Session 02 - Primes

Review

for i← 72 to 3n+19 step 5 //O(n)

for j ← n … i

f(); // O(1)

for i← 1 to n //O(n)

for j ← n … i // O(n)

f(); // O(1)

n + (n-1) + … 1 = n(n+1)/2 = n2 + n = O(n2)

//O(n) Ω(1)

isPrime(n)

for i← 2 to n-1

if n mod i == 0

return false

end

return true

end

isPrime(n)

for i← 2 to sqrt(n)

if n mod i == 0

return false

end

return true

end

Eratosthenes

density of primes at n O(log n)

O(n log log n)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 1 | 1 | 0 | 1 | 00 | 1 | 0 | 0 | 00 | 1 | 00 | 1 | 0 | 00 | 0 | 1 | 00 | 1 | 00 | 0 |

eratosthenes(n)

isPrime ← new boolean[n+1]

isPrime[\*] ← true

for i ← 2 to n

if isPrime[i]

for j ← 2\*i to n step i for (j = 2\*i; j <= n; j += i) n/2 n/3 +n/5 +n/7 +

isPrime[j] ← false

end

end

end

end

Improved eratosthenes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

imagine a larger prime: 97 97\*2 97\*3 97\*4 97\*5

every prime number except 2 is odd

odd \* odd → odd 97\*97 → odd + 97 → even

//O(n log log n)

BetterEratosthenes(n)

isPrime ← new boolean[n]

isPrime[\*] ← true

for i ← 4 to n step 2

isPrime[i] ← false

for i ← 2 to n-1

if isPrime[i]

for j ← i\*i to n step 2i

isPrime[j] ← false

end

End

end

Prime number wheel

3579

Bits[0] = 1110110110…….

Bits[1] =

Bits[2] =

Bits[105] = 1110110110……. = bits[0]

Bits[106] = bits[1]

a=100000000000000

b=100000000050000

long a, b

int[]x = new int[1000000];

for (long i = a; i <= b;...)

x[(int)(i-a)]

10000000000000 10000000000005

2 sqrt(10000000000005)

# 3 Great Algorithms

Greatest Common Denominator (GCD)

Lowest Common Multiple (LCM) 24, 36

Power(xn)

Greatest Common Denominator (GCD) Greatest Common Factor (GCF)

//O(min(a,b))

gcd(a,b)

m ← min(a,b)

biggest = 1

for i← 2 to m

if a mod i == 0 and b mod i == 0

biggest = i

end

end

return biggest

end

int gcd(int a, int b) {

for i = min(a,b); i >= 2; i--) {

if (a % i == 0 && b % i == 0)

return i;

}

return 1;

}

gcd(12,18)

gcd(1096, 14282)

# Euclid : 500BC (2500 ago)

gcd(a,b) = gcd(b, b mod a)

gcd(a, 0) = a

gcd(3025, 1025)

gcd(12, 18) a = 12, b = 18 → a = 18, b = 6 → a = 6, b = 6 mod 18 ⇒ 6, (6,0)

gcd(12, 18) = gcd(18, 12 mod 18) = gcd(18, 12) = gcd(12, 18 mod 12) =

gcd(12, 6) = gcd(6, 12 mod 6) = gcd(6, 0)

gcd(3025, 1025)

gcd(1025, 975) 1025 MOD 3025

gcd(975, 50) gcd(3025,1500)

gcd(50, 25)

gcd(25, 0)

1,1,2,3,5,8,13,21,34,55 Worst case:Fibonacci in reverse logФ

gcd(3025, 3024)

gcd(3024, 1)

gcd(3025, 2)

gcd(2, 1)

O(logn)

int gcd(int a, int b) {

while (b != 0) {

int temp = a % b;

a = b;

b= temp;

}

return a;

}

gcd(a,b)

if b == 0

return a

end

return gcd(b, a mod b)

end

lcm(a,b) lowest common multiple

lcm(12, 18) = 36

12 = 2\*2\*3 18 = 3\*3\*2

What is the complexity to factor(n)?

N = 28

1, 2,4 sqrt(n) 7,14, 28

factor(n)

for i ← 2 to sqrt(n)

If n mod i == 0

Add to list of factors

End

end

return list

2\*3 in common \*3 \*2

//O(log n)

lcm(a,b) = a \* b / gcd(a,b) // 1 + log(n)

# Big Arithmetic

n=4

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 9 | 8 | 7 |

+

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 4 | 3 | 5 |

= // answer is O(n+1) size

// 3n = O(n)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 2 | 2 |

n=4

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | 9 | 9 | 9 |

\*

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 4 | 3 | 5 |

=

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 4 | 9 | 9 | 9 | 5 |
|  |  |  | 9 | 9 | 7 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

//size = O(2n)

// O(n2+n) = O(n2) → O(n log n) using convolutions (we will do this with matrices)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 9 | 9 | 5 |

ab = a \* a \* a \* a \* a….

5555555555555555555552

25555555555555555555555555555555555

O(2bab)

//raise x to the power n

bruteforcepower(x, n)

prod ← 1

for x ← 1 to n

prod ← prod \* x

end

return prod

end

x17 = x1x16

x16 = (x8)2

x8 = (x4)2

x4 = (x2)2

x2 = x \* x

//O(log n)

**//raise x to the power n**

**power(x, n)**

**prod ← 1**

**while n > 0**

**if (n mod 2 != 0)**

**prod ← prod \* x**

**end**

**x ← x \* x**

**n ← n / 2**

**end**

**return prod**

**end**

power(2, 11)

prod = 1\*2\*4\*256

x = 256

n = 1

# 198274192847192875192857192857192857198271

# Fermat

cn!= an+bn n > 3 last theorem → useless movie: Fermat’s Room

“little” theorem for any prime p ap-1 mod p == 1

prime candidate p

witness a, 1 < a < p

if p is prime ⇒ ap-1 mod p == 1

if ap-1 mod p == 1⇒ p is ***probably prime***

if ap-1 mod p != 1⇒ p is ***definitely NOT prime***

p = 12847192875177

a = 2

ap-1 → gigantic

n! 5! = 1\*2\*3\*4\*5 = 120

6!=720

10!

n = 1012 1012! is completely intractable

give me the last 6 digits of 1012! = 000000

10! 3628800

(a\*b) mod p = (a mod p) \* ( b mod p) mod p

**//xn mod m**

**powermod(x, n, m)**

**prod ← 1**

**while n > 0**

**if (n mod 2 != 0)**

**prod ← prod \* x mod m**

**end**

**x ← x \* x mod m**

**n ← n / 2**

**end**

**return prod**

**end**

Fermat(p, k)

for i ← 1 to k

a ← random(2, p-1)

if powermod(a, p-1, p) != 1

return false

end

end

return true (probably)

end

# Carmichael Number

composite (not a prime)

square free

first Carmichael number is 561 = 3\*11\*17

Will always return true for Fermat test UNLESS the witness is one of the factors

isPrime(561)

100000000000000000000000000000000000000001(124121100000000000000000000000001)

# Miller-Rabin

x2 mod m == 1 (x mod m)(x mod m)



11000000

a mod 11000001 == 1

<https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>

n = 221 prime or not?

d = n-1 = 220 = 128 + 64 + 16 + 4

220 bits = 11010100

2sd d = 55, s = 2

MillerRabin(n, k)

WitnessLoop: for i ← 1 to k // k trials  
 a← random[2, *n* − 2]  
 *x* ← *ad* mod *n*  
 **if** *x* = 1 or *x* = *n* − 1 **then**  
 **continue** WitnessLoop  
 for j ← 1 to *s* − 1  
 *x* ← *x*2 mod *n*  
 **if** *x* = 1 **then**  
 **return** false *composite (Carmichael?)*  
 **if** *x* = *n* − 1 **then**  
 **continue** WitnessLoop  
 **end**

**return** false (*composite)*

*end*  
 **return** *probably prime*

*end*

# Agrawal, Kayal, Saxena, AKS

Proved deterministically, that isPrime(p)

1977. Rivest, Shamir, Adelman, RSA is based on n = pq

IBM announced that Quantum computers would be capable of cracking RSA within 5 years.

Discrete Logarithm (Susanne Wetzl)

Elliptical Integrals

p, q 4096 bits

n = 8192 bits

factor(n) O(sqrt(28192)) = 4096 bits

1, **2**, **3**, 4, **5**, 6, **7**, 8, 9, 10

GenRandomNumber()

do

p ← random(24000, 24200) // make sure 1 at the end...

while not MillerRabin(p)

return p

end

<https://en.wikipedia.org/wiki/RSA_(cryptosystem)>

p← 61, q ← 53

n ← p q = 3233

1 < *e* < 3120 e ← 17

d e mod

d← 2753

<https://en.wikipedia.org/wiki/Modular_multiplicative_inverse>

c ← me mod n

m ← cd mod m

m=65 ‘A’

c = m17 mod 3233 → 2790

27902753 mod 3233 → 65

RSA cannot be used to encrypt messages because it is subjection to plaintext attack.

That is, **if you know the text being encrypted, you can figure out the key.**

1. pick a random number
2. Use RSA to exchange the random number with both parties (Diffie-Hellman key exchange)
3. Use the random number for symmetric encryption (today, AES-256, Ringdael)
4. https: works

grid xxx weeather yyyy

David Kahn The Codebreakers

Bruce Schneier Cryptography

OpenSSL

# Prime Number Wheel

countPrimes(a, b) {

if (a mod 2 == 0)

a ++;

for i ← a to b step 2

if isPrime(i)

}

boolean isPrime(p)

for i ← 3 to sqrt(p) step 2

what about 2,3

2 \* 3 = 6

2, 3

6, 7, 8, 9, 10, 11 12,13,14,15

n mod 6 = 0 NOT PRIME multiple of 2,3

n mod 6 = 1

n mod 6 = 2 NOT PRIME multiple of 2

n mod 6 = 3 NOT PRIME, multiple of 3

n mod 6 = 4 NOT PRIME multiple of 2

n mod 6 = 5

for k ← 6 to n step 6

isPrime(k+1) isPrime(k+5)

<https://en.wikipedia.org/wiki/Wheel_factorization>

countPrimes(n1, n2)

count ← 0   
 for i ← n1 to n2 step 2

if isPrime(i)

count++

end

isPrime(n)

for i ← 3 to sqrt(n) step 2

if n mod i == 0

return false

end

end

2, 3, 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

n mod 6 = 0 NOT PRIME

n mod 6 = 1

n mod 6 = 2 MOD 2 NOT PRIME

n mod 6 = 3 MOD 3 NOT PRIME

n mod 6 = 4 MOD2 NOT PRIME

n mod 6 = 5

2, 3, 5 = 30

HW: For n < 232 print out prime or not prime using Miller-Rabin

100730001

your program prints out

prime or not prime

Next time:

bubble sort

selection sort

insertion sort

quicksort

heapsort

merge sort

shuffling

searching (linear, binary, golden mean)

n elements

x = [ 1 2 3 4 5 6 7 8 9 ]

n! orders

entropy has something to do with this! ***It is easier to disorder than to order….***

x = [ 5 1 9 2 8 7 3 4 6]

x = [ 1 5 2 8 7 3 4 6 9]

void sort(int [] x) {

for (int j = 0; j < x.length-1; j++)

for (int i = 0; i < x.length-1; i++) //O(n)

if (x[i] > x[i+1]) {

int temp = x[i];

x[i] = x[i+1]

x[i+1] = temp;

}

Tricks for High Speed Bitvector implementations

initialize all n bits n

zero out all even bits n/2

zero out all mult. 3 n/3

n/5

1. Don’t write out even bits at all. map odd numbers using n/2 → n >> 1

1 3 5 7 9

0 1 2 3

wheel 2 \* 3 \* 5 \* 7 = 210

11 12 13 14 15 16 17 18 19 20 21 22

1 0 1 0 0 0 1 0 1 0 0 0

210 relative prime to 64

2\*105 2^6

123456789

011010100010101010101010101010101010101010101010

65 66 67

101010101010101010101010101010101010101010101010

HW:

Eratosthenes Sieve (improved version) j = i\*i stepping 2i

Use Bitvector. You can write your own, or you can use a library.

Your input is two numbers a and b from standard input

100000000000000 100000000050000

a,b

in order to solve, first compute prime numbers up to????

Eratosthenes (sqrt(b))

Eratosthenes store an array of bits

0 1 2 3 4 5 6

a a+1 a+2

eratosthenes(long a, long b) {

// allocate a bit vector with the right number of bits

// if you were using an array of boolean this would work:

new boolean[(int)(b - a)] ; // notice b-a is long, must convert to int!!!

I WILL PAY ADDITIONAL 200 points to anyone who implements the block-wheel implementation described above

Announce algorithm challenge: