Lecture 27: Prime Time

COSC 273: Parallel and Distributed Computing

Spring 2023

This Week

- Homework 03 Due Friday
- Final Project group & topic selection, also due Friday
  - Option 1: computing prime numbers ← today
  - Option 2: sorting ← Friday
  - Option 3: choose your own adventure

Computing Prime Numbers

Recall

The natural numbers $0, 1, 2, 3, \ldots$

Given natural numbers n, d:

- d **divides** n if $n = d \cdot q$ for some natural number q
  - q is the **quotient** of n and d
  - n is a **multiple** of d
- d is a **proper divisor** of n if it is a divisor and $d \neq 1, n$
- In Java
  - `(n % d == 0)` returns `true` if and only if `d` divides `n`

Proper divisors of 12? 13?

12: 2, 3, 4, 6

13: none

Prime Definitions

A natural number $p > 1$ is **prime** if it has no proper divisors. 13 is prime

- A natural number $n > 1$ that is not prime is **composite**

Examples:

13

- 2, 3, 5, 7, 11, 17, 19 are prime
- 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20 are composite

Who Cares?

- Mathematicians
  - primes are atomic building blocks of natural numbers
  - understanding how prime numbers are distributed is a central goal of number theory
- Computer Scientists
  - prime numbers are essential for RSA encryption
  - error correcting codes
- Everyone
  - RSA encryption is most widely used public key encryption
  - used for secure communication everywhere

Final Project, Option 1

The Task. Generate an array `int[] primes` that contains every prime number that can be stored as an `int` in Java in increasing order.

- `Integer.MAX_VALUE` = `2_147_483_647`
- there are $105,097,565$ primes up to this value
  - $\Longrightarrow \sim 400$ MB of primes!

Testing if a Number is Prime

Method 1: Trial Division

- Check all numbers less than n to see if n is divisble by them:

```
public boolean isPrime(int n) {
    if (n <= 1) return false;

    for (int d = 2; d < n; ++d) {
        if (n % d == 0)
            return false;
    }

    return true;
}
```

trial divisor

Example

Is 91 prime?

d = 2 : X

d = 3 : X

→ d = 4 : don't need to check!

d = 5 : X

→ d = 6 : ~~—————————~~

d = 7 : 91 = 7 * 13

Not prime

Is Trial Division Efficient?

Can we improve trial division?

- Do we have to check all possible divisors up to n−1?

– no: can omit composite #

$d = d_1 \cdot d_2$

d divides n?

Only need to check prime d

– stop @ $\sqrt{n}$

$n = d \cdot q$ w/ $d, q > \sqrt{n}$

$\Rightarrow d \cdot q > (\sqrt{n})(\sqrt{n}) = n$

Making Things More Efficient

Claim 1. If n is composite, then it has a divisor d with $d \leq \sqrt{n}$

Why?

Prev. slide

Making Things More Efficient

Claim 1. If n is composite, then it has a divisor d with $d \leq \sqrt{n}$

Why?

Conclusion. Only need to check divisors up to $\sqrt{n}$

A Faster Procedure

```
public boolean isPrime(int n) {
    if (n <= 1) return false;

    for (int d = 2; d * d <= n; ++d) {
        if (n % d == 0)
            return false;
    }

    return true;
}
```

Can Procedure Be Improved More?

Claim 2. If n is composite, then it has a *prime* divisor at most $\sqrt{n}$.

So we only need to check *primes* up to $\sqrt{n}$

Example:

- To determine if a number less than...
  - ...100 is prime, need only check divisibility by 2, 3, 5, 7
  - ...1,000 is prime, need only check divisibility by 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31
  - ...1,000,000 is prime, need only check divisibility by primes up to 1,000

Generating Primes

Suppose we want to generate *all* primes up to N...

How should we do this?

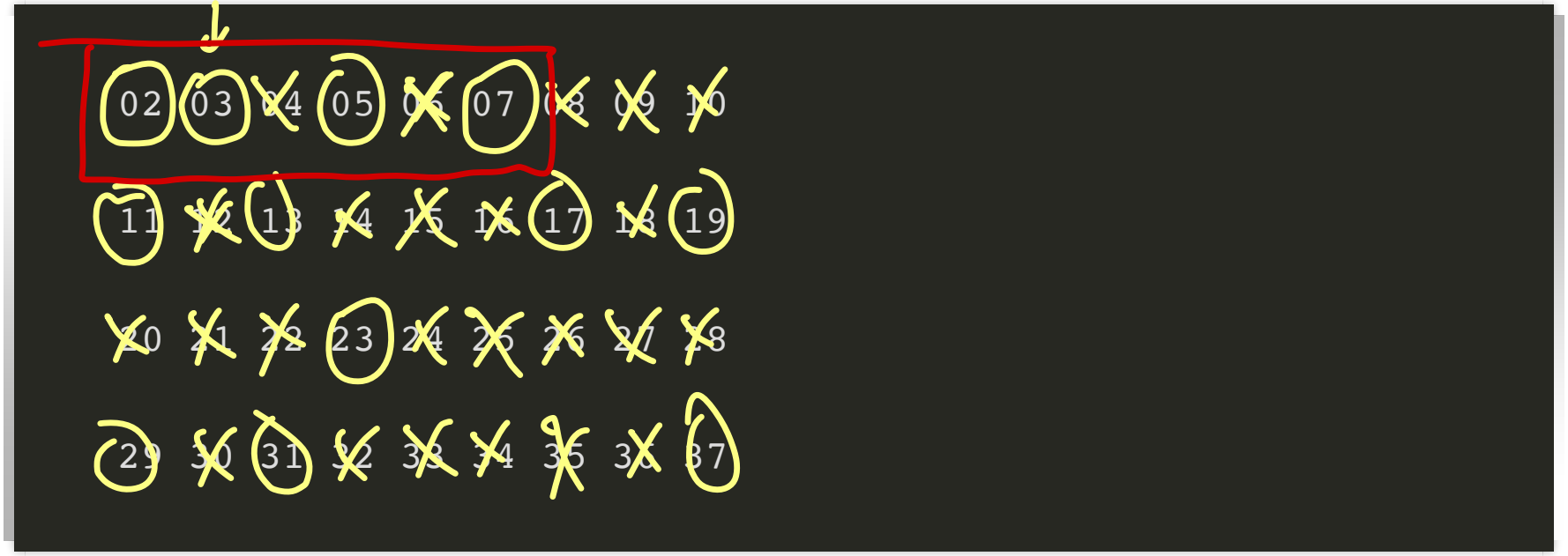
→ start w/ all #s

when find a prime p_1

remove mults. of p_1

Sieve of Eratosthenes

1. Write numbers 2 through `N`
2. Read numbers in order:
   - if a number is not crossed out, it is prime
   - then cross out all multiples



Observation/Optimization

In SoE, once we find primes up to `Math.sqrt(N)`, we can stop!

Why?

Eratosthenes in Code

1. Make boolean array `isPrime` of size `N`
   - interpretation: `isPrime[i] == true` if `i` is prime
2. Initialize `isPrime[i]` to `true` for all `i >= 2`
3. Iterate over indices `i` up to `Math.sqrt(N)`:
   - if `isPrime[i]`:
     - set `isPrime[j] = false` for all `j` that are multiples of `i`
   - otherwise, do nothing

When done: `isPrime[i]` is true precisely for prime `i`

Eratosthenes in Java

```
boolean[] isPrime = new boolean[N];
for (int i = 2; i < N; ++i) {
    isPrime[i] = true;
}
for (int i = 2; i < N; ++i) {
    if (isPrime[i]) {
        for (int j = 2 * i; j < N; j += i) {
            isPrime[j] = false;
        }
    }
}
```

Activity

Let's compute the primes up to $225 = 15^2$

To start, here are the primes up to 15:

- $2, 3, 5, 7, 11, 13$

Primes up to 225:

1–44: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 37, 41, 43

45–89: 47, 53, 61, 67, 71, 73, 79, 83

90–134: 97, 101, 103, 107, 109, 113, 127, 131

·90–134: 137, 139, 149, 157, 163, 167, 173, 179

· 135–179:

180–225:

Project Technical Challenges

1. Storing `boolean isPrime` of size `Integer.MAX_VALUE` is already on the order of 1GB of memory
2. How can we partition the problem to exploit parallelism?
   - multithreading?
   - vector operations?
3. How to *synchronize* between different sub-tasks?