The Rule Of Six: Prime Number Sieving Algorithm Using 6K±1

Alex Lieberman
Christopher Tino
Christopher Marino
Alex Lieberman is an alumni of Cornell University and is attending the Masters of Science Program at Pace University. In his spare time he enjoys tackling problems of all kinds and developing new ideas for products and business. Alex has developed an application in the Google Play store and currently has more on the way.

Christopher Tino has more than 10 years experience as both a web and software developer, with a client portfolio that includes marquee brands such as Nike, GE and Target. He began his Masters of Science in Computer Science at Pace University in June 2013. Prior to studying at Pace, Chris earned his Bachelor of Arts from the University of Richmond. He currently works as a Lead Developer for a creative agency in New York City.

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif

Christopher Marino started his Masters of Science in Computer Science at Pace University in January 2014. He came to Pace holding a Bachelors of Science in Computer Science from St. Johns University in Jamaica, New York. During 2013 he held an internship at an eCommerce company where he evaluated clients needs and the costs to build a website using the eCommerce platform Magento.
The Rule Of Six: Prime Number Sieving Algorithm Using 6K±1

Alex Lieberman, Christopher Marino, and Christopher Tino

Computer Science Department, Pace University
1 Pace Plaza, New York, NY 10038 USA
{al87638n, cm35853n, ct32706n}@pace.edu

Abstract. Generating prime numbers quickly and efficiently is essential to modern technology and computing theory, specifically in regards to public-key cryptology, hashing and prime number factorization\cite{5}. Prime number identification and generation is most accurately achieved by sieving algorithms. The Sieve of Eratosthenes has been the most outlasting algorithm for generating primes since it was discovered over two thousand years ago and has a time complexity O(n log log n). The algorithm being presented improves efficiency of prime number sieving by an upper bound of approximately 70% in comparison to the Sieve of Eratosthenes. It accomplishes these results by skipping numbers that will never be prime and eliminating non-primes in a manner that reduces unnecessary repetition. This method is easily implemented in code and more easily taught in schools for handwritten calculations.

1 Introduction

The Rule of Six prime number generator, or sieve, is designed to retrieve all prime numbers between 2 and $N$ in linear $O(n)$ time. Our algorithm will count primes by automatically skipping known primes and multiple of primes. We rely on the number 6, which is a product of the first two primes (2*3), to determine primes and factors of primes. Furthermore, the Primality Test tells us that all primes are of the form $6k\pm1$, except for two and three\cite{8}. Finally, The Rule of Six can be implemented both computationally and as a naive, handwritten algorithm. We will explain the algorithm in more detail later.

What is a prime number?

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself\cite{6}. As such the only way to calculate prime numbers is by testing if they are divisible by any numbers less than the square root of themselves.

The list of the first ten primes is inclusive of the following numbers:

| 2 3 5 7 11 13 17 19 23 27 |

Figure 1. A list of the first 10 primes

Observations:

A. All even numbers are excluded from the list above as they are all divisible by the number 2.

B. The numbers 6, 9, 12, 15, 18, and 21, 24 are excluded. If we look at the second number ‘3’, we can find that if we multiple three by any number greater than itself, it can never be divisible by only one and itself.
C. We can notice the numbers 10, 15, 20, and 25 are excluded from this, because they are all a factor of the third number on this prime list ‘5’.

D. Finally, we will also witness that any numbers on this list after 5 will not be divisible by 7, 11, 13, 17, 19, 23, 27 and all other numbers that are prime.

As is witnessed, some of these numbers may overlap, such as the number 15, which has factors of both 3 and 5. The method you are about to learn will help you count primes by automatically skipping every number witnessed in A & B. It will also help you quickly skip some numbers that may overlap.

2 A Brief History of Prime Number Sieving

Methods for calculating prime numbers date back to the early Egyptians and Greeks. In the third century B.C., the Greek astronomer Eratosthenes developed one of the most effective methods for discovering primes, known as the Sieve of Eratosthenes. The efficiency of Eratosthenes’ sieve is attributed to an implied tradeoff between time complexity and storage space. The algorithm proceeds as follows:

1. Start with a list of consecutive integers from 2 through n, where p=2 (the first prime).

2. Enumerate multiples of p by counting to n in increments of p, and mark them in the list (as 2p, 3p, 4p, etc.).

3. 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.

The method developed by Eratosthenes has a time complexity of O(n log log n) and has prevailed for over two millennia. Attempts to speed up this algorithm are often done at the sacrifice of storage complexity, as was the case with B.A. Chartres, who modified Eratosthenes’ sieve with smaller arrays and partial sorting. Likewise in 2004, The Sieve of Atkin was developed by A.O. L. Atkin and Daniel J. Bernstein and has a time complexity of O(n / log log n).

3 The Rule of Six Algorithm

Our algorithm hinges on the number 6, which we have determined to be the most important number for discovering primes. The number 6 is the first factorization of any two prime numbers (2 x 3 = 6). As such, 6 holds great significance when calculating prime numbers, because it sieves out the most numbers without missing any primes. The aforementioned Primality Test tells us that all primes are of the form 6k±1. The formula for the Rule of Six is as follows:

The Rule of Six Algorithm:
1. Initialize list with initial values (2, 3).
2. Use the formula 6k ± 1 where k = 1 and must increment by 1. Store values to list.
3. Compute step two up to a product n, where n > 7 if k = 1.
4. Starting with x = 5, remove all products of x times y (up to n), where y starts at x and increments to the next number in the list as x remains the same.

5. Repeat step four by assigning x to the next number in the list. Stop when x times y > n.

6. Remove all factors of the square of every prime number (25, 49).

The following pseudocode illustrates the algorithm:

<table>
<thead>
<tr>
<th>Input: an integer n &gt; 1, where n is even</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i = 5, i += 6, j = 7, j += 6, i or j not exceeding n:</td>
</tr>
<tr>
<td>A[i] is true</td>
</tr>
<tr>
<td>if j &lt; n:</td>
</tr>
<tr>
<td>A[j] = true</td>
</tr>
<tr>
<td>for a = 5, a += 2, where a*a not exceeding n:</td>
</tr>
<tr>
<td>if A[a] is true:</td>
</tr>
<tr>
<td>c = a*a</td>
</tr>
<tr>
<td>while c &lt;= n</td>
</tr>
<tr>
<td>A[c] = false for all c = c + (a*a+2)</td>
</tr>
<tr>
<td>b = a</td>
</tr>
<tr>
<td>do b += 2 while A[b] is false</td>
</tr>
<tr>
<td>while a*b &lt;= n</td>
</tr>
<tr>
<td>A[a*b] = false</td>
</tr>
<tr>
<td>do b += 2 while A[b] is false</td>
</tr>
<tr>
<td>Output: all i such that A[i] is true.</td>
</tr>
</tbody>
</table>

Figure 2. Pseudo-code of The Rule of Six Algorithm

The Rule of Six can also be presented as a naive algorithm, delivering a fast and accurate prime generation approach for use in educational purposes. The algorithm would be presented as follows:

The Naïve Algorithm

1. Initialize your list with 2 and 3.
2. On a separate list count by sixes up to the largest number you want to calculate.
3. For every six, write down the number one less than it and then the number one greater than it.
4. Sieve the list starting at 25 (the square of 5) and cross out all multiples of five by any other prime. After, move to the next largest prime and multiply it by itself and all other primes greater than itself, sieving out the product. Continue this process until the prime you are about to sieve out is greater than the square of the number you are computing up to.

4 ALGORITHM ANALYSIS
The first portion of the algorithm (step 2) allows us to quickly calculate potential primes up to the value \( n \). The benefit here is that we can narrow down the list of potential primes before final sieving (step 3) is attempted.

![Potential Primes vs Actual Primes](image1.png)

**Figure 3.** Spread between primes, non-primes, potential primes, and \( N \).

The Rule of Six algorithm allows us to easily skip over known primes, which results in a major time savings. The graph below illustrates the volume of numbers that are skipped per each iteration of the algorithm. This is in comparison to the Sieve of Eratosthenes, which skips no values.

![Skipped Numbers Per Iteration](image2.png)

**Figure 4.** The number of skipped numbers for prime testing.

The prime factorizations of any two primes will never be prime. We are aware that any number multiplied by two, three, four or five can be prime, but why does counting by six yield a sieve result of all primes and fewer potential primes than any other? It is precisely because it is the factorization of the lowest two primes, which are also the smallest integers that will never be prime when factored by any other number. As such, it eliminates all possibilities of counting by either of these numbers and yields the most whittled down list of possible potential primes. The table below illustrates the effectiveness of using \( 6k\pm1 \) to generate primes.
As discussed, all primes are of the form 6k±1. In our algorithm, we initialize all numbers of this format as a boolean array with value true, and then iterate over the array eliminating multiples of prime numbers. If the value is a multiple, it sets the array position to false and never touches the number again. This will produce a time function of O(n) since each number is only tested once, as opposed to the time function produced by the Sieve of Eratosthenes. This is because the Sieve of Eratosthenes will test the same number multiple times, even if it has already proven to be not prime.

5 EXPERIMENTATION

The Rule of Six algorithm was tested against a standard Sieve of Eratosthenes with regards to time complexity. \( N \) was scaled in both test cases from 0 to 1,000,000,000. As \( N \) approached numbers greater than 100 million it was scaled by 100 million between tests.

As \( N \) climbed from 5 million to 1 billion – the difference in runtime widened. Whereas the algorithm started around 50% faster at \( N = 5,000,000 \) as \( N \) approached 1 billion, there was a significant speed difference of about 71-73%. Further testing on more advanced computing systems and controlled environments is necessary to determine the exact rate of growth in the difference through significantly larger \( N \).

The table below lists the runtime results of \( N \) per algorithm:

<table>
<thead>
<tr>
<th>( N )</th>
<th>The Rule of Six</th>
<th>Sieve of Eratosthenes</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000,000</td>
<td>1.811</td>
<td>6.435</td>
</tr>
<tr>
<td>600,000,000</td>
<td>2.347</td>
<td>8.81</td>
</tr>
<tr>
<td>700,000,000</td>
<td>2.92</td>
<td>10.68</td>
</tr>
<tr>
<td>800,000,000</td>
<td>3.545</td>
<td>12.43</td>
</tr>
<tr>
<td>900,000,000</td>
<td>3.849</td>
<td>14.143</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>4.396</td>
<td>15.776</td>
</tr>
</tbody>
</table>

Figure 6. Table demonstrating the runtime vs. The Sieve of Eratosthenes.
5 CONCLUSION

A major drawback to the Sieve of Eratosthenes is that it will continually attempt to remove a non-prime that may have been removed already, resulting in a higher time complexity[2]. Thus, a faster approach could be achieved by eliminating these extra iterations. Using the Primality Test of $6k\pm1$, we are able to generate a list of primes and potential primes in the first phase of the algorithm, while simultaneously omitting non-primes. Then, in the second phase, we test our potential primes against the products of primes from the previously generated list. The result is a linear time algorithm that, in our testing, is effective up to and past $N = 1,000,000,000$. Furthermore, our naive algorithm is easily implemented and can be taught readily to students learning to generate prime numbers.

References

Alex Lieberman is an alumni of Cornell University and is attending the Masters of Science Program at Pace University. In his spare time he enjoys tackling problems of all kinds and developing new ideas for products and business. Alex has developed an application in the Google Play store and currently has more on the way.

Christopher Tino has more than 10 years experience as both a web and software developer, with a client portfolio that includes marquee brands such as Nike, GE and Target. He began his Masters of Science in Computer Science at Pace University in June 2013. Prior to studying at Pace, Chris earned his Bachelor of Arts from the University of Richmond. He currently works as a Lead Developer for a creative agency in New York City. 

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif

Christopher Marino started his Masters of Science in Computer Science at Pace University in January 2014. He came to Pace holding a Bachelors of Science in Computer Science from St. Johns University in Jamaica, New York. During 2013 he held an internship at an eCommerce company where he evaluated clients needs and the costs to build a website using the eCommerce platform Magento..
The Seidenberg School of Computer Science and Information Systems
Pace University

Technical Report Series

EDITORIAL BOARD

Editor in Chief:
Dr. Amar Gupta, Dean, The Seidenberg School of Computer Science and Information Systems, Pace University

Associate Editors:
Sung-Hyuk Cha, Computer Science, Pace-New York
Catherine Dwyer, Information Technology, Pace- New York

Members:
D. Paul Benjamin, Computer Science, Pace-New York
Nicholas DeLillo, Computer Science, Manhattan College
Edgar G. Ducasse, Mathematics, Pace-New York
Daniel Farkas, Information Technology, Pace-Westchester
Fred Grossman, Information Technology; Doctor of Professional Studies, Pace- New York & White Plains
Francis Marchese, Computer Science, Pace-New York
John Molluzzo, Information Technology, Pace- New York
Narayan Murthy, Computer Science, Pace-Westchester
Christelle Scharff, Computer Science, Pace-New York
Namchul Shin, Information Technology, Pace- New York
Sotirios Skevoulis, Computer Science, Pace-New York
Lixin Tao, Computer Science, Pace-Westchester
Charles C. Tappert, Computer Science; Doctor of Professional Studies, Pace- Westchester & White Plains

The Seidenberg School of Computer Science and Information Systems, through the Technical Report Series, provides members of the community an opportunity to disseminate the results of their research by publishing monographs, working papers, and tutorials. Technical Reports is a place where scholarly striving is respected.

All technical reports are published online at http://support.csis.pace.edu/TechReport. New manuscripts are read by two members of the editorial board; the editor decides upon publication. Statements of policy and mission are found in issues #29 (April 1990) and #34 (September 1990)

Please direct submissions to:

Sung-Hyuk Cha
The Seidenberg School of Computer Science and Information Systems
Pace University
1 Pace Plaza
New York, NY 10038