Preface

Integer factorization is an integral part of modern cybersecurity and encryptions. Being both computationally heavy and time consuming, the prime factorization can require weeks of work by hand, and hours of work using a computer. This is known as the factoring problem, and currently, the only known method of quickly cracking an encryption is using a quantum computer that is yet to exist. For this reason, quantum cryptography is one of the most anticipated fields of study. Since the quantum computer that can break modern encryptions is yet to exist, one of the most common methods of solving for prime factors is known as the Pollard’s Rho algorithm. I will be attempting to learn, understand and code this algorithm as my main assignment for this class (Computer Programming 2). I was unable to understand the algorithm the first time I heard about it in February and this will be my attempt to learn a concept that I was incapable of 6 months ago.

The Algorithm

Pollard’s Rho is an algorithm named solely based on the creator’s name and the algorithm’s shape. Fundamentally, the algorithm works using a few concepts. The first concept is known as the birthday paradox. This concept states that given a group of people *n*, the chances of 2 people having the same birthday goes up drastically, reaching 50.7% when *n* = 23. This concept is important because when applied to Pollard’s Rho, it means that the number of “steps” required before a repetition is found is(This is a common notation for computational complexity). In comparison, the current best factoring algorithm takes time. The other concept is known as the Floyd cycle finding algorithm, or the “tortoise and hare” algorithm. The algorithm works by having 2 different variables, the first being the tortoise and the other as the hare. For every “step” of Pollard’s Rho the tortoise does, the hare does 2 steps, essentially moving quicker than the tortoise. After taking the GCD (greatest common divisor) of tortoise - hare and the number you are trying to factor, if the GCD 1, that GCD is one of the factors of N. Though the concept might be a little difficult to understand, it is simpler to just understand the processes and code instead.

Given a number *n*, randomly choose 2 integers, *x* and *c* such that those 2 integers are less than *n*, *x = y* and g() = mod *n.* For the purpose of the code, *x* and *c* and be randomly chosen or inputted by the user but *n* must be inputted by the user. To perform the first “step” of the tortoise algorithm, update the value of x to g(*x*) and for the first “step” of the hare algorithm, update the value of y to g(g(y)). Afterwards, take the GCD of mod n. If the GCD 1, the GCD is a factor. If not, repeat the algorithm as the second “step” and if it fails again, continue until the GCD 1, or the code fails. If it fails, try again using new parameters or test for primality.

Table Trace (examples)

Table 1.1 (*n* = 713, *x =* 20, *c* = 1)

|  |  |  |
| --- | --- | --- |
| *x* | *y* | GCDmod *n* |
| 20 | 20 |  |
| 401 | 377 | 1 |
| 377 | 584 | 1 |
| 243 | 584 | **31** |

713/31 = 23

Table 1.2 (*n =* 12209*, x* = 100, *c =* 53)

|  |  |  |
| --- | --- | --- |
| *x* | *y* | GCDmod *n* |
| 100 | 100 |  |
| 10053 | 8969 | 1 |
| 8969 | 9218 | 1 |
| 10122 | 5510 | 1 |
| 9218 | 3442 | 1 |
| 9146 | 4031 | 1 |
| 5510 | 2079 | 1 |
| 8579 | 9454 | 1 |
| 3442 | 7531 | 29 |

12209/29 = 421