Amazing Prime Numbers Project (Fall 2018)

First Name: Tyson Last Name: Stinson
UNI: TJS2174

November 30, 2018

1.) Euclid's Algorithm

The first function of program is GCD(a,b). The function takes two integers as its instance variables and uses Euclid's Algorithm to calculate the greatest common denominator of the two numbers. It accomplishes this through recursion. The first step is calculating the remainder from dividing integer a by integer b. This is calculated using the modulus operation. Then if the remainder is zero, integer b is returned as this is base case for the recursive process. If the remainder is not zero, then function call itself with integer b and the remainder as its instance variables. Some examples of the inputs and outputs of this function are shown below.

Table 1: Results for the GCD function.

Integer A	Integer B	Greatest Common Denominator
4278	8602	46
12345	123456	3
406	555	1
244	354	2

2.) Generating Prime Numbers

The second function of the program is called prime(n) and it takes in a integer as its only instance variable. This variable,n, is used to determine range of prime numbers that we generating. For example if you entered 100, then every prime number from 2 to 97 would be generated then added to set then eventually returned as a sorted list. The way that this is accmoplished is through the Sieve of Eratosthenes algorithm. In between the range of 2 and n, we store prime numbers in a set called primes starting with 2 and store all multiples of that prime in the set called composites. After 2, we only evaluates odd numbers for efficiency's sake. Repeating this process for each number that is not in composites we end up with set of all prime numbers in this range. But becuase sets are cannot be indexed and are not ordered we must converted this set to an ordered list which is returned by the funciton. Some examples of the input an output are compiled here. The highest order of magnitude that I could generate before the program started having issues was 10^6. Using sets instead of arrays helped with efficiency but I am sure they are other ways to optimize the method that I dont't quite understand at the moment that would both improve the reach and speed of my function.

Table 2: Results for the Prime function.

Range	Primes
10	4
100	25
1000	168
10000	1229
1000000	78498

3.) Primality Test

For testing the primality of an integer, there are two functions in my program, trialdivision(n) and primalitysieve(n). The first function uses a method known a trialdivision to determine a number is a prime or a composite. It accomplishes this by testing if is an integer between 2 and the square root of the integer you are testing that is a factor. There exists a factor within this range exists then the number you are testing,n, is a composite. If not it is a composite.

Table 3: Results for the trialdivision and primalitysieve functions.

Input	Primality
17	Composite
177	Prime
123	Composite
1234	Composite
12345	Composite
123456	Composite
1234567	Prime
15485863	Prime

The second function, simply uses the prime function described earlier to generate a list of primes and check if the number n exists in that list. If it is, then its a prime. Otheriwse, it is a composite. Because the function relies on the prime funnction it too is limited to range with 10^6.

4.) Prime Factorization

For finding the prime factorization of an integer, there is factorization(n) which is simply a modified version of the trialdivision(n) function. The instance variable is still the number we are analyzing. And the method follows the same steps as trialdivision(n) except now it keeps a list of prime numbers that factors of the number n. If the factors is in the list generated by prime(n) then it is added to the list. At the end of the function this list is returned. One limitation of using the function prime(n) to determine whether the number is prime or not is that it limits the range to 10^6 like many of the other functions. In the future, I would probably use a recursive version to avoid this problem or optimize the prime(n) function.

Table 4: Results for the Factorization function.

Input	Factorization
12	[2,3]
123	[3]
1234	[2]
12345	[3,5]
123456	[2,3]
1234567	[127]

5.) Prime Distribution

Probably the most ambitious of this project functions, the distribution(n) functions takes in the range of numbers you want to analyze then uses the prime(n) method to generate a list of prime numbers in that range. It was why it was important to have prime(n) return an order list, because the first part of the function determines how many primes end with the digits: 1,3,5,7,9 . It is worth noting that besides 2 all primes must end in one of these digits and that 5 is the only prime number that ends with the digit 5. Then for each section of prime numbers, we find how many primes ending with 1,3,5,7,9 follow prime number in that set. For example if the counter end5to7 would be 1 in the range of 10 because in the prime numbers below 10 there is one prime number ending with 5 followed by 7. It is worth noting that the only number that has couter to 5 greater than 0 is 3 because there are no other primes other than 5 that end with the digit 5.

Table 5: Results for the Prime function.

Ratio of...	Primes Ending With 1	Primes Ending With 3	Primes Ending With 5
All	0.250	0.251	$1.274 * 10^{-5}$
Followed By Primes Ending 1	0.041	0.056	0.0
Followed By Primes Ending 3	0.080	0.036	0.0
Followed By Primes Ending 5	0.000	$1.274 * 10^{-5}$	0.0
Followed By Primes Ending 7	0.083	0.074	$1.274 * 10^{-5}$
Followed By Primes Ending 9	0.045	0.085	0.0

Next, the program uses the list from prime(n) to calculate the amount of twin primes in a range. A twin prime is a prime that has another prime within 2 integers of it. For example the first set of twin primes is (3,5). Note, (2,3) is not considerd a twin prime. The way that my program calculates this is by going through each prime and seeing the integer 2 more than it is a prime. That way each set of twin primes is only counted once.

Table 6: Results for the Factorization function.

Range	Amount of Twin Primes
10	2
100	8
1,000	35
10,000	205
100,000	1224
1,000,000	8169

Finally, the last part of the function generates a graph with matplotlib that demonstrates how the amount of primes changes in respect to the range given.
.

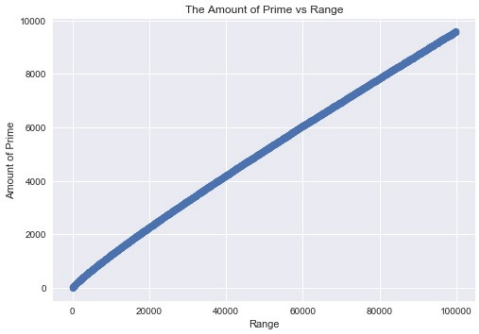

Figure 1: A Graph Relating the amount prime to a given range

6.) Visualization

The results from the distribution section, inspired me to make a visualization that artistically displays the how the numbers are distrubuted. In order to this I used a library called turtles to generate a function to represent each prime number. Given that the distribution function, showed how each prime was distrubuted I figured it would interesting to have this distribution determine a property of the function. So the function takes a list of primes from the prime(n) function as generates sqaures from a common point based on the following properties: the size of the prime number and what digit it ends with. The first property determines the size of the square and the angle at which it is generated. The second property determines the color out the square's outline. The result is a sort of rainbow flower that shows the distribution of primes in a visually appealing manner.

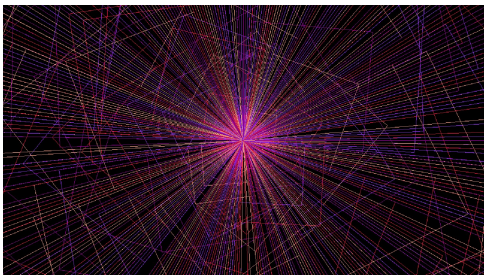

Figure 2: A Graph Screenshot of the Visualizer.