The running time of Euclid’s algorithm

- We analyze the worst-case running time of EUCLID as a function of the size of \( a \) and \( b \)
- Assume w.l.g. that \( a > b \geq 0 \)
- The overall running time of EUCLID is proportional to the number of recursive calls it makes
- Our analysis makes use of the Fibonacci numbers \( F_k \)

**Lemma 31.10** If \( a > b \geq 1 \) and the call EUCLID\((a, b)\) performs \( k \geq 1 \) recursive calls, then \( a \geq F_{k+2} \) and \( b \geq F_{k+1} \).

**Proof** The proof proceeds by induction on \( k \). For the basis of the induction, let \( k = 1 \). Then, \( b \geq 1 = F_2 \), and since \( a > b \), we must have \( a \geq 2 = F_3 \). Since \( b > (a \mod b) \), in each recursive call the first argument is strictly larger than the second; the assumption that \( a > b \) therefore holds for each recursive call.
Assume inductively that the lemma holds if \( k - 1 \) recursive calls are made; we then prove that the lemma holds for \( k \) recursive calls. Since \( k > 0 \), we have \( b > 0 \), and \( \text{EUCLID}(a, b) \) calls \( \text{EUCLID}(b, a \mod b) \) recursively, which in turn makes \( k - 1 \) recursive calls.

The inductive hypothesis then implies that \( b \geq F_{k+1} \) (thus proving part of the lemma), and \( a \mod b \geq F_k \).

We have
\[
b + (a \mod b) = b + (a - b\lfloor a/b \rfloor) \leq a,
\]
since \( a > b > 0 \) implies \( \lfloor a/b \rfloor \leq 1 \).

Thus,
\[
a \geq b + (a \mod b) \geq F_{k+1} + F_k = F_{k+2}.
\]

**Theorem 31.11** (Lamé’s theorem) For any integer \( k \geq 1 \), if \( a > b \geq 1 \) and \( b < F_{k+1} \), then the call \( \text{EUCLID}(a, b) \) makes fewer than \( k \) recursive calls.

- Show (by induction on \( k \)) that the upper bound of this theorem is the best possible because the call \( \text{EUCLID}(F_{k+1}, F_k) \) makes exactly \( k - 1 \) recursive calls when \( k \geq 2 \).
- Since \( F_k \approx \phi^k / \sqrt{5} \), where \( \phi^k \) is the golden ratio \( (1 + \sqrt{5})/2 \), the number of recursive calls in \( \text{EUCLID} \) is \( O(\log b) \).
31.6 Powers of an element

- Just as we often consider the multiples of a given element \( a \), modulo \( n \), we consider the sequence of powers of \( a \), modulo \( n \), where \( a \in \mathbb{Z}_n^* : a^0, a^1, a^2, a^3, \ldots \) modulo \( n \).
- Indexing from 0, the 0th value in this sequence is \( a^0 \mod n = 1 \), and the \( i \)th value is \( a^i \mod n \).
- For example, the powers of 3 modulo 7 are:

| \( i \mod 7 \) | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3( i \mod 7 )</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>...</td>
</tr>
</tbody>
</table>

Above \( \mathbb{Z}_n^* \) stands for a **multiplicative group modulo** \( n \): \((\mathbb{Z}_n^*, \cdot)\).
- The elements of this group are the set \( \mathbb{Z}_n^* \) of elements in \( \mathbb{Z}_n \) that are relatively prime to \( n \):

\[
\mathbb{Z}_n^* = \{ [a]_n \in \mathbb{Z}_n : \gcd(a, n) = 1 \}
\]
- An example of such a group is

\[
\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\} \]
The powers of 2 modulo 7 are

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^i \mod 7$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>…</td>
</tr>
</tbody>
</table>

Let $\langle a \rangle$ denote the subgroup of $\mathbb{Z}_n^*$ generated by $a$ by repeated multiplication, and let $\text{ord}_n(a)$ (the “order of $a$, modulo $n$”) denote the order of $a$ in $\mathbb{Z}_n^*$.

E.g., $\langle 2 \rangle = \{1, 2, 4\}$ in $\mathbb{Z}_7^*$, and $\text{ord}_7(2) = 3$.

The size of $\mathbb{Z}_n^*$ is denoted $\phi(n)$.

This function is known as Euler’s phi function.

It satisfies the equation

$$\phi(n) = n \prod_{\substack{p|n \text{ prime} \\text{and} \ p \mid n}} \left( 1 - \frac{1}{p} \right)$$

so that $p$ runs over all the primes dividing $n$ (including $n$ itself, if $n$ is prime).

**Theorem 31.30** (Euler’s theorem)

For any integer $n > 1$,

$$a^{\phi(n)} \equiv 1 \pmod{n} \text{ for all } a \in \mathbb{Z}_n^*.$$
Theorem 31.31 (Fermat’s little theorem)
If \( p \) is prime, then
\[
a^{p-1} \equiv 1 \pmod{p}
\]
for all \( a \in \mathbb{Z}_p^* \).

- Fermat’s little theorem applies to every element in \( \mathbb{Z}_p \) except 0, since \( 0 \notin \mathbb{Z}_p^* \).
- For all \( a \in \mathbb{Z}_p \), however, we have
  \[
a^p \equiv a \pmod{p}
\]
  if \( p \) is prime.
- E.g., \( 2^{7-1} = 2^6 = 64 \) and \( 64 \mod 7 = 1 \), while
  \( 2^{6-1} = 2^5 = 32 \) and \( 32 \mod 6 = 2 \):
  hence 6 is not prime.
- We showed that 6 is a composite number without factoring it!

Fermat’s little theorem, thus, (almost) gives a test for primality.
We say that \( p \) passes the Fermat test at \( a \), if
\[
a^{p-1} \equiv 1 \pmod{p}
\]
Call a number \( p \) pseudoprime if it passes Fermat tests for all smaller \( a \) relatively prime to it.
Only infrequent Carmichael numbers are pseudoprime without being prime.
If a number is not pseudoprime, it fails at least half of all Fermat tests.
We easily get a pseudoprimality algorithm with an exponentially small error probability.
**PSEUDOPRIME** \((p)\)  
1. Select random \(a_1, \ldots, a_k \in \mathbb{Z}_p\)  
2. Compute \(a_i^{p-1} \mod p\) for each \(i\)  
3. If all values are 1 **accept**, otherwise **reject**  
   - If \(p\) isn’t pseudoprime, it passes each randomly selected test with probability at most  
   - Probability that it passes all \(k\) tests is thus \(\leq 2^{-k}\)  
   - The algorithm operates in polynomial time  
   - To convert this algorithm to a primality algorithm, we should still avoid the problem with the Carmichael numbers

- A number \(x\) is a **square root of 1**, modulo \(n\), if it satisfies the equation \(x^2 \equiv 1 \pmod{n}\)  
- The number 1 has exactly two square roots, 1 and \(-1\), modulo any prime \(p\)  
- For many composite numbers, including all the Carmichael numbers, 1 has 4 or more square roots  
- E.g., \(\pm 1\) and \(\pm 8\) are the 4 square roots of 1 mod 21  
- We can obtain square roots of 1 if \(p\) passes the Fermat test at \(a\) because  
  - \(a^{p-1} \mod p \equiv 1\) and so  
  - \(a^{(p-1)/2} \mod p\) is a square root of 1  
- We may repeatedly divide the exponent by two, so long as the resulting exponent remains an integer
**PRIME**($p$) % accept = input $p$ is prime

1. **if** $p$ is even, accept if $p = 2$, otherwise **reject**
2. Select random $a_1, \ldots, a_k \in \mathbb{Z}_p^*$
3. **for each** $i \in \{1, \ldots, k\}$
4. Compute $a^{p-1} \mod p$, **reject** if different from 1
5. Let $p - 1 = st$ where $s$ is odd and $t = 2^h$ is a power of 2
6. Compute the sequence $a^{s\cdot 2^0}, a^{s\cdot 2^1}, \ldots, a^{s\cdot 2^h}$ modulo $p$
7. **if** some element of this sequence is not 1, find the last element that is not 1 and **reject** if that element is not $-1$
8. All test have been passed, so **accept**

---

**Lemma** If $p$ is an odd prime,

$$\Pr[\text{PRIME accepts } p] = 1.$$  

**Proof** If $p$ is prime, no branch of the algorithm rejects: Rejection in line 4 means that $p(a^{p-1} \mod p) \neq 1$ and Fermat’s little theorem implies that $p$ is composite.

If rejection happens in line 7, there exists some $b \in \mathbb{Z}_p^*$ s.t.

$$b \not\equiv \pm 1 \pmod{p} \quad \text{and} \quad b^2 \equiv 1 \pmod{p}.$$  

Therefore $b^2 - 1 \equiv 0 \pmod{p}$.  

Factoring yields
\[(b - 1)(b + 1) \equiv 0 \pmod{p},\]
which implies that \[(b - 1)(b + 1) = cp\] for some positive integer \(c\).

Because \(b \not\equiv \pm 1 \pmod{p}\), both \(b - 1\) and \(b + 1\) are in the interval \([0, p]\).
Therefore \(p\) is composite because a multiple of a prime number cannot be expressed as a product of numbers that are smaller than it is. \(\square\)

- The next lemma shows that the algorithm identifies composite numbers with high probability

- An important elementary tool from number theory, *Chinese remainder theorem*, says that a one-to-one correspondence exists between \(\mathbb{Z}_{pq}\) and \((\mathbb{Z}_p \times \mathbb{Z}_q)\) if \(p\) and \(q\) are relatively prime:
  - Each number \(r \in \mathbb{Z}_{pq}\) corresponds to a pair \((a, b)\), where \(a \in \mathbb{Z}_p\) and \(b \in \mathbb{Z}_q\) s.t.
    - \(r \equiv a \pmod{p}\) and
    - \(r \equiv b \pmod{q}\)
Lemma If $p$ is an odd composite number, 

$$\Pr[\text{PRIME accepts } p] \leq 2^{-k}.$$ 

Proof Omitted, takes advantage of the Chinese remainder thm. \qed

- Let $\text{PRIMES} = \{n \mid n \text{ is a prime number in binary}\}$
- The preceding algorithm and its analysis establishes: $\text{PRIMES} \in \text{BPP}$
- Note that the probabilistic primality algorithm has one-sided error. When it rejects, we know that the input must be composite. An error may only occur in accepting the input.

Thus an incorrect answer can only occur when the input is a composite number
- For all primes we get the correct answer
- The one-sided error feature is common to many probabilistic algorithms, so the special complexity class $\text{RP}$ is designated for it:

**Definition** $\text{RP}$ is the class of languages that are recognized by probabilistic polynomial time Turing machines where inputs in the language are accepted with a probability of at least $\frac{1}{2}$ and inputs not in the language are rejected with a probability of 1.

- Our algorithm shows that $\text{COMPOSITES} \in \text{RP}$
Primes $\in P$

- A generalization of Fermat's little theorem:

**Theorem A.** Let $a$ and $p$ be relatively prime and $p > 1$. $p$ is a prime number if and only if $(X - a)^p \equiv X^p - a \pmod{p}$

- $X$ is not important here, only the coefficients of the polynomial $(X - a)^p - (X^p - a)$ are significant

- For $0 < i < p$, the coefficient of $X^i$ is $\binom{p}{i} a^{p-i}$.

Supposing that $p$ is prime, $\binom{p}{i} = 0 \pmod{p}$ and hence all the coefficients are zero

- Therefore, we are left with the first term $X^p$ and the last one $-a^p$, which is $-a$ modulo $p$

- Unfortunately, deciding the primality of $p$ based on this requires an exponential time

- Agrawal (1999): it suffices to examine the polynomial $(X - a)^p$ modulo $X^r - 1$

- If $r$ is large enough, the only composite numbers that pass the test are powers of odd primes

- On the other hand, $r$ should be quite small so that the complexity of the approach does not grow too much

- Kayal & Saxena (2000): $r$ doesn’t have to be larger than $4(\log^2 p)$, in which case the complexity of the test procedure is only of the order $O(\log^3 n)$; i.e., belongs to $P$

- The result is based on an unproven claim
• A pair of odd numbers is called **Sophie Germain primes** if both \( q \) and \( 2q + 1 \) are primes (related to Fermat's last theorem).

• Agrawal, Kayal & Saxena (2002): If one can find a pair of SG primes \( q \) and \( 2q + 1 \) s.t.
  \[
  q > 4(\sqrt{2q + 1}) \cdot \log p
  \]
  then \( r \) does not need to be larger than
  \[
  2(\sqrt{2q + 1}) \cdot \log p
  \]

• Unfortunately this test is recursive and has time requirement of \( O(\log^{12} n) \) instead of the \( O(\log^{3} n) \) mentioned above.

---

**DETERMINISTIC-PRIME(\( p \))**

1. **if** \( p = ab \) for some \( b > 1 \) **then reject**
2. \( r \leftarrow 2 \)
3. **while** \( r < p \) **do**
4. **if** \( \gcd(p, r) \neq 1 \) **then reject**
5. **if** DETERMINISTIC-PRIME(\( r \)) **then** \% \( r > 2 \)
6. Let \( q \) be the largest factor of \( r - 1 \)
7. **if** \( q > 4\sqrt{r} \cdot \log p \) **and** \( p^{(r-1)/q} \neq 1 \) (mod \( r \)) **then break**
8. \( r \leftarrow r + 1 \)
9. **for** \( a \leftarrow 1 \) **to** \( 2\sqrt{r} \cdot \log p \) **do**
10. **if** \( (x - a)^p \neq x^p - a \) (mod \( x^{r-1} \cdot p \)) **then reject**
11. **accept** the input;
- The test of line 1 removes the powers of odd primes as required by the test of Agrawal (1999)
- The loop in lines 3–8 searches a pair of Sophie Germain primes $q$ and $r$
- Line 4 tests for Theorem A that $p$ and $r$ are relatively prime
- The loop in line 9 examines primality using a variation of Theorem A (Agrawal, 1999) up to value $2\sqrt{r} \log p$ (AKS, 2002)
- Because Theorem A holds if and only if $p$ is prime, the decision of the algorithm is correct
- The other variations only affect the complexity of the algorithm, not its correctness

Thank You!

MAT-72306 Randomized Algorithms
MAT-72606 Approximation Algorithms
in Spring 2015 term