Prime numbers

Jordi Cortadella

Department of Computer Science

Prime number

- A **prime number** is a natural number that has exactly two *distinct* divisors: 1 and itself.

```
         1  is not prime
         2  is prime
         6  is not prime
        11  is prime
      1023  is not prime
2110454939  is prime
```

Prime number: specification

```
// Pre:  n >= 0
//
// Returns true if n is prime,
// and false otherwise

bool isPrime(int n);
```

Strategy: try all possible divisors from 2 to n-1

Check that *n* is divisible by *d*: `n%d == 0`

Is it prime?

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Prime number

```
// Pre:  n >= 0
// Returns true if n is prime, and false otherwise

bool isPrime(int n) {

    if (n <= 1) return false;

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

    return true;
}
```

Is it fast enough?

Prime number: doing it really fast

- If *n* is not prime, we can find two numbers, *a* and *b*, such that:

$$n = a \cdot b, \qquad \text{with } 1 < a \leq b < n$$

and with the following property: $a \leq \sqrt{n}$

- There is no need to find divisors up to *n*-1. We can stop much earlier.

- Note: $a \leq \sqrt{n}$ is equivalent to $a^2 \leq n$

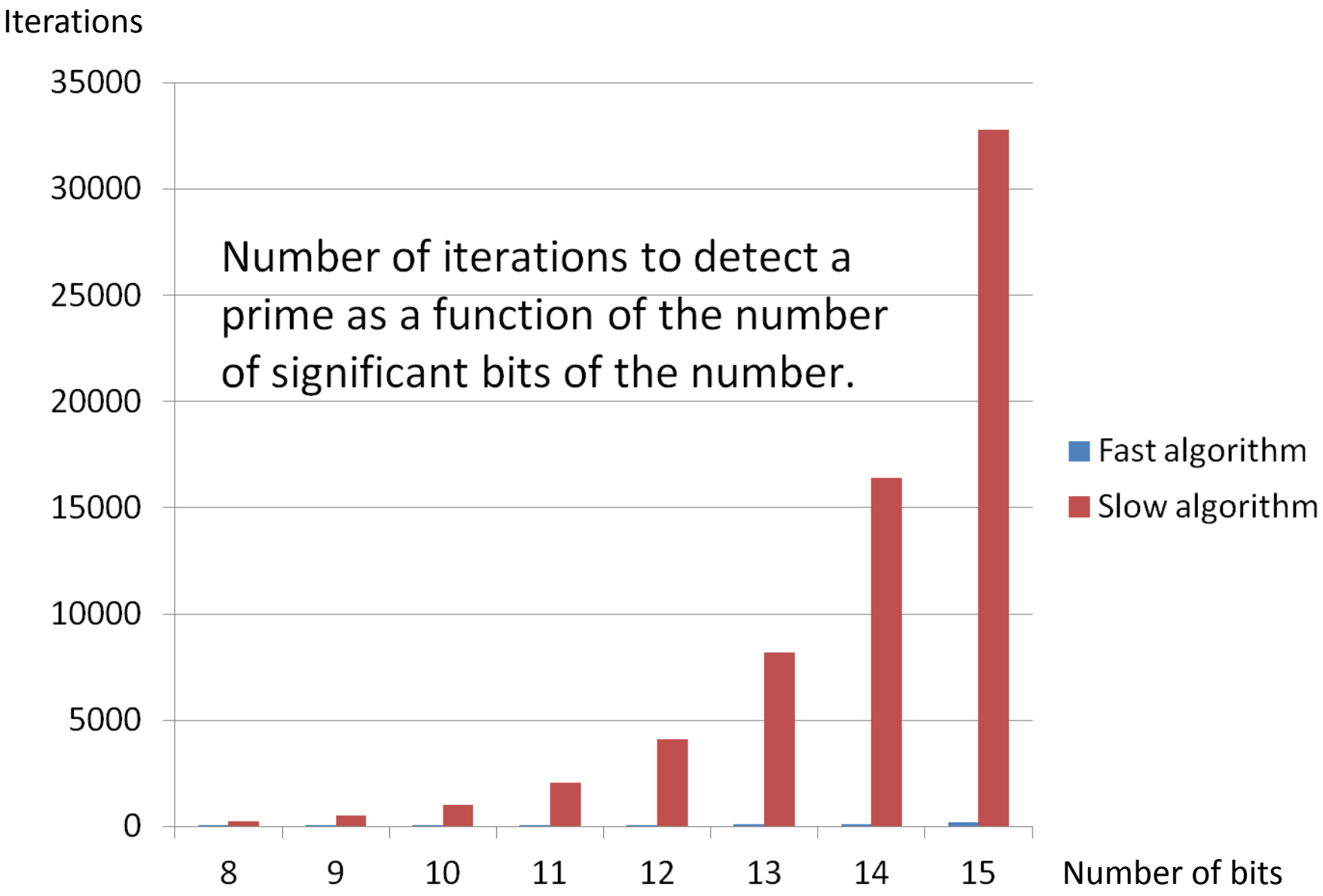
Is it prime?

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Prime number

```
// Pre:  n >= 0
// Returns true if n is prime or false otherwise

bool isPrime(int n) {

    if (n <= 1) return false;

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

    return true;
}
```

Is there any real difference?



In real time (n = 2110454939)

```
> /usr/bin/time prime_slow
2110454939
is prime
6.72 seconds
```

```
> /usr/bin/time prime_fast
2110454939
is prime
0.00 seconds
```

Prime factors

- Write a function that prints the decomposition of a number in prime factors
  - Example:
    input: 350
    output: Factors of 350: 2 5 5 7

- Intuitive algorithm:
  - Try all potential divisors d, starting from 2
    - If divisible by d, divide and try again the same divisor
    - If not divisible, go to the next divisor
  - Keep dividing until the number becomes 1

Prime factors

n	d	divisible	write
350	2	yes	2
175	2	no	
175	3	no	
175	4	no	
175	5	yes	5
35	5	yes	5
7	5	no	
7	6	no	
7	7	yes	7
1	**finish**		

The algorithm will never write a non-prime factor. Why ?

Prime factors

```
// Pre: n > 1
// Prints the decomposition of n in prime factors

void print_prime_factors(int n) {
    int d = 2;  // Variable to store divisors
    cout << "Factors of " << n << ":";

    // Divide n by divisors from 2 in ascending order
    while (n != 1) {
        if (n%d == 0) { // Check if divisible
            cout << " " << d;
            n = n/d;
        }
        else d = d + 1;
    }
    cout << endl;
}
```

Conclusions

- Many algorithms exist to check for primality, but few of them are simple and efficient.
- Use your knowledge about the problem to figure out how to solve it efficiently.
- If you want to find all prime numbers up to a limit, use the *sieve of Eratosthenes*.

  http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes