Randomness and Computation: Some Prime Examples
Checking Our Work

Suppose we want to check \( p(x) \cdot q(x) = r(x) \), where \( p, q \) and \( r \) are three polynomials.

\[(x-1)(x^3+x^2+x+1) = x^4-1\]

If the polynomials are long, this requires \( n^2 \) mults by elementary school algorithms -- or can do faster with fancy techniques like the Fast Fourier transform.

Can we check if \( p(x) \cdot q(x) = r(x) \) more efficiently?
Great Idea: Evaluating on Random Inputs

Let $f(x) = p(x) q(x) - r(x)$. Is $f$ zero?

Idea: Evaluate $f$ on a random input $z$.

If we get $f(z) = 0$, this is evidence that $f$ is zero everywhere.

*If $f(x)$ is a degree $2n$ polynomial, it can only have $2n$ roots. We’re unlikely to guess one of these by chance!*
Equality checking by random evaluation

1. Fix a sample space $S=\{z_1, z_2, \ldots, z_m\}$ with arbitrary points $z_i$, for $m=2n/\delta$.

2. Select random $z$ from $S$ with probability $1/m$.

3. Evaluate $f(z) = p(z) q(z) - r(z)$

4. If $f(z) = 0$, output “equal” otherwise output “not equal”
Equality checking by random evaluation

What is the probability the algorithm outputs “not equal” when in fact \( f = 0 \)?

Zero!

If \( p(x)q(x) = r(x) \), always correct!
Equality checking by random evaluation

What is the probability the algorithm outputs “equal” when in fact $f \neq 0$?

Let $A = \{z \mid z$ is a root of $f\}$.

Recall that $|A| \leq$ degree of $f \leq 2n$. Therefore: $P(A) \leq 2n/m = \delta$.

We can choose $\delta$ to be small.
Equality checking by random evaluation

By repeating this procedure k times, we are "fooled" by the event

\[ f(z_1) = f(z_2) = \ldots = f(z_k) = 0 \]

when actually \( f(x) \neq 0 \)

with probability no bigger than

\[ P(A) \leq (2n/m)^k = \delta^k \]
Wow! That idea could be used for testing equality of lots of different types of “functions”!
Yes! E.g., a matrix is just a special kind of function.

Suppose we do a matrix multiplication of two \( n \times n \) matrices:

\[
AB = C
\]

The idea of random evaluation can be used to efficiently check the calculation.
What does “evaluate” mean?

Just evaluate the “function” $C$ on a random bit vector $r$ by taking the matrix-vector product $C \times r$.

$$AB = C$$

$$\begin{bmatrix} 1 & 0 & 3 & -4 & 8 \\ 7 & 0 & 0 & 2 & 9 \\ 13 & 5 & -6 & 0 & -7 \\ 1 & 6 & 21 & 9 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 9 \\ 7 \\ 31 \end{bmatrix}$$
So to test if $AB = C$ we compute
\[ x = Br, \quad y = Ax (= Abr), \quad \text{and} \quad z = Cr \]

If $y = z$, we take this as evidence that the calculation was correct.

The amount of work is only $O(n^2)$. 

Claim: If $AB \neq C$ and $r$ is a random $n$-bit vector, then $Pr(ABr = Cr) \leq \frac{1}{2}$. 
Claim: If $AB \neq C$ and $r$ is a random $n$-bit vector, then $\Pr(ABr = Cr) \leq \frac{1}{2}$. 
So, if a complicated, fancy algorithm is used to compute $AB$ in time $O(n^{2.236})$, it can be efficiently checked with only $O(n^2)$ extra work, using randomness!
“Random Fingerprinting”

Find a small random “fingerprint” of a large object.

- the value $f(z)$ of a polynomial at a point $z$
- the value $C_r$ at a random bit vector $r$

This fingerprint captures the essential information about the larger object: if two large objects are different, their Fingerprints usually are different!
Earth has huge file X that she transferred to Moon. Moon gets Y.

Did you get that file ok? Was the transmission accurate?

Uh, yeah.
Let \( \pi(n) \) be the number of primes between 1 and \( n \). I wonder how fast \( \pi(n) \) grows?

**Conjecture [1790s]:**

\[
\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1
\]
Their estimates

<table>
<thead>
<tr>
<th>$x$</th>
<th>$\pi(x)$</th>
<th>Gauss' Li</th>
<th>Legendre</th>
<th>$x/(\log x - 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>168</td>
<td>178</td>
<td>172</td>
<td>169</td>
</tr>
<tr>
<td>10000</td>
<td>1229</td>
<td>1246</td>
<td>1231</td>
<td>1218</td>
</tr>
<tr>
<td>100000</td>
<td>9592</td>
<td>9630</td>
<td>9588</td>
<td>9512</td>
</tr>
<tr>
<td>1000000</td>
<td>78498</td>
<td>78628</td>
<td>78534</td>
<td>78030</td>
</tr>
<tr>
<td>10000000</td>
<td>664579</td>
<td>664918</td>
<td>665138</td>
<td>661459</td>
</tr>
<tr>
<td>100000000</td>
<td>5761455</td>
<td>5762209</td>
<td>5769341</td>
<td>5740304</td>
</tr>
<tr>
<td>1000000000</td>
<td>50847534</td>
<td>50849235</td>
<td>50917519</td>
<td>50701542</td>
</tr>
<tr>
<td>10000000000</td>
<td>455052511</td>
<td>455055614</td>
<td>455743004</td>
<td>454011971</td>
</tr>
</tbody>
</table>
Two independent proofs of the Prime Density Theorem [1896]:

\[
\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1
\]
The Prime Density Theorem

This theorem remains one of the celebrated achievements of number theory.

In fact, an even sharper conjecture remains one of the great open problems of mathematics!
The Riemann Hypothesis [1859]

\[
\lim_{n \to \infty} \frac{\pi(n) - n / \ln n}{\sqrt{n}} = 0
\]
Slightly easier to show
\[ \pi(n)/n \geq 1/(2 \log n). \]
Random \text{log} \text{n} bit number is a random number from 1..n

\[ \frac{\pi(n)}{n} \geq \frac{1}{2} \log n \]

means that a random \text{log} \text{n}-bit number has at least a \( \frac{1}{2} \log n \) chance of being prime.
Random $k$ bit number is a random number from $1..2^k$.

$$\frac{\pi(2^k)}{2^k} \geq \frac{1}{2^k}$$

means that a random $k$-bit number has at least a $1/2^k$ chance of being prime.
Really useful fact

A random \( k \)-bit number has at least a \( \frac{1}{2^k} \) chance of being prime.

So if we pick \( 2^k \) random \( k \)-bit numbers, the expected number of primes on the list is at least 1.
Picking A Random Prime

Many modern cryptosystems (e.g., RSA) include the instructions:

“Pick a random n-bit prime.”

How can this be done efficiently?
Picking A Random Prime

“Pick a random n-bit prime.”

Strategy:
1) Generate random n-bit numbers
2) Test each one for primality

[more on this later in the lecture]
Picking A Random Prime

"Pick a random n-bit prime."

1) Generate $k^n$ random n-bit numbers

Each trial has a $\geq \frac{1}{2^n}$ chance of being prime.

Pr[ all $k^n$ trials yield composites ]

\[ \leq (1 - \frac{1}{2^n})^{k^n} = (1 - \frac{1}{2^n})^{2^n} \times \frac{k}{2} \leq \frac{1}{e^{k/2}} \]
Picking A Random Prime

“Pick a random n-bit prime.”

Strategy:
1) Generate random n-bit numbers
2) Test each one for primality

For 1000-bit primes, if we try out 10000 random 1000-bit numbers, chance of failing $\leq e^{-5}$
Moral of the story

Picking a random prime is "almost as easy as" picking a random number.

(Provided we can check for primality. More on this later.)
Earth has huge file $X$ that she transferred to Moon. Moon gets $Y$.

Did you get that file ok? Was the transmission accurate?

Uh, yeah.
Are X and Y the same n-bit numbers?

\[ p = \text{random } 2^\log n \text{-bit prime} \]

Send \((p, X \mod p)\)

Answer to “\(X \equiv Y \mod p?\)”
Why is this any good?

Easy case:
If $X = Y$, then $X \equiv Y \pmod{p}$
Why is this any good?

Harder case:

What if $X \neq Y$? We mess up if $p \mid (X-Y)$.

Define $Z = (X-Y)$. To mess up, $p$ must divide $Z$.

$Z$ is an $n$-bit number.

$\Rightarrow$ $Z$ is at most $2^n$. (duh.)

But each prime $\geq 2$.

Hence $Z$ has at most $n$ prime divisors.
Almost there...

Z has at most $n$ prime divisors.

How many $2\log n$-bit primes?

A random $k$-bit number has at least a $\frac{1}{2k}$ chance of being prime.

$\Rightarrow$ at least $\frac{2^{2\log n}}{2\times2^{\log n}} = \frac{n^2}{4\log n} \gg 2n$ primes.

Only (at most) half of them divide $Z$.

$\Rightarrow$ make mistake with prob $< \frac{1}{2}$. 
Theorem: Let $X$ and $Y$ be distinct $n$-bit numbers. Let $p$ be a random $2\log n$-bit prime.

Then

$$\text{Prob } [X = Y \mod p] < \frac{1}{2}$$

Earth-Moon protocol makes mistake with probability at most $1/2$!
Are $X$ and $Y$ the same $n$-bit numbers?

Pick $k$ random $2\log n$-bit primes: $P_1, P_2, \ldots, P_k$.
Send $(X \mod P_i)$ for $1 \leq i \leq k$.

$k$ answers to “$X = Y \mod P_i$?“
Exponentially smaller error probability

If $X=Y$, always accept.

If $X \neq Y$,

$$\text{Prob } [X = Y \mod P_i \text{ for all } i] \leq (1/2)^k$$
Picking A Random Prime

“Pick a random n-bit prime.”

Strategy:
1) Generate random n-bit numbers
2) Test each one for primality

How can we test primality efficiently?
Primality Testing:
Trial Division On Input n

Trial division up to \( \sqrt{n} \)

\[
\text{for } k = 2 \text{ to } \sqrt{n} \text{ do} \\
\quad \text{if } k \mid n \text{ then} \\
\quad \quad \text{return "n is not prime"} \\
\quad \text{otherwise return "n is prime"}
\]

about \( \sqrt{n} \) divisions
Trial division performs $\sqrt{n}$ divisions on input $n$.

Is that efficient?

For a 1000-bit number, this will take about $2^{500}$ operations.
That’s not very efficient at all!!!

More on efficiency and run-times in a future lecture...
Do the primes have a fast decision algorithm?
Euclid gave us a fast GCD algorithm.

Surely, he tried to give a faster primality test than trial division.

But Euclid, Euler, and Gauss all failed!
But so many cryptosystems, like RSA and PGP, use **fast primality testing** as part of their subroutine to generate a random n-bit prime!

What is the fast primality testing algorithm that they use?
There are fast *randomized* algorithms to do primality testing.

Strangely, by allowing our computational model an extra instruction for flipping a fair coin, we seem to be able to compute some things faster!
If \( n \) is composite, what would be a certificate of compositeness for \( n \)?

A non-trivial factor of \( n \).

But... even using randomness, no one knows how to find a factor quickly.

We will use a different certificate of compositeness that does not require factoring.
Recall that:

Fermat: \( a^{p-1} = 1 \mod p \).

When working modulo prime \( p \), for any \( a \neq 0 \), \( a^{(p-1)/2} = \pm 1 \).

\( x^2 = 1 \mod p \) has at most 2 roots. 1 and -1 are roots, so it has no others.
“Euler Certificate” Of Compositeness

When working modulo a prime $p$, for any $a \neq 0$, $a^{(p-1)/2} = \pm 1$.

We say that $a$ is a certificate of compositeness for $n$, if $a \neq 0$ and $a^{(n-1)/2} \neq \pm 1$.

Clearly, if we find a certificate of compositeness for $n$, we know that $n$ is composite.
"Euler Certificates" Of Compositeness

\[ EC_n = \{ a \in \mathbb{Z}^*_n \mid a^{(n-1)/2} \neq \pm 1 \} \]

\[ NOT-EC_n = \{ a \in \mathbb{Z}^*_n \mid a^{(n-1)/2} = \pm 1 \} \]

If \( NOT-EC_n \neq \mathbb{Z}^*_n \) then \( EC_n \) is at least half of \( \mathbb{Z}^*_n \)

In other words, if \( EC_n \) is not empty, then \( EC_n \) contains at least half of \( \mathbb{Z}^*_n \).
Proof

\( \text{EC}_n = \{ a \in \mathbb{Z}^*_n \mid a^{(n-1)/2} \neq \pm 1 \} \)
\( \text{NOT-EC}_n = \{ a \in \mathbb{Z}^*_n \mid a^{(n-1)/2} = \pm 1 \} \)

**Claim:** NOT-EC\(_n\) is a subgroup of \( \mathbb{Z}^*_n \)

**Proof:**

*Closure:* if \( a, b \in \text{NOT-EC}_n \), then \( ab \in \text{NOT-EC}_n \)

Hence, by Lagrange’s theorem, \( |\text{NOT-EC}_n| \) divides \( |\mathbb{Z}^*_n| \)

\[ \Rightarrow |\text{NOT-EC}_n| \leq \frac{1}{2} |\mathbb{Z}^*_n| \]

\[ \Rightarrow |\text{EC}_n| \text{ contains at least half of } |\mathbb{Z}^*_n| \]
"Euler Certificates" Of Compositeness

\[ EC_n = \{ a \in \mathbb{Z}_n^* \mid a^{(n-1)/2} \neq \pm 1 \} \]

\[ NOT-EC_n = \{ a \in \mathbb{Z}_n^* \mid a^{(n-1)/2} = \pm 1 \} \]

If \( NOT-EC_n \neq \mathbb{Z}_n^* \), then \( EC_n \) is at least half of \( \mathbb{Z}_n^* \).

In other words, if \( EC_n \) is not empty, then \( EC_n \) contains at least half of \( \mathbb{Z}_n^* \).
Randomized Primality Test

Let’s suppose that $E_{C_n}$ contains at least half the elements of $\mathbb{Z}^*_n$.

Randomized Test:

For $i = 1$ to $k$:

1. Pick random $a_i \in [2 \ldots n-1]$;
2. If $\text{GCD}(a_i, n) \neq 1$, Halt with “Composite”;
3. If $a_i^{(n-1)/2} \neq \pm 1$, Halt with “Composite”;

Halt with “I think $n$ is prime. I am only wrong $(\frac{1}{2})^k$ fraction of times I think that $n$ is prime.”
Is $EC_n$ non-empty for all primes $n$?

Unfortunately, no.

$$EC_n = \{ a \in \mathbb{Z}_n^* \mid a^{n-1} \neq \pm 1 \}$$

Certain numbers *masquerade* as primes.

A **Carmichael number** is a number $n$ such that $a^{n-1} = 1 \pmod{n}$ for all numbers $a$ with $\gcd(a,n)=1$.

Example: $n = 561 = 3 \times 11 \times 17$ (the smallest Carmichael number)

$1105 = 5 \times 13 \times 17$

$1729 = 7 \times 13 \times 19$

And there are many of them. For sufficiently large $m$, there are at least $m^{2/7}$ Carmichael numbers between 1 and $m$. 
The saving grace

The randomized test fails only for Carmichael numbers.

But, there is an efficient way to test for Carmichael numbers.

Which gives an efficient algorithm for primality.
Randomized Primality Test

Let’s suppose that \( \mathbb{E} \mathbb{C}_n \) contains at least half the elements of \( \mathbb{Z}_n^* \).

Randomized Test:

For \( i = 1 \) to \( k \):

Pick random \( a_i \in [2 .. n-1] \);

If \( \gcd(a_i, n) \neq 1 \), Halt with “Composite”;

If \( a_i^{(n-1)/2} \neq \pm 1 \), Halt with “Composite”;

If \( n \) is Carmichael, Halt with “Composite”

Halt with “I think \( n \) is prime. I am only wrong \( (\frac{1}{2})^k \) fraction of times I think that \( n \) is prime.”
Randomized Algorithms

The test we outlined made one-sided error:
It never makes an error when it thinks $n$ is composite.
It could just be unlucky when it thinks $n$ is prime.

Another one-sided algorithm that never makes a mistake when it thinks $n$ is prime.

Yet another algorithm makes 2-sided error.
Sometimes it is mistaken when it thinks $n$ is prime,
sometimes it is mistaken when it thinks $n$ is composite.
n prime means half of a's satisfy
\[ a^{(n-1)/2} = -1 \mod n \]

If n is prime, then \( \mathbb{Z}_n^* \) has a generator \( g \). Then \( g^{(n-1)/2} = -1 \mod n \).

A random \( a \in \mathbb{Z}_n^* \) is given by \( g^r \) for uniformly distributed \( r \).

Half the time, \( r \) is odd:
\[ (g^r)^{(n-1)/2} = -1 \mod n \]
Another Randomized Primality Test

Suppose \( n \) is not even, nor is it the power of a number.

Randomized Test:

For \( i = 1 \) to \( k \):

Pick random \( a_i \in [2 .. n-1] \);

If \( \text{GCD}(a_i, n) \neq 1 \), Halt with “Composite”;

If \( a_i^{(n-1)/2} \neq \pm 1 \), Halt with “Composite”;

If all \( k \) values of \( a_i^{(n-1)/2} = +1 \), Halt with “I think \( n \) is composite. I am only wrong \((\frac{1}{2})^k\) fraction of the times.”

Halt with “I think \( n \) is prime. I am only wrong \((\frac{1}{2})^k\) fraction of times I think that \( n \) is prime.”
We can prove that if \( n \) is an odd composite, not a power, and there is some \( a \) such that \( a^{(n-1)/2} = -1 \), then \( EC_n \neq \emptyset \).

Hence, \( EC_n \) is at least a half fraction of \( Z^*_n \).

This algorithm makes 2-sided error. Sometimes it is mistaken when it thinks \( n \) is prime, sometimes it is mistaken when it thinks \( n \) is composite.
Many Randomized Tests

Miller-Rabin test

Solovay-Strassen test
In 2002, Agrawal, Saxena, and Kayal (AKS) gave a deterministic primality test that runs in time $O((\log n)^{12})$.

This was the first deterministic polynomial-time algorithm that didn't depend on some unproven conjecture, like the Riemann Hypothesis!
Picking A Random Prime

“Pick a random n-bit prime.”

Strategy:
1) Generate random n-bit numbers
2) Do fast randomized test for primality
Primality Testing Versus Factoring

**Primality** has a fast randomized algorithm.

**Factoring** is not known to have a fast algorithm.

In fact, after thousands of years of research, the fastest randomized algorithm takes $\exp(O(n \log n \log n)^{1/3})$ operations on numbers of length $n$. With great effort, we can currently factor 200 digit numbers.
<table>
<thead>
<tr>
<th>number</th>
<th>digits</th>
<th>prize</th>
<th>factored</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-100</td>
<td>100</td>
<td></td>
<td>Apr. 1991</td>
</tr>
<tr>
<td>RSA-110</td>
<td>110</td>
<td></td>
<td>Apr. 1992</td>
</tr>
<tr>
<td>RSA-120</td>
<td>120</td>
<td></td>
<td>Jun. 1993</td>
</tr>
<tr>
<td>RSA-129</td>
<td>129</td>
<td>$100</td>
<td>Apr. 1994</td>
</tr>
<tr>
<td>RSA-130</td>
<td>130</td>
<td></td>
<td>Apr. 10, 1996</td>
</tr>
<tr>
<td>RSA-140</td>
<td>140</td>
<td></td>
<td>Feb. 2, 1999</td>
</tr>
<tr>
<td>RSA-150</td>
<td>150</td>
<td></td>
<td>Apr. 16, 2004</td>
</tr>
<tr>
<td>RSA-155</td>
<td>155</td>
<td></td>
<td>Aug. 22, 1999</td>
</tr>
<tr>
<td>RSA-160</td>
<td>160</td>
<td></td>
<td>Apr. 1, 2003</td>
</tr>
<tr>
<td>RSA-200</td>
<td>200</td>
<td></td>
<td>May 9, 2005</td>
</tr>
<tr>
<td>RSA-576</td>
<td>174</td>
<td>$10,000</td>
<td>Dec. 3, 2003</td>
</tr>
<tr>
<td>RSA-640</td>
<td>193</td>
<td>$20,000</td>
<td>Nov 2, 2003</td>
</tr>
<tr>
<td>RSA-704</td>
<td>212</td>
<td>$30,000</td>
<td>open</td>
</tr>
<tr>
<td>RSA-768</td>
<td>232</td>
<td>$50,000</td>
<td>open</td>
</tr>
<tr>
<td>RSA-896</td>
<td>270</td>
<td>$75,000</td>
<td>open</td>
</tr>
<tr>
<td>RSA-1024</td>
<td>309</td>
<td>$100,000</td>
<td>open</td>
</tr>
<tr>
<td>RSA-1536</td>
<td>463</td>
<td>$150,000</td>
<td>open</td>
</tr>
<tr>
<td>RSA-2048</td>
<td>617</td>
<td>$200,000</td>
<td>open</td>
</tr>
</tbody>
</table>

Google: RSA Challenge Numbers
The techniques we’ve been discussing today are sometimes called “fingerprinting.”

The idea is that a large object such as a string (or document, or function, or data structure...) is represented by a much smaller “fingerprint” using randomness.

If two objects have identical sets of fingerprints, they’re likely the same object.