

Trial Division

What is Trial Division?

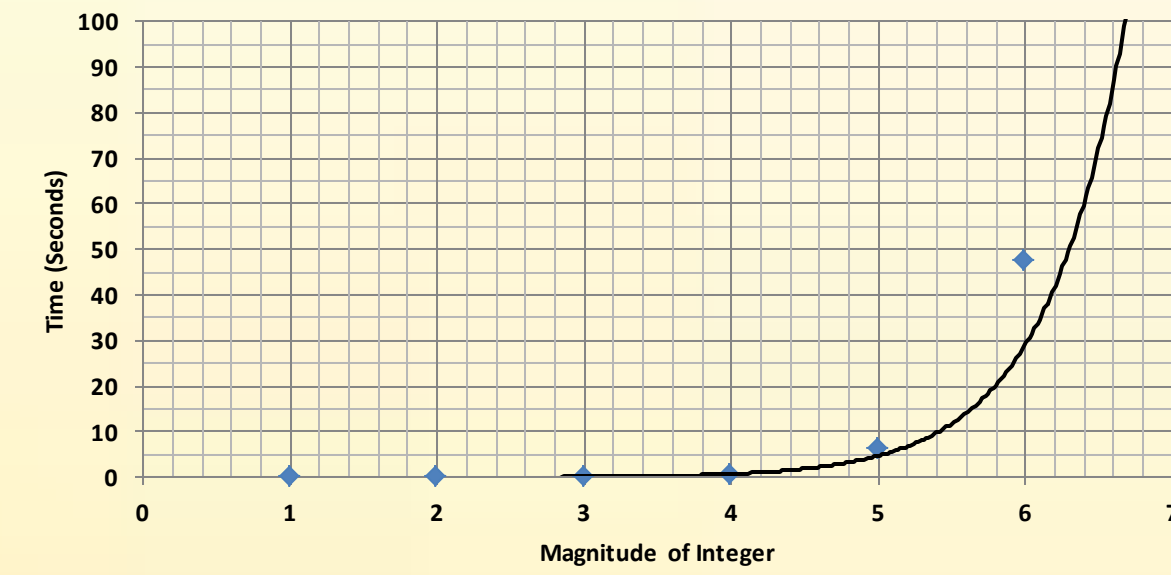
Trial Division is a factoring method involving testing various primes less than the largest integer less than the square root of a non-prime integer until a factor is found.

In our example, we are testing a composite number “n” and assigning the square root of n to “b”. We test each consecutive prime number starting from 2, and output the number if it evenly divides n.

Algorithm

```
trialDiv := proc (n)
local b, i, d;
d := 0;
b := floor(sqrt(n));
x := 1;
while x < 2 to b do
x := nextprime(x);
if modp(n,x) = 0 then
return x, n/x;
fi;
od;
end proc;
```

Simulation



Here, each integer and the time it took to find its first non-trivial factor is plotted. This plot displays how trial division takes considerably longer each time the number factored is expanded another digit and how it is slower than both the other methods. The simulation follows an exponential growth curve.

Pros and Cons

Pros	Cons
Works best for smaller numbers	Loses efficiency with larger numbers.
88% of all composite numbers have a prime factor under 100	The most resource intensive procedure for factoring
92% of all composite numbers have a prime factor under 1000	Takes the most time.
Will always find a prime factor	Must check every prime from 2 to the square root of the number being factored

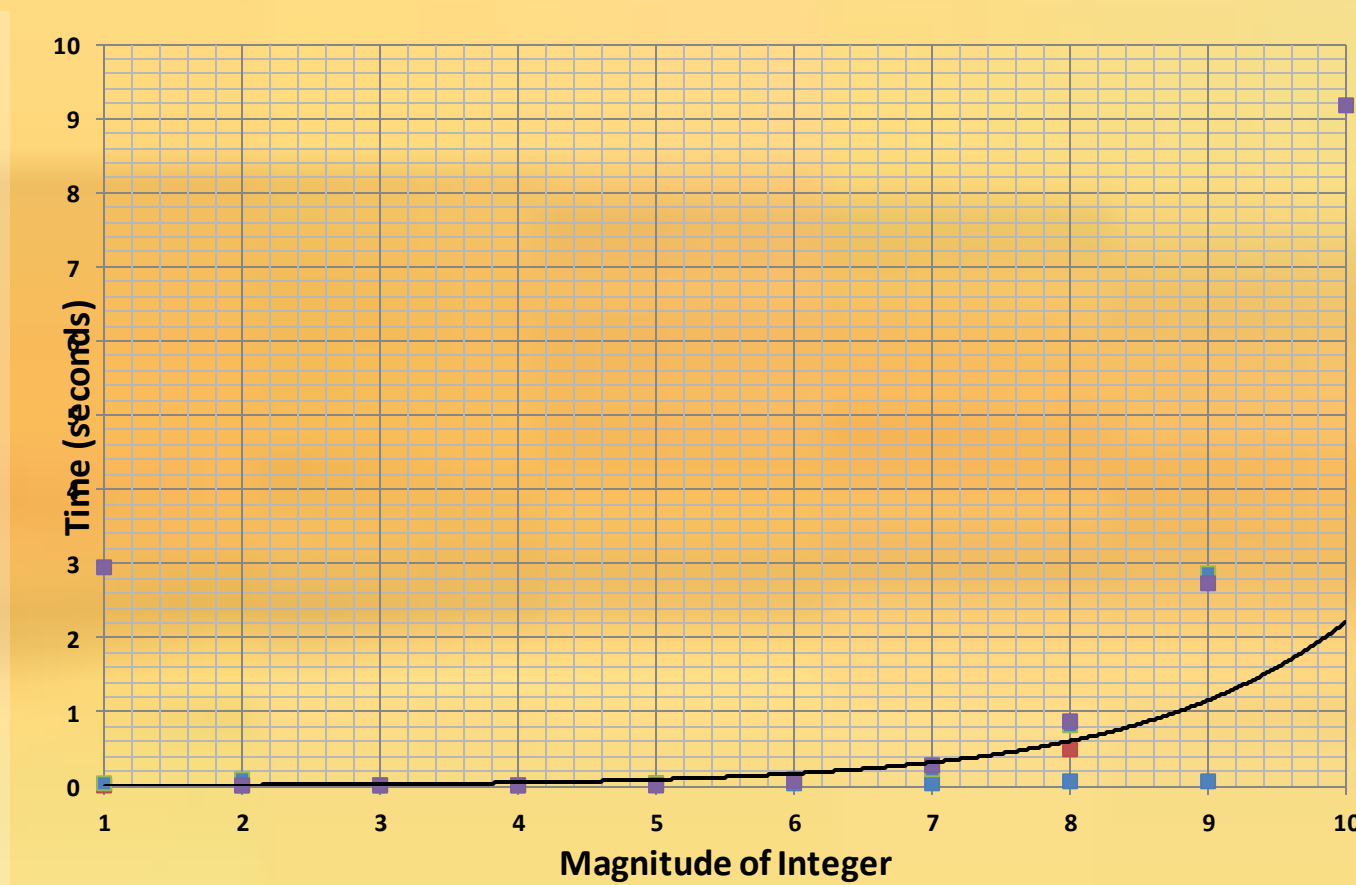
Pollard's Rho (ρ)

Pollard's Rho Algorithm, invented by John Pollard in 1975, is used to find small, non-trivial factors for composite integers. This is done by creating a function that loops until a collision occurs within itself. A collision is defined as an area the function has already encountered.

Algorithm

```
pollRho := proc (m)
local a, b, i, d;
a := 2;
b := 2;
for i to infinity do
a := mod(a^2+1, m);
b := mod(b^2+1, m);
d := gcd(a-b, m);
if 1 < d and d < m then
return d;
elif d = m then
return 0;
end if;
end do;
end proc;
```

Simulation

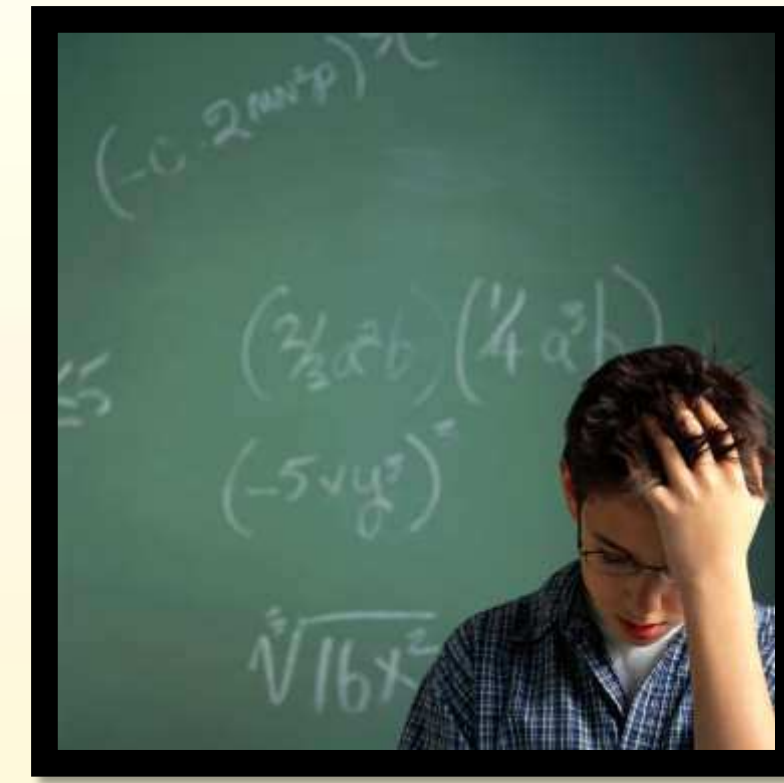


Pros and Cons

Pros	Cons
Can yield a number very quickly depending on the chosen random numbers	Can take a very long time to yield simple factors based on chosen numbers
Considerably more efficient than trial division	Will not always yield a prime factor
Offers a good balance between running time and probability of success	Multiple tests may be necessary to generate a prime factor
Running multiple tests is easy; variables can be manipulated easily	

Fact-or Fiction

Participants:
Alexander Melton, Peter Rodriguez, Paul Mayer, Brandon Huynh
Supervised by:
Frank Madrid, Dr. Charles Lam

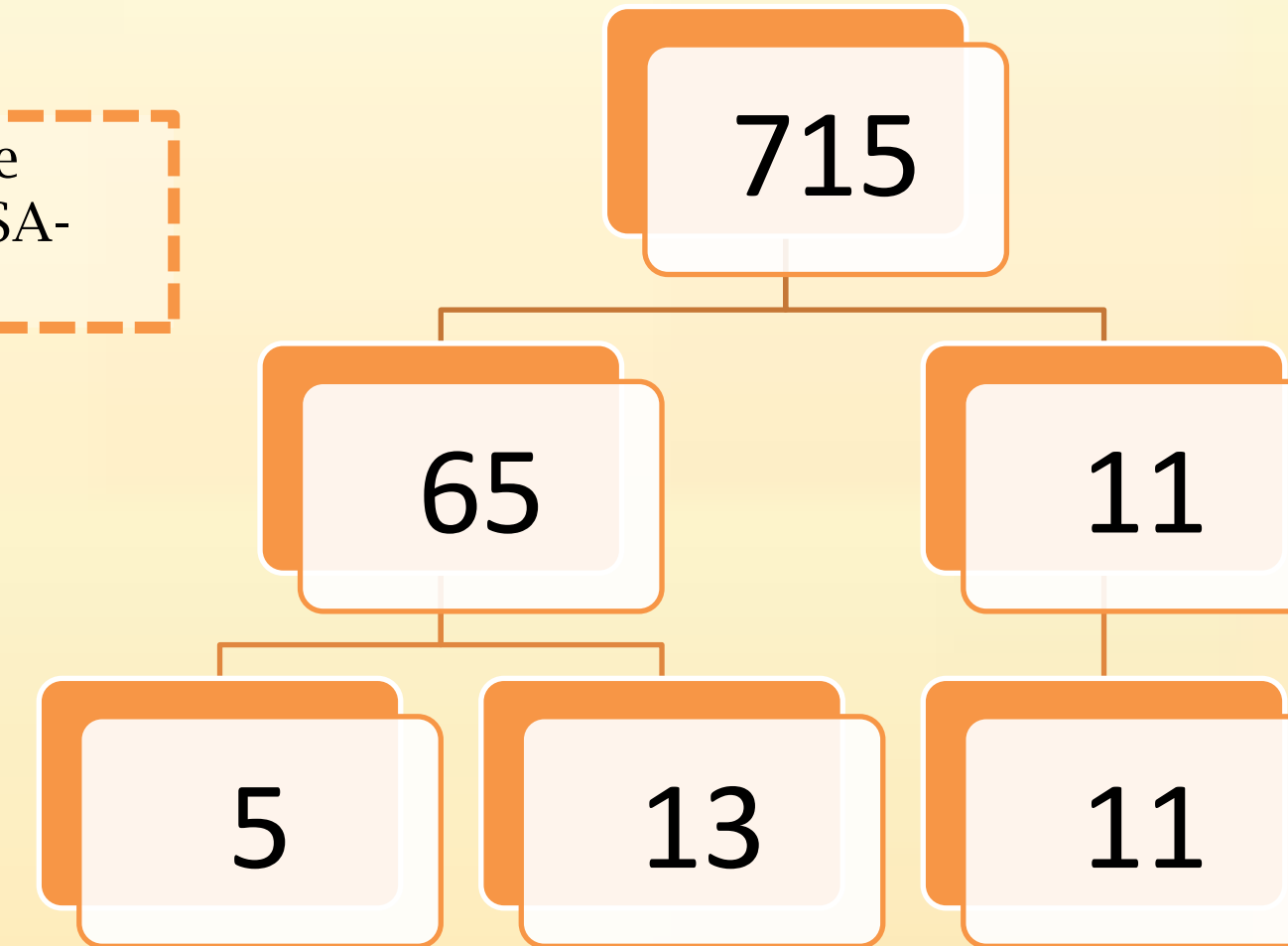


Background

Integer/Prime Factorization
-Breaking down a composite integer into its prime factors, which when multiplied together results in the original number.
-There are several special methods used known as factorization algorithms.



Example of Prime Factorization:



Fun fact-or: There is a \$200,000 cash prize for the successful factorization of a number known as RSA-2048 (its 617 decimal digits long)!

Fun fact-or: The largest semi prime number ever factorized was RSA-768, a 768 bit number 232 digits long!

It would take over 2,000 years on a 2.2 GHz AMD Opteron processor.

Information about the tests:
All the tests conducted, while using different methods of factoring, use the same procedural method of choosing the numbers to be factored to maintain the integrity of the experiment. The magnitude of integer axis on the graphs, for example, represents the range of random prime numbers p and q chosen by the computer. The range of each is (1*10^x...1*10^{x+1}). These two primes are then multiplied together to get the number that will be factored by each method (p*q).

What do these two completely different subjects have anything to do with each other?

Fun fact-or: Semi-prime numbers are difficult to find. 88% of all integers have a factor under 100, and 92% have a factor under 1000. Therefore, precautions must be made to choose semi primes that are not made up of smaller, more easily tested primes.

Factoring & Cryptography

Factoring large composite numbers into its prime factors has long been a goal of mathematicians. While several methods do exist with a variability in the degree of success, no definitive way has emerged as superior to all for factoring each number. Rather, different cases require different methods to break these numbers down.

Not all large composites (at least 9 digits long) can easily be factored into smaller prime numbers. For example, semi-prime numbers are difficult to factorize because they are the product of two primes. While testing all primes up to the square root of a number is feasible for smaller numbers, RSA encryption schemes recommend 232 or more digits, a process that can take years with this method, even when using hundreds of computers simultaneously. Therefore other practical methods must be discovered to factor these numbers.

There are two types of factoring methods, general purpose and special purpose. The time taken by general purpose algorithms depends solely on the size of the integer to be factored, while that of special purpose algorithms depend on the size of the smallest prime factor.

The security of many cryptographic systems, such as the RSA encryption/decryption algorithms, depend on factoring large numbers due to this difficulty. If a way was produced to easily factor large composites, especially semi-primes, RSA and other such methods would be rendered obsolete.

RSA Algorithm:

Key Generation:

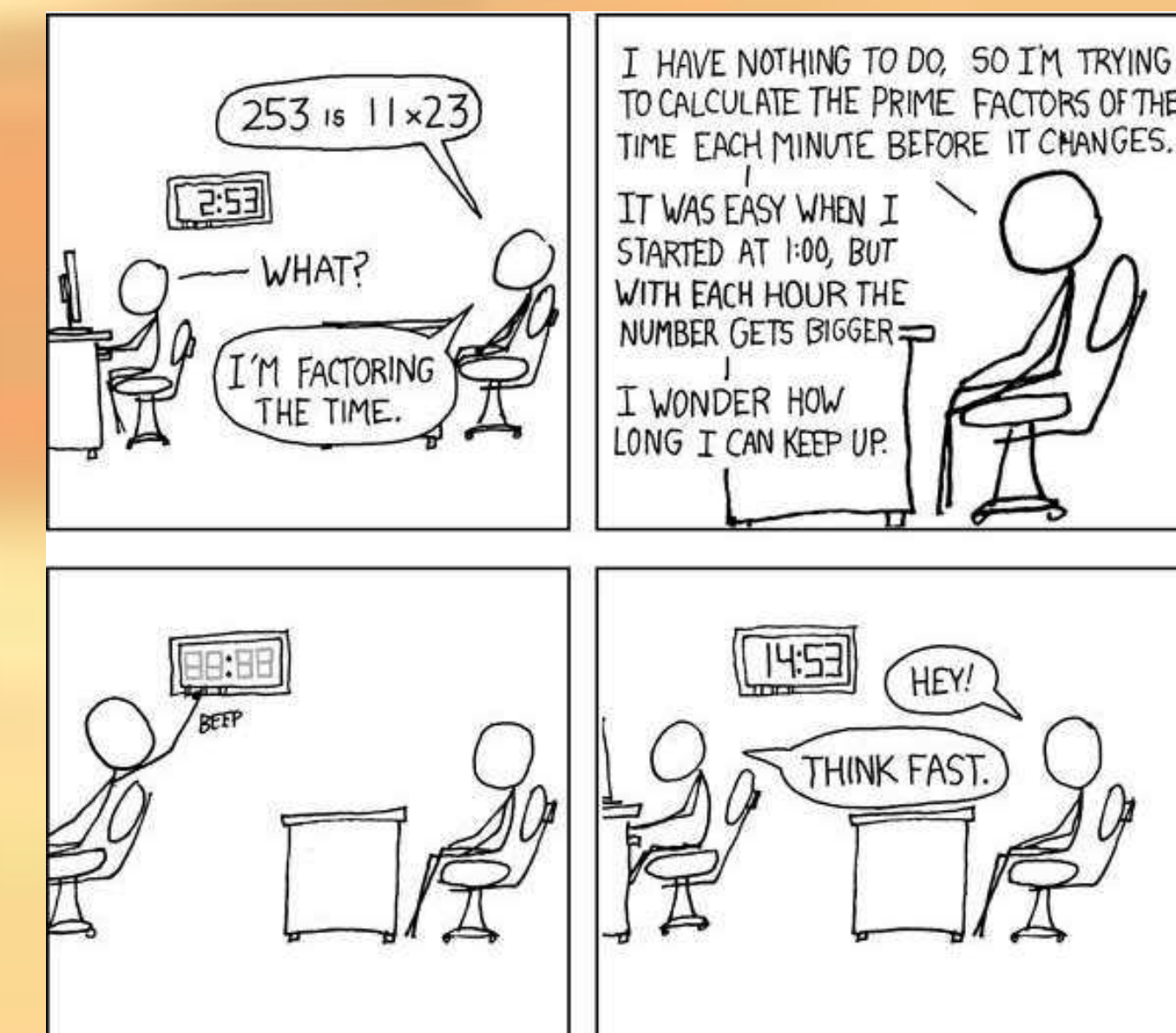
- Choose 2 large primes (P) and (Q).
- Compute n=(p)(q).
- Compute $\phi(n)=(p-1)(q-1)$.
- Select a public exponent e, where $1 \leq e \leq \phi(n) - 1$ and $\gcd(e, \phi(n))=1$.
- Compute private exponent (d) such that $(e)(d) \equiv 1 \pmod{\phi(n)}$.
- Alice's public key is (n,e) and her private key is d.

Encryption:

- Convert message to an integer m, where $1 \leq m \leq n$.
- Compute $c=m^e \pmod{n}$.
- The encrypted message is c. Bob sends c to Alice.

Decryption:

- Compute $m'=c^d \pmod{n}$.
- The decrypted message is m'=m.



Pollard's P-1

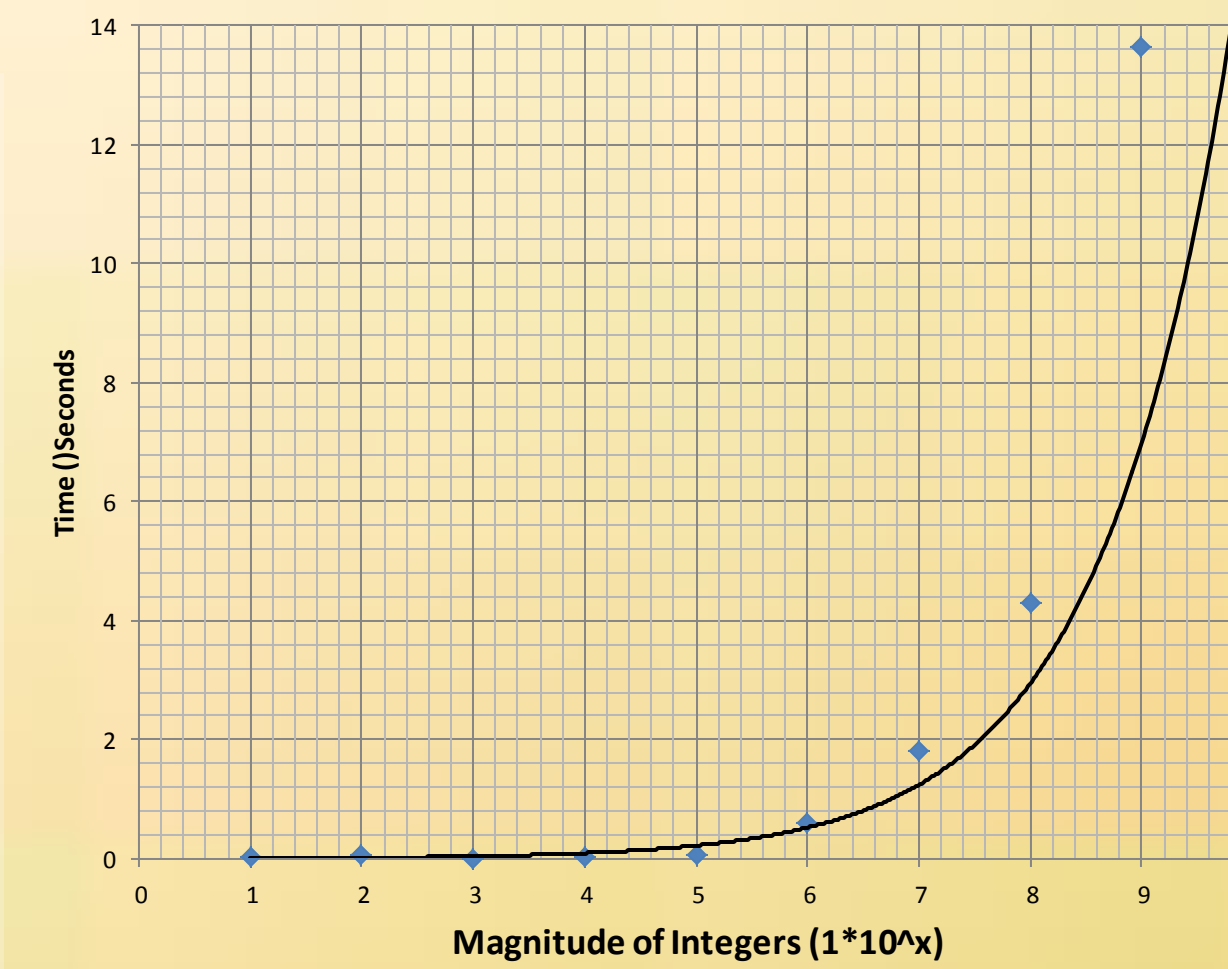
What is Pollard's P-1?

Pollard's p-1 is an “algebraic-group factoring algorithm” invented by John Pollard in 1974 and used to find prime factors of a composite integer n for which p-1 considered “smooth” with respect to a small bound B, where p is the prime factor. A number is “B-smooth” when all of its prime factors are less than B.

Algorithm

```
pollPm0 := proc (o)
local a, b, d, l, i;
a := 'mod'(rand(o), o-2)+1;
print(a);
b := 56;
d := gcd(a, o);
if 2 <= d then
return d
else for i from 2 to b do
if typed(i, prime) then
l := floor(ln(o)/ln(i));
a := 'mod'(a^(i^l), o);
d := gcd(a-1, o);
if d = 1 or d = o then
print("EPC FAIL")
else return d
end if
end if
end do
end if
end proc;
```

Simulation



The relationship between magnitude of integers tested and the time in seconds to factor closely follows an exponential growth model; in other words, it takes considerably longer each time the number is expanded than the previous time.

Pros and Cons

Pros	Cons
The longer the number, the larger the chance the algorithm will yield a factor	The longer the number, the longer the algorithm takes
Considerably faster and less resource intensive than Trial Division	Each successive trial takes longer and uses more resources
Can be run incrementally for increasing values of “b”	Does not always yield a non-trivial prime factor of the original number
Can be modified to find a prime larger than “B”	This does require a modification. And a separate algorithm

Conclusion

As one can see, each of the three factoring methods discussed, Trial Division, Pollard's Rho and Pollard's P-1, all have their own unique strengths and weaknesses. Trial Division, for example, takes longer than both Pollard's Rho and Pollard's P-1, however it yields more results. Many other methods also exist, and many more are currently in development. Obviously, the need for factoring still remains, and it will still remain a “prime” goal of many mathematicians.

Other Factoring Methods...

- Random square factoring
- Quadratic sieve factoring
- Number field sieve factoring

References

- Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. [Handbook of Applied Cryptography](#)
- Randall Munroe. XKCD

