# COMP 446 / 546 ALGORITHM DESIGN AND ANALYSIS

## LECTURE 5 NUMBER THEORETIC ALGORITHMS

ALPTEKİN KÜPÇÜ

Based on slides of Juris Viksna, Yücel Yemez, and Shafi Goldwasser

# RANDOMIZED ALGORITHMS

- **Last time**
  - Las Vegas: Randomized Quicksort
- **Today**
  - Monte Carlo: Primality Testing

# BASIC NUMBER THEORY

- **Theorem: Let *a, b, c* be integers.**
  - If $a \mid b \;\wedge\; a \mid c$ then $a \mid (b + c)$
  - If $a \mid b$ then $a \mid bc \quad \forall c$
  - If $a \mid b \;\wedge\; b \mid c$, then $a \mid c$

- **Proof:**
  - **1.** If $a \mid b \;\wedge\; a \mid c$ then $\exists k_1, k_2$ integers s.t. $b = k_1 a \;\wedge\; c = k_2 a \Rightarrow$ $b + c = k_1 a + k_2 a = (k_1 + k_2)a$ where $(k_1 + k_2)$ is an integer $\Rightarrow a \mid (b + c)$
  - **2.** *??*
  - **3.** *??*

# BASIC NUMBER THEORY

- **Definition: If $a \equiv b \pmod{m}$, then $a$ is *congruent (equivalent)* to $b$ modulo $m$. Furthermore, we have:**
  - $a \equiv b \pmod{m} \leftrightarrow (a \bmod m) = (b \bmod m)$
  - $a \equiv b \pmod{m} \leftrightarrow m \mid a-b$
  - $a \equiv b \pmod{m} \leftrightarrow \exists k \in Z\ a = b + km$

- **The set of all integers congruent to $a$ modulo m constitutes a congruence (equivalence) class.**
  - Remember, there are m pairwise-disjoint equivalence classes modulo m.

# BASIC NUMBER THEORY

- **Theorem: If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then**
  - $a + c \equiv b + d \pmod{m}$
  - $ac \equiv bd \pmod{m}$

- **Proof: *??***

- **Corollary:**
  - (a+b) mod m = ((a mod m) + (b mod m)) mod m
  - (ab) mod m = ((a mod m)(b mod m)) mod m

# PRIME NUMBERS

- **A positive integer $p \geq 2$ is prime if the only positive integers that divide $p$ are $1$ and $p$.**
- **Positive integers $N \geq 2$ which are not prime are called composite.**
- **The unique prime factorization of a positive integer $N$ is an expression of $N$ as a product of primes**
  - $N = p_1\ p_2\ p_3 \ldots p_k$
- **Theorem: *(The Fundamental Theorem of Arithmetic)***
  - Every positive integer (greater than 1) can be written uniquely as the product of primes.

# CLASSIC AND MODERN PROBLEMS

1. **Density**
   - How many primes are in {1…N}?
2. **Listing**
   - List all the primes in {1…N}.
3. **Testing**
   - Given a positive integer N, is N prime?
4. **Generating**
   - Pick a random prime number in {1...N}.

- **Modern cryptography is largely built on these problems**

# 1. DENSITY OF PRIMES

- Let $\pi(N)$ = the number of primes in $\{1 \ldots N\}$
  - $\pi(10) = 4$ $\{2,3,5,7\}$
  - $\pi(20) = 8$ $\{2,3,5,7,11,13,17,19\}$
- **Theorem (Euclid): There are infinitely-many primes.**
- **Proof: (by contradiction) Assume all $n$ primes are $p_1, p_2, \ldots, p_n$**
  - Let $q = p_1 * p_2 * \ldots * p_n + 1$
  - By the Fundamental Theorem of Arithmetic, $q$ is either prime or can be written as a product of primes.
  - No prime $p_i$ divides $q$ since it means $p_i \mid 1$, which is impossible.
  - Therefore either $q$ must be prime, or $q$ must have another prime divisor such that $p \neq p_i \ \forall i \ 1 \leq i \leq n$. Contradiction.
    - This is a non-constructive existence proof

# 1. DENSITY OF PRIMES

**Theorem (Euler / Hadamard):** ***(Prime Number Theorem)***

$$\lim_{n \to \infty} \frac{\pi(n)}{n/lnn} = 1$$

In other words, as $n \to \infty$, $\pi(\mathbf{n}) \to n \,/\, \ln n$

- **There are about *n / ln n* primes that are less than (or equal to) *n*.**
- **Given a random number *y* s.t. *1 < y < n*, the probability that *y* is prime is *1/ln n***

# 2. LISTING PRIMES

- List all the primes in {1...N}

Sieve of Eratosthenes

```
set prime [2..N] = 1
for p = 2 to n do
    if prime [p] = 1 then
        print "p is prime"
        for m = 2 to N/p do
            prime [mp] = 0
```

For each prime p, the Sieve eliminates all multiples of p.
No prime will ever be eliminated, and every composite (which must have a prime factor smaller than itself) is guaranteed to be eliminated before the outer loop reaches it.

Running time:
$O(N\ \pi(N))$ multiplications

# 3. TESTING PRIMALITY

- **Pseudocode?**
- **Theorem: If $N$ is composite, then $N$ has a prime factor $p \leq \sqrt{N}$.**
- **Proof: By contradiction.**
  - Suppose some composite $N$ has a prime factorization $N = p_1 p_2 \ldots p_k$ where all $p_i > \sqrt{N}$.
  - Then $N= p_1 p_2 \ldots p_k > (\sqrt{N})^k$
  - This only holds for $k < 2$, which means $N$ is prime.
  - Contradiction.

# 3. TESTING PRIMALITY

**ISPRIME (N)**

```
for k = 2 to √N do
    if k | N then
        return "N is not prime, the evidence is k"
return "N is prime"
```

**Runtime: $O(\sqrt{N})$ divisions**

**Input N has length $n = \log N$.**

**In terms of input length, the runtime is $O(2^{n/2})$**

**EXPONENTIAL!!!**

# RANDOMIZED PRIMALITY TESTING

- **General Idea:**
  - On input N, use randomness to look for evidence that N is composite
  - If evidence found, output "N is composite"
  - Otherwise, output "N is probably prime"
- **History:**
  - Miller-Rabin Test 1975
    - Monte Carlo
  - Adleman Huang Test 1987
    - Las Vegas
  - Agrawal-Kayal-Saxena Test 2002
    - Deterministic polynomial time with no errors
    - Not practical
      - Hard to implement
      - Large Big-Oh constant

# BASIC GROUP THEORY

- **A group $(G, *)$ is a set G and a binary operation $*$ as long as there is $e \in G$ such that for all $a,b,c \in G$:**
  - $a * b \in G$ *(closure)*
  - $(a * b) * c = a * (b * c)$ *(associativity)*
  - $a * e = a$ and $e * a = a$ *(identity)*
  - There exists a *unique* $a^{-1}$ such that $a * a^{-1} = e$ and $a^{-1} * a = e$ *(inverses)*
- **Some Basic Groups: Let N > 0**
  - $Z_N = \{0,1,2\ldots N-1\}$ under addition (mod N), identity is 0
  - $Z_N^* = \{ x \mid 1 \leq x < N \text{ and } \gcd(x, N) = 1 \}$ under multiplication (mod N), identity is 1
    - EX: $Z_6^*=\{1,5\}$
    - EX: $Z_7^*=\{1,\ldots,6\}$

# BASIC GROUP THEORY

- **$(H, *)$ is a subgroup of $(G, *)$ if $H \subseteq G$ and $(H, *)$ is a group:**
  - $\forall x,y \in H$, $x * y \in H$ (closure)
  - $\forall x \in H$, $x^{-1} \in H$ (inverse)
  - and the identity of G is in H ($e \in H$)
  - associativity follows directly from the property of operation $*$
- **Let $|G|$ denote the order of $G$: the number of elements in $G$.**
- **Theorem (Lagrange): Let $G$ be a group, and $H$ be a subgroup of $G$. Then $|H|$ divides $|G|$.**
- **Define Euler's phi function as $\phi(N) = |Z_{N^*}|$**
- **Theorem (Euler): For $N > 1$ and all $a \in Z_N^*$ we have $a^{\phi(N)} \equiv 1 \bmod N$**

# FERMAT'S LITTLE THEOREM

**If N is prime, then for every integer a such that $1 \leq a \leq N-1$,**

$$a^{N-1} \equiv 1 \pmod{N}$$

- **Proof:**
  - **Given** $a \in Z_p^*$
  - **Let** $aZ_p^* = \{ ax \mid x \in Z_p^* \} = \{ 1a, 2a, 3a, \ldots, (p-1)a \}$
  - $aZ_p^* = Z_p^*$ **because**
    - $ax \in aZ_p^* \Rightarrow ax \in Z_p^*$ (closure) **and**
    - $x \in Z_p^* \Rightarrow x = a\,(a^{-1}x) \in aZ_p^*$ (closure and inverse)
  - **Multiply** (mod p) **all the elements in each set**
    - $1a \cdot 2a \cdot 3a \cdots (p-1)a = 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p}$
    - $a^{p-1}(1 \cdot 2 \cdot 3 \cdots (p-1)) = 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p}$
    - $a^{p-1} = 1 \pmod{p}$

# FERMAT'S LITTLE THEOREM

If $N$ is prime, then for every integer $a$ such that $1 \le a \le N-1$,

$$a^{N-1} \equiv 1 \pmod{N}$$

- **Primality Testing Idea:** On input $N$, if we can find $1 \le a \le N-1$, such that $a^{N-1} \not\equiv 1 \pmod{N}$, then this proves that $N$ is NOT prime. But we still have no clue on factors of N.

- Example: Given $N=15$ let $a=2$ and test if $2^{14} \stackrel{?}{\equiv} 1 \pmod{15}$

$$2^{14} \equiv 4 \pmod{15}$$

- Bingo! Now we know that 15 is not a prime

# FERMAT'S LITTLE THEOREM

**If N is prime, then for every integer a such that $1 \leq a \leq N-1$,**

$$a^{N-1} \equiv 1 \pmod{N}$$

- **Works for 15. But for $N = 341 = 31*11$, we have $2^{341-1} = 1 \pmod{341}$**
- **Solution: Fermat's theorem says for any a, not just 2, so try other values.**

- **But, for $N = 561 = 51*11$ we have $\forall a$, $a^{N-1} = 1 \pmod{N}$.**
- **These are called Carmichael Numbers and there are infinitely many of them.**
- **Problem: Fermat's theorem is not an if-and-only-if theorem.**

- **Theorem: N is a Carmichael number ⇒ N is not a prime power**

# FERMAT'S PRIMALITY TEST (FOR NON-CARMICHAEL NUMBERS)

- A randomized primality test based on Fermat's Little Theorem.

pick $a \in \{1 \ldots N-1\}$ at random

if $a^{N-1} \neq 1 \pmod N$ then

output "N is composite"

else

output "N may be prime"

Using divide-and-conquer algorithm with O(log N) complexity

**Easy Facts:**

Runs in polynomial time

On prime N, always outputs "N may be prime"

May err on composite N

# FERMAT'S PRIMALITY TEST (FOR NON-CARMICHAEL NUMBERS)

- **Theorem: If N is composite but not a Carmichael number, then the Pr[ algorithm outputs "N may be prime" ] ≤ 1/2.**
- **Proof:**
  - Define $B = \{ a \in Z_N^* \mid a^{N-1} \equiv 1 \pmod{N} \}$
  - B contains all the inputs which cause Fermat's test to produce an error. It is a **subgroup** of $Z_N^*$
    - Closure: $a^{N-1} \equiv 1 \pmod{N}$, $b^{N-1} \equiv 1 \pmod{N} \Rightarrow (ab)^{N-1} \equiv 1 \pmod{N}$
    - Identity: $1^{N-1} \equiv 1 \pmod{N}$
    - Inverses: $a^{N-1} \equiv 1 \Rightarrow (a^{N-1})^{-1} \equiv 1 \Rightarrow (a^{-1})^{N-1} \equiv 1 \pmod{N}$
  - Since N is neither prime nor Carmichael, $B \neq Z_N^*$ so $|B| < |Z_N^*|$
  - Remember Lagrange's Theorem: $|B|$ divides $|Z_n^*| \Rightarrow |B| \leq 1/2\ |Z_n^*|$
  - Thus, Pr[ randomly picking bad a value ] = $|B| / |Z_n^*| \leq 1/2$

# FERMAT'S PRIMALITY TEST (FOR NON-CARMICHAEL NUMBERS)

- **Remember, we can boost the correctness probability of a randomized algorithm as high as we like by repeating it.**
- **Randomized primality test for non-Carmichael N:**

```
repeat k times
    pick a ∈ {1…N -1} at random
    if a^(N -1) ≠ 1 (mod N) then
        return "N is composite"
return "N may be prime"
```

# FERMAT'S PRIMALITY TEST (FOR NON-CARMICHAEL NUMBERS)

- Remember, we can boost the correctness probability of a randomized algorithm as high as we like by repeating it.
- Randomized primality test for non-Carmichael N:

repeat k times

pick $a \in \{1 \ldots N-1\}$ at random

if $a^{N-1} \neq 1 \pmod{N}$ then

return "N is composite"

return "N may be prime"

- Correct with probability $\geq 1 - \frac{1}{2}^{k}$

# QUADRATIC RESIDUE THEOREM (MODULAR SQUARE-ROOTS)

- **Theorem: If $N$ is prime, then the equation $x^2 \equiv 1 \pmod{N}$ has only two solutions in $Z_{N^*}$ :**

$$x \equiv 1 \pmod{N} \text{ and } x \equiv -1 \pmod{N}$$

- **Proof:**
  - Suppose $a^2 \equiv 1 \pmod{N}$.
  - Then $(a+1)(a-1) = a^2 - 1 \equiv 0 \pmod{N}$.
  - Thus $(a+1)(a-1)$ is a multiple of $N$
  - Since $N$ is a prime, then either $(a+1)$ or $(a-1)$ is a multiple of $N$
  - Therefore either $a \equiv 1 \pmod{N}$ or $a \equiv -1 \pmod{N}$
- **Alternative formulation: If there exists an integer $1 < x < n-1$, such that $x^2 \equiv 1 \pmod{n}$, then $n$ is composite.**

# NEW PRIMALITY TEST

- **New Idea for a Primality test:**
  - If, on input N, can find x such that $x^2 \equiv 1 \pmod{N}$ but $x \neq 1 \pmod{N}$ and $x \neq -1 \pmod{N}$, then it's a proof that N is not prime.
  - This idea will work for all N
- ***How do we find square roots of 1 (mod N) different from 1 and -1 ?***
- **Idea:** Take any a such that $a^{N-1} \equiv 1 \pmod{N}$ (any a works for Carmichael numbers)

# NEW PRIMALITY TEST

- **New Idea for a Primality test:**
  - If, on input N, can find x such that $x^2 \equiv 1 \pmod{N}$ but $x \neq 1 \pmod{N}$ and $x \neq -1 \pmod{N}$, then it's a proof that N is not prime.
  - This idea will work for all N
- ***How do we find square roots of 1 (mod N) different from 1 and -1 ?***
- **Idea:** Take any a such that $a^{N-1} \equiv 1 \pmod{N}$ (any a works for Carmichael numbers)
  - Find s,t with $N-1=2^s t$ with t odd. (how is this possible?)
  - Compute $a^{\frac{N-1}{2}}$ mod N, $a^{\frac{N-1}{4}}$ mod N … $a^{\frac{N-1}{2^s}}$ mod N
  - Find the first value that is different from 1 and -1
- **Theorem:** N Carmichael ⇒ Pr[ finding a root different from 1 and -1] ≥ $\frac{1}{2}$

# MILLER-RABIN PRIMALITY TEST

- **INPUT:** N > 2 odd with $N-1 = 2^s t$ such that t is odd.
- **OUTPUT:** “probably prime” or “composite”

Check if N is a perfect power.
***Can we do it in polynomial time?***

**MILLER-RABIN (N)**

if $N = a^b$ with a,b>1 then return “composite”

pick random integer a in $Z_N^*$

if $a^{N-1} \neq 1 \pmod{N}$ then return “composite”

compute the sequence $a^{\frac{N-1}{2}}, \ldots, a^{\frac{N-1}{2^S}}$ mod N

find the first element $y \neq 1 \pmod{N}$ in the sequence

(if it doesn’t exist, return “composite”)

if $y \neq -1 \pmod{N}$ then return “composite”

else return “probably prime”

# MILLER-RABIN PRIMALITY TEST: CORRECTNESS

- **If N is a prime**
  - No matter how we choose a, algorithm always says "probably prime" since we can never find
    - a s.t. $a^{N-1} \neq 1 \pmod{N}$
    - or a non-trivial root of 1 (mod N).
- **If N is composite**
  - Need to prove there are many choices of a for which the algorithm will output "composite"
  - Show: Pr[ algorithm outputs "composite" on composite N ] $\geq 1/2$

# MILLER-RABIN PRIMALITY TEST: CORRECTNESS

- Consider the set B of bad choices of a such that Miller-Rabin test says that N is prime when N is indeed composite.
- We want to prove $|B| \leq (N-1)/2$.
- We do it by proving that B is always contained in a proper sub-group of $Z_N^*$ and therefore (using Lagrange's Theorem) $|B| \leq ½ |Z_N^*|$
- Proof will be skipped.

# MILLER-RABIN PRIMALITY TEST

- **For every composite N**
  - The probability that Miller-Rabin makes a mistake saying that N is probably prime is ≤ ½.
  - Repeat the test k times, and say probably prime if and only if test never says N is composite.
  - Probability of making a mistake is $\leq \frac{1}{2^k}$
- **Deterministic polynomial-time always-correct primality testing algorithm exists, but not practical.**
- **In practice, use Miller-Rabin test with k=80.**

# 4. GENERATING RANDOM PRIMES

- **Goal: Pick a random prime number in {1..N}**
- **Brute force algorithm**
  - List all the primes in {1…N}
  - Pick one at random
- **Listing the primes takes time polynomial in N.**
  - Remember, $O(N\ \pi(N))$ multiplications
- **Remember, the size of the input N is n = log N.**
- **Thus, an algorithm polynomial in N is indeed exponential in its input size ($N = 2^n$).**
  - $O(N\ \pi(N))$ multiplications = $O(2^n\ \pi(2^n))$ multiplications
- **We want an algorithm that is polynomial in n.**

# 4. GENERATING RANDOM PRIMES

**RANDOM-PRIME (N)**

Pick a random number m ∈ {1…N}

Test if m is prime (e.g., run MILLER-RABIN (m))

If m is prime, return m

else, try again

- ***What is the expected number of tries before finding a prime?***
- **Let p be the probability of picking a prime on one try**
  - $p = \pi(N) / N = 1 / \ln N$ (Prime Number Theorem)
- **Expected number of tries is**
  - $1/p = \ln N = O(\log N) = O(n)$

# INTEGER OPERATIONS

- **Until now, we have only considered algorithm running times in terms of the number of integer operations**
  - Multiplication, addition, division
- ***But, in terms of input length, how long do these operations take?***
  - i.e., if we multiply two n-bit integers, how many bitwise operations will it take?
  - Are multiplication, addition, division all one-step operations taking the same amount of time?
  - What about modular multiplication, addition, division? Are they simpler because we have modulus N that simplifies the operations, or harder?

# REPRESENTATIONS OF INTEGERS

- **Theorem: Every positive integer n can be written uniquely as**
  - $n = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b^1 + a_0 b^0$ $\quad k \in N,\ 0 < a_i < b,\ 1 < b,\ a_k \neq b$

- **Definition: The base b expansion (representation) of n is denoted by $(a_k a_{k-1} \ldots a_1 a_0)_b$**
  - $b = 2$ ➔ Binary
  - $b = 16$ ➔ Hexadecimal
  - $b = 8$ ➔ Octal
  - $b = 10$ ➔ Decimal

# ALGORITHMS ON BINARY INTEGERS

- **n: number of bits in the binary representation of an integer**
  - Thus, for integer N we have n = log N
- **Consider operations on n-bit integers.**
- **Addition: Takes O(n) steps.**
- **Multiplication: Takes $O(n^2)$ steps.**
- **Division: Takes $O(n^2)$ steps with optimized algorithms.**
- **Overall: addition is easy, multiplication is hard, division is harder.**

# COMPLEXITY OF MODULAR OPERATIONS

- **Modular Multiplication: Compute ab (mod m)**
  - regular multiplication and then division, i.e. time complexity $O(n^2)$
- **Modular Division: Compute a/b (mod m)**
  - Compute $b^{-1}$ (mod m) first and then multiply $ab^{-1}$ (mod m)
  - Inverse does not always exists - we need gcd(b,m) = 1
- **Modular Inversion:**
  - Done via Extended Euclidean algorithm
- **Euclidean Algorithm**
  - **GCD (*a*, *b*)** // assume w.l.o.g. $a \geq b$
    - if $b = 0$ then return $a$
    - else return GCD($b$, $a$ mod $b$)

# CORRECTNESS OF EUCLIDEAN ALGORITHM

- **Lemma: Let $a = bq + r$, where $a, b, q$ and $r$ are integers. Then we have $\gcd(a, b) = \gcd(b, r)$**
- **Proof:**
  - **(i)** Assume $(c \mid a \wedge c \mid b)$. Then show $c \mid r$
    - We know $c \mid a - bq$ ***(WHY ??)*** Remember $r = a - bq$
    - Hence any common divisor of $a$ and $b$ is also a common divisor of $b$ and $r$.
  - **(ii)** Assume $(c \mid b \wedge c \mid r)$. Then show $c \mid a$
    - We know $c \mid bq + r$ ***(WHY ??)*** Remember $a = bq + r$
    - Hence any common divisor of $b$ and $r$ is also a common divisor of $a$ and $b$.
  - **(i)** and **(ii)** together mean that all common divisors of (a,b) and (b,r) pairs are the same. Therefore their *greatest* common divisor must also be the same.

# COMPLEXITY OF EUCLIDEAN GCD ALGORITHM

**GCD ($a$, $b$)**
  if $b = 0$ then return $a$
  else return GCD($b$, $a$ mod $b$)

- **Theorem: If $a > b \geq 0$ and the invocation of *GCD* performs $k \geq 1$ recursive calls, then $a \geq F_{k+2}$ and $b \geq F_{k+1}$.**
  - (where $F_k$ is the $k^{th}$ Fibonacci number)
- **Proof: (by induction)**
  - $k = 1$ $\Rightarrow$ $b \geq 1 = F_2$, $a \geq 2 = F_3$ Base case OK
  - $k = n - 1$ $\Rightarrow$ $b \geq F_n$, $a \geq F_{n+1}$ Inductive Hypothesis
  - $k = n$: $\Rightarrow$ $b = a \bmod b \geq F_n$, $a = b \geq F_{n+1}$

  $\Rightarrow$ $a \geq b + a \bmod b \geq F_{n+1} + F_n = F_{n+2}$ OK
- ***Running Time:***
  - $F_k \approx ((1 + \sqrt{5}) / 2)^k / \sqrt{5}$ $\Rightarrow (\sqrt{2})^k < F_k < 2^k$
  - $n = \max\{\log a, \log b\}$ - number of bits to encode $a$ and $b$
    - $n =\sim \log F_{k+2} =\sim k$
  - $\Theta(k) = \Theta(n)$ recursive calls with one division at each call (for a mod b)
  - $\Theta(n^3)$ complexity in terms of bit operations

# EXTENDED EUCLIDEAN ALGORITHM

- **Theorem: There exist integers *s* and *t* such that *gcd(a,b) = as + bt***
- **(gcd, s, t) = ExtendedGCD (*a*, *b*)**
  - if $b = 0$ then return $(a,1,0)$
  - $(d',x',y') \leftarrow$ ExtendedGCD$(b, a \bmod b)$
  - $(d,x,y) \leftarrow (d',y',x' - \text{floor}(a/b)\, y')$
  - return $(d,x,y)$
- **Complexity: $O(n^3)$**
  - Modular Inversion: $O(n^3)$
  - Modular Division: $O(n^3) + O(n^2) = O(n^3)$
    - Compute $b^{-1} \pmod m$ as the $s$ value in ExtendedGCD(b,m) where $\gcd(b,m) = 1$
    - Then perform a modular multiplication $ab^{-1} \pmod m$
    - Harder than regular division, harder than modular multiplication

# COMPLEXITY OF MODULAR OPERATIONS

- **Modular Exponentiation: Compute $b^a \pmod m$**
  - Trivial algorithm:
    - Compute $b \pmod m$, $b^2 \pmod m$, $b^3 \pmod m$, …, $b^a \pmod m$
    - $O(a)$ modular multiplications. $a \sim 2^n$ → $O(2^n)$ → EXPONENTIAL!!
  - Square-and-multiply algorithm:
    - Compute $b \pmod m$, $b^2 \pmod m$, $b^4 \pmod m$, $b^8 \pmod m$, …
    - Then compute $b^a \pmod m$ by multiplying powers of $b$ where the corresponding bit of $a$ is 1.
    - Takes $O(n)$ modular multiplications → $O(n^3)$ bit operations. Lots of research in the area.
  - Overall: (Modular) Exponentiation is the hardest. Know these when creating algorithms!!!

# CHINESE REMAINDER THEOREM

- **Theorem: *(The Chinese Remainder Theorem)***
- **Let $m_1 , m_2 , \ldots, m_n$ be pairwise relatively-prime positive integers. The system**

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

$$\vdots$$

$$x \equiv a_n \pmod{m_n}$$

- **has a unique solution in modulo $m = m_1 \cdot m_2 \ldots m_n$**

# CHINESE REMAINDER THEOREM

- **Solution:**
  - Let $M_k = m / m_k$ for $k = 1, 2, \ldots, n$.
  - Hence $\gcd(m_k, M_k) = 1$ (since $m_i$ are pairwise relatively prime)
  - Therefore $\exists y_k$ inverse of $M_k$ mod $m_k$ s.t. $M_k y_k \equiv 1 \pmod{m_k}$
  - The solution can then be given as:
  - $x = a_1 M_1 y_1 + a_2 M_2 y_2 + \ldots + a_n M_n y_n$
- **Check:**
  - Note: $M_i \equiv 0 \pmod{m_k}$ when $i \neq k$
  - and $M_k y_k \equiv 1 \pmod{m_k}$
  - $x \pmod{m_k} = a_1 M_1 y_1 \pmod{m_k} + a_2 M_2 y_2 \pmod{m_k} + \ldots + a_n M_n y_n \pmod{m_k}$
  - $= 0 \pmod{m_k} + \ldots + 0 \pmod{m_k} + a_k M_k y_k \pmod{m_k} + 0 \pmod{m_k} + \ldots + 0 \pmod{m_k}$
  - $\equiv a_k \pmod{m_k}$

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- **The Chinese Remainder Theorem is extremely useful when working with large numbers (e.g., cryptography, signal processing) just as the (Extended) Euclidean Algorithm is.**
- **Computer Arithmetic with Large Numbers**
  - Suppose our computer can perform arithmetic operations with integers $< 100$ much faster than with larger integers.
  - In reality, our computers can handle numbers $< 2^{30}$ well
    - But we will say $< 100$ so that we can understand the example
  - Yet, we want to work with much larger numbers (e.g., $2^{2050}$)
    - Again, for the sake of example, we want to work with numbers $< 100000000 = 10^8$
  - Let us pick a group of pairwise relatively-prime numbers: $m_1 = 99$, $m_2 = 98$, $m_3 = 97$, $m_4 = 95$.

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- **By CRT, every number $< m_1 \cdot m_2 \cdot m_3 \cdot m_4 = 99 \cdot 98 \cdot 97 \cdot 95 = 89403930$ can be represented uniquely using these four moduli.**

| | | |
|---|---|---|
| | **modulus** | **99 98 97 95** |
| | **123684:** | **(33, 8, 9, 89)** |
| **+** | **413456:** | **(32, 92, 42, 16)** |
| | **537140:** | **(65, 2, 51, 10)** |

***WHY CAN WE ADD PER MODULI ??***

BECAUSE $(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$

- **We only perform additions per moduli (very fast operations), and then use CRT for the following system to find $x = 537140$:**
  - $x \equiv 65 \bmod 99$
  - $x \equiv 2 \bmod 98$
  - $x \equiv 51 \bmod 97$
  - $x \equiv 10 \bmod 95$

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- Pseudoprime: Composite integer $N$ for which $2^{N-1} \equiv 1 \pmod{N}$
- E.g., $341 = 11 \cdot 31$ $\quad 2^{340} \equiv 1 \pmod{341}$
- *But how to compute* $2^{340} \pmod{341}$ *?*

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- **Pseudoprime: Composite integer $N$ for which $2^{N-1} \equiv 1 \pmod{N}$**
- **E.g., $341 = 11 \cdot 31$   $2^{340} \equiv 1 \pmod{341}$**
- ***But how to compute* $2^{340} \pmod{341}$ *?***
  - $2^{10} \equiv 1 \pmod{11}$   by Fermat's Little Theorem
  - $2^{340} = (2^{10})^{34} \equiv 1 \pmod{11}$

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- **Pseudoprime: Composite integer $N$ for which $2^{N-1} \equiv 1 \pmod{N}$**
- **E.g., $341 = 11 \cdot 31$ $\quad 2^{340} \equiv 1 \pmod{341}$**
- ***But how to compute* $2^{340} \pmod{341}$ *?***
  - $2^{10} \equiv 1 \pmod{11}$ by Fermat's Little Theorem
  - $2^{340} = (2^{10})^{34} \equiv 1 \pmod{11}$
  - $2^5 = 32 \equiv 1 \pmod{31}$ by computing manually
  - $2^{340} = (2^5)^{68} \equiv 1 \pmod{31}$

# APPLICATIONS OF CHINESE REMAINDER THEOREM

- **Pseudoprime: Composite integer $N$ for which $2^{N-1} \equiv 1 \pmod{N}$**
- **E.g., $341 = 11 \cdot 31$ $2^{340} \equiv 1 \pmod{341}$**
- ***But how to compute* $2^{340} \pmod{341}$ *?***
  - $2^{10} \equiv 1 \pmod{11}$ by Fermat's Little Theorem
  - $2^{340} = (2^{10})^{34} \equiv 1 \pmod{11}$
  - $2^5 = 32 \equiv 1 \pmod{31}$ by computing manually
  - $2^{340} = (2^5)^{68} \equiv 1 \pmod{31}$
  - Use Chinese Remainder Theorem:
    - $2^{340} \equiv 1 \pmod{11}$
    - $2^{340} \equiv 1 \pmod{31}$
    - $\rightarrow 2^{340} \equiv 1 \pmod{341}$