Efficiency of the Sieve of Eratosthenes and Comparative Analysis of Prime-Finding Algorithms

History of the Sieve of Eratosthenes

One of the most popular prime number algorithms is the Sieve of Eratosthenes. This algorithm was founded by a famous Greek scientist named Eratosthenes of Cyrene. Eratosthenes had many talents, one of which was in mathematics. Mathematics during his life was nothing compared to modern-day mathematics; thus, it was easier to become talented at mathematics.

Eratosthenes discovered a systematic way to find primes. This system involved starting with a prime and then marking all multiples of that prime as composite. The numbers that do not get crossed off end up being the primes.

The Sieve of Eratosthenes has played a significant role in finding "small" primes back during the time when computers did not exist. Although better algorithms have been found to compute primes using computers, this algorithm is one that every math or computer science student should be taught.

Efficiency of the Sieve of Eratosthenes

Pattern

1. **Initialization**: Create a list of consecutive integers from 2 through n.
2. **Marking Non-Primes**: Starting with the first prime number $p = 2$, mark all multiples of $p(i.e., 2p, 3p, 4p, ...)$ as non-prime.
3. **Iteration**: Move to the next unmarked number in the list, which is the next prime, and repeat the marking process.
4. **Termination**: Continue until $p^2 > n$. All remaining unmarked numbers are primes.

Example of the Sieve of Eratosthenes Start by listing all numbers from 2 to 16. Initially, assume all numbers are prime.

P : Indicates the number is currently considered **prime**. **C** : Indicates the number has been marked as **not prime** (composite).

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P

Step 1: Mark Multiples of 2 The first prime number is 2. Mark all multiples of 2 (except 2 itself) as **not prime**.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P	P	C	P	C	P	C	P	C	P	C	P	C	P	C

Marked Multiples of 2: 4, 6, 8, 10, 12, 14, 16

Step 2: Mark Multiples of 3 The next unmarked number is 3, which is prime. Mark all multiples of 3 (except 3 itself) as **not prime**.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P	P	C	P	C	P	C	C	C	P	C	P	C	C	C

Marked Multiples of 3: 6, 9, 12, 15

Termination The next unmarked number is 5. Since $5^2 = 25 > 16$, we can stop the process. All remaining unmarked numbers are primes.

Final List of Primes:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P	P	C	P	C	P	C	C	C	P	C	P	C	C	C

Primes up to 16: 2, 3, 5, 7, 11, 13

Time Complexity The time complexity of an algorithm measures how the running time increases with the size of the input. For the Sieve of Eratosthenes:

- **Initialization**: Creating the list takes $O(n)$ time.
- **Marking Non-Primes**: For each prime p , marking its multiples up to n takes $O(n/p)$ time.

The total time spent marking multiples is:

$$\sum_{p \leq \sqrt{n}} \frac{n}{p}$$

The sum of the reciprocals of the primes up to $\sqrt{n}$ is approximately $n \log \log n$. Therefore, the overall time complexity is:

$$O(n \log \log n)$$

This is nearly linear, making the sieve exceptionally efficient for large values of n.

Benefit's: The sieve of Eratosthenes is good for generating very large prime numbers due to its time complexity of $O(n \log \log n)$. The sieve is also very straightforward to understand and implement by hand or through algorithms on a computer. The sieve also provides a deterministic way to identify all primes up to n, ensuring that all primes can easily be identified and won't be missed. Unlike probabilistic algorithms, the sieve doesn't rely on chance, offering guaranteed results.

Challenges While the Sieve of Eratosthenes is highly efficient and simple for generating all prime numbers up to a moderate limit, it faces significant challenges when dealing with very large numbers or specific primality testing requirements. In such cases, alternative algorithms may offer better performance and scalability based on the specific needs of the application. It is designed to generate all primes within a specified range rather than efficiently verifying the primality of a single very large number. Additionally, the sieve operates with a fixed upper limit, meaning that the range of numbers to be sieved must be known in advance. If primes beyond this initial range are needed, the sieve must be rerun with a higher n, which can be inefficient and time-consuming.

Real world usage The Sieve of Eratosthenes has been instrumental in generating comprehensive lists of prime numbers efficiently, which are essential for mathematical research and educational purposes. In the field of cryptography, it aids in the generation of large prime numbers necessary for creating secure keys in systems like RSA encryption. The sieve also facilitates prime factorization in computational mathematics by providing a list of primes up to a certain limit, which can be used to decompose integers into their prime components. Additionally, it is a staple in programming contests and competitive programming, where quick prime generation is often required under strict time constraints. Researchers utilize the sieve to study the distribution of primes and to test conjectures related to prime numbers, such as the Goldbach and Twin Prime conjectures. Moreover, the sieve serves as a foundational tool in developing more advanced algorithms and optimizing data structures that rely on prime numbers for enhanced performance. Its simplicity and efficiency make the Sieve of Eratosthenes an invaluable asset across various domains in mathematics and computer science.

Other Prime-Finding Algorithms

Trial Division

Description The Trial Division algorithm is typically the easiest to understand prime number algorithm. This algorithm aims to determine if a number can be factored compared to how the Sieve of Eratosthenes works by eliminating multiples of primes. There is a strategic strategy to determine the possible factors of a number. The possible factors of a number turn out being all numbers less than the square root of said number

The algorithm goes as follows:

1. Take in a positive integer n
2. Determine the range of numbers you need to check as possible factors
3. Iterate through the range of possible factors
4. If the factor divides evenly into the number, then it is composite
5. If all possible factors do not divide evenly into the number, then it is prime.

Efficiency Time Complexity

- **Best Case**: The best-case scenario occurs when the number being tested, n, is divisible by small primes like 2 or 3. In this case, the algorithm can quickly conclude that n is composite after a few iterations. This results in a time complexity of $O(1)$, as the algorithm can terminate early.

- **Worst Case**: In the worst case, if n is prime, the algorithm must check all potential divisors up to $\sqrt{n}$. The number of such potential divisors is proportional to $\sqrt{n}$, resulting in a time complexity of $O(\sqrt{n})$. This makes Trial Division inefficient for large numbers.
- **Average Case**: On average, the algorithm will perform better than the worst case because it will typically find factors before reaching $\sqrt{n}$. However, the average-case time complexity remains $O(\sqrt{n})$ because it still depends on the size of the input number.

Trial Division algorithm is efficient in determining if single numbers are prime; however, the Sieve of Eratosthenes is efficient is finding all the primes in a specified limit. A main trade off for the Trial Division algorithm is that with larger numbers it can take increasing amounts of time.

Miller-Rabin Primality Test

Description The Miller-Rabin Primality Test is a probabilistic primality test. This means that this algorithm determines whether a number is likely prime but does not determine for certain. Gary Miller discovered a deterministic version of this test; however, this relies on a big problem in math being true called the extended Riemann hypothesis. Michael Rabin modified Millers version to make it probabilistic, and thus not dependent on an unproven problem.

This algorithm relies on mathematical concepts such as "Strong probable primes" and "Choice of bases".

Strong probable primes

"For a given odd integer $n > 2$, we can write $n - 1$ as $2^s d$ where s is a positive integer and d is an odd positive integer. Now lets consider an integer a (called a base) which is co-prime to n, Then n is said to be a strong probable prime to base a if one of these congruence relations holds:

- $a^d \equiv 1 \pmod{n}$
- $a^{2^r d} \equiv -1 \pmod{n}$ for some $0 \leq r < s$

If neither of these congruence relations hold, then n is composite and a is considered a **witness** to the compositeness of n" ("Miller-Rabin primality test," n.d.)

Choices of bases

Picking a base a at random will yield a fast probabilistic test. Most bases a will be a witness to n being composite and thus will reduce odds of a false positive to a very small rate. The typical interval for choosing a base is $1 < a < n - 1$

Efficiency The Miller-Rabin Primality Test is a probabilistic algorithm, meaning that it can determine whether a number is "probably prime" or composite with a high degree of confidence. Its performance depends on several factors, including the size of the number being tested and the number of rounds of the test that are performed.

- **Time Complexity**: The Miller-Rabin test has a time complexity of $O(k \cdot \log^3 n)$, where n is the number being tested for primality, and k is the number of rounds or random bases tested.
- **Performance for Large Numbers**: One of the biggest advantages of the Miller-Rabin test is its efficiency for very large numbers. It is significantly faster than the deterministic algorithms, such as Trial Division or the Sieve of Eratosthenes, for primality testing of large integers. The test can be run multiple times to decrease the probability of false positives (incorrectly identifying a composite number as prime), making it a flexible choice for large-scale prime verification tasks.
- **Probabilistic Nature**: Unlike deterministic tests like Trial Division, the Miller-Rabin test does not guarantee absolute certainty that a number is prime. Instead, it gives a probabilistic result that can be made arbitrarily accurate by increasing the number of rounds.
- **Trade-offs**: While the Miller-Rabin test is very efficient and scalable for large numbers, its probabilistic nature means that it may not be suitable for applications where absolute primality is required. For example, in cryptographic settings, a false positive could have serious security implications.

Comparative Analysis

Time Complexity

- **Trial Division** exhibits a time complexity of $O(\sqrt{n})$, making it inefficient for large numbers. As seen in the results below, it struggled to identify larger primes within the given timeframe. However it has the benefit of being potentially very fast for smaller primes.
- **Sieve of Eratosthenes** operates with a time complexity of $O(n \log \log n)$, offering a near-linear performance for finding all primes up to a specified limit. It often outperforms Trial Division in both speed and the size of primes found.
- **Miller-Rabin Primality Test** has a time complexity of $O(k \cdot \log^3 n)$, where k is the number of testing rounds. Its probabilistic nature allows it to handle large numbers, making it highly effective for primality of large numbers.

Scalability

- **Trial Division** scales very poorly. As input sizes increase, the time required grows rapidly, limiting its practicality to small primes.
- **Sieve of Eratosthenes** scales well for generating all primes up to large n, but its memory requirements can become a bottleneck for extremely large ranges when run on a computer.
- **Miller-Rabin** scales very well and is good for testing the primality testing of individual large numbers. This makes it ideal for applications like cryptography where verifying the primality of large integers is important.

Some of these results can be seen from these algorithms being run together, however, there is room for error due to the CPU cycles that is dedicated to each algorithm likely differing based on processor load at the time of execution.

Algorithm	**Largest Prime**	**Elapsed (s)**	**Elapsed (HH:MM:SS)**
Trial Division	1,188,567,577	12,000.00s	03:20:00
Sieve of Eratosthenes	1,188,671,413	12,000.00s	03:20:00
Miller-Rabin	1,188,766,571	12,000.00s	03:20:00

Note tests where run a Ryzen 7 5800x with 32 GB of memory

Conclusion

The analysis of these prime-finding algorithms highlight differences in time complexity and scalability. The Trial Division algorithm has a time complexity of $O(\sqrt{n})$, which indicates it is efficient for small numbers but will progressively start to struggle with larger numbers. In contrast, the Sieve of Eratosthenes has a time complexity of $O(n \log \log n)$ and performs near-linearly while outperforming the Trial Division algorithm. A downside for the sieve is that can become memory intensive for very large ranges. Alternatively, the Miller-Rabin Primality Test has a time complexity of $O(k \cdot \log^3 n)$ where k is the number of testing rounds. Being a probabilistic test allows it to handle large individual primes well.

The Sieve of Eratosthenes remains one of the most efficient algorithms for finding all the prime numbers up to a specified limit. This directly relates to the algorithms simplicity and ability to generate primes for large limits while maintaining a relatively small computational complexity. Due to the efficiency of this algorithm, this makes it a preferred choice for prime number generation.

While the Sieve of Eratosthenes is one of the most efficient algorithms for finding primes, it can be seen that the Sieve is most efficient when computing a large set of primes. In turn, when a smaller set of primes needs to be calculated or checking smaller single numbers for primality, other algorithms such as the Miller-Rabin probabilistic test or the Trial Division algorithm shine in these cases.

Future research could explore possible optimizations to the sieve, such as using parallel computing to help increase efficiency. Additionally, exploring an algorithm that combines different prime-finding algorithms based on size and scale of the problem. Prime numbers are important in cryptography; studying prime generation in cryptography (and other related fields) could help promote new algorithms.

References

- https://en.wikipedia.org/wiki/Sieve_of_Atkin
- https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test#