Discrete Math II
Project A-4
"Optimal Pairs"

Gabriel Smith

Problem

Two relatively prime numbers m and n (that is, gcd(m,n) = 1), with $m < n$, are called ***optimal pairs*** if the sum of all primes between m and n (inclusive) is equal to the product of m and n. That is:

where **P** is the set of primes. *E.g.*, (2,5) is an optimal pair because $2 + 3 + 5 = 2 \cdot 5$.

Find all optimal pairs where both numbers are less than 10^7.

Find all optimal pairs where both numbers are prime and less than 10^{12}.

Overview

My algorithm, written in Java, solves this problem using a series of nested for loops. The first loop cycles through the integers between 1 and the upper bound; its current value assigned to m. The second loop, contained within the first, cycles through the integers between m and the upper bound, assigning its value to n. By looping this way, we can assure that m is always less than n. For each pair of values *(m, n)* the program goes through a series of checks to determine whether it is an optimal pair. It first checks if m and n are relatively prime. If the greatest common divisor of m and n is one, then m and n are relatively prime. If m and n are relatively prime, the program then checks whether *(m, n)* is an optimal pair. It does so by cycling through the integers between m and n and keeping a running total of all the prime numbers it encounters. At the end of this loop, if the total = $m \cdot n$ then *(m, n)* is an optimal pair.

Because of the way this algorithm was implemented, its time complexity is O(n^2). As n, the upper bound, gets large, the time it takes to run the algorithm increases on an exponential scale. As you will see from the program's output, running the algorithm with an upper bound greater than 1500 requires a tremendous amount of time. Because of this, I was unable to calculate the optimal pairs using max bounds of 10^7 and 10^{12}.

Programs Used

Eclipse SDK 3.3.0 for writing, compiling and running the Java code.

Maplesoft Maple 11.0 for graphing and analyzing the collected data.

Solution

Due to the time required to run this algorithm using large upper bounds, I ran a series of tests using smaller upper bounds and compared the results with the time required to finish. The largest upper bound I was able to use was 5000, which took almost 10 ½ hours. The output for each run is shown below.

Upper bound: **1**

```
Optimal pairs found between 0 and 1: 0
Total run time: 0:00:00:000
```

Upper bound: **10**

```
1, 2
2, 5
Optimal pairs found between 0 and 10: 2
Total run time: 0:00:00:015
```

Upper bound: **100**

```
1, 2
2, 5
3, 13
5, 31
7, 53
Optimal pairs found between 0 and 100: 5
Total run time: 0:00:00:203
```

Upper bound: **500**

```
1, 2
2, 5
3, 13
5, 31
7, 53
12, 103
15, 149
17, 169
29, 314
Optimal pairs found between 0 and 500: 9
Total run time: 0:00:28:672
```

Upper bound: **1000**

```
1, 2
2, 5
3, 13
5, 31
7, 53
12, 103
15, 149
17, 169
29, 314
43, 507
55, 676
57, 713
Optimal pairs found between 0 and 1000: 12
Total run time: 0:04:33:016
```

Upper bound: **1500**

```
1, 2
2, 5
3, 13
5, 31
7, 53
12, 103
15, 149
17, 169
29, 314
43, 507
55, 676
```

```
57, 713
Optimal pairs found between 0 and 1500: 12
Total run time: 0:17:06:703
```

Upper bound: **5000**

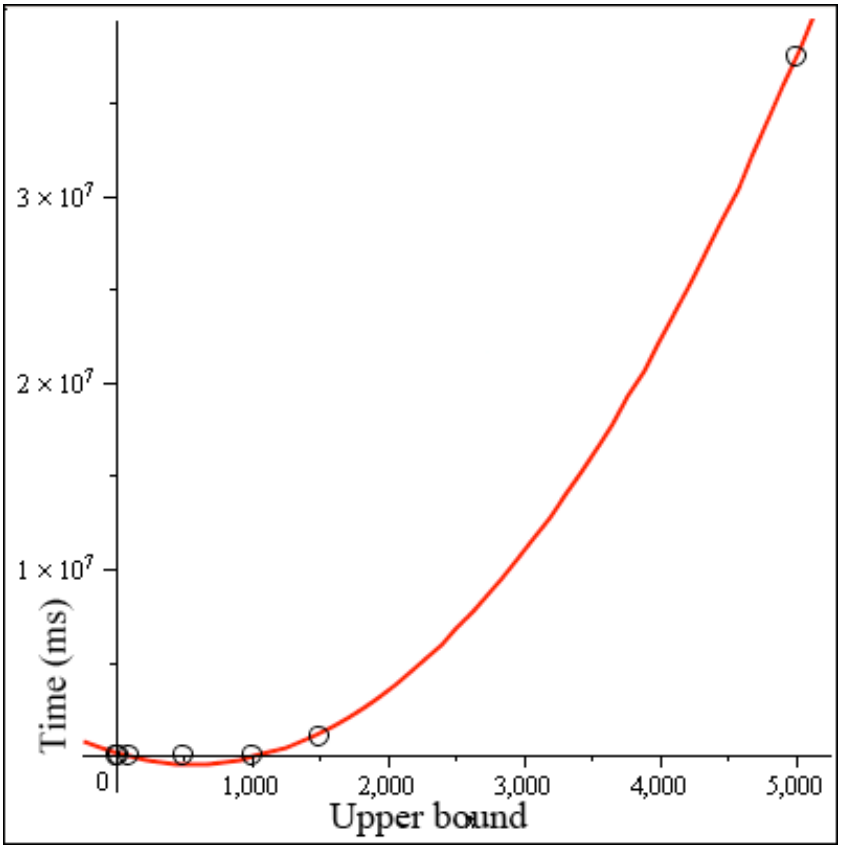
```
1, 2
2, 5
3, 13
5, 31
7, 53
12, 103
15, 149
17, 169
29, 314
43, 507
55, 676
57, 713
141, 2065
Optimal pairs found between 0 and 5000: 13
Total run time: 10:25:34:005
```

Note: An attempt was made to run the algorithm with an upper bound of 10000. At about 30 hours into the run, the optimal pair (332, 5453) was found. This run was unable to complete.

Runtime

Upper Bound	Runtime	Runtime (in milliseconds)
1	0:00:00:000	0 ms
10	0:00:00:015	15 ms
100	0:00:00:203	203 ms
500	0:00:28:672	28,672 ms
1000	0:04:33:016	273,016 ms
1500	0:17:06:703	1,026,703 ms
5000	10:25:34:005	37,534,005 ms

Plotting the upper bound vs. runtime we get the graph:

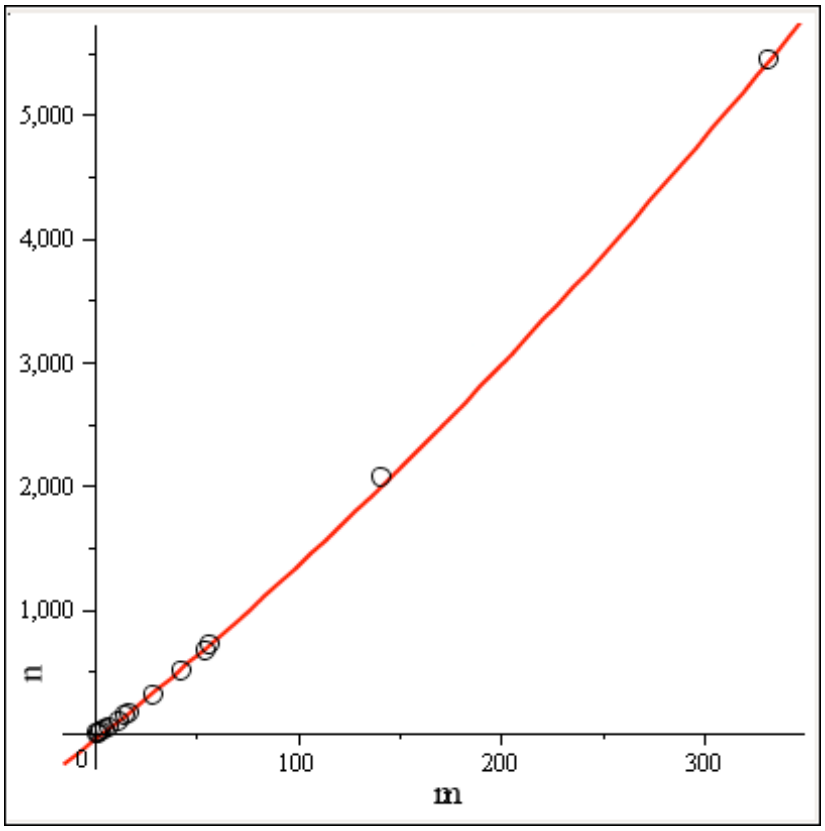


The red line represents the best fitting curve for these points, which follows the equation:

If we plug 10^7 into this equation, we get 190,261,946,331,860 ms, which is approximately **6,029 years**. Plugging 10^{12} into the equation gives us approximately1,902,823,997,954,616,000,170,000 ms, which is an amazing **6,029,814,180,000 years**! As you can see, both of these numbers are virtually impossible to handle with the hardware I was working on. With the help of a few supercomputers, it may be possible to computer the optimal pairs up to 10^7 or even 10^{12}.

Analysis

Using the optimal pairs I was able to find, we can arrange them in a graph to look for any patterns. The optimal pairs found were {(1, 2), (2, 5), (3, 13), (5, 31), (7, 53), (12, 103), (15, 149), (17, 169), (29, 314), (43, 507), (55, 676), (57, 713), (141, 2065), (332, 5453)}. Graphing these points as *m* vs. *n* we get:



The red line represents the best fitting curve for these points, which follows the equation:

These points do seem to follow an almost-linear path. As *m* get large, the space between each seems to increase exponentially, suggesting the next optimal pair will have an *m* value of about 750 and an *n* value over 10000. Given much more time to run the algorithm, and possibly with the help of a few supercomputers, we would probably find that the pattern continues, with the space between each successive optimal pair increasing exponentially.