# CIS*3190 – Assignment 4 Sieve of Eratosthenes Reflection Report

## Implementing C:

My experience using C to implement the Sieve of Eratosthenes algorithm was pleasant as expected. Considering the fact that C has been the language I have used the most out of any other language, it is not a surprise that I was comfortable using the data structures, syntax and system time functions. The benefits of a language like C in my mind compared to the non-C languages we have covered is easy readability with the syntax. This could just be a symptom of extensive experience with the language, but I have grown comfortable with things like semi-colons at the end of lines, loops with "()" for the condition and "{}" for the loop. I find it easy to separate the code and identify what function goes with what. Another useful feature in C is the use of block comments. Although small, it seems almost intuitive to have this option in a programming language in my mind as it looks easier to read and it is easier for the programmer to make function and program headings. The system clock is very easy to use in C as you simply need to call "clock()" to get to current time and there is a built in struct called "clock_t" for you to use to analyze the return. Furthermore, C has very good data structures and functions that made it easy to program this algorithm from the Boolean array, to the memset() function that allowed me to initialize the whole array at once. To speak to the limitations of C in comparison to Ada, Fortran, Python and Cobol (The other languages I used), I would say that C not having dynamic typing is a limitation in my mind. Besides this, I found C to be very useful and easy to use for implementing this algorithm.

## Implementing Python:

Python was in my opinion, the easiest language to use to implement this algorithm. Some of the reasons I say this is because it was very easy to initialize the array to have the values I needed. Much like in C when I used the "memset()" function, Python allowed me to set the first two values to "False" and the rest to "True" when I was initializing the array. Another reason I found python easy to implement was because of the dynamic typing. This meant I didn't have to worry about the type the system time was returning or anything along those lines. Speaking of the system time, Python also has a very useful function called "time.time()" that returns the system time down to even more specific than a nanosecond and I found that extremely useful. The file I/O in Python was straight forward as expected and it resembled C in many ways. Lastly, Python makes it very easy to read user input was the "input()" function. On the negative side, Python is an extremely inefficient language because it is interpreted at runtime and not compiled beforehand. In my case, I wasn't very concerned with the efficiency of the algorithm but I can see why people wouldn't want to use Python if they were looking for an implementation that is easy as well as efficient.

## Implementing Fortran:

The Sieve of Eratosthenes was easy to implement in Fortran. Much like C, I used a simply 3 loop structure which was composed of do loops instead of "for" loops. I would consider it a limitation that you are not able to use "for" loops in Fortran because although it is very similar to the do loops in usability, it doesn't make as much sense to me to have do. I associate do with a command and "for" for a condition. Fortran also has some different syntax choices that I would not recommend using and I wasn't a fan of, such as the use of colons in variable declarations. Regardless, Fortran was and still is my favourite language that we learned in this course for many reasons. Its data structures are simple and its usability, as far as built in functions, is as good as C if not better in my mind. It's use indentation rather than brackets which was different for me but I found it nice (Definitely something I wouldn't mind getting used to). Its CPU time function (CPU_Time()) was very easy to implement and took in a real and updated that. This allowed for simple and specific measurements of the run time in milliseconds. Lastly, the file I/O in Fortran is very basic and once I saw one example online I found it very easy to implement the read and writes to files throughout the course.

## Implementing Ada:

Of the languages I am writing about in this report, Ada is my 2nd least favourite (In front of Cobol). This is most likely because of the unfamiliar syntax and structure. This starts with things like ":=" instead of "=" and includes other things such as "with ____ use _____". The structure of the language, specifically when it wants us to initialize all variables and functions at the top of each function they are used in makes it a bit harder to read in my opinion, especially when there are multiple embedded functions. Another thing that made Ada a bit harder in my opinion was the distinction between bounded and unbounded strings. I found myself asking why they even use strings that aren't unbounded. On the positive side, Ada is a relatively fast language (Compared to python and Cobol) that contained all the data types I needed to complete this algorithm such as Arrays, strings and integers. Ada's built in system time function "Clock" was very simple and easy to implement and there is a data type in Ada called "Time" that works with Clock to indicate the run time of the program. Lastly, the loops and conditional statements were easy to use and intuitive because of their similarity to many other languages.

## Implementing Cobol:

Cobol is my least favourite language that I used in this course as I expected it would be. The main thing that gave me trouble in Cobol was getting used to using the sections and declaring variables and records in the right place and correctly. Cobol ending up being roughly twice as long as most of the other programs I wrote and the declaration of variables was the main reason for this. In regards to timing the run time of the Cobol program, after researching different ways to do it, I came across a solution that returned me the seconds of run time but not any more specific than that. Then, I found that the built-in function "current-date" that returned me a string of the date and time, down to the millisecond. I used this when testing for

the run time of the Cobol algorithm. The only limitation to this method is that the decimal stops at milliseconds so it rounds to that and is not more specific like the built-in system times in other languages. Another difficulty I had with Cobol is understanding how they meant for us to use the "paragraph" syntax and when to put "." at the end of statements or not. All this being said, it wasn't all bad things. A few good features that the newer version of Cobol contained were that it used a very simple "display" function that printed to standard out as well as loops that were very similar to Ada and Fortran in my opinion.

## Which language had the best usability?

(e.g. was easiest to program, had the most appropriate structures, is easy to maintain).

In my opinion, the language that had the best usability (Without regards to efficiency) of the five I used for this assignment is python. Python used the least amount of lines to get the job done because of multiple reasons, starting with the syntax and not having brackets for loops, to having easy to use functions like "time.time()" built in to get the system time and "input()" to get user input. The main reason I found python the easiest to implement is because of the dynamic typing. Of the 5 languages I used to implement this algorithm, python was the only one that did not need you to specify a type for its variables. This allowed me to work even fast and implement this algorithm without much thinking. Furthermore, much like C, python had appropriate structures for writing to a file that made it very easy and effortless to implement.

## Run Times

| Language | 100,000 Limit | 1,000,000 Limit |
|---|---|---|
| **C** | 0.832ms | 7.880ms |
| **Python** | 26.761ms | 183.830ms |
| **Fortran** | 1.094ms | 14.077ms |
| **Ada** | 1.246ms | 9.339ms |
| **Cobol** | 5ms | 48ms |

## Was there any difference in efficiency?

(Show using various test data, in table form)

There were many differences in the efficiencies of the languages I used. First and foremost, I noticed a much longer run time for python than any of the other languages. This instantly made sense to me because python is the only language of the bunch that is interpreted at run time. C, Ada, Fortran and Cobol all must be compiled beforehand and converted into machine language so then they can be run. Additionally, there was a slight run time difference between the other programming languages. The fastest being C and then Fortran, Ada and last was Cobol. I suspect that Cobol simply because it is the oldest of the 4 languages and C is the fastest because it is the most maintained, modern and regularly updated language.