Sieve of Eratosthenes Explained

Explain seive of erothesis

The **Sieve of Eratosthenes** is a simple and efficient algorithm to find all prime numbers up to a given number n. It works by iteratively marking the multiples of each prime number starting from 2. Here's how it works step by step:

Steps of the Algorithm

1. **Initialize the list**:
   - Create a boolean array `isPrime[]` of size $n + 1$.
   - Set all elements to `true`, assuming all numbers are prime initially.
   - Set `isPrime[0]` and `isPrime[1]` to `false` because 0 and 1 are not prime.
2. **Iterate through the numbers**:
   - Start with the first prime number, $p = 2$.
   - For each number p in the array:
     - If `isPrime[p]` is `true`, it is a prime number.
     - Mark all multiples of p (starting from p^2) as `false` since they cannot be prime.
3. **Repeat until $p^2 > n$**:
   - Once $p^2 > n$, the process stops because all non-prime numbers have already been marked.
4. **Collect primes**:
   - The indices of `true` values in `isPrime[]` are the prime numbers up to n.

Example

Find all primes up to $n = 10$.

1. Initialize:
   $isPrime = [false, false, true, true, true, true, true, true, true, true, true]$
   (Index 0 and 1 are `false`).

2. Start with $p = 2$:
   Mark multiples of 2: $4, 6, 8, 10$.
   $isPrime = [false, false, true, true, false, true, false, true, false, true, false]$.

3. Move to $p = 3$:
   Mark multiples of 3: 9.
   $isPrime = [false, false, true, true, false, true, false, true, false, false, false]$
   .

4. Stop at $p = 4$ because $4^2 > 10$.

5. Collect primes:
   Primes are $2, 3, 5, 7$.

Time Complexity

- **O(n log log n)**: Efficient for large n.
- Space complexity: **O(n)** for the boolean array.

This method is widely used for generating prime numbers in number theory problems due to its simplicity and efficiency. Let me know if you'd like an example in Java or need further clarification!

Explain the same when n=100

Let's use the **Sieve of Eratosthenes** to find all prime numbers up to $n = 100$.

Steps

1. **Initialization**:
   - Create an array `isPrime[]` of size 101 (indices 0 to 100).
   - Initially, set all values to `true`, except for indices 0 and 1, which are set to `false` because 0 and 1 are not prime.

   ```arduino
   isPrime = [false, false, true, true, true, ..., true]  // Array size: 101
   ```

2. **Marking Non-Primes**:
   - Start with $p = 2$, the first prime number.
   - Mark all multiples of 2 (starting from $2^2 = 4$) as `false`.
     Multiples of 2: $4, 6, 8, \ldots, 100$.
   - Updated `isPrime`:

     ```arduino
     isPrime = [false, false, true, true, false, true, false, true, false, ...,
     false]
     ```

   - Move to $p = 3$, the next prime.
     Mark all multiples of 3 (starting from $3^2 = 9$) as `false`.
     Multiples of 3: $9, 12, 15, \ldots, 99$.
   - Updated `isPrime`:

     ```arduino
     isPrime = [false, false, true, true, false, true, false, true, false, false,
     false, ..., false]
     ```

   - Continue to $p = 5$:
     Mark all multiples of 5 (starting from $5^2 = 25$) as `false`.
     Multiples of 5: $25, 30, 35, \ldots, 100$.
   - Updated `isPrime`:

     ```arduino
     isPrime = [false, false, true, true, false, true, false, true, false, false,
     ```

```
false, ..., false]
```

- Proceed to $p = 7$:
  Mark all multiples of 7 (starting from $7^2 = 49$) as `false`.
  Multiples of 7: $49, 56, 63, ..., 98$.
- Updated `isPrime`:

```arduino
isPrime = [false, false, true, true, false, true, false, true, ..., false]
```

- Stop at $p = 10$ because $10^2 = 100 > n$.

3. **Collect Primes**:
   - The indices of `true` values in `isPrime[]` represent the prime numbers up to 100.

Result

The prime numbers up to $n = 100$ are: **2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97**.

Efficiency

- **Time Complexity**: $O(100 \cdot \log(\log(100)))$, approximately linear for small n.
- **Space Complexity**: $O(101)$ for the `isPrime[]` array.

Let me know if you'd like further clarifications or an example implementation!

Understand the flow. Why it works

The **Sieve of Eratosthenes** works based on the fundamental properties of prime numbers and their multiples. Here's why it works step by step:

Key Idea: What is a Prime Number?

A prime number is a number greater than 1 that has no divisors other than 1 and itself.

- For example, 2, 3, 5, and 7 are prime because they cannot be expressed as products of smaller positive integers.
- Non-prime (composite) numbers, such as 4, 6, and 9, have divisors other than 1 and themselves.

Why Does the Algorithm Work?

1. Marking Multiples

- The algorithm marks all multiples of a prime number p as non-prime because they are not divisible only by 1 and themselves.
  - Example: For $p = 2$, multiples like $4, 6, 8, \ldots$ are marked since they are divisible by 2.
- When marking multiples, we start from p^2 because smaller multiples of p (like $p \times 1, p \times 2, \ldots$) have already been marked by smaller primes.

2. Stopping at $\sqrt{n}$

- It is sufficient to stop the process once $p^2 > n$ because any composite number n must have at least one factor $\leq \sqrt{n}$.
  - Example: For $n = 100$, $\sqrt{100} = 10$. All composite numbers ≤ 100 will have factors ≤ 10, so we only process $p = 2, 3, 5, 7$.

3. Preserving Primes

- The algorithm never marks a prime as non-prime. This is because:
  - We only start marking multiples from p^2, ensuring p itself is not affected.
  - Any composite number will be marked by one of its smaller prime factors before we reach it.

Example to Understand the Flow

Let's focus on $n = 10$ as a simplified example:

1. **Initial Array**:

   ```arduino
   isPrime = [false, false, true, true, true, true, true, true, true, true, true]
   ```

   (0 and 1 are not prime; assume all others are prime.)

2. **Step 1: Process $p = 2$**:
   - Start at $p^2 = 4$, mark multiples of 2:

     ```css
     isPrime[4], isPrime[6], isPrime[8], isPrime[10] = false
     ```

   Updated `isPrime`:

   ```arduino
   [false, false, true, true, false, true, false, true, false, true, false]
   ```

3. **Step 2: Process $p = 3$**:
   - Start at $p^2 = 9$, mark multiples of 3:

     ```arduino
     isPrime[9] = false
     ```

   Updated `isPrime`:

   ```arduino
   [false, false, true, true, false, true, false, true, false, false, false]
   ```

4. **Stop at $p = 4$** (since $4^2 > 10$).

5. **Collect Primes**:
   - Indices with `true` are $2, 3, 5, 7$, the primes up to 10.

Why It Works Mathematically

1. **All composite numbers get marked**:
    - Any composite number n has at least one prime factor $p \leq \sqrt{n}$. The multiples of p are systematically marked.
2. **Primes remain unmarked**:
    - A number is only marked as non-prime if it is a multiple of a smaller prime. This ensures primes remain `true` in the array.

Intuition

- Think of it as "crossing out" numbers:
    - Start with a clean slate where all numbers are prime.
    - Cross out numbers systematically by marking multiples of each prime.
    - What remains are the primes!

This systematic marking ensures efficiency and correctness. Let me know if you'd like any part clarified further!