Today we will see another powerful technique in the design of algorithms: Use of Randomness.

**Randomized Algorithm**

Up to now our algorithms used as their basic operations: $+,*,/,,\$, test for 0 On the same input, our algorithms always had a unique output.

This lecture we consider randomized algorithms: algorithms which can toss coins as a basic step. Namely, the algorithm can toss a fair coin whose outcome is $H$ or $T$ with probability $\frac{1}{2}$. And on the same input may behave differently depending on the outcome of the coins tossed during the execution. Behave differently: take a different amount of steps, may even have different outputs.

In this lecture we will speak of a sub-class of randomized (also called probabilistic) algorithms which are called Monte Carlo algorithms.

The guarantee that a Monte-Carlo algorithm for a problem P provides are:

- For every input (instance), regardless of coins tossed, the algorithm runs in polynomial time.
- For every input (instance),

\[
prob(\text{output of the algorithm is correct}) \geq \text{high}
\]

1. You can control this probability and modify a Monte Carlo algorithm to make $\text{high}$ make it exponentially close to 1, but still there will be a non 0 probability that the algorithm is incorrect.

2. The probabilities is taken over the coins tossed by the algorithm, not the input. Namely, for every input, the probability is high.

Q: Why should we use Randomness?
• Some problems cannot be solved, even in principle, if the computer is deterministic and cannot toss coins: the problem of asynchronous consensus in distributed computing is an example (see distributed algorithms class).

• Sometimes, only exponential time deterministic algorithms are known, whereas we can get polynomial time randomized algorithms.

• Sometime, get a significant polynomial time speedup by going from deterministic to probabilistic. Namely, say from $\Theta(n^8)$ to $\Theta(n^3)$.

Q: Is there an intuition for why tossing coins may possibly help?

A: When we design an algorithm and want to show it is correct and has a good running time, one can liken it to fighting an adversary who prepares the input in the worst possible way to slow down the algorithm the most. In a sense if the algorithm tosses coins, it means that the adversary who prepared the input does not exactly know what the algorithm will do (even the algorithm does not know till the coins are tossed), and thus cannot trick the algorithm into running for a longer amount of time.

In making statements concerning probabilities we will always have a sample space $S$ which is a finite set of elementary events that can happen and a probability distribution which is a function $Pr: S \rightarrow \mathbb{R}$ s.t. $\sum_{a \in S} Pr(a) = 1$

EX: say the sample space used in an algorithm is the set of all possible outcomes of the coins tossed.

Say $n = 3$ flips of a coin

$S = HHH, HHT, \ldots TTT$, $Pr(a) = 1/8$

When all elementary events have the same probability $Pr$ is the uniform distribution.

For the purpose of this lecture, we will always think of our sample space as a range of integers $\{1, 1, \ldots, N-1\}$ and the randomized step of our algorithms will be ‘choose a random $a \in \{1, \ldots, N-1\}$’. Namely, for all $a$, $Pr(a$ is chosen) = $1/\{N-1\}$

MODEL: recall that last lecture when we talked about the first example of integer multiplication we counted bit-operations, but when we talked about Power and Fibonacci algorithms we assumed that arithmetic operations such as multiplying and adding integers were unit cost $\Theta(1)$. Here again, we will count bit operations, so our multiplications will NOT be free.

**Distinguishing Primes from Composites**

Throughout this lecture we will develop a randomized algorithm for one of the oldest questions in number theory: distinguishing Prime numbers from composite numbers. Along
the way we will learn about various other number theoretic algorithms: greatest common
divisors, solving quadratic equations modulo primes, modular arithmetic.

Let the fun begin!

A positive integer $N \geq 2$ is prime if there exists a $d$ such that $d \neq N, 1$ and $d|N$ (we use $d|N$ to mean $d$ divides $N$) otherwise it is not prime and we call it a composite number. Traditionally, mathematicians looked at the:

1. DENSITY of primes: how many primes are there in $\{1, ..., N\}$?
   Prime Density Theorem (Euler, Hadamard 1750): as $n \rightarrow \infty$, $\pi(N) = \text{number of primes in } \{1, ..., N\} = \frac{N}{\log N}$

2. GENERATING all primes: find all primes in $\{1, ..., N\}$.
   The sieve of Eratosthenes which works as follows: for each $I = 1...n$ cross all multiples of $I$, by the time you get to an $I$ if it’s not crossed out it’s a prime.

3. TESTING: Given as input an integer $N \geq 2$, output PRIME if $N$ is a prime number, and COMPOSITE if it is composite (we call the problem for which the answer is one of two choices, a decision problem).

4. Modern Problem: Generate a large prime in an interval between $\{1, ..., N\}$. Strategy: choose a random $n$-bit number $N$ and test if it’s a prime. Important problem in cryptography. By the density theorem it means that if you pick a 100-bit random number say, the chance it is prime is $\frac{1}{\ln 10^{100}}$

   The general idea for primality testing: Look for evidence that the input $N$ is not a prime. If you find it, say $N$ is COMPOSITE otherwise say $N$ is a PRIME.

How? First natural idea would be to Find $d|N$ such that $d$ not 1 or $N$

**Trial Division**

input: $N$ s.t. $\log N = n$, (namely: $N$ can be represented with $o(n)$ bits
output: YES if $N$ is prime, NO if $N$ is composite
for all $i = 1$ to $\sqrt{N}$
if $i|N$ then output YES
otherwise, output NO

The running for this algorithm is: Let $D(n)$ be the cost of division for $n$-bit integers (it is $\Theta(n^2)$). $T(n) = O(\sqrt{N}D(n)) = O(2^{n/2}n^2)$. why? since $N$ which can be represented by
$n$ bits can be as large as $2^n$. This is not a polynomial time algorithm in $n$, rather it is Exponential time in $n$. For example, to find a divisor using this method for a 512-bit input $N$ or detect that none exists, takes more than the number of of seconds since the universe begins. In fact, we should not be surprised, as the best algorithm known for factoring integers has sub-exponential runtime $O(2^{n^{1/3}})$. We in contrast want an algorithm which runs in POLYNOMIAL TIME in $n$.

Thus, we will look for different evidence that $N$ is prime. Our modified general idea for primality testing.

Do a randomized search for evidence that the input $N$ is not a prime. 
if you find it, say $N$ is COMPOSITE otherwise say $N$ is 'probably PRIME'

We will dedicate our entire lecture toward achieving this goal. The primality testing problem has taken center stage in theoretical computer science very early since 1976. It is one of these handful of corner-stone problems which many theoreticians in the last 30 some years have struggled with. It also presents a great success story. In 1975 Rabin-Miller and Solovay Strassed presented Monte Carlo probabilistic primality test. We will present the Rabin-Miller test here today. In 2002 (27 years later and many fascinating algorithms later), Agrawal-Kayal-Saxena (AKS) devised finally a deterministic polynomial time test for primality if $N$ is prime. Great algorithm, which you can see in more advanced algorithms classes.

To describe the Miller-Rabin we have to dig deeper and study a (little)-modular arithmetic and some simple algorithms for number theory problems. All numbers we refer to throughout these notes are $n$-bit long.

**Number Theory Background and Algorithms**

$A \mod B$. Note: “$A \mod B$” means (the remainder when you divide $A$ by $B$) the smallest nonnegative number of form $A - kB$ for integer $k$. Can compute in time $O(n^2)$ using grade-school method - keep subtracting till get a number smaller than $A$. ($O(n)$ subtractions each costing $O(n)$.) Will write $A = B \mod N$ to be a short hand to mean $(A \mod N) = (B \mod N)$.

$GCD(A, B) =$ greatest common divisor of $A$ and $B$ where by convention $GCD(A, 0) = A$.
Can we compute $GCD(A, B)$ quickly? over 2000 years old algorithm by Euclid’s is based on observation that $GCD(A, B) = GCD(B, A \mod B)$.

**Proof:**
if $A$ and $B$ are multiples of $d$, then there exists $A'$ and $B'$, such that $A = A'd$ and $B = B'd$, and thus $A - kB = d(A' - kB')$ is a multiple of $d$ too. Similarly, if $d$ divides $B$ and $d$ divides $(A \mod B)$ then $d$ divides $A$.
GCD(A,B)
    if (B=0) return A
    else return GCD (B, A mod B)

E.g., GCD(51,9) = GCD(9,6) = GCD(6,3) = GCD(3,0) = 3.

Remarks:

- The number of iterations is linear in the number of bits in the input i.e \( \Theta(n) \). One way to see this is that if \( A \geq B \) then \( A \mod B \) is guaranteed to have at least one fewer bit than \( A \).

- When \( \text{gcd}(A, B) = 1 \) we say that \( A \) and \( B \) are relatively prime to each other.

- EXTENDED GCD: We can also compute integers \( x \) and \( y \) such that \( d = Ax + By \). For example, \( A = 7, B = 9 \). \( d = 1 = 4 \times 7 - 3 \times 9 \), so \( x = 4, y = -3 \).

  How can we find these coefficients \( x \) and \( y \): can compute as by product of the GCD algorithm above (see section 3.1, page 937, in your text book). Essentially, recursively, running on \( B \) and \( A - kB = (A \mod B) \), we compute \( x', y', d' \) such that \( d' = Bx' + (A - kB)y' \). This in turn means that \( d' = Ay' + B(x' - ky') \). The ability to find these coefficients \( x \) and \( y \) will turn out to be very useful.

EXponentiation mod N:

Given \( A, B, \) and \( D \) all \( n \)-bit integers, compute \( A^B \mod N \) = use divide and conquer Power algorithm from last time modified slightly. Note: you can do work in the exponents, i.e \( A^i A^j = A^{i+j} \mod N \).

Power (A,B,N)
    if B=1 output A
    if B is even, then x=Power(A,B/2,N) mod N; output x*x mod N
    if B is odd, then x=Power(A,(B-1)/2,N) mod N; output (x*x mod N)*A mod N

As before, Power(A,B,N) would take \( O(logB) = O(n) \) multiplications. Note that each time you do a multiplication you reduce your number mod \( N \) to keep your intermediate computations smaller than \( N \), so you are also working with \( n \) bit numbers, and the total complexity counting the cost for multiplications is \( O(n^2) \) or with the best known multiplication \( O(n^3) \).
Modular Arithmetic: Our Basic Groups

Let $\mathbb{Z}_N = \{0, 1, 2, \ldots, N - 1\}$,
$\mathbb{Z}_N^+ = \{1, 2, \ldots, N - 1\}$, and
$\mathbb{Z}_N^* = \{A \in \mathbb{Z}_N : \gcd(A, N) = 1\}$.

Ex: $\mathbb{Z}_7^* = \{1, 2, \ldots, 6\}$ $\mathbb{Z}_6^* = \{1, 5\}$ If $N$ is prime, note that $\mathbb{Z}_N^* = \{1, 2, \ldots, N - 1\}$.

A group $(G, \text{op})$ is a set $G$ and a binary operation $\text{op}$ that satisfies closure, associativity, the existence of inverses, and identity. Examples of groups are: $\mathbb{Z}_N$ under $+$ mod $N$ operation where the identity is 0, and more important for us: $\mathbb{Z}_N^*$ under the operation of multiplication mod $N$.

claim: The set $\mathbb{Z}_N^*$ is a group under the operation of multiplication mod $N$

- Closure: If you multiply 2 numbers in $\mathbb{Z}_N^*$, then their product is in $\mathbb{Z}_N^*$ (easy to see if no divisors in common between $A$ and $N$, and between $B = and N$, then no divisors in common in between $AB$ and $N$).

- Identity element 1 is in $\mathbb{Z}_N^*$.

- Inverses: If $A \in \mathbb{Z}_N^*$, there exists an element (denoted by) $A^{-1}$ in $\mathbb{Z}_N^*$ such that $A^{-1}A = 1 \mod N$. To see this, run Extended-Euclid’s algorithm on the pair $(A, N)$. It returns $x, y$ such that $xA + yN = \gcd(A, N) = 1$. Taking this equation mod $N$, you get that $xA = 1 \mod N$ and thus $x = A^{-1} \mod N$. E.g., what is $5^{-1} \mod 17$?

The starting (and main) idea behind our probabilistic primality test comes from:

Fermat’s little theorem: if $N$ is prime and $1 \leq A \leq N - 1$, then $A^{N-1} = 1 \mod N$.

Proof of Fermat’s little theorem: Lets prove it by induction on $A$. (check if its ok to do induction like this...) Let $A = 1$ then obviously, $1^{N-1} = 1$. Assume it is true for $A$. Prove for $A+1$. $(A + 1)^N = A^N + N A^{(N-1)} + N(N-1)/2 A^{(N-2)} + \ldots + N A^{N-1} + 1 = (since N is prime) (A^N + 1) \mod N = (by induction hypothesis) (A + 1) \mod N, so (A + 1)^N = (A + 1) and by existence of inverses $(A + 1)^{N-1} = 1 \mod N$. QED.

Lets check $N = 7$ and $A = 2$, then $2^7 = 1 \mod 7$ whereas $N = 6$ and $A = 5$, $5^6 - 1 = 5 \neq 1 \mod 6$.

FIRST ALGORITHMIC KEY IDEA: Fermat’s little theorem suggest us a nice way to prove that a number $N$ is not prime without necessarily having *any* idea how to factor it: if we
can find an \( a < N \) such that \( a^{N-1} \neq 1 \mod N \), then this proves that \( N \) is not prime so we know that \( N \) is composite. (Remember, we can do exponentiation quickly even though \( N \) is a huge number).

Here is our first attempt at a randomized primality test.

**FERMAT-PSEUDO-PRIME-TEST (N)**

1. pick a random \( A \) between 1 and \( N - 1 \).

2. if \( A^{N-1} \neq 1 \mod N \), return COMPOSITE else return “Probably PRIME”.

(Note: don’t even have to test \( \gcd(A,N) \) since if \( \gcd \) is not 1 then for sure \( A^{N-1} \neq 1 \mod N \)).

We observe the following:

- First, the algorithm runs in \( O(n^3) \) steps, so its polynomial time.
- Second, on input \( N \) prime, it will always say ‘probably PRIME’, so the algorithm is always correct on \( N \) prime.

The question is: what happens when the input \( N \) is composite? Fermat’s theorem does not tell us what happens in this case. Namely,

1. it is possible that also when \( N \) is composite, for some \( 1 \leq A \leq (N - 1) \) \( A^{(N-1)} = 1 \mod N \) even when \( N \) is composite. e.g. \( N=341 \) and \( A=2 \), \( 2^{340} = 1 \mod (341) \). Namely, Fermat’s theorem does not work in both directions. Just because some numbers \( a \) and \( N \) satisfy, \( a^{N-1} = 1 \mod N \), doesn’t guarantee that \( N \) is prime. So, who says we will pick an \( A \) for which \( A^{N-1} \neq 1 \mod N \)?

2. Even worse: it is possible that for some composite numbers \( N \), for all \( A \), \( A^{(N-1)}=1 \mod N \). Indeed, such composites exist and they are called Charmichael numbers. E.g 561, 1105, 1729. Luckily, they are extremely rare – only 255 of them occur in the first 100,000,000 integers – and they are easy to deal with (as we shall see later), But, lets ignore them for now.

We will now show that ignoring Charmichael numbers we will be ok with good probability.

We need a bit more group theory

We say that a set \( H \subseteq G \) is a **subgroup** of \( G \) if when \( A, B \) are in \( H \) then \( A * B \) is in \( H \) (closure); if \( A \) is in \( H \) then \( A^{-1} \) is in \( H \) (inverse); and if the identity element of \( G \) is in \( H \).
A KEY PROPERTY of groups[Lagrange]: say $G$ is a group and $H$ is a subgroup of $G$ then THE SIZE OF $H$ DIVIDES THE SIZE OF $G$.

Proof: Say $H$ is a subgroup of $G$ and $y$ is not in $H$. Then the coset $yH = \{yh : h \in H\}$ is a set of size $|H|$ (if $y \cdot h1 = y \cdot h2$ then $h1 = h2$) and is disjoint from $H$ (if $y \cdot h1 = h2$ then $y = h2 \cdot h1^{-1}$, which is in $H$ by $H$’s group closure properties). Furthermore, all cosets are disjoint (if $z \cdot h1 = h2$ then $z = y \cdot h3$ for some $h3$ in $H$).

THEOREM: For composite $N$, (where $N$ is NOT a Charmichael number) $Pr$(Fermat Prime Test Says COMPOSITE on input $N$) $> 1/2$

Proof: Since we assumed that $N$ is composite and not a Charmichael number, there exists at least one $A'$ such that $(A')^{N-1} = 1 \pmod{N}$. In order to prove theorem we have to show that there are actually lots of of such A’s. In fact we will show that for at least a half of the A’s in $\mathbb{Z}_N^*$, $A^{N-1} \neq 1 \pmod{N}$.

Define the set $H = \{A^{N-1} = 1 \pmod{N}\}$. This is the set of A’s on which our algorithm says ‘probably PRIME’ even though $N$ is composite. Moreover, this is a subgroup of $\mathbb{Z}_N^*$ (verify it! if $A, B \in H$ then $(AB)^{N-1} = 1$ (closure), if $A^{N-1} = 1 \pmod{N}$, then $B = A^{-1} \pmod{N}$ is such that $B^{N-1} = (A^{N-1})^{-1} = 1 \pmod{N}$, and $1 \in H$).

By our key property about groups above, $|H|$ divides $|\mathbb{Z}_N^*|$ and as $A' \notin H$, $H \neq \mathbb{Z}_N^*$, and thus $|H| \leq \frac{|\mathbb{Z}_N^*|}{2} < \frac{(N-1)}{2}$.

The theorem now follows easily: When the algorithm chooses at random $A \in \mathbb{Z}_N^*$, the probability that that he finds an A in $H$ is at most $\frac{1}{2}$ so the probability that he finds an A such that $A^{N-1} \neq 1 \pmod{N}$ which is out of $H$ is at least 1/2 and the algorithm will say COMPOSITE. QED

Can we do to boost up our probability of our randomized primality to say COMPOSITE on input composite $N$ (and not charmichael) with probability $1 - \frac{1}{2^k}$?

FERMAT-PSEUDO-PRIME-TEST'(N,k)
Repeat k times

1. pick a random A between 1 and $N - 1$.
2. if $A^{N-1} \neq 1 \pmod{N}$, return COMPOSITE

otherwise return Probably PRIME.
Claim: For composite \( N \), where \( N \) is NOT a Charmichael number, \( \Pr( \text{Fermat Prime Test Says COMPOSITE on } N ) \geq 1 - \frac{1}{2^k} \).

Proof: Since the algorithm actually chooses \( k \) random and independent \( A \)'s, the probability that each of them is in \( H \) (defined in the previous proof) is at most \( \frac{1}{2} \) as these are independent choices, the probability that all of them are in \( H \) is at most \( \frac{1}{2^k} \), and thus the probability that for at least one of them \( A^{N-1} \neq 1 \mod N \), is greater than \( 1 - \frac{1}{2^k} \). QED

Are we done? No! We promised to give a Monte Carol algorithm which for all input \( N \), \( \Pr(\text{output of the algorithm is correct } t) \) is high. What about Charmichael Numbers? The full primality test we present is due to Miller-Rabin 1976. They show how to augment the Fermat Test to handle all inputs \( N \).

Whereas above we based our idea on Fermat’s little theorem. Here we need another idea.

**Modular Square Root Theorem:** if \( N \) is prime, then the equation \( x^2 = 1 \mod N \) has only 2 solutions in \( Z_N \), \( x=1 \mod N \) and \( x=-1 \mod N \).

Proof of Modular Square Root Theorem: Do it by counter positive. Suppose that \( x^2 = 1 \mod N \) but \( x \neq 1, -1 \mod N \). Then \( (x^2 - 1) = 0 \mod N \), or \( (x-1)(x+1) = kN \) for some \( k \), but neither \( x-1 \) nor \( x+1 \) are multiples of \( N \), As \( N \) divides the product, that \( N \) has some factors(divisors) that divide \( x+1 \) and other factors(divisor) that divide \( x-1 \) but then \( N \) cannot be prime. \(^1\) QED

**SECOND ALGORITHMIC KEY IDEA:** This theorem suggests another way to find evidence that a number \( N \) is composite: On input \( N \): if you find a square root of 1 mod \( N \) which is not 1 or -1. Then you have evidence that \( N \) is composite. Luckily this time, it turns out that if \( N \) is a charmmichael number (actually any composite \( N \) which is not even or a prime power), then the equation \( x^2 = 1 \mod N \) has at least 4 solutions, so there exists at least 2 roots of 1 which are not 1 or -1. Moreover for Charmichael numbers, these are easy to find.

The final missing piece is: how do we quickly find square roots of 1 which are not +1 or -1???

Here is the final idea.

**THIRD ALGORITHMIC KEY IDEA:** When we chose an \( A \) in the Fermat test, and \( N \) was a Charmichael numbers, \( A^{N-1} = 1 \mod N \), we now use this equation to try to find square roots of 1 which are not +1 or -1. Compute \( A^{(N-1)/2} \mod N \) which is a square root

\(^1\)actually this means that \( \gcd(x-1,N) \) gives us a factor \( N \), recall \( \gcd \) is efficient so actually this means that Charmichael numbers are easy to factor.
of 1. If its value is still 1, keep dividing the exponent by two: Namely, compute $A^{(N-1)/2^i} \mod N$ for $i = 1, 2, \ldots$ as long as $(N - 1)/2^i$ is an integer (i.e till reach $u$ such that $u$ is odd $(N - 1) = u2^k$); if you ever hit a number different than 1 or -1, Bingo! You found a root of 1 which is not 1 or -1 and therefore N is composite. If you hit -1, choose another A and try again. What can be formally shown (actually easily using our key property of groups) is that if N is Charmichael number, then with each A you have a probability $1/2$ of finding a square root of 1 which is not 1 or -1 and therefor detecting that N is composite.

Finally!

Miller-Rabin Primality Test (N,k)[Modified]

1. if N is 2, output PRIME
2. if N is even, output ‘COMPOSITE’
3. if N is a perfect power, i.e $N = a^b$ for $a, b > 1$ output ‘COMPOSITE’
4. Let $(N - 1) = u2^l$ (where u is odd)  
   Repeat $k$ times
      (a) Pick a random A in $\{1, \ldots, N - 1\}$ compute $A^{N-1}$. If its not 1, output COMPOSITE
      (b) Compute the sequence $A^{N-1}/2, A^{(N-1)/4}, \ldots, A^u \mod N$
      (c) find the first element in this sequence which is not 1 (if such element exists), and if its not -1, output COMPOSITE
5. If did not output COMPOSITE yet, output ‘Probably PRIME’

THEOREM: The Miller-Rabin test on input N

- runs in $\Theta(kn^4)$
- on input prime N always outputs ‘probably PRIME’
- on input composite N, outputs COMPOSITE with probability $1 - \frac{1}{2^k}$

To prove this theorem, we need to bound the one final lemma.

FINAL LEMMA: if N is a Charmichael number then $\text{Prob}(\text{Miller Rabin on input N outputs COMPOSITE})$ greater than $1 - \frac{1}{2^k}$

PROOF: Let $t < N$ and $H^t = \{A : A^t = 1 \mod N \text{ or } A^t = -1 \mod N\}$ It is easy to see that H is a sub-group of $Z_N^*$. We claim that
(1) if there exists an $A \in Z_N^*$ such that $A^t \neq 1 \mod N$ then $H^t$ is a proper subgroup of $Z_N^*$ (i.e there exists an $x$ such that $x^t \neq 1 \mod N$ and $x^t \neq -1 \mod t$ (i.e $x \notin H$) and thus by our important property about groups, at least half of the B’s in $Z_N^*$ are not in $H^t$.

(2) if $t$ is odd, then there always exist $x \in Z_N^*$ such that $x^t \neq 1 \mod N$.

Both (1) and (2) use the Chinese remainder theorem (see book chapter 31)
(1) Take the $A$ such that $A^t \neq 1 \mod N$. $N = N_1N_2$ for $gcd(N_1, N_2) = 1$ since $N$ is not a perfect prime power. Now, set $1 \leq x < N$ to be the $x$ which satisfies the following two equations: $x = A \mod N_1$ and $x = 1 \mod N_2$ (use notation CRT says there is always a unique $x \mod N$ which satisfies these equations). Look at $x^t = A^t \mod N_1$ and $x^t = 1^t = 1 \mod N_2$. Since $A^t \neq 1 \mod N$, we also know that $A^t \neq 1 \mod N_1$. So, we found an $x$ such that $x^t \neq -1, 1 \mod N$.

(2) This time, set $1 \leq x < N$ to be the $x$ which satisfies the following two equations: $x = 1 \mod N_1$ and $x = -1 \mod N_2$ (CRT says there is always an $x$ which satisfies this and its unique). Since $x^t = 1 \mod N_1$ and $x^t = -1 \mod N_2$ (since $t$ is odd), then $A^t \neq 1 \mod N$ (by CRT if the latter was true then both $A^t = 1 \mod N_1$ and $A^t = 1 \mod N_2$).

Having shown (1) and (2) it is easy to establish our theorem. With probability at least one half we will choose an A in the algorithm (when N is charmichael) such that $A^{N-1} = 1 \mod N$ and the first $i$ when $A^{\frac{N-1}{2^i}} \neq 1 \mod N$ also will be different than $-1$.

QED