Number Theory in CS

Hmm interesting font

Similar to the sorting lecture, I'll leave most of the more difficult mathematical proofs of theorems to the end of class. Feel free to leave before them if you're not interested or stick around for some cool number theory

GCD and LCM

Greatest common divisor

Euclidean algorithm: gcd(a, b) = gcd(b%a, a) (% means mod)

Psuedocode:

```
gcd(int a, int b) // assuming a < b
    If (a == 0)
        Return b
    Else
        Return gcd(b%a, a)
```

Complexity: O(log(min(a, b))

Least common multiple: just use the fact that gcd(a, b) * lcm(a, b) = a*b, so lcm(a, b) = a*b/(gcd(a, b))

If you know the prime factorizations of a and b, you can also find gcd and lcm by taking the min/max of each prime power in a and b

Example Euclidean Algortihm

gcd(268, 1004)
= gcd(200, 268)
= gcd(68, 200)
=gcd(64, 68)
=gcd(4, 64)
=gcd(0, 4)
= 4

Quick detour into mods

a = b (mod m) means a and b have the same remainder when divided by m

For instance, 4 = 24 (mod 5)

If you take a mod equation and do a bunch of multiplication / addition to both sides, stuff is still true

4 * 3 = 24 * 3 (mod 5) (12 = 72 (mod 5))

4 *3 + 4 = 24 * 3 + 4 (mod 5) (16 and 76)

You can also replace numbers with numbers equivalent to them in the mod and get true equations still

For instance, 4 * 7 = 3*6 + 10 (mod 5), and thus 4 * 2 = 3 * 1 + 0 (mod 5)

Primality Testing

Miller-Rabin: based on two fundamental ideas

- For all primes p, $a^{p-1} = 1 \pmod{p}$; the converse DOES NOT HOLD
  - This is called Fermat's Little Theorem
- If $x^2 = 1 \pmod{m}$ and $x \neq 1, -1 \pmod{m}$, then m is NOT prime

For testing a number n: let $n-1 = 2^t * u$, where u is odd. Check from i = 0 to t-1,

- If $a^{2^{(i+1)}} = 1 \pmod{n}$, and $a^{2^i} \neq -1, 1 \pmod{n}$, then n is composite

And also if $a^{n-1} \neq 1 \pmod{n}$, then n is composite

It turns out that if n is composite, 75% of the time, we will find that n is composite. But a can be pretty much any number, so if we choose 25 values for a, then the chances of us saying n is prime when it's not is $1/2^{50}$, which is so small that it will never happen

Miller Rabin pseudocode and Complexity

```
u = n-1
t = 0; while (u%2 == 0) { u = u/2; t=t+1 }
Bool prime = true
Repeat 25 times
    a = randint();
    If (gcd(a, n) != 1)
        Well that just means n isn't prime so gg
    Else
        X = a^u%n
        For i=0 to t-1
            If x^2 % n == 1 and x%n != -1, 1
                Composite, prime = false
            X = (x*x) % n
        If x%n != 1
            Composite, prime = false
Return prime
```

Complexity: k(log n)^3

K is 25 here

One log n from i = 0 to t-1

The others: apparently multiplying two numbers in mod n for large enough n is $(\log n)^2$ time

Not quite sure why but big numbers go brr

Does the number 561 pass the Miller-Rabin test?

Solution

Using base 2, let $561 - 1 = 35 \times 2^4$, which means $m = 35$, $k = 4$, and $a = 2$.

Initialization:	$T = 2^{35} \bmod 561 = 263 \bmod 561$	
$k = 1$:	$T = 263^2 \bmod 561 = 166 \bmod 561$	
$k = 2$:	$T = 166^2 \bmod 561 = 67 \bmod 561$	
$k = 3$:	$T = 67^2 \bmod 561 = +1 \bmod 561$	$\rightarrow$ **a composite**

Binary exponentiation

Fast way to compute a^b (mod c): we a, a^2, a^4, a^8 … (mod c) and multiply together the ones in the binary representation of b

```
Answer = 1
While b > 0
    If b%2 == 1
        Answer = (answer * a)%c
    b = b/2
    a = (a*a)%c
```

Time complexity: O(log b) (we need that many operations to get all the squared powers, then we just multiply some of them together)

Binary exponentiation example

3^{11} (mod 5): 11 in binary: 1011

1 * 3 = 3 (mod 5), after this: a at 3^2 = 4 (mod 5); 1011

3 * 4 = 2 (mod 5), after this: a at 4^2 = 1 = 3^4(mod 5); 1011

Result still at 2: a at 1^2 = 1 = 3^8 (mod 5); 1011

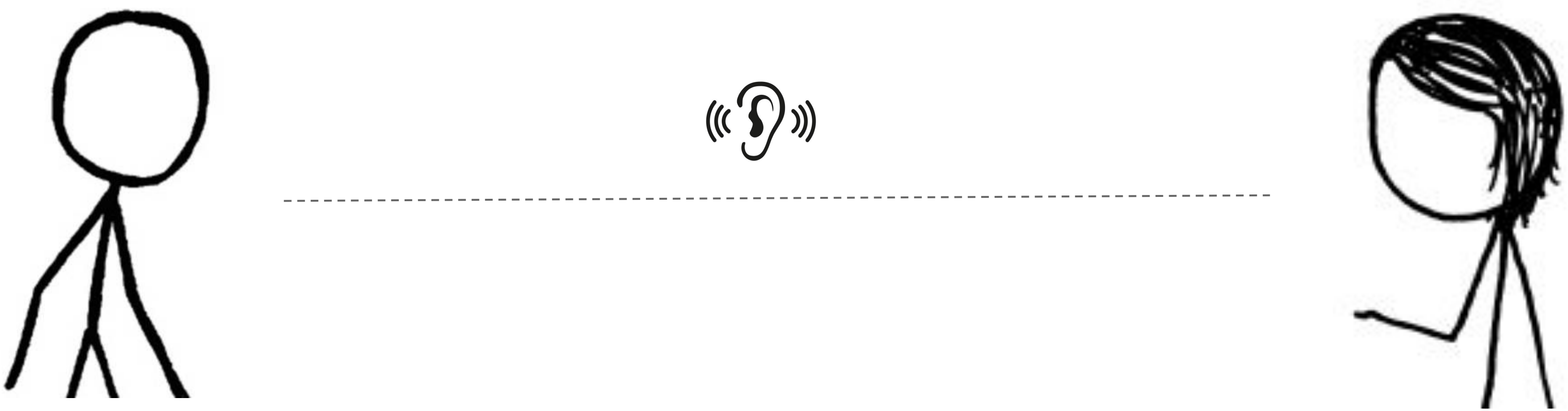
2 * 1 = 2 (mod 5); 1011

Final answer 2

We did $3^1 * 3^2 * 3^8$ (mod 5)

RSA

How do Cueball and Megan communicate privately while not being able to develop a strategy beforehand?



RSA: THE DETAILS

Choose two large primes p, q

N = pq, e is some usually predetermined number (I think 65537 is standard)

Private key: number d such that e*d = 1 (mod (p-1)(q-1))

Encoding the message: take C = M^e (mod n) and send it over

Decoding the message: take C^d (mod n)

RSA: WHY IT WORKS

Given n, it's incredibly difficult to find pq (you basically have to brute force)

Why does $M^{de} = M \pmod{n}$?

- **Euler's Theorem**: $x^{phi(m)} = 1 \pmod{m}$ if $gcd(x, m) = 1$
  - We don't have to worry much about phi(m): just know that $phi(pq) = (p-1)(q-1)$
- Then $x^{de} = x^{y * (p-1)(q-1) + 1} = (x^{(p-1)(q-1)})^y * x = 1^y * x = x \pmod{pq}$

So we get the original message back!

The steps of taking exponent are done quickly using binary exponentiation

RSA: Computing modular inverse of e

Extended Euclidean algorithm

For any a, b, there exist integers x, y, such that ax + by = gcd(a, b) and the extended euclidean algorithm lets us find this (x, y)

```
Extended Euclid (a, b) // returns (x, y)
    If a== 0
        Return (0, 1)
    b= ak + r (division)
    (x,y) = Extended Euclid(r, a)
    Return (y-kx, x)
```

RSA: Computing modular inverse of e intuition

What we're essentially doing is back-substituting: consider gcd(34, 20)

Euclidean Algorithm

34 = 20 * 1 + 14

20 = 14 * 1 + 6

14 = 6 * 2 + 2 ← the gcd

6 = 2 * 3 + 0

Extended Part

2 = 14 - 2*6

= 14 - 2*(20 - 14*1) = 3*14-2*20

= 3*(34-20*1) - 2*20

= **3*34 - 5*20**

RSA FINAL SLIDE

We can find d by finding (x,y) such that x*e + y*(p-1)(q-1) = 1 (assuming that gcd(e, (p-1)(q-1)) = 1)

Then x*e = multiple of (p-1)(q-1) + 1, so x*e = 1 (mod (p-1)(q-1)

To recap

1. Person 1 generates n = pq, e and sends (n, e). They then use the Extended Euclidean algorithm on e and (p-1)(q-1) to find d
2. Person 2 encodes their message M by taking M^e (mod n) and sends it to person 1
3. Person 1 decodes the message by taking $(M^e)^d$ (mod n), getting back M

Prime factorization

Recall that primes are numbers p with only 1 and p as factors

Every integer has a unique prime factorization (product of prime powers)

120 = 2^3 * 3^1 * 5^1

Basic O(sqrt n) method

Essentially, brute force for the prime divisors of n

===

Divide out all the 2’s

Loop over 3, 5, 7, … ~sqrt(n) and for each value divide n by that # until you can’t anymore

If the leftover is over 1, it’s a prime

84 -> 2^2 * 21 -> 2^2 * 3 * 7

123 -> 3 * 41

27 -> 3^3

Basic O(sqrt n) method analysis

Say that i is the thing doing the iterating.

If a prime p divides n, n keeps those powers until i gets up to p; then we remove all of them

By going 2, 3, 5, 7, … sqrt(n), we remove all the powers of primes <= sqrt(n)

There can only be one prime divisor > sqrt(n), and it can only have exponent 1: otherwise, the product would be > sqrt(n)^2 = n

Thus, the thing left must be a prime

The number of times we have to divide is at most O(log n), since each time we divide n by something at least 2, and we iterate over $O(\sqrt{n})$ elements, so $O(\log n + \sqrt{n}) = O(\sqrt{n})$

Sieve of Eratosthenes method – precomputation

Essentially, the lowest thing not marked yet is a prime, and then we mark all multiples of that

Complexity: O(n log(log(n))

=================================

Keep array lowest_divisor[n] that returns the lowest prime divisors

Iterate i from 2 to n

    If lowest_divisor[i] == 0

        Iterate j multiples of i from min(i^2, n+1) to n, set lowest_divisor[j] = i if it’s 0 before

Sieve of Eratosthenes method

Now that we have the lowest_divisor array, we can just go through and repeatedly take out the lowest divisor

==

```
While (n > 1)

    Add lowest_divisor[n] onto prime factorization

    n = n / lowest_divisor[n]
```

This part takes O(log n) time, since you’re dividing by at least 2 each time

Total time is O(nlog(log(n)) + qlogn), where q is the number of queries of prime factorizations

	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	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

Assorted Proofs

Aka stuff that I knew or was able to find within like 20 minutes of surfing google and college websites

Proof of Euclidean Algorithm

Say that a < b; we’ll use the notation x | y means y is divisible by x

Let b = aq + r, 0 <= r < a (this is division)

gcd(a, b) | b, gcd(a, b) | aq, => gcd(a, b) | b - aq = r; since gcd(a, b) | a and gcd(a, b) | r, gcd(a, b) | gcd(a, r)

gcd(r, a) | r, gcd(r, a) | aq, => gcd(r, a) | aq+r = b => gcd(r, a) | gcd(a, b)

Thus gcd(a, b) = gcd(r, a) (if they divide each other and are both positive, then they’re the same value

Proof of complexity of Euclidean Algorithm

Say we have gcd(a, b), a < b; we'll prove that at each point in the process, the remainder b%a is less than b/2

If b < 2a, then our next step takes us to gcd(b-a, a), with b-a < b - b/2 = b/2

If b > 2a, then our next step takes us to gcd(b%a, a) with b%a < a < b/2

If b = 2a, then we only have one more step

This means every two steps, the larger number is halved. After one step, the smaller number is the larger number, so it takes about 1 + 2log(min(a, b)) time, thus O(log(min(a, b))

Proof of extended euclidean algorithm

Recursive algorithms lend themselves well to proofs by induction. We'll induct on min(a, b)

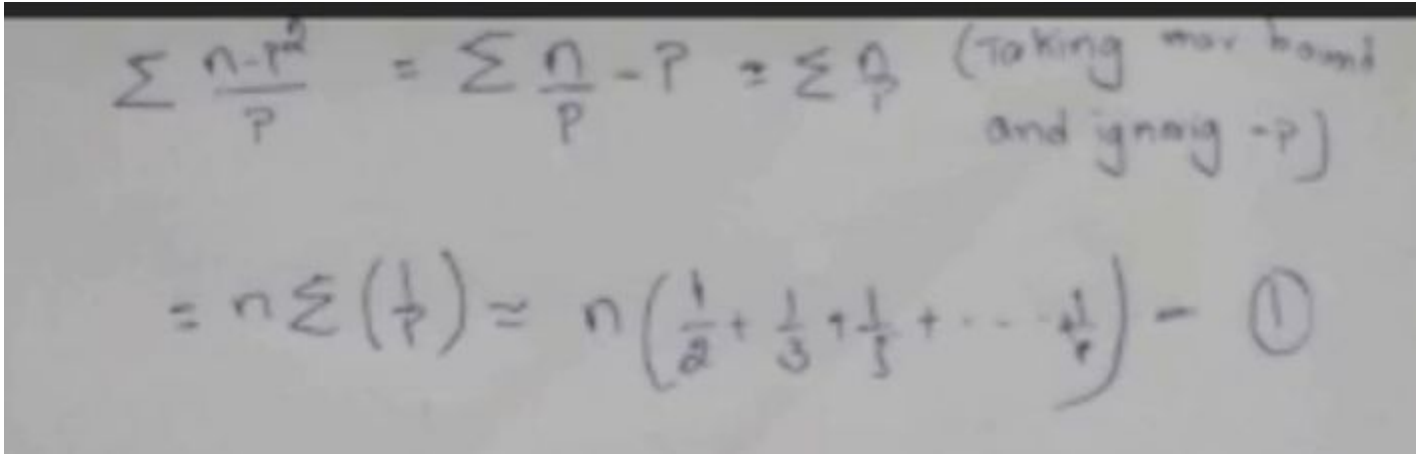
Obviously if a == 0, 0*0 + 1*b = gcd(0, b) = b, so the base case works

Assume that the function returns correct x, y for all lower min(a, b). We'll show it gives working integers for (a, b)

If xr + ya = d, then substituting, x(b-ak) + ya = d, so a(y-kx) + bx = d

Recall that gcd(a, b) = gcd(r, a) = d

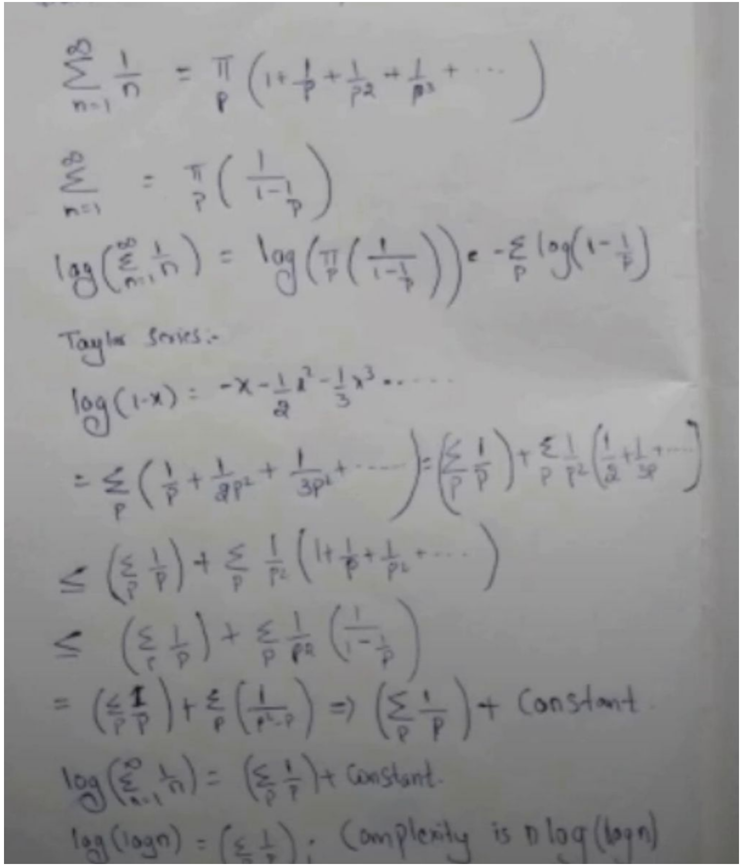
Sieve of Eratosthenes time complexity proof



For each prime p, we go through less than (by about a constant factor) n/p of its multiples: thus, we need the complexity of the sum of the reciprocals of primes up to n

You can prove rigorously using formulas for the sum of the prime numbers under n that the -p term does not matter to the overall complexity

Sieve of Eratosthenes time complexity proof



$$\sum_{n=1}^{\infty} \frac{1}{n} = \prod_p \left(1+\frac{1}{p}+\frac{1}{p^2}+\frac{1}{p^3}+\cdots\right)$$

$$\sum_{n=1}^{\infty} = \prod_p \left(\frac{1}{1-\frac{1}{p}}\right)$$

$$\log\left(\sum_{n=1}^{\infty} \frac{1}{n}\right) = \log\left(\prod_p \left(\frac{1}{1-\frac{1}{p}}\right)\right) = -\sum_p \log\left(1-\frac{1}{p}\right)$$

Taylor series:-

$$\log(1-x) = -x-\frac{1}{2}x^2-\frac{1}{3}x^3\ldots$$

$$= \sum_p \left(\frac{1}{p}+\frac{1}{2p^2}+\frac{1}{3p^3}+\cdots\right) = \left(\sum_p \frac{1}{p}\right)+\sum_p \frac{1}{p^2}\left(\frac{1}{2}+\frac{1}{3p}+\cdots\right)$$

$$\le \left(\sum_p \frac{1}{p}\right)+\sum_p \frac{1}{p^2}\left(1+\frac{1}{p}+\frac{1}{p^2}+\cdots\right)$$

$$\le \left(\sum_p \frac{1}{p}\right)+\sum_p \frac{1}{p^2}\left(\frac{1}{1-\frac{1}{p}}\right)$$

$$= \left(\sum_p \frac{1}{p}\right)+\sum_p \left(\frac{1}{p^2-p}\right) \Rightarrow \left(\sum_p \frac{1}{p}\right) + \text{Constant}$$

$$\log\left(\sum_{n=1}^{\infty} \frac{1}{n}\right) = \left(\sum_p \frac{1}{p}\right) + \text{Constant}$$

$\log(\log n) = \left(\sum \frac{1}{p}\right)$; Complexity is n log(log n)

- Choose one value from each parentheses for the prime power; for instance, for 1/(2^2*3), it's 1/(2^2) * 1/3 * 1 * 1 * 1….
- geometric series formula
- taking log of both sides and using log(ab) = log a + log b
- using the fact that log(1-x) = -x - ½ x^2 - 1/3x^3 + … for 0 < x < 1

- plugging in 1/p as x to the above and separating the first terms; **the next bit of work is to show that the other part converges, or is bounded by a constant**
  - ½ < 1, ⅓ < 1, etc obviously
  - again geometric series formula
  - the sum of the 1/(p^2 - p) is less than sum of 1/(k^2-k) for all integers k, which converges by telescoping: it's 1/(k-1) - 1/k + 1/k - 1/k+1 + … , so it's equal a constant
- **the fact that complexity of 1/1 + 1/2 + … + 1/n = log n**

Proof of two propositions behind Miller-Rabin

If p is prime, $a^2 = 1 \pmod{p}$ means $p \mid a^2-1$, or $p \mid (a-1)(a+1)$

Since p is prime, this can only happen if $p \mid a-1$ or $p \mid a+1$, so $a = 1, -1 \pmod{p}$

To prove Fermat's Little Theorem quickly: note that {1, 2, … p-1} are all relatively prime to p. Also, {a, 2a, … a(p-1)} are all distinct mod p since otherwise ia = ja —> (i-j)a = 0 —> i = j

Since they're the same numbers, $1 * 2 * \ldots * p-1 = a * 2a * \ldots * a(p-1) = a^{p-1} * 1 * \ldots * p-1$ —> $a^{p-1} = 1 \pmod{p}$

Proving the ¾ number is way way too hard for a short lecture