list after 3 is 5; cross out every 5th number in the list after 5 by counting up from 5 in increments of 5 (i.e. all the multiples of 5):

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30.

The next number not yet crossed out in the list after 5 is 7; the next step would be to cross out every 7th number in the list after 7, but they all have been crossed out at this point, as these numbers (14, 21, 28) are also multiples of smaller primes because 7 × 7 is greater than 30. The numbers not crossed out in the list at this point are all the prime numbers below 30:

2 3 5 7 11 13 17 19 23 29.

This algorithm produces all primes not greater than n. It includes a common optimization, which is to start enumerating the multiples of each prime i from i2. The time complexity of this algorithm is O(n log log n).

**Lab Task**

* Use Functional Programming.
* Develop a program using Scala to calculate prime numbers using the Sieve of Eratosthenes method.
* Prime numbers should be calculated between 1 and a large number (provided by the user).
* Write unit tests to check the correctness of your program.
* Profile the memory and CPU utilization of your program for prime numbers generation upto 1000, 10000 and 10,00,000.

## Deliverables

* Each submission is individual with the following composition:
  + Source Code
  + Unit Tests
  + Documentation(Introduction, Approach, Design, How to Run and Analysis)
  + Link to the public repo on GitHub
* Convert your submission files into a zip folder and name it as given below, finally upload the zip folder to LMS.
  + Name – Registration No. – Section

## Grade Criteria

This lab will be graded on the following rubric: 