CS 317 - Fall 2019 - Lab Project 2

In this assignment, we will write and run a simple algorithm. The algorithm is designed to "Find all of the perfect numbers". We will find that the initial algorithm is very slow and inefficient. We will attempt several "improvements" on the initial algorithm. For each improvement, we will determine the degree of improvement realized. Our goal here is to explore a variety of approaches which (hopefully) increase the time efficiency of our algorithm. For each of several approaches, we will update our algorithm to implement the improvement, then run the program. We will then decide if there was noticeable time improvement.

As discussed in class, a perfect number, n, is one for which the sum of all proper divisors of n is equal to n. For example:

n	**Proper Divisors**	**Sum of Proper Divisors**
6	$1, 2, 3$	6
28	$1, 2, 4, 7, 14$	28
496	$1, 2, 4, 8, 16, 31, 62, 124, 248$	496
8128	$1, 2, 4, 8, 16, 32, 64, 127, 254, 508, 1016, 2032, 4064$	8128
33550336	$1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096,$ $8191, 16382, 32764, 65528, 131056, 262112, 524224,$ $1048448, 2096896, 4193792, 8387584, 16775168$	33550336

Implementing this idea will be the first of several approaches. The others are listed below. Each approach should begin with: "for each integer, n, starting at 2 and counting up, do:

1. **Basic Algorithm:** Find all proper divisors of n; Sum them; If the sum $= n$, then n is perfect.
2. **Using a known property of perfect numbers:** If n is prime, then if $2^n - 1$ is prime, then $(2^n - 1)(2^{n-1})$ is perfect. To determine if $2^n - 1$ is prime, use the basic methods to determine primality.
3. **Use a esoteric mathematical theorem on divisibility:** If n is prime, then if $2^n - 1$ is prime, then $(2^n - 1)(2^{n-1})$ is perfect. To determine if $2^n - 1$ is prime, use the Lucas-Lehmer test. Write your program in Java.
4. **Compare two languages:** If n is prime, then if $2^n - 1$ is prime, then $(2^n - 1)(2^{n-1})$ is perfect. To determine if $2^n - 1$ is prime, use the Lucas-Lehmer test. Write your program in Python.
5. **Multi-threading:** Using your Lucas-Lehmer approach, discover how to utilize multiple processors on your computer, and write and run a program which implements this.

6. **Deleting redundancy from your program:** Regenerating a list of prime numbers is redundant and can slow you down. Try an approach with avoids this

7. **Manipulation at the bit (processor) level:** Notice that, for a given prime, p, the number $2^p - 1$ takes the binary form: 111...11, a sequence of p 1's. Notice also that the number 2^{p-1} takes the binary form 1000...00, or 1 followed by $p - 1$ 0's. For instance, $2^5 - 1 = 11111_2$, and $2^{11-1} = 10000000000_2$. Rewrite either your Java or Python Lucas-Lehmer programs to construct the values $2^p - 1$ and 2^{p-1} by setting and clearing the bits in these numbers, instead of multiplying or computing exponentials.

Goal: Find as many perfect numbers, in order, as you can using the approaches below, and use an appropriate metric for evaluating and comparing each approach.

Assignment: You will need to write Java and Python code which implements each of the improvement suggestions above. As you run each program segment, you should see a list of perfect numbers scroll down your screen. You might also print the corresponding prime number, the number of elapsed nanoseconds, or seconds, or hours, etc. Perhaps you wish to print a certain number of primes so that your screen is active, and doesn't appear to have stopped.

You will need to discuss the efficiency of your approaches above. For instance, you'll need to state whether the Lucas-Lehmer approach enhances the speed of your algorithms, and by roughly how much? Or does setting and clearing bits to form $2^p - 1$ speed of your program a great deal or not? Do any of the approaches slow you down? A meaningful graph helps. Should you pick a specific perfect number and graph each method against the time required to produced that perfect number? Or should you plot a different curve for each enhancement showing the time required to find each perfect number? It's up to you. This program is all about trying things and analysing them. In the process, you will learn of the many types of things people do to speed up algorithms.

By the end of this assignment, you should be very impressed with i) how much faster we can make programs run with wise design decisions, and ii) be disappointed that at your very best, you won't be able to find as many perfect numbers as you hope. Well ... unless you go beyond this lab and try even more and better ideas!

Conclusion: At the end of this project, you should write a statement of conclusion. What have you learned? What impressed you and what didn't? List at least one idea of something more you could do to make this algorithm ever faster!