Sieve of Eratosthenes & Sieve of Atkin

Tyler Ciapala-Hazlerig
CPSC 450
Fall 2024

Summary

This project investigates two different algorithms for the problem of finding prime numbers. The two algorithms, the Sieve of Eratosthenes and the Sieve of Atkin, were implemented and tested in C#. In the performance tests, it was observed that the Sieve of Eratosthenes performed worse than the Sieve of Atkin for large n with the opposite happening for small n. Both appear to run in near linear time according to the observed performance tests.

1. ALGORITHM SELECTED

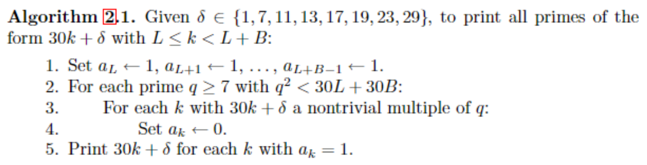
The problem in question is that of finding prime numbers. This problem consists of finding all the prime numbers in a given range of numbers. For my implementations, we are using a range of 0-n.

The simple algorithm chosen to solve this problem is the Sieve of Eratosthenes. This Sieve runs in $O(nloglogn)$ time [1]. This is due to how primes are marked in the sieve. In my implementation, the outer for-loop runs $\sqrt{n}$ times and the inner for-loop runs approximately $\frac{n}{i}$ times. According to O'Neill [4], we can use a harmonic series to approximate the complexity of the two for loops. Note that in the figure below, the $O(n)$ is from the sieve array initialization and gets factored out from the final complexity due to $O(nloglogn)$ being more dominant.

$$\sum_{i=1}^{\pi(\sqrt{n})} \frac{n}{p_i} \approx \frac{n}{2} + n \sum_{i=2}^{\frac{2\sqrt{n}}{\ln n}} \frac{1}{i \ln i} \approx \frac{n}{2} + n \int_{i=2}^{\frac{2\sqrt{n}}{\ln n}} \frac{1}{i \ln i} \approx n \ln \ln n + \mathrm{O}(n)$$

Figure 1. Approximation of the Sieve of Eratosthenes complexity.[4]

My implementation for the Sieve of Eratosthenes is roughly based on the following pseudocode from Atkin [2]:



Algorithm 2.1. Given $\delta \in \{1,7,11,13,17,19,23,29\}$, to print all primes of the form $30k+\delta$ with $L \le k < L+B$:

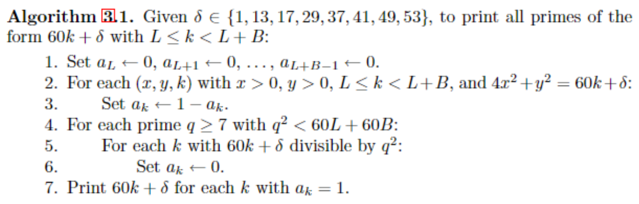
```
1. Set $a_L \leftarrow 1$, $a_{L+1} \leftarrow 1$, $\ldots$, $a_{L+B-1} \leftarrow 1$.
2. For each prime $q \ge 7$ with $q^2 < 30L + 30B$:
3.     For each $k$ with $30k+\delta$ a nontrivial multiple of $q$:
4.         Set $a_k \leftarrow 0$.
5. Print $30k+\delta$ for each $k$ with $a_k = 1$.
```

Figure 2. Pseudocode for the Sieve of Eratosthenes.

My implementation includes a few modifications to this pseudocode. The most obvious one is that my algorithm is given an n, not a set of numbers, where n is the maximum number to "search" for primes in. I also initialize the sieve to 0 and set them to 1 at line 4. This is opposite to the pseudocode.

The complex algorithm chosen for this problem was the Sieve of Atkin. This is a more modern sieve that runs in $O(\frac{n}{loglogn})$ time according to Atkin [2]. This is true because of the bounds imposed on the second set of for-loops. Let's look at my implementation in more detail to understand where this complexity comes from. The initialization of the sieve if $O(n)$. In the first set of for-loops, which is responsible for marking all potential primes in the sieve, the outer loop runs approximately $\sqrt{n}$ times. The inner loop runs approximately $\sqrt{n}$ times for each iteration of the outer loop. This makes the first set of for-loops take approximately $O(n)$ time. In the second set of for-loops, which is responsible for eliminating all the squares of primes, the outer loop runs approximately $\sqrt{n}$ times with the inner loop looping through all the multiples of i from the outer loop. According to Atkin [2] this causes this set of loops to have a time complexity of $O(\frac{n}{loglogn})$. Finally, the last for-loop, which is responsible for enumerating the primes from the sieve, iterates through the entire sieve making it take O(n) time. Since $O(\frac{n}{loglogn})$ dominates over the other O(n) loop complexities, we conclude that the algorithm has a time complexity of $O(\frac{n}{loglogn})$.

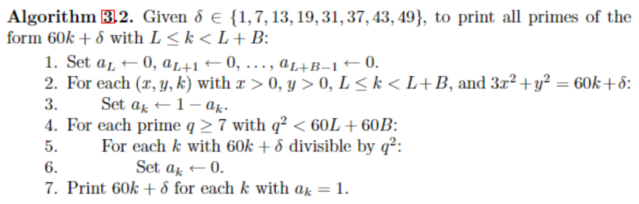
Atkins original paper split the algorithm into three different sub-algorithms. Thus, my implementation of the Sieve of Atkin was based on the following three sub-algorithms [2].



Algorithm 3.1. Given $\delta \in \{1,13,17,29,37,41,49,53\}$, to print all primes of the form $60k+\delta$ with $L \le k < L+B$:

```
1. Set $a_L \leftarrow 0$, $a_{L+1} \leftarrow 0$, $\ldots$, $a_{L+B-1} \leftarrow 0$.
2. For each $(x, y, k)$ with $x > 0$, $y > 0$, $L \le k < L+B$, and $4x^2+y^2 = 60k+\delta$:
3.     Set $a_k \leftarrow 1 - a_k$.
4. For each prime $q \ge 7$ with $q^2 < 60L + 60B$:
5.     For each $k$ with $60k+\delta$ divisible by $q^2$:
6.         Set $a_k \leftarrow 0$.
7. Print $60k+\delta$ for each $k$ with $a_k = 1$.
```

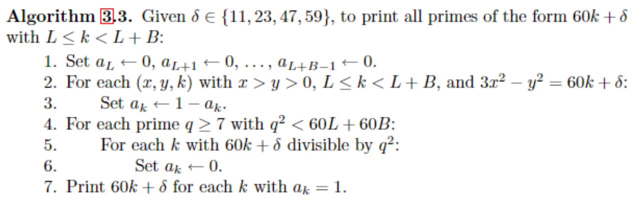
Figure 3. First part of the pseudocode for the Sieve of Atkin.



Algorithm 3.2. Given $\delta \in \{1,7,13,19,31,37,43,49\}$, to print all primes of the form $60k+\delta$ with $L \le k < L+B$:

```
1. Set $a_L \leftarrow 0$, $a_{L+1} \leftarrow 0$, $\ldots$, $a_{L+B-1} \leftarrow 0$.
2. For each $(x, y, k)$ with $x > 0$, $y > 0$, $L \le k < L+B$, and $3x^2+y^2 = 60k+\delta$:
3.     Set $a_k \leftarrow 1 - a_k$.
4. For each prime $q \ge 7$ with $q^2 < 60L + 60B$:
5.     For each $k$ with $60k+\delta$ divisible by $q^2$:
6.         Set $a_k \leftarrow 0$.
7. Print $60k+\delta$ for each $k$ with $a_k = 1$.
```

Figure 4. Second part of the pseudocode for the Sieve of Atkin.



Algorithm 3.3. Given $\delta \in \{11, 23, 47, 59\}$, to print all primes of the form $60k + \delta$ with $L \leq k < L + B$:

```
1. Set $a_L \leftarrow 0$, $a_{L+1} \leftarrow 0$, $\ldots$, $a_{L+B-1} \leftarrow 0$.
2. For each $(x, y, k)$ with $x > y > 0$, $L \leq k < L + B$, and $3x^2 - y^2 = 60k + \delta$:
3.     Set $a_k \leftarrow 1 - a_k$.
4. For each prime $q \geq 7$ with $q^2 < 60L + 60B$:
5.     For each $k$ with $60k + \delta$ divisible by $q^2$:
6.         Set $a_k \leftarrow 0$.
7. Print $60k + \delta$ for each $k$ with $a_k = 1$.
```

Figure 5. Third part of the pseudocode for the Sieve of Atkin.

2. IMPLEMENTATION

Both algorithms' implementations were written in C#.

My implementation of the Sieve of Eratosthenes varied from the pseudocode in a few different ways. First, the pseudocode was designed in a way to take a set of numbers as input and print the prime numbers in that set. My implementation simply takes an n and finds all the prime numbers in the range of 0-n. My implementation uses a Boolean array to represent the sieve. It initializes the sieve to all 0's and sets each index to 1 when a prime is found. This is opposite to the pseudocode. Rather than printing each prime number, my implementation iterates through again after the main algorithm as run and adds each index that is 1 to a C# standard library list data structure, then returns this list.

My implementation of the Sieve of Atkin is substantially different from the originally published pseudocode since the original is split into three different sub-algorithms. However, the main ideas still stand. My implementation combines the first for-loop in each of the three algorithms together and performs conditional checks to determine which action to perform on the sieve. The second for-loop in each pseudocode sub-algorithm is the same. Thus, I simply included it in my implementation mostly unchanged. Just like on the Sieve of Eratosthenes implementation, the final step was to iterate through the sieve and add each true index to a C# standard library List data structure and return the list.

Testing the algorithm simply consisted of unit tests that ran the algorithm and tested it against a known good list of prime numbers. The tests consisted of different input sizes at increments of 10, 50, and 1000, in addition to invalid input sizes of 0 and negative numbers. The same tests were run for each algorithm.

3. PERFORMANCE TESTS

The performance tests performed on my implementations consisted of timing how long each algorithm took to generate primes up to n (primes between 0-n). Two tests of this nature were performed, a small-n test and a large-n test. The small-n test started with an input size of 0, increased by 100, and stopped at 900. The large-n test started with an input size of 0, increased by 100,000, and stopped at an input size of 900,000. Each test resulted in 10 data points for each algorithm.

The tests were performed on a Linux virtual machine running the NixOS operating system inside of the Windows Subsystem for Linux (WSL). The virtual machine was running on a computer with an Intel Core i7 11700k and 32 GB of memory.

The tests were performed in a shell environment as defined by the direnv.nix file at the root directory of the repository.

To run the performance tests, enter the "./PerformanceTests" directory in the project and run the command "dotnet run". Note: you will need to install the .NET 9 SDK on your computer to run the tests. If you are using the Nix package manager or NixOS you can use the provided direnv.nix file to load the environment.

4. EVALUATION RESULTS

The large input size performance graph can be seen below. It looks about as expected. The Sieve of Eratosthenes is outperformed by the Sieve of Atkin as n increases. This makes sense given that the Sieve of Eratosthenes has a time complexity of $O(nloglogn)$ while the Sieve of Atkin has a time complexity of $O(\frac{n}{loglogn})$. $O(nloglogn)$ is worse, which is seen in the graph below.

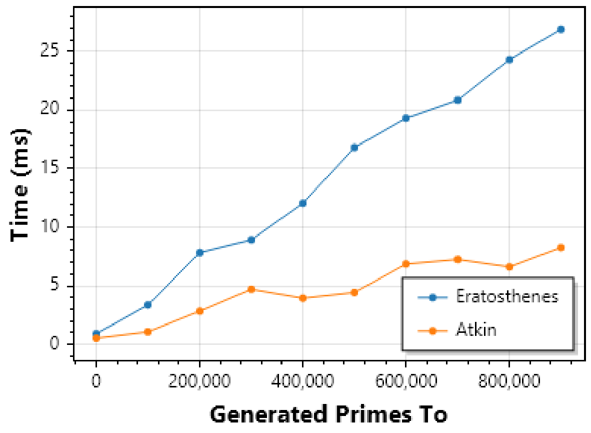


Figure 6. Large-n performance test results for Sieve of Eratosthenes vs Sieve of Atkin.

Things look a little different when we look at the graph from the small input size performance tests. This test looks in detail at how the algorithms behave for small input sizes. Here, we can see that the difference between the two algorithms is negligible. In fact, the Sieve of Eratosthenes outperforms, or is nearly equal to, the Sieve of Atkin for many samples. According to Loncaric [3], this is likely because the Sieve of Eratosthenes is simpler than the Sieve of Atkin. Thus, at a lower input size, the overhead from the Sieve of Atkin being more complex results in it being slower than the simpler algorithm. For example, my implementation of the Sieve of Atkin must compute the square of a number a little less than $8n$ times throughout one pass of the algorithm. This could be slightly improved with dynamic programming techniques of storing the already calculated squares, however, this would never reach the simplicity, at least for small-n, of the Sieve of Eratosthenes which only need to calculate the square of a number approximately $nloglogn$ times. At small-n, $loglogn$ is much smaller than the constant 8.

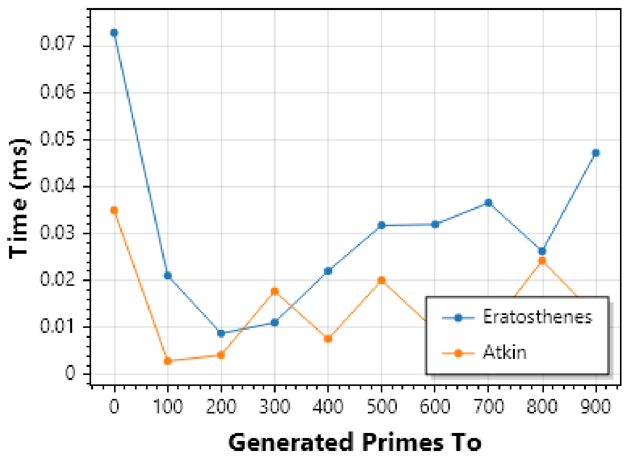


Figure 7. Small-n performance test results for Sieve of Eratosthenes vs Sieve of Atkin.

Examining the time complexities of each algorithm allows us to better understand why we observe this behavior at small n. At small n, $loglogn$ is close to 1, at least compared to very large n. With the $loglogn$ term being in the denominator of the fraction for the Sieve of Atkin, we nearly get $O(\frac{n}{1})$ or $O(n)$. A similar fact is true for the Sieve of Eratosthenes. With the $loglogn$ being close to 1 we get nearly $O(n * 1)$ or $O(n)$. Given that their time complexities are so similar for small-n, we can look for other inefficiencies, such as the number of squares needed to be calculated in the Sieve of Atkin, to determine why the Sieve of Atkin performs better for small-n.

5. REFLECTION

The most interesting thing about the algorithms I chose was the vastly different ways they can be implemented. For example, I found pseudocode for the Sieve of Eratosthenes that was different than the pseudocode I based my implementation on, even though they both achieved the same algorithm.

The most difficult part of this assignment was reading the original paper by Bernstein and Atkin. It was very dense and difficult to read. It was also difficult to fully understand how to implement the algorithm based on the pseudocode provided. This was because their pseudocode consisted of three different sub-algorithms that I then had to combine into one. Also, the original paper defined their algorithms as generic as possible, which I then had to use to figure out how to implement my not so generic algorithm. I did end up having to use a few other resources to help with this.

If I had more time for this project, I would develop some tests to test the space complexity of each implementation because space complexity was talked about a lot in the Bernstein and Atkin paper. I would also test the difference between the Sieve of Eratosthenes and a Segmented Sieve. I might also try to implement the Sieve of Atkin using dynamic programming to store the already calculated squares and see if that improves its performance at small input sizes

6. RESOURCES

[1] This paper gave me an understanding of the Sieve of Eratosthenes. The pseudocode for the Sieve of Eratosthenes in this paper helped me to understand how the algorithm worked even though it wasn't the one I based my algorithm off. This paper also gave me an introduction into prime number generation algorithms and sieves. Due to the generic nature of this paper, this was the starting point for my research.

[2] This was the paper that originally published the Sieve of Atkin. This paper was very difficult to read, however, it gave me a starting point for implementing the Sieve of Atkin. This paper also helped me understand the complexity of the Sieve of Atkin.

[3] This resource was used to help me sanity check my findings. More importantly, it helped me to understand why I was seeing the two algorithms behave similarly at small n. None of the other papers I found mentioned why this happens.

[4] This paper helped me a lot in understanding the time complexity of the Sieve of Eratosthenes. This paper contained the necessary harmonic series that allows us to approximate the loop with the time complexity of $O(nloglogn)$.

7. REFERENCES

[1] Jonathan Sorenson. 1990. An Introduction To Prime Number Sieves. Department of Computer Sciences. University of Wisconsin-Madison, Madison, WI. https://research.cs.wisc.edu/techreports/1990/TR909.pdf.

[2] A. O. L. Atkin and D. J. Bernstein. 2003. Prime Sieves Using Binary Quadratic Forms. Mathematic of Computation 73, 246, 1023-1030. https://www.ams.org/journals/mcom/2004-73-246/S0025-5718-03-01501-1/S0025-5718-03-01501-1.pdf

[3] Martin Loncaric. 2016. Prime Sieves. (August 2016). Retrieved December 5, 2024 from https://graphallthethings.com/posts/sieves/.

[4] Melissa E. O'Neill. The Genuine Sieve of Eratosthenes. Harvey Mudd College, Claremont, CA, USA. https://www.cs.hmc.edu/~oneill/papers/Sieve-JFP.pdf