# Error Detection in Number-Theoretic and Algebraic Algorithms

by

Troy Michael John Vasiga

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

**Abstract**

CPU's are unreliable: at any point in a computation, a bit may be altered with some (small) probability. This probability may seem negligible, but for large calculations (i.e., months of CPU time), the likelihood of an error being introduced becomes increasingly significant. Relying on this fact, this thesis defines a statistical measure called *robustness*, and measures the robustness of several number-theoretic and algebraic algorithms.

Consider an algorithm $A$ that implements function $f$, such that $f$ has range $O$ and algorithm $A$ has range $O'$ where $O \subseteq O'$. That is, the algorithm may produce results which are not in the possible range of the function. Specifically, given an algorithm $A$ and a function $f$, this thesis classifies the output of $A$ into one of three categories:

1. Correct and feasible – the algorithm computes the correct result,
2. Incorrect and feasible – the algorithm computes an incorrect result and this output is in $O$,
3. Incorrect and infeasible – the algorithm computes an incorrect result and output is in $O' \backslash O$.

Using probabilistic measures, we apply this classification scheme to quantify the robustness of algorithms for computing primality (i.e., the Lucas-Lehmer and Pepin tests), group order and quadratic residues. Moreover, we show that typically, there will be an "error threshold" above which the algorithm is unreliable (that is, it will rarely give the correct result).

**Acknowledgements**

I would first like to thank Professor Jeffrey Shallit who supervised me during this extremely long journey. His outstanding editorial skills, patience and dedicated pursuit of mathematical truths were touchstones that kept me moving forward.

I would also like to thank my examining committee: Professor Ian Munro, Professor George Labahn, Professor Cameron Stewart and Professor Leslie Davison. I very much appreciate their use of some of their summer reading energy for this work, which is about as far removed from summer reading material as one can get.

Thank you $\epsilon$.

Finally, I would like to thank Krista for her patience, love and understanding.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction and Motivation

Computers currently are, and always have been, unreliable.

Historically, when "computers" were those people who calculated extensive arithmetic formulae (sequences, factors of integers, logarithms of integers, etc.) mistakes (such as an incorrectly written digit, or transposed digits) would invariably occur.

For example, the French mathematician Edouard Lucas devoted some of his computational energy towards determining perfect numbers. Recall that a *perfect number* is a positive integer $n$ such that $\sum_{\substack{d|n \\ 1 \leq d < n}} d = n$. Euclid demonstrated that if $2^p - 1$ is prime, then $(2^p - 1)2^{p-1}$ is a perfect number. Furthermore, Euler proved that all even perfect numbers must be of the form $(2^p - 1)2^{p-1}$. Lucas [61], relying on the above theorems and his extensive computations stated that "Nous pensons avoir démontré par de très longs calculs qu'il n'existe pas de nombres parfaits pour $p = 67$ et $p = 89$"[1]. Powers [78] later demonstrated that Lucas made an error in his computations, by proving that $2^{89} - 1$ was prime, which implies that there is a perfect number of the form $(2^p - 1)2^{p-1}$ where $p = 89$. (It is worth noting that Lucas was correct in determining that $(2^{67} - 1)2^{66}$ is not a perfect number.)

A more well-known computational error was that of William Shanks (c. 1873) [39]. Shanks claimed to have computed $\pi$ to 707 decimal digits using applications of trigonometric expansions [86]. In 1944, D. F. Ferguson reviewed Shanks' computations more thoroughly, and discovered that Shanks introduced an error at the $528^{th}$ decimal place, which revealed that the last 179 digits in Shanks' computation were incorrect. Ferguson and Wrench [33] formalized these results.

[1] "We think we have demonstrated by lengthy calculations that there do not exist perfect numbers for the values $p = 67$ and $p = 89$."

In more modern times, silicon-based computation can be adversely affected by *soft errors*. A soft error can be loosely defined as an error that is a random, non-destructive event that alters one or more bits of information stored in a RAM cell or CPU register [101]. In early electronic circuits, soft errors could be introduced by faulty power systems or surges of electricity (e.g., lightning). The problem of maintaining an unwavering power supply was solved by computer manufacturers by shielding and grounding power supplies (see Ziegler's introduction [101]). Once the electrical control problems were solved, research focused on soft errors caused by radioactive particles present in earthly materials, as well as the soft errors caused by cosmic radiation emitted from space [101]. In fact, a "solar maximum" occurs every 11 years, causing increased bursts of radiation, which can disturb radio and satellite communication [69]. One result that provides additional motivation for studying soft errors is that as computing devices are moved to higher altitudes from the surface of the earth, the soft error rate increases dramatically [101] As an example, computers in high altitude aircraft have a soft error rate that is approximately 100 times the rate at sea-level [66]. Heidergott [45] provides an excellent overview of soft errors from an engineering perspective, as well as interesting measures of the rate of errors. As an example, Heidergott states that in some satellite systems, there are more than 200 soft errors per day, with peak error rates of 32 soft errors per minute "during intense solar particle events" [45].

Soft errors differ considerably from *hard errors* which are more physical in their nature. A manufacturing defect in RAM that causes predictable, repeatable memory failure, such as a bit always being set to 0, is one type of hard error. Possibly the most "famous" (or at least, largest in terms of media exposure and economic consequences) hard error was the incorrect division wiring on the Pentium chip: Blum and Wasserman [15] provide a mathematical reflection on this hard error. Stallings [88] provides a thorough introduction to hard and soft errors.

While it is debatable whether economics should motivate research, it is the case that "NASA alone expects to spent up to $10 million on the quest for 100% reliable software" [37]. As further motivation, researchers at IBM [20] have concluded "[f]or every 256 Megabytes of memory, you'll get one soft-error a month."

In this light, the work in this thesis illustrates how certain, well-known algorithms compare (in terms of accuracy and probability of success) when soft errors occur.

## 1.1 Related work

There has been considerable analysis of algorithms under error.

Von Neumann [96] studied the effects of errors on deterministic finite automata. In fact, von Neumann viewed error "not as an extraneous and misdirected or misdirecting accident, but as an essential part of the process under consideration." [96, page 1]. This work has since been extended, notably by Delyon and Maler [30], who define the concept of a synchronizing deterministic finite automaton.

For example, one problem that has been studied is the problem of searching under error: in an ordered list of elements, we wish to locate a particular element. However, the query mechanism (such as "Is the current value less than the desired value?") may be faulty (e.g., errors cause the response of queries to be incorrect). Rivest et al. [82] explored binary searching where the output is incorrect by finding an optimal continuous solution and using it to solve the discrete problem. Czumaj and Sohler [27] introduced soft kinetic data structures, which allow some (fixed) variation in the sorted properties, much in the same way that AVL trees (originally presented in Adel'son-Vel'skii and Landis [1] and thoroughly explained by Lewis and Denenberg [57]) allow flexibility in the definition of balanced binary search trees. Feige et al. [32] modelled the computation of binary search algorithms by way of decision trees and gave bounds on the number of repeated executions by way of "majority voting" (i.e., running the algorithm in parallel, and using the output which is given by a majority of processors). Pelc considered searching where there are a fixed number of erroneous queries [73] and also a fixed probability that queries are erroneous [74]. Borgstrom and Rao Kosaraju [16] used amortized analysis to quantify searching when there is a fixed proportion of erroneous queries, which was also considered by Aslam and Dhagat [4]. Muthukrishnan [68] used the *defective coins problem* to correctly find an element when there are at most $O(\log \log N)$ erroneous queries out of a total of $N$ queries. Ravikumar and Lakshmanan [80] consider searching when there is a known set of possibly lying patterns.

Adler et al. [2] extended the work of Ravikumar and Lakshmanan to modelling an elimination tournament that minimizes the number of games required, using the fact that there are "erroneous" results of games if the expected winner does not, in fact, win.

A related problem is the problem of sorting: given a sequence of elements, place them in (increasing) order. Again, much work has been done in this area, of which we discuss a small sampling. Lakshmanan, Ravikumar and Ganesan [54] considered the problem of sorting in both the "half-lie" case (only "no" answers to

queries may be erroneous; all "yes" answers are guaranteed to be correct) and "full-lie" case (either "yes" or "no" answers may be erroneous), and they established a lower-bound on the full-lie case. Long [60] used the search results of Rivest et al. [82] to improve on the running time found by Lakshmanan, Ravikumar and Ganesan. Ravikumar [81] considered an upper bound on the number of erroneous queries that could occur in merge-sort algorithm and yet still have runtime $O(n \log n)$.

## 1.2 Self-checking algorithms

Some algorithms have more inherent ability to detect (and possibly recover) from errors that occurred during computation. For instance, an integer factorization algorithm will be able to verify that the product of the outputted factors equals the original integer input.

In other algorithms, however, it may be more difficult to perform a verification step to determine if the output is correct. As an example, primality testing typically outputs whether or not an integer is prime or composite, with no short "proof" of the correctness of the output (other than the possibility of viewing the sequence of calculations that derived the result).

There have been several arguments that certain algorithms have a "high likelihood" of success. Powers stated that Lucas' method of determining primality (see Chapters 5 and 6) "is free from any uncertainty as to the accuracy of the conclusion that the number under consideration is prime...since an error in calculating any term of the series would have the effect of preventing the appearance of the residue 0 [which indicates primality]" [78]. Uhler took a less absolute opinion of the accuracy of Lucas' test under errors in computation. He claimed that the chance of stating that a composite number of the form $2^p - 1$ is prime is "utterly negligible, although not impossible" [93]. However, the opposite case of claiming a prime number of the form $2^p - 1$ is composite is "possible", even with the "utmost care and impeccable honesty" in the computations [93].

It is worth noting, however, that the preceding arguments offer little in the way of formally proving any of the claims, and as such, the claims are mere opinion.

Blum presented a more formal approach to determining the reliability of algorithms that have errors introduced: Blum defined the concepts of *program checking* and *self-checking programs*. In short, a *program checker for program* $P$ is a program $C$ that takes an (input, output) pair, and either certifies that the output of $P$ on the input was correct, or declares that $P$ is "buggy"(i.e., incorrect). A strict bound exists, however, on the running time of the checker $C$, in the sense that the

running time of $C$ is $o(P)$. This running time restriction ensures that the checker is not just the original program $P$, and also that running the checker $C$ will not dramatically impact the running time of the main program $P$. *Self-checking* (or *self-testing*) models are similar to the program checker model described above, but rely on the fact that program $P$ is not "too faulty on average" [14] to create a more general program checker. A more thorough explanation to program checking is presented by Blum and Kannan [13] and Wasserman and Blum [97]. Self-checking is described in full by Blum, Luby and Rubinfeld [14]. The work by Blum has also been extended to *self-correcting* algorithms (which rely on *spot-checkers*) as outlined by Ergüun et al [31].

In summary, program checking has moved from informal arguments to more formal methods. The work in this thesis branches off from the work of Blum on numerical problems (e.g., matrix multiplication) to the area of number-theoretical problems. Most crucially, however, the work in this thesis differs from the work of Blum et al. in that Blum's checker algorithms must run (asymptotically) faster the program which they are checking. Our work is not about checking, certifying results or detecting errors. Rather, our work measures the probability of correctness and the probability of detecting error in our computational model, without any computational overhead.

## 1.3 Error model for computation

The thrust of this thesis is to quantify the effect of errors on various algorithms. That is, we must formally specify a model to capture how, when and where errors may occur in order to determine what effect their occurrence has on the correctness of the algorithm's output.

We begin by stating where errors could be introduced into the algorithms:

- The input may be changed before any computation occurs.
- The output of any non-trivial calculation may be incorrect. For example, multiplication of two integers, neither of which is 0 or 1, is non-trivial. Division by two or a decrement of 1 is trivial.
- At any point in the algorithm's execution, a register containing any value (either a computed value or a constant) may be altered.
- The output of the algorithm may be changed after all computation occurs.

In contrast, the following examples illustrate those operations that we will assume to be correct:

- Computing subtraction by 1 (i.e., $p-1$) is error free.
- Given the previous point, computing $\frac{p-1}{2}$ is error free.
- Computing $x \pmod{p}$ will output an element $y$ such that $x = kp+y$ for some integer $k$, where $0 \leq y \leq p-1$.

It is worth noting that since we will assume that modular reduction does produce a "reasonable" result, we will view an error as some event that causes a (correct) value $x$ to be replaced by another value $y \pmod{N}$ with uniform probability. In other words, if an error occurs, it will change the value $x$ to particular value $y$ in the range $0, \ldots, N-1$ with probability $1/N$. Modelling errors using only incorrect modular computation as the form of error can be justified for a few reasons. First, this error model can capture a wide variety of errors. For example, an "off-by-one" error is equivalent to an error in modular computation. Second, this error model mimics the effect of the Intel division bug (as outlined earlier in this chapter), in that modular reduction can be viewed as repeated division. Third, this model provides mathematically interesting results concerning the underlying structure of the algorithms we analyze in this thesis.

Next, we outline our model of fault-detection. Consider an algorithm $A$ that implements function $f$, such that $f$ has range $O$ and algorithm $A$ has range $O'$ where $O \subseteq O'$. That is, the algorithm may produce results which are not in the possible range of the function. Specifically, given an algorithm $A$ and a function $f$, the output of $A$ can be classified as belonging to one of three categories:

1. Correct and feasible – the algorithm computes the correct result,
2. Incorrect and feasible – the algorithm computes an incorrect result and this output is in $O$,
3. Incorrect and infeasible – the algorithm computes an incorrect result and output is in $O' \backslash O$.

It is also worth noting that for the algorithms that we will be examining, an input from a large domain is mapped to an output value in a small range. For example, quadratic residue computation takes as input a value from 0 to $p-1$ (for some large prime $p$), and computes a value in the set $\{-1, 0, 1\}$. Thus, since we

have a large input space and a small output space, case (3) above is a possibility, and as such, we will need to worry about detecting such errors.

Additionally, some algorithms may not produce infeasible results. For example, an algorithm which adds two integers $x$ and $y$ modulo $N$ can produce any element in $\mathbb{Z}/(N)$. Thus, there are no infeasible results. However, we can modify this algorithm (and extend this to any algorithm of this form) by grouping two executions of the algorithm as an ordered pair. For example, we can modify the addition algorithm outlined above and transform it into an algorithm that produces an ordered pair, by taking the output of one execution of the algorithm as the first element in the ordered pair, and the second execution of the algorithm as the second element. Therefore, this new composite algorithm would produce:

- correct and feasible output if the output was the pair $(a, a)$ where $a = x + y \pmod{N}$,
- incorrect and feasible output if the output was the pair $(b, b)$ where $b \neq x + y \pmod{N}$
- incorrect and infeasible output if the output was the pair $(c, d)$ where $c \neq d$ and $c, d \in \mathbb{Z}/(N)$.

This repeated execution technique easily generalizes to any algorithm that produces only feasible outputs.

We now define the idea of a *robust* algorithm. It is desirable to avoid having output which is incorrect but feasible, since there is often no simple way to determine if the output is in fact incorrect. Thus, we wish our algorithms to produce either correct output or output which we can identify as infeasible, and thus, incorrect. Formalizing this idea, we consider an algorithm to be computing a map from the input space $I$ to the output space $O'$, $O \subseteq O'$, where $O$ is the space of feasible outputs and $O'$ is the set of all outputs. We define a *robust* algorithm as an algorithm that, on an input of size $n$, produces output that is

- correct with probability $p_c(n)$;
- incorrect and infeasible with probability $p_i(n)$;
- incorrect and feasible with probability $p_f(n)$;

where $p_c(n) > b > 0$ (for some constant $b$) and $p_f(n) < p_i(n)$. The rationale for this definition is that if an algorithm satisfies this definition, we can execute

the algorithm repeatedly to increase the confidence of the output. By way of explanation, the least desirable category of output is incorrect and feasible, since the output is not correct yet cannot be detected as incorrect, whereas incorrect and infeasible can be determined to be incorrect since it is infeasible. Thus, if an algorithm is repeatedly executed, the likelihood of having either correct or infeasible output will increase, and thus, we lessen the likelihood of feasible but incorrect output.

In some situations, we may use an equivalent definition of robustness, where we replace $p_f(n) < p_i(n)$ with the inequality $\frac{p_i(n)}{p_f(n)+p_i(n)} = \frac{p_i(n)}{1-p_c(n)} > 0.5$. By some simple rearrangement, in order to show robustness, we need to show that

$$p_f(n) < p_i(n),$$

which holds if and only if

$$\frac{p_f(n)}{p_i(n)} < 1,$$

which is equivalent to

$$\frac{p_i(n) + p_f(n)}{p_i(n)} < 2,$$

and by reciprocating, we have

$$\frac{p_i(n)}{p_i(n) + p_f(n)} > 0.5.$$

Should an algorithm not be robust for certain inputs, it may be possible to convert it to a robust algorithm by running two independent executions of the input on particular input. The following lemma shows that combining two independent trials of a robust algorithm maintains robustness.

**Lemma 1.3.1** *Suppose algorithm $A$ is robust. Then algorithm $A^*$, which runs algorithm $A$ on two independent executions on the same input, is also robust.*

**Proof:** Since algorithm $A$ is robust, we know that $p_c(n) > b > 0$ for some constant $b$ and $p_i(n) > p_f(n)$, where $n$ is the size of the input given to algorithm $A$. Let $p_c^*(n)$, $p_f^*(n)$ and $p_i^*(n)$ be the probabilities for correct output, feasible but incorrect output and infeasible output (respectively) for algorithm $A^*$. We need to show that $p_c^*(n) > b^* > 0$ for some constant $b^*$ and $p_i^*(n) > p_f^*(n)$ for any input size $n$.

Since the output of algorithm $A^*$ is correct if and only if both independent executions of algorithm $A$ are correct, the probability that algorithm $A^*$ is correct

is $p_c^*(n) = (p_c(n))^2$. Since $p_c(n) > b > 0$, it follows that $p_c^*(n) = (p_c(n))^2 > b^2 > 0$. Thus the constant $b^* = b^2$ satisfies this part of the definition of robustness.

In order to show that $p_i^*(n) > p_f^*(n)$, we consider what the probabilities are.

To begin to show this inequality, we consider what an infeasible output from algorithm $A^*$ is. Certainly, if either of the outputs from either independent execution is infeasible, the output is infeasible. In particular, if the "first" execution has infeasible output, it does not matter what the "second" execution gives. Additionally, if the first output is not infeasible, but the second execution is, this would be infeasible output from algorithm $A^*$.

To formulate this mathematically, $p_i^*(n) \geq p_i(n) + p_i(n)(1 - p_i(n))$. It is worth noting that in practice, this inequality will almost always be strict, since there are cases, such as one output being correct and the other being feasible but incorrect, which would increase the value $p_i^*(n)$.

Additionally, in order to have feasible but incorrect output from algorithm $A^*$, we would need both feasible but incorrect outputs to match. Formulating this, we have $p_f^*(n) \leq p_f(n)^2$. It is worth noting that this inequality becomes strict if there is more than one possible feasible but incorrect output (since if there are different feasible outputs from the two independent executions, the output of algorithm $A^*$ would be deemed infeasible).

Notice that since $p_i(n) < 1$, it follows that $p_i(n)^2 < p_i(n)$, and combining this with our inequalities for $p_i^*(n)$ and $p_f^*(n)$, we have

$$p_i^*(n) \geq p_i(n) + p_i(n)(1 - p_i(n)) > p_i(n)^2 > p_f(n)^2 \geq p_f^*(n)$$

which proves that algorithm $A^*$ is robust. ■

## 1.4 A motivating example

We illustrate how errors affecting a particular algorithm can produce each of the three categories of output outlined in Section 2.6. Suppose we want to determine the validity of a credit card. Moreover, we are sending the credit card information one digit at a time (say, over the phone to a salesperson) and a digit may be written down incorrectly. We will suppose (for the time being) that the last digit is the check digit. More specifically, the credit card number is an $n$-digit sequence, $d_1, d_2, ..., d_n$, where each digit is in the range 0 to 9, and we have the following

equation that must be satisfied:

$$d_n = \left( \sum_{i=1}^{n-1} d_i \right) \bmod 10, \tag{1.1}$$

that ensures that the check digit is dependent on the other $n-1$ digits. We shall call Equation (1.1) the *check digit equation*.

For our error model, we will assume that with probability $\epsilon$, each digit $d_j$ ($1 \leq j \leq n$) is changed (with uniform distribution) to a *different* digit chosen from $\{0, 1, ..., 9\} - \{d_j\}$. For example, in the transmission of the credit card digits from point $A$ to point $B$, digits can be altered with probability $\epsilon$.

Notice that this model differs from the model that is used throughout the remaining chapters, in that an error in this example must change the value of of $d_j$ to be something other than itself. We alter the error model for this example only simply in order to make the analysis cleaner.

It is worth noting that for the subsequent chapters in this thesis, we are assuming the input size is large enough such that either model (forcing errors to alter values, or not requiring alteration) will yield the same results, since the effect of the additional "non-modifying error case" is negligible. For example, the difference between $1/2^{2^{1000}}$ and $1/(2^{2^{1000}} + 1)$ is insignificant for the purposes of the results obtained in this thesis.

- *Correct*

  In order to ensure that that the error is correct, no error can occur. Thus, since the probability that no error occurs in a single digit is $(1 - \epsilon)$, the probability that the output is correct is

  $$p_c(n) = (1 - \epsilon)^n.$$

- *Incorrect and feasible*

  By incorrect and feasible, we mean that an error or errors occurred that modified the original digit sequence (thus the transmitted credit card number is not the same as the original credit card number); however, the transmitted card is feasible, in the sense that the check digit equation is satisfied. More formally, after transmission, we have a new credit card number sequence $d'_1, d'_2, ..., d'_n$ such that

  $$d'_n = \left( \sum_{i=1}^{n-1} d'_i \right) \bmod 10$$

where $d'_j \neq d_j$ for at least one $j$, $1 \leq j \leq n$.

Notice that if one exactly error occurs in $d_0, d_1, \ldots, d_n$, there is no possibility of feasibility. Specifically, if one value of $d_j$ is modified, then only one side of Equation (1.1) will be altered, and thus, the equation cannot be valid.

If two errors occur, these errors must, in effect, "cancel out." First, observe that if $d_j$ is altered to $d'_j$, then $d'_j = (d_j + a) \bmod 10$. If the other error affects the check digit $d_n$, then $d'_n = (d_n + a) \bmod 10$ in order for Equation (1.1) to be satisfied. If the second error affects a digit other than the check digit, in order to cancel, we must have $d'_k = (d_k - a) \bmod 10$ in order to satisfy Equation (1.1).

Notice that out of the $n$ digits, there are $\binom{n}{2}$ ways of "choosing" 2 digits to modify. The chance that exactly two errors occur is $\epsilon^2(1-\epsilon)^{n-2}$. Finally, given one error, there is precisely one other error that causes cancellation: formally, there is a $\frac{1}{9}$ probability that the second error cancels out the first error. In total, if two errors occur, the probability that the output is incorrect and feasible is

$$\binom{n}{2}(1-\epsilon)^{n-2}\epsilon^2\frac{1}{9}.$$

We now extend this case to analyze the case when $k$ errors occur. An argument analogous to the previous paragraph yields $\binom{n}{k}$ ways of picking $k$ digits to be altered, and $(1-\epsilon)^{n-k}\epsilon^k$ as the probability that exactly $k$ errors occur. To determine the probability that $k$ errors can "cancel" out, we consider the probability that $k-1$ errors do not cancel out. That is, iff the $k-1$ errors do not cancel out, we can use the $k$th digit to compensate for the remainder. Letting $f(k)$ be the probability that $k$ digits cancel out, we have the following recurrence relation:

$$\begin{aligned} f(1) &= 0 \\ f(k) &= \frac{1 - f(k-1)}{9}, \ (k > 1). \end{aligned}$$

This recurrence relation can be solved to yield

$$f(n) = \frac{9^{n-1} - (-1)^{n-1}}{10 \cdot 9^{n-1}}, \ (n \geq 1).$$

Since we have as few as 1 error and as many as $n$ errors, the probability that the output is incorrect and feasible is:

$$p_f(n) = \sum_{k=1}^{n} \binom{n}{k}(1-\epsilon)^{n-k}\epsilon^k \frac{9^{k-1} - (-1)^{k-1}}{10 \cdot 9^{k-1}}.$$

We can simplify this expression as follows:

$$\begin{aligned}
p_f(n) &= \sum_{k=1}^{n} \binom{n}{k} (1-\epsilon)^{n-k} \epsilon^k \frac{9^{k-1} - (-1)^{k-1}}{10 \cdot 9^{k-1}} \\
&= \frac{1}{10} \sum_{k=1}^{n} \binom{n}{k} (1-\epsilon)^{n-k} \epsilon^k - \frac{1}{10} \sum_{k=1}^{n} \binom{n}{k} (1-\epsilon)^{n-k} \epsilon^k \left(\frac{-1}{9}\right)^{k-1} \\
&= \frac{1}{10}(1-(1-\epsilon)^n) + \frac{9}{10} \sum_{k=1}^{n} \binom{n}{k} (1-\epsilon)^{n-k} \epsilon^k \left(\frac{-1}{9}\right)^{k} \\
&= \frac{1}{10}(1-(1-\epsilon)^n) + \frac{9}{10} \left( \left(1 - \epsilon - \frac{\epsilon}{9}\right)^n - (1-\epsilon)^n \right) \\
&= \frac{1}{10}(1-(1-\epsilon)^n) + \frac{9}{10} \left( \left(1 - \epsilon - \frac{\epsilon}{9}\right)^n - (1-\epsilon)^n \right) \\
&= \frac{1}{10} - (1-\epsilon)^n + \frac{9}{10} \left(1 - \frac{10\epsilon}{9}\right)^n .
\end{aligned}$$

- *Incorrect and infeasible*

  In this case, we can rely on the analysis performed in the previous case. In particular, there are $\binom{n}{k}$ ways of picking $k$ digits to be altered, and $(1-\epsilon)^{n-k}\epsilon^k$ is the probability that exactly $k$ errors occur. The only remaining task is to determine the probability that $k$ digits do not "cancel" out (that is, Equation (1.1) is violated).

  Let $g(k)$ be the probability that $k$ errors do not cancel out. Using the incorrect and feasible result from above, it is clear that $g(k) = 1 - f(k)$; that is, either errors cancel out or they do not. Thus, we have:

$$\begin{aligned}
g(k) &= 1 - f(k) \\
&= 1 - \frac{9^{k-1} - (-1)^{k-1}}{10 \cdot 9^{k-1}} \\
&= \frac{9^k - (-1)^{k-1}}{10 \cdot 9^{k-1}}.
\end{aligned}$$

  In summary, the probability that the output is incorrect and infeasible is:

$$p_i(n) = \sum_{k=1}^{n} \binom{n}{k} (1-\epsilon)^{n-k} \epsilon^k \frac{9^k - (-1)^k}{10 \cdot 9^{k-1}}.$$

Since $p_c(n) + p_f(n) + p_i(n) = 1$, we can write $p_i(n)$ as

$$\begin{aligned} p_i(n) &= 1 - p_c(n) - p_f(n) \\ &= 1 - (1-\epsilon)^n - \left(\frac{1}{10} - (1-\epsilon)^n + \frac{9}{10}\left(1 - \frac{10}{9}\epsilon\right)^n\right) \\ &= \frac{9}{10} - \frac{9}{10}\left(1 - \frac{10}{9}\epsilon\right)^n. \end{aligned}$$

We now look at some asymptotic results in these cases.

- $\epsilon \gg \frac{1}{n}$

  For simplicity, let us assume that $\epsilon = \frac{1}{c}$ for some constant $c > 1$.

  In the "correct" output case, we have

  $$(1-\epsilon)^n = \left(\frac{c-1}{c}\right)^n,$$

  which becomes vanishingly small as $n$ grows. Thus, we would not rely on the algorithm in this case, since we require $p_c(n) > d$ for a constant $d$ for appropriately large $n$.

  In the "incorrect but feasible case", we have

  $$p_f(n) = \frac{1}{10} - (1-\epsilon)^n + \frac{9}{10}\left(1 - \frac{10\epsilon}{9}\right)^n,$$

  and since both $(1-\epsilon)^n$ and $\frac{9}{10}\left(1 - \frac{10\epsilon}{9}\right)^n$ become vanishingly small as $n$ grows, we have

  $$p_f(n) \doteq \frac{1}{10}.$$

  Similarly, in the "incorrect and infeasible case", we have

  $$\begin{aligned} p_i(n) &= \frac{9}{10} - \frac{9}{10}\left(1 - \frac{10}{9}\epsilon\right)^n \\ &\doteq \frac{9}{10}. \end{aligned}$$

  Thus, if the probability of error is $\frac{1}{c}$ our algorithm does detect errors (when they occur) 90% of the time.

- $\epsilon \ll \frac{1}{n}$

  In the "correct" output case, using the binomial theorem, we can approximate $(1-\epsilon)^n$ as $1 - n\epsilon + \frac{n(n-1)}{2}\epsilon^2$.

  Therefore, there is a constant lower-bound on $p_c(n)$.

  In the "incorrect but feasible case", we have

$$\begin{aligned} p_f(n) &= \frac{1}{10} - (1-\epsilon)^n + \frac{9}{10}\left(1 - \frac{10\epsilon}{9}\right)^n \\ &\approx \frac{1}{10} - (1 - n\epsilon + \frac{n(n-1)}{2}\epsilon^2) + \frac{9}{10}\left(1 - \frac{10}{9}n\epsilon + \frac{n(n-1)}{2}\left(\frac{10}{9}\epsilon\right)^2\right) \\ &= \frac{n(n-1)\epsilon^2}{18}. \end{aligned}$$

  In the "incorrect and infeasible case", we have

$$\begin{aligned} p_i(n) &= \frac{9}{10} - \frac{9}{10}\left(1 - \frac{10}{9}\epsilon\right)^n \\ &\approx \frac{9}{10} - \frac{9}{10}\left(1 - \frac{10}{9}n\epsilon + \frac{n(n-1)}{2}\left(\frac{10}{9}\epsilon\right)^2\right) \\ &= n\epsilon - \frac{10n(n-1)}{18}\epsilon^2. \end{aligned}$$

  To demonstrate robustness, we must show that $p_f(n) < p_i(n)$. We can show this as follows:

$$\begin{aligned} p_f(n) < p_i(n) &\iff \frac{n(n-1)\epsilon^2}{18} < n\epsilon - \frac{10n(n-1)}{18}\epsilon^2 \\ &\iff \frac{11n(n-1)}{18}\epsilon^2 < n\epsilon \\ &\iff \frac{11(n-1)}{18}\epsilon < 1 \\ &\iff \epsilon < \frac{18}{11(n-1)}. \end{aligned}$$

  Thus, the algorithm would be robust so long as $\epsilon < \frac{18}{11(n-1)}$.

- *Limit as* $n \to \infty$ *if* $\epsilon = \frac{1}{n}$

In the "correct" output case, we have

$$p_c(n) = \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \doteq 0.3679.$$

In the "incorrect but feasible" case, we have

$$\begin{aligned} p_f(n) &= \lim_{n\to\infty} \left(\frac{1}{10} - (1-\epsilon)^n + \frac{9}{10}\left(1 - \frac{10\epsilon}{9}\right)^n\right) \\ &= \frac{1}{10} - \frac{1}{e} + \frac{9}{10}e^{-\frac{10}{9}} \\ &\doteq 0.0284. \end{aligned}$$

In the "incorrect and infeasible case", we have

$$\begin{aligned} p_i(n) &= \lim_{n\to\infty} \left(\frac{9}{10} - \frac{9}{10}\left(1 - \frac{10}{9}\epsilon\right)^n\right) \\ &= \frac{9}{10} - \frac{9}{10}e^{-\frac{10}{9}} \\ &\doteq 0.6037. \end{aligned}$$

Therefore, the errors would be detected approximately $\frac{0.6037}{0.6321} = 95.5\%$ of the time, making the algorithm very robust in this case.

To summarize the previous analysis, incorrect output that is feasible can occur with a non-trivial probability if $\epsilon \gg \frac{1}{n}$. However, in practice, $\epsilon \ll \frac{1}{n}$, and thus this algorithm would be robust.

Finally, we remark that this simple checksum analysis can be applied to other variants of checksum. For example, the *Luhn algorithm* [64] for credit card checksums works as follows, given a card number `C`:

```
sum ← 0
size ← length(C)
parity ← size mod 2
for i from 0 to size - 1 {
  digit ← C[i]
  if (i mod 2 = parity)
    digit ← 2 * digit
  if (digit > 9)
    digit ← digit - 9
```

```
  sum ← sum + digit
}
return (sum mod 10) = 0
```

There are two crucial items needed in order to observe the isomorphism between Luhn's algorithm and the simple checksum analyzed above. First, notice that the card is valid iff the checksum is divisible by 10 in both algorithms. Second, notice that there is a 1-1 correspondence between the digits of the simple checksum and the digits in Luhn's algorithm. Specifically, the table below makes this correspondence explicit.

| digit (simple) | value of "odd" position (Luhn's) | value of "even" position (Luhn's) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 2 |
| 2 | 2 | 4 |
| 3 | 3 | 6 |
| 4 | 4 | 8 |
| 5 | 5 | 1 |
| 6 | 6 | 3 |
| 7 | 7 | 5 |
| 8 | 8 | 7 |
| 9 | 9 | 9 |

Due to this mapping, the probability space is precisely the same under both of these algorithms, and thus, our earlier analysis applies to Luhn's algorithm.

## 1.5 Outline of the remaining chapters

In the remaining chapters of this thesis, we use the definition of robustness from Section 1.3 to quantify the robustness of algorithms for computing primality (i.e., the Lucas-Lehmer and Pepin tests), group order and quadratic residues. Moreover, we show that typically, there will be an "error threshold" above which the algorithm is unreliable (that is, it will rarely give the correct result).

The main results of this thesis can be classified into three categories. The first category of results are those related to algorithmic structure, by which we mean algebraic classification of how particular algorithms operate on their input.

The main results describe the underlying structure of Pepin's test for primality, as proved in Corollary 4.4.3; the structure underlying the Lucas-Lehmer primality test, represented in two different formulations in Corollary 5.4.6 and Theorem 6.2.1; and, the algebraic structure of computing the order of a group element as a Markov chain in Lemma 7.1.1.

The second category of results proven in this thesis concern statistical measures related to the underlying structures of algorithms. Specifically, we look at the statistical results concerning tail lengths and cycle lengths of the digraph based on the Pepin test in Theorem 4.7.1 and digraph based on the Lucas-Lehmer test in Corollary 5.7.1. We also provide interesting asymptotic measures for Pepin's test in Theorems 4.7.4 and 4.7.5, and analogous theorems related to the Lucas-Lehmer test in Theorems 5.7.2 and 5.7.3

The final category of results of this thesis relate to quantifying robustness of algorithms. Specifically, we determine conditions that ensure algorithms are robust for computing quadratic character (Theorems 3.6.7 and 3.7.3), Pepin's test for primality (Theorem 4.4.7 and 4.5.12), Lucas-Lehmer test for primality (Corollary 5.6.5) and computing group order of an element (Theorem 7.1.6).

We now outline the remainder of this thesis. In Chapter 2 we present some notation and definitions used throughout the remaining chapters. In Chapter 3, we analyze the robustness of two algorithms for computing the quadratic character of an integer modulo $p$ ($p$ prime), showing that in order to make the algorithms robust, two independent executions of the algorithm are required. In Chapter 4, we analyze the robustness of Pepin's primality test for numbers of the form $2^{2^k}+1$ ($k \geq 1$), showing robustness results for this algorithm for prime input, as well as heuristic and explicit results for composite input. In showing these robustness results for Pepin's test, we also present some statistical measures concerning the digraph formed by the iteration $x \to x^2$, and these statistical measures give insight into the underlying algorithmic process of Pepin's test. In Chapter 5, we apply techniques similar to those used in Chapter 4 on the Lucas-Lehmer primality test for numbers of the form $2^p - 1$ ($p$ prime). Again, we prove robustness results for both the prime and composite input cases, and we also give statistical measures concerning the digraph formed by the iteration $x \to x^2-2$. In Chapter 6, we present an alternative proof of one of the main theorems of Chapter 5 (Corollary 5.4.6, which deals with the structure of the digraph formed by the iteration $x \to x^2 - 2$), using Dickson polynomials and Lucas functions. We move away from primality tests in Chapter 7, and extend the analysis techniques used in the credit card verification example from Section 1.4 to an algorithm that computes the order of an element in a group.

# Chapter 2

# Definitions and Notation

## 2.1 Introduction

In this chapter, we briefly define mathematical structures and notation that will be used throughout the rest of this thesis.

## 2.2 Galois fields and multiplicative groups

We denote the Galois field of $q$ elements as $\mathrm{GF}(q)$. When $q = p$, a prime, we denote the elements of $\mathrm{GF}(p)$ as elements of $\mathbb{Z}/(p)$. We let $\mathrm{GF}(p)[x]$ denote the ring of polynomials (in $x$) with coefficients from $GF(p)$.

If $H$ is a multiplicative group and $h \in H$, we define the *order of $h$ (in $H$)* as the least integer $i$ ($i \geq 1$) such that $h^i = 1$. We usually denote the order of $h$ in $H$ as $\mathrm{ord}_H\ h$. If $H = (\mathbb{Z}/(N))^*$, we write $\mathrm{ord}_N\ h$.

We let $\varphi$ denote the Euler-phi function. One identity we will make use of frequently is $\sum_{d\,|\,n} \varphi(d) = n$.

Let $\nu_b(d)$ denote the largest power of $b$ that divides $d$ (that is, $\nu_b(d) = w$ if $d = b^w l$ for some $l$ where $b \not| \ l$).

Let $\pi(x; l, k)$ denote the number of primes $\leq x$ that are congruent to $k \pmod{l}$.

## 2.3 Iterated functions

Define a function $f : D \to R$ for sets $D$ and $R$ ($R \subseteq D$). We can define the *iteration of $f$* as:

$$f^i(x) = \begin{cases} x, & \text{if } i = 0; \\ f(f^{i-1}(x)), & \text{otherwise.} \end{cases}$$

## 2.4 Directed graphs

In this thesis, we define a particular *directed graph* (also called a *digraph*) $G_{N,a} = (\mathcal{V}, \mathcal{E})$ where the vertices are given by

$$\mathcal{V} = \{x \ : \ x \in (\mathbb{Z}/(N))^*\}$$

and the directed edges are given by

$$\mathcal{E} = \{(x, (x^2 - a) \bmod N) \ : \ x \in \mathcal{V})\}.$$

It should be noted that the notation $(v, w)$ indicates the edge is directed *from $v$ to $w$*.

We also consider the digraph $G^R_{N,a}$ to be the digraph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \mathbb{Z}/(N)$ and $\mathcal{E} = \{(x^2 - a, x) \ : \ x \in \mathbb{Z}/(N)\}$. In other words, $G^R_{N,a}$ is the graph $G_{N,a}$ with the edge directions reversed.

We will be dealing with certain directed graphs. A *complete binary tree of height $h$*, denoted $B_h$, is a directed graph with $2^i$ nodes at depth $i$, for $0 \leq i \leq h$, with the property that every non-leaf node has exactly two children. The graph $B_h$ contains $2^{h+1} - 1$ nodes in total.

In order to make explicit statements about the robustness of number theoretic algorithms, we will need to discuss the "shape" of these digraphs $G_{N,a}$. Since the vertex set is fixed, the defining factor in the "shape" of the graph is the parameter $a$ in the map $x \to x^2 - a \pmod N$. As an example, these types of digraphs have been considered by Harris [44], Kravitz [53] and recently by Teske & Williams [92] in the context of Cunningham chains of primes.

If we pick a particular $x \in \mathcal{V}$, and let $f$ be any function we can consider the *orbit* of $x$ (the directed path in $G_{N,a}$ beginning at $x$) under $f$. Since $\mathcal{V}$ is finite, there must exist a least positive integer $s = s(x)$ such that $f^s(x) \in \{f^0(x), f^1(x), ..., f^{s-1}(x)\}$. Let $t = t(x)$ be the least such non-negative integer such that $f^t(x) = f^s(x)$. We

call $t$ the *tail length* and the elements $x, f(x), f^2(x), ..., f^t(x)$ the *tail*. If we set $c = c(x) = s(x) - t(x)$, we have $f^t(x) = f^{t+c}(x)$. We call $c$ the *cycle length* and the elements $f^{t+1}(x), ..., f^{t+c-1}(x)$ the *cycle*. For a visual definition, see Figure 2.1.

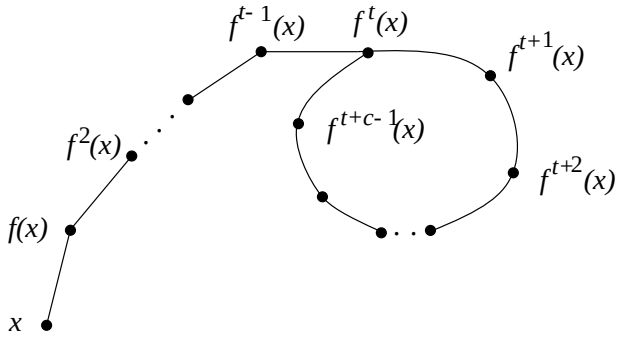


Figure 2.1: The tail and cycle length.

We will also use some statistical measures to describe the digraphs $G_{N,a}$. In particular, we define:

- $TC(N, a) :=$ total number of cycles;
- $T_0(N, a) :=$ total number of elements in all cycles, i.e., the number of $g \in (\mathbb{Z}/(N))^*$ with $t(g) = 0$;
- $AC(N, a) :=$ average length of a cycle;
- $C(N, a) :=$ average value of $c(g)$ over all $g \in (\mathbb{Z}/(N))^*$;
- $T(N, a) :=$ average value of $t(g)$ over all $g \in (\mathbb{Z}/(N))^*$;
- $ST_0(M) := \sum_{2<p\leq M, p \text{ prime}} T_0(p, a)$;
- $ST(M) := \sum_{2<p\leq M, p \text{ prime}} \sum_{1\leq x<p} t(x)$.

These measures were studied by Pollard [77] in his $\rho$-factoring method. Pollard's $\rho$-method, which is based on iterating a random quadratic map and using a cycle-finding algorithm attributed to Floyd (outlined in Knuth [51]), provides asymptotic results on the quantity $AC$, specifically. Pollard's method was improved upon by Brent [18] using a variation on the cycle finding algorithm. Thus, as one example, determining bounds on these asymptotic measurements is useful for classifying and comparing running times of various factoring methods.

## 2.5 Congruence solutions

In this section, we define what is meant by a solution to a congruence and what is meant by the number of solutions to individual or simultaneous congruences.

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ where $n \geq 1$ and not all $a_i, 0 \leq i \leq n$ are zero. Let $N$ be a positive integer. Consider the congruence

$$f(x) \equiv 0 \pmod{N}.$$

We call $x^*$ a *solution to the congruence* if $f(x^*) = mN$ where $0 \leq x^* \leq N$ and $m \in \mathbb{Z}$. Certainly, if there is such an $x^*$ then for any integer $t$, $x^* + tN$ satisfies the congruence: however, we view these other satisfying integers as being in the equivalence class of $x^*$ and not as distinct solutions.

The *number of solutions to the congruence* is $n$ if there are distinct solutions $x_i^*$ (as described above) for $1 \leq i \leq n$.

A *system of simultaneous congruences* is a collection of congruences of the form:

$$\begin{aligned} f_1(x) &\equiv 0 \pmod{N_1} \\ f_2(x) &\equiv 0 \pmod{N_2} \\ &\vdots \\ f_k(x) &\equiv 0 \pmod{N_k}, \end{aligned}$$

where $N_i$ are distinct positive integers.

We call $x^*$ a *solution to a system of simultaneous congruences* if $f(x^*) = m_i N_i$ for some integer $m_i$ where $1 \leq i \leq k$ and $0 \leq x^* < \Pi_{j=1}^{k} N_j$. We can similarly define the number of solutions to a simultaneous congruence as the number of distinct $x^*$ that are solutions to the system of simultaneous congruence.

## 2.6 A computational model

We now formalize our computational model.

Our model is based on the *Random Access Machine* model which is outlined in Papadimitriou [71, Section 2.6], as well as many other computational analysis texts. Specifically, we have the following descriptive points:

- The main data structure is a collection of *registers*, where each register can store an arbitrarily large positive or negative integer. We label registers with $x, y, \ldots$ .

- Algorithms are finite sequence of instructions executed sequentially, where each instruction is either an assignment, comparison, iteration or conditional instruction.
- We may *assign* a value to a register by way of the $\leftarrow$ operator. For example, the instruction $x \leftarrow y$, semantically gives register $x$ the value contained in register $y$, without modifying the contents of $y$ (or any other register).
- We may *compare* a register to another register or value by way of the $=$ operator. For example, the instruction $x = 3$ would evaluate to true iff the value 3 is stored in register $x$.
- We may *iterate* a sequence of instructions by way of the following construct:

  ```
  for R from b to v do
    X
  end for
  ```

  which will execute the instructions called `X` a total of $v - b + 1$ times, where on the $i$th iteration, $X$ is executed with $R$ having value $b + i - 1$.
- We may also *conditionally* evaluate instructions using the following construct:

  ```
  if C1 then
    X1
  else if C2 then
    X2
  else if ...
    ⋮
  else if CN then
    XN
  else
    Y
  end if
  ```

  which will execute instructions `Xi` iff conditions `C1, ..., C(i-1)` evaluate to false and condition `Ci` evaluates to true. The instructions labeled `Y` are executed if and only if all conditions `Ci` evaluate to false.
- We will also use mathematical operations such as arithmetical operators $(+, -, *, /)$, exponentiation $(x^y)$ and modular reduction ($x$ mod $y$) when we are computing a value or condition.

## 2.7 Asymptotic analysis

We define asymptotic bounds on functions in the standard way: see, for example Lewis and Denenberg [57]. Let $f, g$ be functions from non-negative real numbers to non-negative real numbers. We say $f \in O(g)$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. For lower bounds, we use the notation $f \in \Omega(g)$ to indicate that there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$. We say that $f \in \Theta(g)$ if $f \in O(g)$ and $f \in \Omega(g)$.

## 2.8 Conclusion

Having provided some definitions, we are now ready to apply these on algorithms for quadratic character, primality testing and group order computation in the remaining chapters of this thesis.

# Chapter 3

# Quadratic residue algorithms

## 3.1 Introduction

In this chapter, we determine the robustness of two quadratic residue calculations. Specifically, we outline a naive algorithm that uses linear exponentiation, and a slightly more sophisticated algorithm that uses binary exponentiation. We then analyze these two algorithms using our error model outlined in Section 1.3, concluding that both algorithms are robust under reasonable assumptions about the rate of error.

## 3.2 Basics

In this section, we provide some basic definitions concerning quadratic residues and the computational model that will be used to analyze the robustness of the algorithms used to compute quadratic residues.

To begin, consider an integer $m > 0$ and a non-zero integer $a$ (with $\gcd(a, m) = 1$). Then $a$ is a *quadratic residue* $\bmod m$ iff $x^2 \equiv a \pmod{m}$ has a solution. If $\gcd(a, m) = 1$ and $a$ is not a quadratic residue, then we call $a$ a *quadratic nonresidue*.

If we restrict the modulus $m$ to be some prime $p$, we can determine the quadratic character computationally, using what is known as Euler's criterion.

**Lemma 3.2.1** *Let $p$ be an odd prime and let $\gcd(a, p) = 1$. Then $a$ is a quadratic residue mod $p$ if*

$$a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$$

*and $a$ is a quadratic nonresidue mod $p$ if*

$$a^{\frac{p-1}{2}} \equiv -1 \pmod{p}.$$

**Proof:** See Corollary 5.7.3 in Bach and Shallit [5]. ■

The two algorithms for computing the quadratic character of a number (which will be presented in the next two sections) both use Lemma 3.2.1 as the skeleton for their algorithms.

To aid us notationally, we also define the *Legendre symbol* to be the following:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \pmod{p}.$$

We conclude this section by capturing the cardinality of quadratic residues and nonresidues. This lemma will be useful later when computing various probabilities in the robustness analysis.

**Lemma 3.2.2** *$GF(p)^*$, where $p$ is prime, has $\frac{p-1}{2}$ quadratic residues, and $\frac{p-1}{2}$ quadratic nonresidues.*

## 3.3 Quadratic residue computation: linear algorithm

As outlined in the previous section, Lemma 3.2.1 provides a calculation for computing quadratic residues. Essentially, to compute the quadratic residue of $a$ modulo $p$, we must raise $a$ to the $(p-1)/2$ power and reduce modulo $p$. For this first algorithm, we compute the quadratic residue by multiplying the element $a$ repeatedly to obtain $a^{\frac{p-1}{2}}$ modulo $p$. This algorithm is described formally in Algorithm 3.1.

## 3.4 Quadratic residue computation: binary algorithm

We can improve on the running time of Algorithm 3.1 by using the binary representation on the number, and repeatedly squaring our intermediate results. The method of multiplication is outlined in, for example, Cormen, Leiserson, Rivest and Stein [26]. This algorithm is outlined in Algorithm 3.2.

$T \leftarrow 1$
**for** $i$ from 1 to $\frac{p-1}{2}$ **do**
  $T \leftarrow T * a \pmod{p}$
**end for**
**if** $T = 1$ **then**
  **print** "$a$ is a quadratic residue"
**else if** $T = -1$ **then**
  **print** "$a$ is a quadratic nonresidue"
**else if** $T = 0$ **then**
  **print** "$a \equiv 0 \pmod{p}$"
**else**
  **print** "An error causing incorrect and infeasible output has occurred."
**end if**

Algorithm 3.1: Linear Quadratic Residue Algorithm

Compute the binary representation for $\frac{p-1}{2}$
Denote it by $B = b_k b_{k-1} \cdots b_1 b_0$
$T \leftarrow a^{b_0} \pmod{p}$
$P \leftarrow a$
**for** $i$ from 1 to $k$ **do**
  **if** $b_i = 1$ **then**
    $T \leftarrow T * P \pmod{p}$
  **end if**
  $P \leftarrow P * P \pmod{p}$
**end for**
**if** $T = 1$ **then**
  **print** "$a$ is a quadratic residue"
**else if** $T = -1$ **then**
  **print** "$a$ is a quadratic nonresidue"
**else if** $T = 0$ **then**
  **print** "$a \equiv 0 \pmod{p}$"
**else**
  **print** "An error causing incorrect and infeasible output has occurred."
**end if**

Algorithm 3.2: Binary Quadratic Residue Algorithm

## 3.5 Robustness preliminaries

We now measure the accuracy of both the linear and binary quadratic residue algorithms (Algorithms 3.1 and 3.2) when errors occur during computation.

For both algorithms, observe that if we obtain a value of $T$ which is outside of the set $\{-1, 0, 1\}$ the output is infeasible (and incorrect).

We now concern ourselves with determining errors that cause incorrect output that is feasible.

To begin, we consider the case where an error is introduced in the algorithm before the first step. Formally, by "introducing an error", we simply mean that input value of $a$ can become any element in $\mathbb{Z}/(p)$ with uniform probability. (It is worth noting that we use $\mathbb{Z}/(p)$ instead of $GF^*(p)$ in order to make the probability space easier to work with.) Specifically, we let $\epsilon$ represent the probability that an event occurred (i.e., a stored value has been tampered with). We also assume that this error is equally likely to occur before any particular step in the algorithm.

Moreover, since we assume that we reduce modulo $p$ correctly, this error simply replaces $a$ with an element $b \in \mathbb{Z}/(p)$ (where we allow the possibility that $b = a$). For simplicity, we suppose that all of the remaining steps in the algorithms are computed correctly. We now perform analysis on the various cases for $a$ and $b$.

If the input $a$ is a quadratic nonresidue, and $b$ is a quadratic nonresidue, then the output of our algorithm will be correct: the algorithm run on input $a$ without error produces the same output as the algorithm run on input $b$. Similarly, if $a$ was a quadratic residue and $b$ is a quadratic residue, the output will be correct.

If the input $a$ is 0 and $b$ is anything other than 0, the output will be incorrect (i.e., the output will be either 1 or $-1$) but feasible.

As described in Section 1.3, we use the notation $p_c$ to denote the probability the output of the given algorithm is correct; $p_f$ to denote the probability the output of the given algorithm is incorrect but feasible; and $p_i$ to denote the probability the output is incorrect and infeasible. Using this notation, we summarize our observations concerning the effect of an error introduced into the algorithm. We can apply these observations into the following lemma.

**Lemma 3.5.1** *Consider an element $a \in \mathbb{Z}/(p)$ for some prime $p$. If an error is introduced in the input $a \neq 0$ to either the linear or binary quadratic residue algorithms, and the algorithm continues execution without error, we have*

*(a)* $p_c(p) = \frac{1}{2} - \frac{1}{2p}$,

*(b)* $p_f(p) = \frac{1}{2} + \frac{1}{2p}$, *and*

*(c)* $p_i(p) = 0$.

**Proof:** We begin by proving part (c). From our earlier observations, there are no infeasible outputs if the error occurs before the first step, since the input to the algorithm will still be an element from $\mathbb{Z}/(p)$. Since there are no infeasible outputs, $p_i(p) = 0$.

To show part (b), we observed that if the input is altered and the algorithm proceeds without error, the output is guaranteed to be feasible. To have the output be incorrect, we must have the altered input value be 0 or an element which does not have the same quadratic character as the original input. There are a total of $\frac{p+1}{2}$ such values (one value of 0 and $\frac{p-1}{2}$ values of the opposite quadratic character). Since we are assuming that any of the $p$ values from $\mathbb{Z}/(p)$ are equally likely to be the altered value, the probability is $\frac{p+1}{2p}$ which simplifies to the desired result.

Part (a) follows from the fact $p_c(p) + p_i(p) + p_f(p) = 1$. ■

The case when the input is 0 is similar to analyze.

**Lemma 3.5.2** *Consider a prime $p$. If an error is introduced in the input $a = 0$ to either the linear or binary quadratic residue algorithms, and the algorithm continues execution without error, we have*

*(a)* $p_c(p) = 1/p$,

*(b)* $p_f(p) = 1 - 1/p$, *and*

*(c)* $p_i(p) = 0$.

**Proof:** Part (c) follows the same reasoning used in Lemma 3.5.1.

Part (b) results from the earlier observation that if the input is replaced by a random element of $\mathbb{Z}(p)$ and the algorithm proceeds without error, the output is guaranteed to be feasible. To have the output be incorrect, the element must be something other than 0. Since $p - 1$ of the $p$ values in $\mathbb{Z}/(p)$ are not 0, we have $p_f(p) = 1 - 1/p$.

Part (a) follows from $p_c(p) + p_i(p) + p_f(p) = 1$. ■

Next, we compute these probabilities for the case of an error being introduced during the execution of the linear algorithm, Algorithm 3.1.

**Lemma 3.5.3** *Consider a prime $p$, and an element $a \in \mathbb{Z}/(p)$ where $a \not\equiv 0 \pmod{p}$. In the linear quadratic residue algorithm (Algorithm 3.1), if the last error is introduced at the start of iteration $i$ of the loop (where $2 \leq i \leq \frac{p-1}{2}$) such that the value $T$ is replaced with a new value, we have*

*(a)* $p_c(p) = \frac{1}{p}$,

*(b)* $p_f(p) = \frac{2}{p}$, *and*

*(c)* $p_i(p) = 1 - \frac{3}{p}$.

**Proof:** We begin with part (a). If the value of $T$ is replaced at the start of iteration $i$ with the value $r$, the algorithm will proceed to completion and terminate with the value

$$T = r \cdot a^{\frac{p-1}{2}-i}.$$

The output will be correct iff

$$r \cdot a^{\frac{p-1}{2}-i} = \left(\frac{a}{p}\right) \pmod{p}.$$

This equation has a unique solution for $r$, which is

$$r = a^i \pmod{p}.$$

Therefore, the probability that the error introduced resulted in correct output is $\frac{1}{p}$.

For part (b), we use the same reasoning as used in proving part (a) to conclude that the output will be feasible but incorrect iff

$$r \cdot a^{\frac{p-1}{2}-i} = -\left(\frac{a}{p}\right) \pmod{p}$$

or

$$r \cdot a^{\frac{p-1}{2}-i} = 0 \pmod{p}.$$

These equations each yield one solution for $r$, which is

$$r = -a^i \pmod{p}$$

and

$$r = 0$$

respectively. Therefore, $p_f(p) = \frac{2}{p}$.

Part (c) follows immediately. ∎

The same results apply for the binary quadratic residue algorithm, Algorithm 3.2, whose proof follows a similar structure to Lemma 3.5.3.

**Lemma 3.5.4** *Consider a prime $p$, and an element $a \in \mathbb{Z}/(p)$ such that $a \not\equiv 0 \pmod{p}$. If the last error is introduced at iteration $i$ of the loop (where $2 \leq i \leq k$) in the binary quadratic residue algorithm (Algorithm 3.2) such that the value $T$ is replaced with a new value, we have*

*(a)* $p_c(p) = \frac{1}{p}$,

*(b)* $p_f(p) = \frac{2}{p}$, *and*

*(c)* $p_i(p) = 1 - \frac{3}{p}$.

**Proof:** We begin by proving part (a). If the value of $T$ is replaced at the start of iteration $i$ with the value $r$, the algorithm will proceed to completion and terminate with the value:

$$T = r \cdot a^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k},$$

where $b_i$ is the $i$th bit of the binary representation of $\frac{p-1}{2}$. The output will be correct iff

$$r \cdot a^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k} = \left(\frac{a}{p}\right) \pmod{p}.$$

This equation has a unique solution (for $r$), which is

$$r = \left(\frac{a}{p}\right) (a^{-1})^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k} \pmod{p}.$$

Therefore, the probability that the error introduced resulted in correct output is $\frac{1}{p}$.

To show part (b), we note that the output will be feasible but incorrect iff

$$r \cdot a^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k} = -\left(\frac{a}{p}\right) \pmod{p}$$

or

$$r \cdot a^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k} = 0 \pmod{p}.$$

These two equations each have one solution (for $r$), which is

$$r = -\left(\frac{a}{p}\right) (a^{-1})^{2^{i+1} \cdot b_{i+1} + \cdots + 2^k \cdot b_k} \pmod{p}$$

or $r = 0$ (respectively). Therefore, $p_f(p) = \frac{2}{p}$.

Part (c) follows immediately. ∎

We now consider the case where the value $a$ is a multiple of $p$. In particular, we begin by considering the linear quadratic residue algorithm.

**Lemma 3.5.5** *Consider a prime $p$ and an element $a \in \mathbb{Z}/(p)$ with $a \equiv 0 \pmod{p}$. If the last error is introduced during the execution of the linear quadratic residue algorithm before the last multiplication ($a * T$), we have*

*(a)* $p_c(p) = 1$, *and*

*(b)* $p_f(p) = p_i(p) = 0$.

**Proof:** In the linear quadratic residue algorithm described in Algorithm 3.1, the last multiplication will be $a * T$ modulo $p$, where $T$ may be any element in $\mathbb{Z}/(p)$ (due to possible errors introduced in earlier steps). However, since $a \equiv 0 \pmod{p}$, this final multiplication produces the correct output. Thus $p_c(p) = 1$. It follows that $p_f(p) = p_i(p) = 0$. ■

For the binary quadratic residue algorithm, we get the same result as in Lemma 3.5.5, using the consistency of the register $P$ to yield the result.

**Lemma 3.5.6** *Consider a prime $p$ and an element $a \in \mathbb{Z}(p)$ such that $a \equiv 0 \pmod{p}$. If an error is introduced during the execution of the binary quadratic residue algorithm (Algorithm 3.2) before the last $T \leftarrow T * P \pmod{p}$ operation we have*

*(a)* $p_c(p) = 1$, *and*

*(b)* $p_f(p) = p_i(p) = 0$.

**Proof:** Since $a \equiv 0 \pmod{p}$, if there is an error altering the value of $T$ before the last assignment of $T * P \pmod{p}$, we know that the value of $P$ will be computed correctly. That is,

$$P \equiv a^{b_0+2b_1+\cdots+2^k b_k} \equiv 0 \pmod{p}.$$

Therefore, on computing the last $T$ value will result in $T \equiv 0 \pmod{p}$, which guarantees the output will be correct. Thus $p(c) = 1$ and, therefore, $p_f(p) = p_i(p)$. ■

The only remaining case to consider is where errors are introduced into the value of $P$ during the binary quadratic residue computation. Unlike the previous few lemmas, there is not a concise closed-form for the probabilities that occur when the value $P$ is modified.

**Lemma 3.5.7** *Consider a prime $p$ and an element $a \in \mathbb{Z}/(p)$ such that $a \not\equiv 0 \pmod{p}$. If an error is introduced into the binary quadratic residue algorithm (Algorithm 3.2) which causes $P$ to take on a new value at step $i$, where $1 \leq i \leq k$, then the probability that the output will be feasible but incorrect ($p_f(p)$) is $\frac{c_r}{p}$, where $c_r$ is the number of solutions to*

$$r^{b_{i+1}+\cdots+2^{k-i-1}b_k} = -\left(\frac{a}{p}\right) \cdot (a^{-1})^{b_0+2b_1+\cdots+2^i b_i} \pmod{p}.$$

**Proof:** If the value of $P$ is changed to $r$ at step $i$, we have at step $i$ the values

$$T = a^{b_0+2b_1+\cdots+2^i b_i}$$

and

$$P = r.$$

After the remaining $k - i$ steps of the algorithm, we have

$$T = a^{b_0+2b_1+\cdots+2^i b_i} r^{b_{i+1}+\cdots+2^{k-i-1}b_k}$$

and

$$P = r^{2^{k-i}}.$$

Thus, using an argument similar to the proofs above, the output will be feasible but incorrect iff the output value $T$ is $-\left(\frac{a}{p}\right)$, which holds iff the equation

$$r^{b_{i+1}+\cdots+2^{k-i-1}b_k} = -\left(\frac{a}{p}\right) \cdot (a^{-1})^{b_0+2b_1+\cdots+2^i b_i} \pmod{p}.$$

has a solution, which proves the result. ∎

This result could be improved by finding tight bounds on the number of solutions to

$$r^{b_{i+1}+\cdots+2^{k-i-1}b_k} = -\left(\frac{a}{p}\right) \cdot (a^{-1})^{b_0+2b_1+\cdots+2^i b_i} \pmod{p}.$$

However, work in this direction did not yield any non-trivial bounds.

## 3.6 Robustness results for the linear algorithm

We now turn to formally determining the robustness of Algorithms 3.1 and 3.2, using the Lemmas presented in the previous section. In particular, we will show that both the linear and binary versions of the quadratic residue algorithms are robust, using the error model introduced in Section 1.3 and refined for the quadratic residue computation context in Section 3.5.

We will consider the linear quadratic residue algorithm in this section, and then perform similar analysis for the binary quadratic residue algorithm in the next section.

We begin by computing the probability of incorrect but feasible output for the linear quadratic residue algorithm for $a \neq 0$.

**Theorem 3.6.1** *Consider a prime p. If the probability of an error occurring before any particular step in the linear quadratic residue algorithm (Algorithm 3.1) is $\epsilon$, and the input $a \neq 0$, then the probability that the output of the algorithm will be*

*(a) correct is*

$$p_c(p) = (1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2} - \frac{1}{2p}) + \frac{1}{p}\left(1 - (1-\epsilon)^{\frac{p-3}{2}}\right),$$

*(b) feasible but incorrect is*

$$p_f(p) = \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2} + \frac{1}{2p}) + \frac{2}{p}\left(1 - (1-\epsilon)^{\frac{p-3}{2}}\right),$$

*(c) infeasible and incorrect is*

$$p_i(p) = \left(1 - \frac{3}{p}\right)(1 - (1-\epsilon)^{\frac{p-3}{2}}).$$

**Proof:** We begin by proving part (b), and we do this by breaking up the proof into two components. If the last error occurs before the first step, then there are $\frac{p-3}{2}$ steps which must occur without error. The probability of this is $\epsilon(1-\epsilon)^{\frac{p-3}{2}}$. From Lemmas 3.5.1, the probability that this error will cause feasible but incorrect output is $\frac{1}{2} + \frac{2}{p}$, so we compute the product of these quantities to compute the value of $p_f(p)$ if the last error affects the initial input value.

The second component is when the error occurs somewhere after the first step. In this case, we consider all possible steps as the last error step: that is, if the last error occurs at step $i$ of the algorithm, the remaining $\frac{p-1}{2} - i$ steps must occur without error. We combine this sum with Lemma 3.5.3 multiplicatively, and summing these two cases, we derive

$$\begin{aligned}
p_f(p) &= \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}+\frac{1}{2p}) + \sum_{i=2}^{\frac{p-1}{2}}(1-\epsilon)^{\frac{p-1}{2}-i}\epsilon\frac{2}{p} \\
&= \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}+\frac{1}{2p}) + \frac{2}{p}(1-\epsilon)^{\frac{p-1}{2}}\epsilon\sum_{i=2}^{\frac{p-1}{2}}\left(\frac{1}{1-\epsilon}\right)^i \\
&= \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}+\frac{1}{2p}) + \frac{2}{p}(1-\epsilon)^{\frac{p-1}{2}}\epsilon\frac{(\frac{1}{1-\epsilon})^{(p+1)/2} - (\frac{1}{1-\epsilon})^2}{\frac{1}{1-\epsilon}-1} \\
&= \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}+\frac{1}{2p}) + \frac{2}{p}(1-\epsilon)^{\frac{p-1}{2}}\epsilon\left(\frac{1}{1-\epsilon}\right)^2\left((\frac{1}{1-\epsilon})^{\frac{p-3}{2}}-1\right)\frac{1-\epsilon}{\epsilon} \\
&= \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}+\frac{1}{2p}) + \frac{2}{p}\left(1-(1-\epsilon)^{\frac{p-3}{2}}\right),
\end{aligned}$$

which proves the result.

For part (a), we need to compute the probability of correct output. First observe that if no errors occur, then the output will be correct: the probability of having no errors is $(1-\epsilon)^{\frac{p-1}{2}}$, since each of the $\frac{p-1}{2}$ steps must have no error, and the probability of no error occurring is $(1-\epsilon)$. In addition to this observation, we combine Lemmas 3.5.1 and 3.5.3 using the same reasoning as in our computation of $p_f(p)$ above, along with the same arithmetic reductions used to compute $p_f(p)$. Specifically, we derive

$$\begin{aligned}
p_c(p) &= (1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}-\frac{1}{2p}) + \sum_{i=2}^{\frac{p-1}{2}}(1-\epsilon)^{\frac{p-1}{2}-i}\epsilon\frac{1}{p} \\
&= (1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{1}{2}-\frac{1}{2p}) + \frac{1}{p}\left(1-(1-\epsilon)^{\frac{p-3}{2}}\right),
\end{aligned}$$

which is the desired result.

For part (c), in order for the output to be infeasible, recall that from Lemma 3.5.1, if the last error occurs before the first step of the Algorithm 3.1, the output cannot

be infeasible. Thus, we only need to consider the last error occurring between steps 2 to $\frac{p-1}{2}$, and the probability if the last error occurs in these steps is $1 - \frac{3}{p}$, by Lemma 3.5.3. Taking this sum over all possible steps, and using the simplifications used to compute $p_f(p)$, we derive

$$\begin{aligned} p_i(p) &= \sum_{i=2}^{\frac{p-1}{2}} \epsilon(1-\epsilon)^{\frac{p-1}{2}-i}\left(1-\frac{3}{p}\right) \\ &= \epsilon\left(1-\frac{3}{p}\right)(1-\epsilon)^{\frac{p-1}{2}} \sum_{i=2}^{\frac{p-1}{2}} \left(\frac{1}{1-\epsilon}\right)^i \\ &= \left(1-\frac{3}{p}\right)(1-(1-\epsilon)^{\frac{p-3}{2}}), \end{aligned}$$

which is the desired probability measure. ■

We are now ready to show the main result of this chapter, which is to show that Algorithm 3.1 is robust.

**Theorem 3.6.2** *If the probability of error $\epsilon < \frac{1}{p}$ and $p \geq 11$ is a prime, Algorithm 3.1 is robust on input $a \neq 0$.*

**Proof:** To show robustness from the definitions outlined in Section 1.3 we need to show that $p_c(p) > b$ for some constant $b > 0$ and $\frac{p_f(p)}{p_i(p)} < 1$. To show that $p_c(p) > b > 0$, notice that

$$\begin{aligned} p_c(p) &= (1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}-\frac{1}{2p}\right) + \frac{1}{p}\left(1-(1-\epsilon)^{\frac{p-3}{2}}\right) \\ &= \left(1-\frac{1}{p(1-\epsilon)}\right)(1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}-\frac{1}{2p}\right) + \frac{1}{p}. \end{aligned}$$

To show this inequality, we know that since $p > 11$, we have

$$(p-1)\ln(1-\frac{1}{p}) > 11\ln(1-\frac{1}{12}) > -0.95 > -1.$$

Thus $(1-\frac{1}{p})^{p-1} > \frac{1}{e}$. Combining this inequality with $\epsilon < \frac{1}{p}$, we have

$$(1-\epsilon)^{\frac{p-1}{2}} = ((1-\epsilon)^{p-1})^{\frac{1}{2}} > \left(\frac{1}{e}\right)^{\frac{1}{2}} > \frac{1}{2}.$$

Returning to $p_c(p)$, we have

$$\begin{aligned} p_c(p) &= \left(1-\frac{1}{p}\right)(1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}-\frac{1}{2p}\right)+\frac{1}{p} \\ &> \left(1-\frac{1}{11}\right)\left(\frac{1}{2}\right) \\ &= \frac{5}{11} \\ &> 0 \end{aligned}$$

which proves that $p_c(p) > b > 0$.

It is worth noting that to this point, we have only required $p \geq 3$.

To show $\frac{p_f(p)}{p_i(p)} < 1$, notice this is equivalent to showing that $p_i(p) > p_f(p)$. Equivalently, we must show the following inequality holds:

$$p_i(p) - p_f(p) = \left(1-\frac{5}{p}\right)(1-(1-\epsilon)^{\frac{p-3}{2}}) - \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}+\frac{2}{p}\right) > 0.$$

Rearranging, we must show

$$\left(1-\frac{5}{p}\right)(1-(1-\epsilon)^{\frac{p-3}{2}}) > \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}+\frac{2}{p}\right).$$

Notice that if we assume $p > 11$ and $\epsilon < \frac{1}{3}$, then

$$\begin{aligned} 1-(1-\epsilon)^{\frac{p-3}{2}} &> 1-(1-\epsilon)^4 \\ &= 4\epsilon - 6\epsilon^2 + 4\epsilon^3 - \epsilon^4 \\ &= \epsilon(4-6\epsilon+4\epsilon^2-\epsilon^3) \\ &> \epsilon(4-6\epsilon) \\ &> 2\epsilon \\ &> \frac{11}{6}\epsilon. \end{aligned}$$

Thus,

$$\begin{aligned} \left(1-\frac{5}{p}\right)(1-(1-\epsilon)^{\frac{p-3}{2}}) &> \frac{6}{11}(1-(1-\epsilon)^{\frac{p-3}{2}}) \\ &> \epsilon \\ &> \epsilon(1-\epsilon)^{\frac{p-3}{2}}\left(\frac{1}{2}+\frac{2}{p}\right), \end{aligned}$$

which proves the desired inequality, showing that $p_i(p) > p_f(p)$. Thus, the linear quadratic residue algorithm is robust. ■

To complete the analysis of robustness of the linear quadratic residue algorithm, we need to consider the case when the input to the algorithm is a multiple of the prime $p$. We use similar reasoning as was used in proving the previous theorem. As we will demonstrate, however, Algorithm 3.1 is not robust according to our definition.

To begin, prove a theorem analogous to Theorem 3.6.1.

**Theorem 3.6.3** *Consider a prime $p$. If the probability of an error occurring before any particular step in the linear quadratic residue algorithm (Algorithm 3.1) is $\epsilon$, and input $a \equiv 0 \pmod{p}$, then the probability that the output of the algorithm will be*

*(a) correct is*

$$p_c(p) = (1-\epsilon)^{\frac{p-1}{2}} + \epsilon(1-\epsilon)^{\frac{p-3}{2}}\frac{1}{p} + \sum_{i=2}^{\frac{p-1}{2}} 1 \cdot \epsilon(1-\epsilon)^{\frac{p-1}{2}-i},$$

*(b) feasible but incorrect is*

$$p_f(p) = \epsilon(1-\epsilon)^{\frac{p-3}{2}}(1-\frac{1}{p}),$$

*(c) infeasible and incorrect is*

$$p_i(p) = 0.$$

**Proof:** To show part (a), notice that there are three cases to consider. The first case is where no errors occur, which has probability $(1-\epsilon)^{\frac{p-1}{2}}$. The second case is where an error occurs before the first step, and the remaining $\frac{p-3}{2}$ steps occur without error. Using the result of Lemma 3.5.2, we have probability $\epsilon(1-\epsilon)^{\frac{p-3}{2}}\frac{1}{p}$ in this case. The third case to consider is the last error occurs at some step $i$ $(2 \leq i \leq \frac{p-1}{2})$. For this third case, we use Lemma 3.5.5 to derive the probability of correctness as $\sum_{i=2}^{\frac{p-1}{2}} 1 \cdot \epsilon(1-\epsilon)^{\frac{p-1}{2}-i}$. Summing these three quantities together gives the desired result.

For part (c), both Lemma 3.5.2 and 3.5.5 show that $p_i(p) = 0$. Rephrasing this result, regardless of when an error occurs, there is no possibility of incorrect and infeasible output. Thus $p_i(p) = 0$.

Part (b) follows from the fact that $p_c(p) + p_f(p) + p_i(p) = 1$. ∎

In order to show that Algorithm 3.1 is robust if $a$ is a multiple of $p$, we need to show that $\frac{p_i(p)}{1-p_c(p)} > 0.5$. However, since $p_i(p) = 0$, this inequality cannot hold, and thus, the algorithm is not robust in this case.

Thus, we need to alter the output of algorithm slightly. As we will show, if we combine the results of two independent executions of Algorithm 3.1, the algorithm satisfies the definition of robustness.

To begin, we define $p_c^*(p)$ to be the probability of correct output under this combined execution model. Notice that $p_c^*(p) = p_c(p)^2$, since the independent trials must both give the same correct answer. In order for the output to be incorrect and feasible, the output on both independent trials must match, but must not be the correct quadratic residue for the given input. In other words, if the input $a \neq 0$, then the output of both independent trials must match, in that they both must be 0 or both must be $-\left(\frac{a}{p}\right)$.

We calculate this probability in the following lemma.

**Lemma 3.6.4** *Suppose that integer $a \neq 0$ and that $p$ is prime. If two independent trials of Algorithm 3.1 are run on input $a$, the probability that the output (of the combined trials) is feasible but incorrect is*

$$p_f^*(p) = 2\left(\frac{1-(1-\epsilon)^{\frac{p-1}{2}}}{p}\right)^2.$$

**Proof:** From the proof of Lemmas 3.5.1 and 3.5.3, we can observe that the probability that an output is 0 is exactly the probability that the last error which causes the value stored in $T$ to be 0. Similarly, the output will be $-\left(\frac{a}{p}\right)$ iff the value of $T$ is altered to hold the value $a^{-i} \pmod p$ if the error occurs at step $i$ of Algorithm 3.1. In both of these cases, there is exactly one value that $T$ can acquire in order to produce the desired result. We let $p_0(p)$ denote the probability that the output will be 0 in one execution. We compute this probability as

$$p_0(p) = \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-3}{2}} + \sum_{i=2}^{\frac{p-1}{2}} \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-1}{2}-i},$$

where the first term arises from the error replacing the value $a$ with 0 occurring before the first step, and the second term arises from considering cases where 0

replaces $T$ during the execution of the algorithm. We simplify this probability $p_0(p)$ as

$$\begin{aligned}
p_0(p) &= \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-3}{2}} + \sum_{i=2}^{\frac{p-1}{2}} \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-1}{2}-i} \\
&= \frac{1}{p}\epsilon \sum_{i=1}^{\frac{p-1}{2}} (1-\epsilon)^{\frac{p-1}{2}-i} \\
&= \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-1}{2}} \sum_{i=1}^{\frac{p-1}{2}} (1-\epsilon)^{-i} \\
&= \frac{1}{p}\epsilon(1-\epsilon)^{\frac{p-1}{2}} \frac{1-\epsilon}{\epsilon}\left(\frac{1}{(1-\epsilon)^{\frac{p+1}{2}}} - \frac{1}{1-\epsilon}\right) \\
&= \frac{1}{p}(1-(1-\epsilon)^{\frac{p-1}{2}}).
\end{aligned}$$

Notice that $p_0(p)$ is also the probability that the output is $-\left(\frac{a}{p}\right)$, in one of the independent trials. Since the other independent trial must match the first one, we have to square $p_0(p)$. Moreover, we could have either 0 or $-\left(\frac{a}{p}\right)$ as the possible values, therefore, we must multiply this probability by 2, so that $p_f^*(p) = 2(p_0(p))^2$ which yields the result. ■

In the case that the input is $a = 0$, the feasible but incorrect outputs must both be 1 or both be $-1$. Again, we calculate the probability that we have feasible and incorrect output in this case.

**Lemma 3.6.5** *Suppose that integer $a = 0 \pmod p$ and that $p$ is prime. If two independent trials of Algorithm 3.1 are run on input $a$, the probability that the output (of the combined trials) is feasible but incorrect is*

$$p_f^*(p) = 2\left(\epsilon(1-\epsilon)^{\frac{p-3}{2}}\frac{p-1}{2p}\right)^2.$$

**Proof:** From Lemma 3.5.2 and 3.5.5, the only possible way to have output 1 or $-1$ from the algorithm is for an error to occur during the first step of the algorithm and the remaining $\frac{p-3}{2}$ steps are free from error. The probability that the output is 1 is exactly the same as the probability that the output is $-1$ (there are an equal number of quadratic residues and non-residues). Therefore, the probability that one

independent trial of Algorithm 3.1 results in output 1 on input 0 is $\epsilon(1-\epsilon)^{\frac{p-3}{2}}\frac{p-1}{2p}$, which is the same as the probability that the output is $-1$ on input 0. Thus, since we have two independent trials that must agree, we must square this result. Again, as in the proof of Lemma 3.6.4, we must multiply this quantity by 2 in order to capture the two possible feasible answers. ■

We are now ready to discuss the robustness results in this new light.

**Theorem 3.6.6** *If the probability of error $\epsilon < \frac{1}{2}$ and $p$ is a prime, an algorithm which is the combination of two independent trials of Algorithm 3.1 on input $a = 0$ is robust.*

**Proof:** We first have to show that $p_c^*(p) > b > 0$ for some constant $b$. Since we have shown that $p_c(p) > d > 0$ for some constant $d$, it follows that $p_c^*(p) = (p_c(p))^2 > d^2 > 0$, where $d^2$ is a constant.

Next, we need to demonstrate that

$$\frac{p_i^*(p)}{p_i^*(p) + p_f^*(p)} > 0.5.$$

Using the fact that $p_c^*(p) + p_i^*(p) + p_f^*(p) = 1$, we can rewrite this inequality as

$$1 - \frac{p_f^*(p)}{1 - p_c^*(p)} > 0.5$$

which is equivalent to showing

$$\frac{1 - p_c^*(p)}{p_f^*(p)} > 2.$$

Notice that from Theorem 3.6.3, we can write $p_c(p) = 1 - p_f(p) = 1 - \epsilon(1-\epsilon)^{\frac{p-3}{2}}\frac{p-1}{p}$. Using the result of Lemmas 3.6.5 for the quantity $p_f^*(p)$ and the fact that $p_c^*(p) =$

$(p_c(p))^2$, we have

$$\begin{aligned}
& \frac{1-\left(1-\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{p})\right)^2}{2\epsilon^2(1-\epsilon)^{p-3}(\frac{p-1}{2p})^2} \\
= & \frac{\left(2-\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{p})\right)\left(\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{p})\right)}{2\epsilon^2(1-\epsilon)^{p-3}(\frac{p-1}{2p})^2} \\
= & \frac{2-\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{p})}{2\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{2p})} \\
= & \frac{1}{\epsilon(1-\epsilon)^{\frac{p-3}{2}}(\frac{p-1}{2p})}-\frac{1}{2} \\
= & (\frac{1}{\epsilon})\left(\frac{1}{(1-\epsilon)^{\frac{p-3}{2}}}\right)\left(\frac{2p}{p-1}\right)-\frac{1}{2}.
\end{aligned}$$

Using the facts that $\epsilon < \frac{1}{2}$ and $p > 2$, we group and prove the inequality. Specifically,

$$\begin{aligned}
& (\frac{1}{\epsilon})\left(\frac{1}{(1-\epsilon)^{\frac{p-3}{2}}}\right)\left(\frac{2p}{p-1}\right)+1 \\
> & (2)(1)(2)-\frac{1}{2} \\
> & 2,
\end{aligned}$$

which proves that this "two-trial" algorithm is robust when $a = 0$. ∎

Combining Lemma 1.3.1 with Theorem 3.6.2, we know that two independent trials of Algorithm 3.1 will be robust when the input is non-zero. We summarize this result in the following theorem, which is one of two main theorems in this chapter.

**Theorem 3.6.7** *When two independent executions of Algorithm 3.1 are run on the same input, the algorithm is robust, so long as the probability of error is $\epsilon < \frac{1}{p}$ and the prime $p$ satisfies $p \geq 11$.*

## 3.7 Robustness results for the binary algorithm

We apply a similar analysis to the binary algorithm (Algorithm 3.2) as was done for the linear algorithm (Algorithm 3.1) in Section 3.6. We will show that Algorithm 3.2 is robust, so long as two independent trials are executed or the input is not 0. For this analysis, we will make the assumption that errors only affect the variable $T$ in Algorithm 3.2. This assumption does not adversely limit the effectiveness of the analysis, since an error introduced into variable $P$ will cause the value of $T$ to also be erroneous, and thus, we capture that case even under this restriction.

It is worth noting that in all cases below, we use log to denote $\log_2$.

**Theorem 3.7.1** *Consider a prime $p$. If the probability of an error occurring before any particular step in the binary quadratic residue algorithm (Algorithm 3.2) is $\epsilon$, and the input $a \neq 0$, then the probability that the output of the algorithm will be*

*(a) correct is*

$$p_c(p) = (1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil} + \epsilon(1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil - 1}(\frac{1}{2} - \frac{1}{2p}) + \frac{1}{p}\left(1 - (1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil - 1}\right),$$

*(b) feasible but incorrect is*

$$p_f(p) = \epsilon(1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil - 1}(\frac{1}{2} + \frac{1}{2p}) + \frac{2}{p}\left(1 - (1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil - 1}\right),$$

*(c) infeasible and incorrect is*

$$p_i(p) = \left(1 - \frac{3}{p}\right)(1 - (1-\epsilon)^{\lceil \log(\frac{p-1}{2}) \rceil - 1}).$$

**Proof:** Relying on the proof used in Theorem 3.6.1, we note the only difference is that there are $\lceil \log\left(\frac{p-1}{2}\right) \rceil$ steps in the binary algorithm. Moreover, Lemma 3.5.1 states that both the binary and linear algorithm behave the same when the input is altered before the first step, and Lemmas 3.5.3 and 3.5.4 yield the same result. Thus, we only need to alter the summation formulas to take into account the $\lceil \log\left(\frac{p-1}{2}\right) \rceil$ steps of the binary quadratic residue algorithm.

We begin by proving part (b), using the same reasoning as in Theorem 3.6.1, and letting $k = \lceil \log\left(\frac{p-1}{2}\right) \rceil$, we have

$$p_f(p) = \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} + \frac{1}{2p}) + \sum_{i=2}^{k}(1-\epsilon)^{k-i}\epsilon\frac{2}{p}$$

which simplifies to

$$p_f(p) = \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} + \frac{1}{2p}) + \frac{2}{p}\left(1 - (1-\epsilon)^{k-1}\right).$$

which proves the result.

Again, using the same reasoning as Lemma 3.6.1 we derive a formula for part (a) as

$$\begin{aligned} p_c(p) &= (1-\epsilon)^k + \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} - \frac{1}{2p}) + \sum_{i=2}^{k}(1-\epsilon)^{k-i}\epsilon\frac{1}{p} \\ &= (1-\epsilon)^k + \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} - \frac{1}{2p}) + \frac{1}{p}\left(1 - (1-\epsilon)^{k-1}\right), \end{aligned}$$

which is the desired result.

For part (c), we have

$$\begin{aligned} p_i(p) &= \sum_{i=2}^{k}\epsilon(1-\epsilon)^{k-i}\left(1 - \frac{3}{p}\right) \\ &= \left(1 - \frac{3}{p}\right)(1 - (1-\epsilon)^{k-1}), \end{aligned}$$

which is the desired probability measure. ■

We now need to demonstrate the robustness of Algorithm 3.2. To do this, we will mimic the proof of Theorem 3.6.2.

**Theorem 3.7.2** *If the probability of error* $\epsilon < \frac{1}{\lceil \log(\frac{p-1}{2}) \rceil}$ *and* $p > 33$ *is prime, Algorithm 3.2 is robust on input* $a \neq 0$.

**Proof:** We must show that $p_c(p) > b > 0$ for some constant $b$ and $\frac{p_f(p)}{p_i(p)} < 1$. To minimize the notation, we let $k = \lceil \log\left(\frac{p-1}{2}\right) \rceil$, and will expand $k$ only when necessary.

We begin by showing $p_c(p) > b > 0$ for some constant $b$. We can rearrange $p_c(p)$ as

$$\begin{aligned} p_c(p) &= (1-\epsilon)^k + \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} - \frac{1}{2p}) + \frac{1}{p}\left(1 - (1-\epsilon)^{k-1}\right) \\ &= \left(1 - \frac{1}{p(1-\epsilon)}\right)(1-\epsilon)^k + \epsilon(1-\epsilon)^{k-1}(\frac{1}{2} - \frac{1}{2p}) + \frac{1}{p}. \end{aligned}$$

Notice that since $\epsilon < \frac{1}{k}$, we have

$$(1-\epsilon)^k > (1-\frac{1}{k})^k > (\frac{1}{e})^2$$

using the facts that $k > 4$ and the function $(1-\frac{1}{k})^k$ is monotonically increasing. Returning to $p_c(p)$, we have

$$\begin{aligned} p_c(p) &= (1-\frac{1}{p})(1-\epsilon)^k + \epsilon(1-\epsilon)^{k-1}(\frac{1}{2}-\frac{1}{2p}) + \frac{1}{p} \\ &> (1-\frac{1}{33})(\frac{1}{e^2}) \\ &> 0, \end{aligned}$$

which proves that $p_c(p) > b > 0$. Notice that to this point, we only required $p \geq 3$ to demonstrate that $p_c(p)$ is bounded below by a constant: we will need a larger lower bound on $p$ in the next part of the proof, however.

To show $\frac{p_f(p)}{p_i(p)} < 1$, notice this is equivalent to showing that $p_i(p) > p_f(p)$. Equivalently, we must show the following inequality holds:

$$p_i(p) - p_f(p) = \left(1-\frac{5}{p}\right)(1-(1-\epsilon)^{k-1}) - \epsilon(1-\epsilon)^{k-1}\left(\frac{1}{2}+\frac{2}{p}\right) > 0.$$

Rearranging, we must show

$$\left(1-\frac{5}{p}\right)(1-(1-\epsilon)^{k-1}) > \epsilon(1-\epsilon)^{k-1}\left(\frac{1}{2}+\frac{2}{p}\right).$$

Notice that if we assume $p > 33$ and $\epsilon < \frac{1}{3}$, then

$$\begin{aligned} 1-(1-\epsilon)^k &> 1-(1-\epsilon)^4 \\ &= 4\epsilon - 6\epsilon^2 + 4\epsilon^3 - \epsilon^4 \\ &= \epsilon(4-6\epsilon+4\epsilon^2-\epsilon^3) \\ &> \epsilon(4-6\epsilon) \\ &> 2\epsilon \\ &> \frac{33}{28}\epsilon. \end{aligned}$$

Thus,

$$\begin{aligned}\left(1-\frac{5}{p}\right)\left(1-(1-\epsilon)^{k-1}\right) &> \frac{28}{33}(1-(1-\epsilon)^{k-1}) \\ &> \epsilon \\ &> \epsilon(1-\epsilon)^{k-1}\left(\frac{1}{2}+\frac{2}{p}\right),\end{aligned}$$

which proves the desired inequality, showing that $p_i(p) > p_f(p)$. Thus, the binary quadratic residue algorithm is robust for input $a \neq 0$. ■

In a similar way to what was demonstrated in the linear quadratic residue algorithm, the Algorithm 3.2 is not robust on input $a = 0$. However, we can show that combining two independent trials of Algorithm 3.2 does result in a robust algorithm. That result is summarized in the second main theorem of this chapter.

**Theorem 3.7.3** *When two independent executions of Algorithm 3.2 are run on the same input, the algorithm is robust.*

**Proof:** Alter Theorem 3.6.6 by substituting the value $\lceil \log\left(\frac{p-1}{2}\right)\rceil - 1$ into the expression $\frac{p-3}{2}$, it follows that two independent trials on input $a = 0$ result in a robust algorithm. We can combine this fact with Theorem 3.7.2 and Lemma 1.3.1 to demonstrate that combining two independent trials of Algorithm 3.2 on non-zero input maintains robustness. Thus, on all inputs, two independent executions of Algorithm 3.2 satisfies the conditions for robustness. ■

## 3.8 Conclusion

We have shown robustness results for both the linear and binary quadratic residue algorithms. In particular, both algorithms require two independent executions in order to meet the specifications of robustness.

In the next chapter, we apply some of the same techniques used in the analysis of quadratic residues to perform analysis of the robustness of algorithms for determining primality. Specifically, we will examine the robustness of Pepin's test for primality.

# Chapter 4

# Robustness of Pepin's primality test

## 4.1 Introduction

In this chapter, we examine the robustness of Pepin's test for primality for numbers of the form $F_k = 2^{2^k} + 1, k \geq 1$. In particular, we prove robustness results for all $F_k$ both explicitly (given the factorization of $F_k$) and heuristically (based only on $k$).

## 4.2 Pepin's algorithm

Pepin's test for primality of Fermat numbers $F_k = 2^{2^k} + 1$ (described in his 1877 paper [75]), relies on the iteration of $f : x \to x^2 \pmod{F_k}$ beginning with $x = 5$. Specifically, we can consider Pepin's test to be equivalent to the following pseudocode:

The correctness of Pepin's algorithm can be easily shown, based on two lemmas. Specifically, we will need Euler's criterion (Lemma 3.2.1 in Chapter 3) in addition to the following lemma, first published by Kraitchik [52] and subsequently by Lehmer [55].

**Lemma 4.2.1** *An integer $n$ is prime if and only if there exists an integer $a$ such that*

$$a^{n-1} \equiv 1 \pmod{n}$$

$x \leftarrow 5$
**for** $i$ from 1 to $2^k - 1$ **do**
  $x \leftarrow x^2 \pmod{F_k}$
**end for**
**if** $x = F_k - 1$ **then**
  **print** "$F_k$ is prime"
**else**
  **print** "$F_k$ is composite"
**end if**

Algorithm 4.1: Pepin's Algorithm

*and*

$$a^{\frac{n-1}{q}} \not\equiv 1 \pmod{n}$$

*for all primes $q$ that divide $n-1$.*

**Proof:** Assume $n$ is prime. Consider a generator, $a$ for $GF(n)^*$. Since $a$ has order $n-1$, it must be the case that $a^{n-1} \equiv 1 \pmod{n}$ and $a^{\frac{n-1}{q}} \not\equiv 1 \pmod{n}$ for all divisors $q$ of $n-1$.

For the converse, assume an $a$ exists such that $a^{n-1} \equiv 1 \pmod{n}$ and $a^{\frac{n-1}{q}} \not\equiv 1 \pmod{n}$ for all primes $q$ that divide $n-1$. We can deduce that the multiplicative order of $a$ in $(\mathbb{Z}/(n))^*$ must divide $n-1$. Suppose, by way of contradiction, that $\mathrm{ord}_n\ a < n-1$. Then there must exist an integer $k > 1$ such that $\mathrm{ord}_n\ a = \frac{n-1}{k}$. Furthermore, there is some prime $p$ that must divide $k$, or equivalently, $k = mp$ for some $m$. Using this relation as an exponent, we have

$$a^{\frac{n-1}{k}} \equiv a^{\frac{n-1}{mp}} \equiv 1 \pmod{n}.$$

Raising this equation to the $m$th power, we have

$$a^{\frac{n-1}{p}} \equiv 1 \pmod{n},$$

which contradicts the fact that all prime $p$ have $a^{\frac{n-1}{p}} \not\equiv 1 \pmod{n}$. Thus $\mathrm{ord}_n\ a = n-1$ and hence, $n$ is prime. ∎

Relying on Lemma 3.2.1 and 4.2.1, we prove the correctness of the algorithm outlined by Pepin [75].

**Theorem 4.2.2** *The algorithm for Pepin's test for primality (Algorithm 4.1) is correct (if no errors occur).*

**Proof:** We need to demonstrate that $F_k$ is prime iff $5^{2^{2^k-1}} \equiv -1 \pmod{F_k}$, since after the for-loop, we know that $x \equiv 5^{2^{2^k-1}} \pmod{F_k}$.

Suppose that $F_k$ is prime. We have $F_k - 1 = 2^{2^k}$. Inductively, $2^{2^2} \equiv 1 \pmod 5$ and if $2^{2^k} \equiv 1 \pmod 5$ for $k \geq 2$, then $(2^{2^k})^2 \equiv 2^{2^{k+1}} \equiv 1 \pmod 5$. Thus, we know that $2^{2^k} \equiv 1 \pmod 5$ for all $k \geq 2$. Using quadratic reciprocity, we have

$$\left(\frac{5}{F_k}\right) = \left(\frac{F_k}{5}\right) = \left(\frac{2}{5}\right) = -1.$$

Since 5 is a quadratic nonresidue of $F_k$ we can use Lemma 3.2.1 to deduce $5^{\frac{F_k-1}{2}} \equiv 5^{2^{2^k-1}} \equiv -1 \pmod{F_k}$.

On the other hand, suppose $5^{2^{2^k-1}} \equiv -1 \pmod{F_k}$. Then

$$5^{\frac{F_k-1}{2}} \equiv -1 \pmod{F_k}$$

which, by squaring both sides, yields

$$5^{F_k-1} \equiv 1 \pmod{F_k}.$$

Since $F_k - 1 = 2^{2^k}$, the only prime divisor of $F_k - 1$ is 2. Using Lemma 4.2.1, we have that all prime divisors, $q$, of $F_k - 1$ satisfy $5^{\frac{F_k-1}{q}} \not\equiv 1 \pmod{F_k}$. Therefore, $F_k$ must be prime. ■

## 4.3 Previous work

Since Pepin's test for primality relies on the map $f : x \to x^2 \pmod N$, and since squaring is the one of the fundamental operations in computing elements in $(\mathbb{Z}/(N))^*$ (see Chapter 3 for evidence of this fact), it is not surprising that the map $x \to x^2 \pmod N$ has been thoroughly studied, for both prime and composite $N$, as well as generalized powers (other than 2). We briefly outline such work in this section.

Chassé [21, 22, 23] proved some basic results regarding the cycle length of iterations of the form $x \to x^2 + d$.

When the modulus is prime, Blanton, Hurd and McCranie [10, 11], Rogers [84] and Flores [35] independently analyzed the structure of the digraph formed by the map $x \to x^2 \pmod N$. Lucheta, Miller and Reiter [63] generalized this iteration

to allow arbitrary powers (i.e., $x \rightarrow x^{\alpha} \pmod{N}$) when the modulus is prime. The iteration $x \rightarrow x^k$ over the $p$-adic numbers was discussed by Khrennikov and Nilsson [50].

When the modulus is composite, Wilson [100] and Brennan and Geist [17] independently analyzed the iteration $x \rightarrow x^{\alpha} \pmod{N}$ for all $\alpha \geq 2$ and composite $N$. Additional work by Somer and Křížek [87] in this area led to two necessary and sufficient conditions for compositeness of Fermat numbers. Martin and Pomerance [65] examine the statistical properties of an upper bound for the length of the period of the iteration $x \rightarrow x^{\alpha} \pmod{N}$, and in doing so, provide some measures for the number of cycles formed by this iteration.

It is also worth noting that the topology of the functional digraph of quadratic maps is related to Shanks' chains of primes, as recently investigated by Teske and Williams [92].

Finally, the iteration $x \rightarrow x^2$ modulo composite numbers is an integral part of modern pseudo-random bit generation, as discussed, for example, in Blum, Blum, and Shub [12].

We will use some of the previous work on the map $x \rightarrow x^2 \pmod{N}$ over $(\mathbb{Z}/(N))^*$ to make both explicit and heuristic arguments for the robustness of Pepin's test for primality in the following sections.

### 4.3.1 Viewing the iteration as a digraph

For the map $x \rightarrow x^2 \pmod{N}$, we illustrate three examples: one for Fermat primes (i.e., primes of the form $2^{2^k} + 1$), one for Mersenne primes (i.e., primes of the form $2^k - 1$) and one for a composite number (i.e., $N = 35$). These are shown in Figures 4.1, 4.2, and 4.3 respectively.

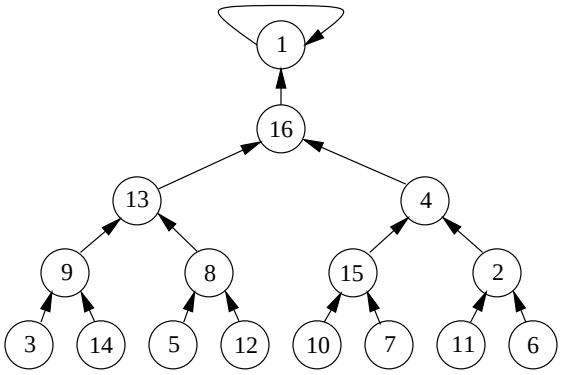


Figure 4.1: The digraph $G_{17,0}$.

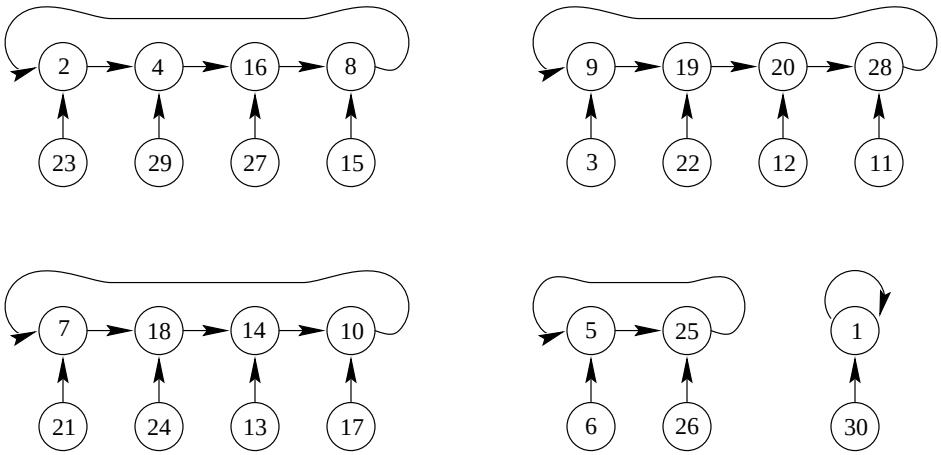


Figure 4.2: The digraph $G_{31,0}$.

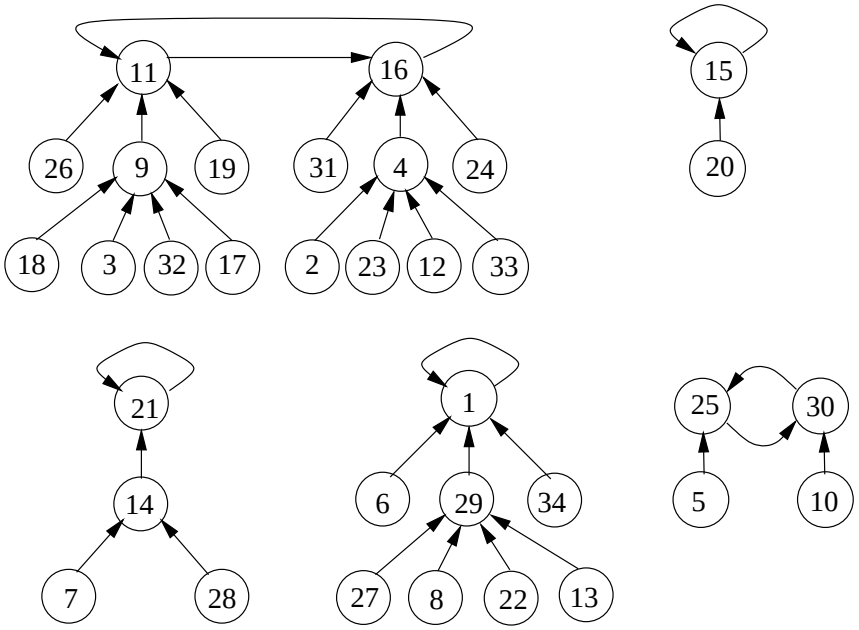


Figure 4.3: The digraph $G_{35,0}$.

## 4.4 The case when $F_k$ is prime

If $F_k$ is prime, we consider how an output of "composite" can occur if an error is introduced before, during or after the for-loop in the algorithm discussed above for the computed value of $x$.

We begin our analysis by considering the case $f : x \to x^2 \pmod{p}$ when $p$ is prime. This case has been thoroughly studied, and we summarize that previous work in the following two theorems.

**Theorem 4.4.1** *Let $p$ be an odd prime, and let the iteration be $f : x \to x^2 \pmod{p}$. The tail length for an element $x \neq 0$ is $t(x) = \nu_2(\mathrm{ord}_p\ x)$ and the cycle length for $x$ is $c(x) = \mathrm{ord}_l\ 2$, where $\mathrm{ord}_p\ x = 2^e \cdot l$ and $e, l$ are non-negative integers with $l$ odd.*

**Proof:** By the definitions of tail and cycle length from Chapter 2, we know that $f^t(x) = f^{t+c}(x)$. It follows we must have $x^{2^t} \equiv x^{2^{t+c}} \pmod{p}$. Rearranging, we have $x^{2^t(2^c-1)} \equiv 1 \pmod{p}$. Write $\mathrm{ord}_p\ x = 2^e \cdot l$ where $l$ is an odd positive integer and $e$ is a non-negative integer. Then by definition of order, we have $2^e \cdot l \mid 2^t(2^c-1)$.

Since $c(x)$ and $t(x)$ are the least such integers that satisfy the necessary relations, we have $t(x) = e = \nu_2(\mathrm{ord}_p\ x)$, and $c = \mathrm{ord}_l\ 2$. ∎

We can characterize the tails of elements in terms of primitive roots, as follows:

**Theorem 4.4.2** *Let $p$ be an odd prime, and let $\gamma$ be a primitive root modulo $p$. Then*

*(a) the set of elements in cycles is*

$$\{a \in GF(p)^* \ :\ t(a) = 0\} = \{\gamma^i \ :\ 0 \le i < p \text{ and } \nu_2(i) \ge \nu_2(p-1)\};$$

*(b) the set of elements that have tail length $k$ (for $1 \le k \le \nu_2(p-1)$) is*

$$\{a \in GF(p)^* \ :\ t(a) = k\} = \{\gamma^i \ :\ 0 \le i < p \text{ and } \nu_2(i) = \nu_2(p-1) - k\};$$

**Proof:** Since $a \in GF(p)^*$, there must exist an integer $i$ such that $a = \gamma^i$. Write $p - 1 = 2^\tau \cdot \rho$, where $\rho$ is odd. To prove (a), we have the following equivalences:

$$\begin{aligned}
t(a) = 0 &\iff a = a^{2^l} \text{ for some } l > 0 \\
&\iff a^{2^l - 1} = 1 \\
&\iff \gamma^{i(2^l-1)} = 1 \\
&\iff p - 1 \,|\, i(2^l - 1) \\
&\iff \rho \,|\, 2^l - 1 \text{ and } \nu_2(i) \ge \nu_2(p-1).
\end{aligned}$$

To see the last equivalence, notice that if we pick $l = \mathrm{ord}_\rho 2$, then $2^l \equiv 1 \pmod{\rho}$ and thus $\rho \,|\, 2^l - 1$.

For (b), we have the following equivalences:

$$\begin{aligned}
t(a) = k \text{ for } k \ge 1 &\iff a^{2^k} = a^{2^{k+l}} \text{ and } a^{2^{k-1}} \ne a^{2^{k+l-1}} \text{ for some } l > 0 \\
&\iff a^{2^k(2^l-1)} = 1 \text{ and } a^{2^{k-1}(2^l-1)} \ne 1 \\
&\iff \gamma^{i2^k(2^l-1)} = 1 \text{ and } \gamma^{i2^{k-1}(2^l-1)} \ne 1 \\
&\iff p - 1 \,|\, i2^k(2^l - 1) \text{ and } p - 1 \nmid i2^{k-1}(2^l - 1) \\
&\iff \rho \,|\, i(2^l - 1) \text{ and } \nu_2(i2^k) = \nu_2(p-1).
\end{aligned}$$

∎

It is worth noting that Theorem 4.4.1 was originally proved in the literature by Blanton, Hurd and McCranie [10, 11]. Corollary 4.4.3 is also due to Blanton, Hurd and McCranie [10, Theorems 2 and 4].

We now prove the first major result of this chapter concerning the structure of Pepin's test.

It follows from the previous theorem that, in general, the topology of the functional digraph $G_{p,0}$ can be described as follows:

**Corollary 4.4.3** *Let $p$ be an odd prime with $p-1 = 2^\tau \cdot \rho$, $\rho$ odd. For each positive divisor $d$ of $\rho$, $G_{p,0}$ contains $\varphi(d)/\mathrm{ord}_d\ 2$ cycles of length $\mathrm{ord}_d\ 2$. There are $\rho$ elements in all these cycles, and off each element in these cycles hangs a reversed complete binary tree of height $\tau - 1$ containing $2^\tau - 1$ elements.*

**Proof:** Let $\gamma$ be a primitive root modulo $p$. Elements that are in a cycle have no tail length, and thus, if $x$ is in a cycle, then $t(x) = 0$. Using Theorem 4.4.2, $x$ must be of the form $\gamma^{j\cdot 2^\tau}$, $0 \le j < \rho$. It follows that there must be a total of $\rho$ elements in all cycles. Since $\gamma^{2^\tau}$ generates a subgroup of $(\mathbb{Z}/(\rho))^*$ of order $\rho$, we must have $\varphi(d)$ elements of order $d$ for each divisor of $\rho$. The elements of order $d$ are $\gamma^{j2^\tau\rho/d}$ for $0 \le j < d, \gcd(j,d) = 1$. Using a similar technique to Theorem 4.4.2, we see that if the cycle length is $c$, then $\gamma^{(j2^\tau\rho/d)(2^c-1)} = 1$, that gives a cycle of length $c = \mathrm{ord}_d\ 2$. It immediately follows that there must be a total of $\varphi(d)/\mathrm{ord}_d\ 2$ distinct cycles.

For the tails of size 1, we must square to an element of the form $\gamma^{j\cdot 2^\tau}$ but not be of this form. That is, the elements that have tail length 1 are $\gamma^{j\cdot 2^{\tau-1}}$. Inductively, if $\gamma^i$ is an element of tail length $t$, $1 \le t < \tau$, the elements with tail length $t+1$ are $\gamma^{i/2}$ and $\gamma^{(i+p-1)/2}$, which are distinct since $\gamma^{(p-1)/2} = -1$. ∎

As an example, let us consider the case $p = 29$, where $\tau = 2$ and $\rho = 7$. See Figure 4.4.

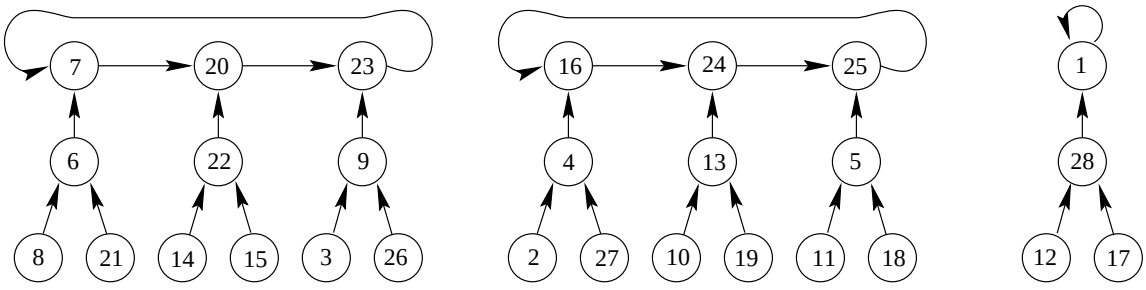


Figure 4.4: The topology of $G_{29,0}$

We can enumerate the number of elements in $GF(p)^*$ that have a given cycle structure, as follows: for each divisor $d$ of $p-1$ there are exactly $\varphi(d)$ elements of $GF(p)^*$ of order $d$. From above, the tail length for each such element is $t = \nu_2(d)$ and the cycle length is of size $\text{ord}_{d/2^{\nu_2(d)}}\ 2$. For example, for $p = 29$ we have the data in Table 1.

| $d$ | $\varphi(d)$ | elements of order $d$ | $t = \nu_2(d)$ | $l = d/2^t$ | $c = \text{ord}_l\ 2$ |
|---|---|---|---|---|---|
| 1 | 1 | $\{1\}$ | 0 | 1 | 1 |
| 2 | 1 | $\{28\}$ | 1 | 1 | 1 |
| 4 | 2 | $\{12, 17\}$ | 2 | 1 | 1 |
| 7 | 6 | $\{7, 16, 20, 23, 24, 25\}$ | 0 | 7 | 3 |
| 14 | 6 | $\{4, 5, 6, 9, 13, 22\}$ | 1 | 7 | 3 |
| 28 | 12 | $\{2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27\}$ | 2 | 7 | 3 |

Table 1: The structure of $G_{29,0}$

We are now ready to use these general theorems in the case when $F_k$ is prime.

**Theorem 4.4.4** *The structure of the digraph $G_{F_k,0}$ when $F_k$ is prime is a reversed complete binary tree of height $2^k - 1$ with root $-1$, attached to a cycle of length 1 on the integer 1. The elements $x$ with $t(x) = a$ are given by $5^{e \cdot 2^{k-a}}, 0 \leq e < 2^a$, $e$ odd.*

**Proof:** Use Theorem 4.4.2 and Corollary 4.4.3. Since $F_k - 1 = 2^{2^k}$, the only odd divisor of $p-1$ is 1. By the argument presented in Theorem 4.2.2, 5 must be a primitive root of $F_k$ when $F_k$ is prime. ■

As an example, we can observe that for Figure 4.1, we have $k = 2$ when $p = 17$, and thus, the height of the binary tree is $2^2 - 1 = 3$.

Having the structure for $G_{F_k,0}$ when $F_k$ is prime, we now explicitly quantify the robustness of Algorithm 4.1 if an error occurs to the register containing $x$. Again, we are assuming the error model outlined in Section 1.3, where we model errors as events that change $x$ randomly to any element of $\mathbb{Z}/(F_k)$.

**Theorem 4.4.5** *Suppose $F_k$ is prime, for some $k \geq 2$. If the last error to occur to $x$ is just before the $i$th squaring step in the for-loop of the algorithm for Pepin's test as outlined in Section 4.2, the probability that the output is "prime" is*

$$\frac{2^{2^k - i}}{F_k}$$

*If the last error is introduced into $x$ after the for-loop, the probability the output is "prime" is $\frac{1}{F_k}$.*

**Proof:** Let $x_1, x_2, ..., x_{2^k-1}$ be the value of $x$ before the first, second, ..., $(2^k-1)$th squaring step, respectively. Let $x_{2^k}$ be the value of $x$ after the $(2^k-1)$th squaring step.

To prove this result, we work backwards. Consider $x_{2^k}$: if there are no more squaring steps, we enter the if-statement. That is, if the error is introduced, it must cause $x_{2^k}$ to remain $F_k - 1$: the chance that this occurs is $\frac{1}{F_k}$.

By Theorem 4.4.4, we know that the element $-1$ is the root of a complete binary tree of height $2^k - 1$. Thus, if we have $j$ more squarings to do before outputting, we must have

$$(x_{2^k-j})^{2^j} = F_k - 1 \pmod{F_k}$$

in order for the output to be "prime." The only solutions to this equation are those elements that are at depth $j$ in the complete binary tree rooted at $-1$, and there are $2^j$ such elements at depth $j$ in the complete binary tree rooted at $-1$. Since $j = 2^k - i$ (that is, if there are $j$ more squarings to do, we must be at step $2^k - i$), there are $2^{2^k-i}$ values that satisfy the above equation. Therefore, the probability that the error introduced into $x_i$ will lead to $F_k - 1$ is $\frac{2^{2^k-i}}{F_k}$. ■

We now use Theorem 4.4.5 to evaluate the robustness of Pepin's test in the case when the input is prime. As outlined in Section 1.3, we have to compute the probability of correct output ($p_c(F_k)$), the probability of incorrect and feasible output ($p_f(F_k)$) and the probability of incorrect and infeasible output ($p_i(F_k)$) on inputs of size $F_k = 2^{2^k} + 1$.

We capture some fundamental probabilities in the following lemma.

**Lemma 4.4.6** *Consider Pepin's algorithm (Algorithm 4.1) on input $F_k$ that is prime. If the probability of an error occurring at any step of the algorithm is $\epsilon$, we have*

1. $p_c(F_k) = (1-\epsilon)^{2^k} + \frac{2^{2^k}}{F_k}\frac{\epsilon}{1-2\epsilon}\left((1-\epsilon)^{2^k} - \frac{1}{2^{2^k}}\right)$
2. $p_f(F_k) = 1 - p_c(F_k)$, *and*
3. $p_i(F_k) = 0$.

**Proof:** To begin, we compute the probability of correctness. There are two components. The first component of this probability is the case when no error occurs, which means that at all $2^k$ steps, no error occurs, and the probability of no error in a single step is $(1-\epsilon)$. If an error does occur, we use the result of Theorem 4.4.5 together with the probability that an error occurs at step $i$, where $i$ ranges over all possible steps. Recall that if the probability of error is $\epsilon$, the probability that the last error occurs at step $j$ is $\epsilon(1-\epsilon)^{2^k-j}$, since there must be an error (this accounts for the $\epsilon$ term) and the remaining $2^k - j$ steps must be error free (which accounts for the $(1-\epsilon)^{2^k-j}$ term). To summarize mathematically, we have

$$\begin{aligned}
p_c(F_k) &= (1-\epsilon)^{2^k} + \sum_{i=1}^{2^k} (1-\epsilon)^{2^k-i} \epsilon \frac{1}{2^i} \frac{2^{2^k}}{F_k} \\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k} \frac{2^{2^k}}{F_k} \sum_{i=1}^{2^k} \left( \frac{1}{2(1-\epsilon)} \right)^i \\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k} \frac{2^{2^k}}{F_k} \frac{1}{2(1-\epsilon)} \frac{(\frac{1}{2(1-\epsilon)})^{2^k} - 1}{\frac{1}{2(1-\epsilon)} - 1} \\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k} \frac{2^{2^k}}{F_k} \frac{1}{2(1-\epsilon)} \left( \left( \frac{1}{2(1-\epsilon)} \right)^{2^k} - 1 \right) \frac{2(1-\epsilon)}{2\epsilon - 1} \\
&= (1-\epsilon)^{2^k} + \frac{2^{2^k}}{F_k} \frac{\epsilon}{2\epsilon - 1} \left( \frac{1}{2^{2^k}} - (1-\epsilon)^{2^k} \right) \\
&= (1-\epsilon)^{2^k} + \frac{2^{2^k}}{F_k} \frac{\epsilon}{1 - 2\epsilon} \left( (1-\epsilon)^{2^k} - \frac{1}{2^{2^k}} \right)
\end{aligned}$$

Since the only other output from Pepin's test will be a valid integer in the range $0, \ldots, 2^{2^k}$, we have $p_f = 1 - p_c$ and $p_i = 0$. That is, there are no infeasible outputs. ■

Using this lemma, we note that Pepin's algorithm is not robust, since the ratio $\frac{p_i}{1-p_c} = 0$ does not satisfy the definition of robustness as stated in Section 1.3. However, if we combine the results of two independent executions of the algorithm, Pepin's test becomes robust.

Under this combined model, the output is correct if both independent trials give the same result. Thus, $p_c^*(F_k) = p_c(F_k)^2$ where $p_c^*$ represents the probability of correct output under this combined model. The output is incorrect and feasible if the results from both independent executions of the algorithm yield the same answer. In other words, an error must occur in one execution, and the other

execution must end up with exactly the same value. Suppose the value of the first execution of the algorithm is $v$. Based on Theorem 4.4.4, we know that in order for the second execution of the algorithm to end up with $v$ we must have

- the last error to occur at step $2^k$ and the error changes its stored value to $v$, or
- in general, the last error occurs at step $2^k - j$ and the error changes the stored value to one of the $2^j$ children of $v$ that are distance $j$ below $v$ in the tree, where $0 \leq j < 2^k$.

Formulating these observations, we have

$$\begin{aligned}
p_f^*(F_k) &= p_f(F_k) \cdot \sum_{j=1}^{2^k} (1-\epsilon)^{2^k - j} \epsilon \frac{2^j}{F_k} \\
&= p_f(F_k) \frac{\epsilon(1-\epsilon)^{2^k}}{F_k} \sum_{j=1}^{2^k} \left( \frac{2}{1-\epsilon} \right)^j \\
&= p_f(F_k) \frac{\epsilon(1-\epsilon)^{2^k}}{F_k} \left( \frac{\frac{2^{2^k+1}}{(1-\epsilon)^{2^k+1}} - \frac{2}{1-\epsilon}}{\frac{2}{1-\epsilon} - 1} \right) \\
&= p_f(F_k) \frac{\epsilon(1-\epsilon)^{2^k}}{F_k} \frac{2}{1-\epsilon} \left( \frac{\frac{2^{2^k}}{(1-\epsilon)^{2^k}} - 1}{\frac{1+\epsilon}{1-\epsilon}} \right) \\
&= p_f(F_k) \frac{\epsilon(1-\epsilon)^{2^k}}{F_k} \frac{2}{1-\epsilon} \frac{1-\epsilon}{1+\epsilon} \left( \frac{2^{2^k}}{(1-\epsilon)^{2^k}} - 1 \right) \\
&= p_f(F_k) \frac{\epsilon(1-\epsilon)^{2^k}}{F_k} \frac{2}{1+\epsilon} \left( \frac{2^{2^k} - (1-\epsilon)^{2^k}}{(1-\epsilon)^{2^k}} \right) \\
&= \frac{2p_f(F_k)\epsilon(2^{2^k} - (1-\epsilon)^{2^k})}{F_k(1+\epsilon)}.
\end{aligned}$$

To show our combined algorithm is robust, we need to show that

$$\frac{p_i^*(F_k)}{p_i^*(F_k) + p_f^*(F_k)} > 0.5,$$

and also show that $p_c^*(F_k)$ is bounded below by a constant larger than 0. We do so in the following theorem, which is one of the main results concerning robustness for Pepin's test.

**Theorem 4.4.7** *Consider Pepin's algorithm (Algorithm 4.1). If the input $F_k$ given to the algorithm is prime, and if the probability of an error occurring is $\epsilon < \frac{1}{2^k}$, the combination of two independent trials of Pepin's algorithm is robust.*

**Proof:** We must show two things: $\frac{p_i^*(F_k)}{p_i^*(F_k)+p_f^*(F_k)} > 0.5$ and $p_c^*(F_k) > b > 0$ for some constant $b$. First, we will show that $p_c^*(F_k)$ is bounded below by a constant larger than 0. We will use the fact that $p_c^*(F_k) = (p_c(F_k))^2$ combined with the definition of $p_c(F_k)$ from Lemma 4.4.6 to prove this first part. We know that

$$p_c(F_k) = (1-\epsilon)^{2^k} + \frac{2^{2^k}}{F_k}\frac{\epsilon}{1-2\epsilon}\left((1-\epsilon)^{2^k} - \frac{1}{2^{2^k}}\right).$$

Since we have $\epsilon < \frac{1}{2^k}$, we can bound the first term in the sum by

$$(1-\epsilon)^{2^k} > \frac{1}{e},$$

which gives a suitable lower bound on $p_c(F_k)$, and thus a lower bound on $p_c^*(F_k)$.

To demonstrate the second part of the theorem, we need to show that

$$\frac{p_i^*(F_k)}{p_i^*(F_k)+p_f^*(F_k)} > 0.5.$$

We have

$$\begin{aligned}
\frac{p_i^*(F_k)}{p_i^*(F_k)+p_f^*(F_k)} &= \frac{1-p_c(F_k)^2-p_f^*(F_k)}{1-p_c(F_k)^2-p_f^*(F_k)+p_f^*(F_k)} \\
&= \frac{1-p_c(F_k)^2-p_f^*(F_k)}{1-p_c(F_k)^2} \\
&= 1-\frac{p_f^*(F_k)}{1-p_c(F_k)^2} \\
&= 1-\frac{2p_f(F_k)\epsilon(2^{2^k}-(1-\epsilon)^{2^k})}{F_k(1+\epsilon)}\frac{1}{1-p_c(F_k)^2} \\
&= 1-\frac{2(1-p_c(F_k))\epsilon(2^{2^k}-(1-\epsilon)^{2^k})}{F_k(1+\epsilon)}\frac{1}{(1+p_c(F_k))(1-p_c(F_k))} \\
&= 1-\frac{2\epsilon(2^{2^k}-(1-\epsilon)^{2^k})}{F_k(1+\epsilon)(1+p_c(F_k))} \\
&> 1-\frac{2\epsilon(F_k-2)}{F_k(1+\epsilon)(1+p_c(F_k))} \\
&= 1-\frac{F_k-2}{F_k}\frac{2\epsilon}{(1+\epsilon)(1+p_c(F_k))}.
\end{aligned}$$

We look at components of the last line of the inequality. Since $k > 0$ then $\frac{F_k-2}{F_k} < 1$. Also, since $k > 0$, it follows that $\epsilon < \frac{1}{4}$ and thus $2\epsilon < \frac{1}{2}$. Additionally, since $\epsilon \geq 0$ and $p_c(F_k) \geq 0$, we have that $(1+\epsilon)(1+p_c(F_k)) > 1$. We can combine these inequalities to yield

$$\begin{aligned} 1 - \frac{F_k - 2}{F_k} \frac{2\epsilon}{(1+\epsilon)(1+p_c(F_k))} &> 1 - \frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

which proves the result. ■

## 4.5 The case when $F_k$ is composite

In this section, we analyze the robustness of Pepin's test when $F_k$ is composite. In order to do this, we will present more general results for arbitrary composite moduli due to Wilson [100] and use these results to prove robustness measures for composite $F_k$.

We begin by presenting some basic results on composite moduli, drawn from the classic text by Hua [47].

**Lemma 4.5.1** *A necessary and sufficient condition for $m$ to have a primitive root is that $m = 2, 4, p^l$ or $2p^l$, where $p$ is prime, $l \geq 1$.*

**Proof:** This is Theorem 3.9.1 of Hua [47]. ■

The following three lemmas rely on the definition of solutions to congruences as defined in Section 2.5.

**Lemma 4.5.2** *Let $f(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$ and define $f'(x) = n a_n x^{n-1} + \cdots + 2a_2 x + a_1$. Let $p$ be prime and $l \geq 1$. If $f(x) \equiv 0 \pmod{p}$ and $f'(x) \equiv 0 \pmod{p}$ have no common solution, then the two congruences $f(x) \equiv 0 \pmod{p^l}$ and $f(x) \equiv 0 \pmod{p}$ have the same number of solutions.*

**Proof:** See Theorem 2.9.3 of Hua [47]. ■

**Lemma 4.5.3** *The congruence $x^k \equiv n \pmod{p}$, ($p \nmid n$, $k \geq 0$) has 0 or $\gcd(k, p-1)$ solutions.*

**Proof:** This is Theorem 3.7.2 of Hua [47]. ∎

We now consider the case when the modulus is a power of 2.

**Lemma 4.5.4** *The congruence* $x^k \equiv n \pmod{2^a}$, $2 \nmid n$ *has* 0 *or* $C$ *solutions, where*

$$C = \begin{cases} 1, & \text{if } a \in \{0,1\}; \\ \gcd(k,2), & \text{if } a = 2; \\ \gcd(k,2) \cdot \gcd(2^{a-2},k), & \text{if } a > 2. \end{cases}$$

**Proof:** If the modulus is $2^a$, it is well known (see Hua [47, Theorem 3.9.4]) that

$$(\mathbb{Z}/(2^a))^* \cong \begin{cases} \{1\}, & \text{if } a \in \{0,1\}; \\ \mathbb{Z}/(2), & \text{if } a = 2; \\ \mathbb{Z}/(2) \times \mathbb{Z}/(2^{a-2}), & \text{if } a > 2. \end{cases}$$

The first two cases give the result trivially for $a \in \{0,1,2\}$. For $a > 2$, we know computing the $k$-th power in $(\mathbb{Z}/(2^a))^*$ is equivalent to multiplying by $k$ in the corresponding additive representation, and since multiplying by $k$ is a $\gcd(2^{a-2},k)$-to-1 map, the result follows. ∎

We combine the previous three lemmas into a result on the indegree of vertices in the component containing $-1$ in the digraph formed by $x \to x^k \pmod{N}$.

**Theorem 4.5.5** *Let* $N = 2^a p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$, *where* $p_i$ *is prime,* $a \geq 0$ *and* $e_i \geq 1$. *In the digraph formed by* $x \to x^k \pmod{N}$, *where* $p_i \nmid k$ *for all* $i$, $1 \leq i \leq m$, *the indegree of every vertex in the component containing* $-1$ *is either* 0 *or* $I$, *where*

$$I = \begin{cases} \Pi_{i=1}^m \gcd(k, p_i - 1), & \text{if } a \in \{0,1\}; \\ \gcd(k,2) \Pi_{i=1}^m \gcd(k, p_i - 1), & \text{if } a = 2; \\ \gcd(k,2) \gcd(2^{a-2},k) \Pi_{i=1}^m \gcd(k, p_i - 1), & \text{if } a > 2. \end{cases}$$

**Proof:** Using the Chinese Remainder Theorem, we know that solving $x^k \equiv -1 \pmod{N}$ is equivalent to solving the system of congruences

$$\begin{aligned} x^k &\equiv -1 \pmod{2^a} \\ x^k &\equiv -1 \pmod{p_1^{e_1}} \\ &\vdots \\ x^k &\equiv -1 \pmod{p_m^{e_m}}. \end{aligned}$$

Since $p_i \nmid k$ for all $i$, we know that $x^k \equiv -1 \pmod{p_i}$ and $kx^{k-1} \equiv 0 \pmod{p_i}$ have no common solution. It follows that we can apply Lemma 4.5.2 with Lemma 4.5.3 to deduce that $x^k \equiv -1 \pmod{p_i^{e_i}}$ has $\gcd(k, p_i - 1)$ solutions. We combine this result with Lemma 4.5.4 using the Chinese Remainder Theorem to derive the value of $I$ as indicated. ■

We now present a key result on the structure of the digraphs formed when the modulus is composite, due to Wilson [100].

**Theorem 4.5.6** *Let $N = 2^a p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$, where $p_i$ is prime, $a \geq 0$ and $e_i \geq 1$. In the digraph $G_{N,0}$, the height of the tree rooted at $-1$ is $h$, where*

$$h = \begin{cases} 0, & \text{if } a > 1; \\ \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}, & \text{otherwise.} \end{cases}$$

**Proof:** If $a > 1$, then $4 \,|\, N$. Therefore, for $x^2 \equiv -1 \pmod{N}$ to have a solution, $x^2 \equiv -1 \pmod{4}$ must have a solution. However, $x^2 \equiv 3 \pmod{4}$ has no solution, and so the height of the tree rooted at $-1$ must be 0 in this case.

If $a \leq 1$, we use the Chinese Remainder Theorem to deduce that $x^2 \equiv -1 \pmod{N}$ has solution $x_0$ iff $x_0$ is a solution to $x^2 \equiv -1 \pmod{p_i^{e_i}}$ for each $i$, $1 \leq i \leq m$. Furthermore, using Lemma 4.5.2, there is a one-to-one correspondence between solutions to $x^2 \equiv -1 \pmod{p_i^{e_i}}$ and solutions to $x^2 \equiv -1 \pmod{p_i}$.

We know by Corollary 4.4.3 that the height of the tree rooted at $-1$ is $\nu_2(p_i - 1) - 1$. In other words,

$$x^{2^{\nu_2(p_i-1)-1}} \equiv -1 \pmod{p_i}$$

has a solution and

$$x^{2^{\nu_2(p_i-1)}} \equiv -1 \pmod{p_i}$$

has no solution, for each $i$. Applying this to the composite case, we have the height of the tree rooted at $-1$ is $h$ iff

$$x^{2^h} \equiv -1 \pmod{N}$$

has a solution and

$$x^{2^{h+1}} \equiv -1 \pmod{N}$$

has no solution, which holds if and only if

$$x^{2^h} \equiv -1 \pmod{p_i}$$

has a solution for all $i$ and

$$x^{2^{h+1}} \equiv -1 \pmod{p_i}$$

has no solution for at least one such $i$. Combining these two facts together, we see that $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$, which proves the result. ∎

The following corollary is immediate.

**Corollary 4.5.7** *Let $N = 2^a p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$, where $p_i$ is prime, $a \geq 0$ and $e_i \geq 1$. In the digraph $G_{N,0}$, the height of the tree rooted at $-1$ is 0 iff $a > 1$ or $p_i \equiv 3 \pmod 4$ for some $i$.*

We now analyze the structure of this tree rooted at $-1$ to determine the number of elements contained within it.

**Theorem 4.5.8** *Let $N = 2^a p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$, where $p_i$ is prime, $a \geq 0$ and $e_i \geq 1$. In the digraph $G_{N,0}$, the tree rooted at $-1$ is a complete $2^m$-ary tree.*

**Proof:** Define $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$. We know from Corollary 4.5.7 that the tree rooted at $-1$ in $G_{N,0}$ is empty if $a > 1$ or if $p_i \equiv 3 \pmod 4$ for some $i, 1 \leq i \leq m$. Thus, we can view this as a complete $2^m$-ary tree: it just happens to have height 0.

Thus, non-empty trees occur only if $a \in \{0, 1\}$ and $p_i \equiv 1 \pmod 4$ for all $i, 1 \leq i \leq m$. It must be that $h > 0$ since $p_i - 1 \equiv 0 \pmod 4$ for $1 \leq i \leq m$.

We know that an element in the tree rooted at $-1$ has indegree 0 or

$$\Pi_{i=1}^{m} \gcd(2, p_i - 1) = 2^m$$

by Lemma 4.5.5. Since elements that are at depth $h$ in the tree rooted at $-1$ are solutions to the equivalence

$$x^{2^h} \equiv -1 \pmod{N},$$

it follows that there must be

$$\Pi_{i=1}^{m} \gcd(2^h, p_i - 1) = (2^h)^m$$

elements at depth $h$ in the tree rooted at $-1$ in $G_{N,0}$. Combining this fact with the fact that all elements have indegree 0 or $2^m$, it follows that the tree rooted at $-1$ must be a complete $2^m$-ary tree. ∎

We now combine the two main theorems of this section into a theorem for the case when $N = F_k = 2^{2^k} + 1$.

**Theorem 4.5.9** *Suppose $F_k$ is composite. Write $F_k = p_1^{e_1} \cdots p_m^{e_m}$. The tree rooted at $-1$ in $G_{F_k,0}$ is a complete $2^m$-ary tree of height $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$.*

**Proof:** We note $F_k \equiv 1 \pmod 4$, and so we may apply Theorems 4.5.6 and 4.5.8 to get the result. ∎

We will now use the previous theorem to prove a theorem on the robustness of Pepin's test in the composite case, based on the factorization of $F_k$.

**Theorem 4.5.10** *Suppose $F_k$ is composite, for some $k \geq 1$. Write $F_k = p_1^{e_1} \cdots p_m^{e_m}$ where all $p_i$'s are distinct odd primes, and $e_i > 0$. Let $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$. If the last error to occur to $x$ is just before the $j$th squaring step in the for-loop of the algorithm for Pepin's test, the probability that the output is "composite" is 1 if $j < 2^k - h$ and*

$$\frac{F_k - 2^{m(2^k - j)}}{F_k}$$

*otherwise. If the last error is introduced into $x$ after the for-loop, the probability the output is "composite" if $F_k$ is composite is $\frac{F_k - 1}{F_k}$.*

**Proof:** As in the proof of the robustness of Pepin's test if $F_k$ is prime, let $x_1, x_2, ..., x_{2^k - 1}$ be the value of $x$ before the first, second, ..., $(2^k - 1)$th squaring step, respectively. Let $x_{2^k}$ be the value of $x$ after the $(2^k - 1)$th squaring step.

To prove this result, we work backwards. Consider $x_{2^k}$: if there are no more squaring steps, we enter the if-statement. That is, if the error is introduced, it must cause $x_{2^k}$ (assuming it is any of the $F_k$ values) to become $F_k - 1$ in order for the output to be "prime". Thus, the chance that "composite" is outputted is $1 - \frac{1}{F_k} = \frac{F_k - 1}{F_k}$.

By Theorem 4.5.9, we know that the element $-1$ roots a complete $2^m$-ary tree of height $h$. Thus, if we have $l$ more squarings to do before outputting, we must have

$$(x_{2^k - l})^{2^l} \equiv F_k - 1 \pmod{F_k}$$

in order for the output to be "prime," with the proviso that $l \leq h$. If $l > h$, either the element $(x_{2^k - l})^{2^l} \pmod{F_k}$ will be 1 (if $x_{2^k - l}$ is in the same component as 1) or it will some other element $y$, where $y$ is not in the same component as 1 (or, more importantly, $-1$) in $G_{F_k,0}$. Thus, we are guaranteed to finish the algorithm with a value of $x \not\equiv -1 \pmod N$, and so the result holds if $l > h$.

If $l \leq h$, we know that the only solutions to the equation

$$(x_{2^k-l})^{2^l} = F_k - 1 \pmod{F_k}$$

are those elements that are at depth $l$ in the complete tree rooted at $-1$, and there are $2^{ml}$ such elements at depth $l$ in the complete tree rooted at $-1$. Since $l = 2^k - i$ (that is, if there are $l$ more squarings to do, we must be at step $2^k - i$), there are $2^{m(2^k-i)}$ values that satisfy the above equation. Therefore, the probability that the error introduced into $x_i$ will lead to $F_k - 1$ is $\frac{2^{m(2^k-i)}}{F_k}$. Reversing this result gives a probability of $\frac{F_k - 2^{m(2^k-i)}}{F_k}$ in the case when $i \geq 2^k - h$, as desired. ∎

We use Theorem 4.5.10 to quantify the probabilities of correct output, incorrect but feasible output and infeasible output, and summaries these statistics in the following lemma.

**Lemma 4.5.11** *Consider Pepin's algorithm (Algorithm 4.1) on composite input. Write $F_k = p_1^{e_1} \cdots p_m^{e_m}$ where all $p_i$'s are distinct odd primes, and $e_i > 0$. Let $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$. If the probability of error is $\epsilon$, we have*

- *the probability that the output is correct ($p_c(F_k)$) is*

$$1 - \left(\frac{\epsilon}{F_k}\right)\left(\frac{1 - ((1-\epsilon)2^m)^{h+1}}{1 - (1-\epsilon)2^m}\right)$$

- *the probability that the output is incorrect but feasible is*

$$p_f(F_k) = 1 - p_c(F_k),$$

- *the probability that the output is incorrect and infeasible is $p_i(F_k) = 0$.*

**Proof:** We begin by looking at computing the probability of correctness. Notice that the output will be correct is no errors occur, and this happens with probability $(1-\epsilon)^{2^k}$. If there is an error, then the last error either occurs before step $2^k - h$, in which case Theorem 4.5.10 indicates the output will be composite with probability 1, or the last error occurs at or after step $2^k - h$, in which case Theorem 4.5.10 indicates the output will be composite with probability $1 - \frac{2^{m(2^k-j)}}{F_k}$. We need to sum these two values over the range $1 \ldots 2^k - h - 1$ and $2^k - h \ldots 2^k$ respectively. Combining these three possibilities together, we get

$$p_c(F_k) = (1-\epsilon)^{2^k} + \sum_{i=1}^{2^k-h-1} 1 \cdot (1-\epsilon)^{2^k-i}\epsilon + \sum_{i=2^k-h}^{2^k} (1-\epsilon)^{2^k-i}\epsilon\left(1 - \frac{2^{m(2^k-i)}}{F_k}\right)$$

which we simplify as

$$
\begin{aligned}
p_c(F_k) &= (1-\epsilon)^{2^k} + \sum_{i=1}^{2^k-h-1} 1\cdot(1-\epsilon)^{2^k-i}\epsilon + \sum_{i=2^k-h}^{2^k}(1-\epsilon)^{2^k-i}\epsilon\left(1-\frac{2^{m(2^k-i)}}{F_k}\right)\\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k}\sum_{i=1}^{2^k}(1-\epsilon)^{-i} - \epsilon(1-\epsilon)^{2^k}\sum_{i=2^k-h}^{2^k}(1-\epsilon)^{-i}\frac{2^{m(2^k-i)}}{F_k}\\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k}\left(\frac{\left(\frac{1}{(1-\epsilon)}\right)^{2^k+1} - \frac{1}{(1-\epsilon)}}{\frac{1}{(1-\epsilon)}-1}\right)\\
&\quad -\epsilon(1-\epsilon)^{2^k}\left(\frac{2^{m2^k}}{F_k}\right)\sum_{i=2^k-h}^{2^k}\left(\frac{1}{(1-\epsilon)2^m}\right)^i\\
&= (1-\epsilon)^{2^k} + \epsilon(1-\epsilon)^{2^k}\left(\frac{1}{1-\epsilon}\right)\left(\frac{1-\epsilon}{\epsilon}\right)\left(\frac{1}{(1-\epsilon)^{2^k}}-1\right)\\
&\quad -\epsilon(1-\epsilon)^{2^k}\left(\frac{2^{m2^k}}{F_k}\right)\left(\frac{(\frac{1}{(1-\epsilon)2^m})^{2^k+1} - (\frac{1}{(1-\epsilon)2^m})^{2^k-h}}{\frac{1}{(1-\epsilon)2^m}-1}\right)\\
&= (1-\epsilon)^{2^k} + (1-\epsilon)^{2^k}\left(\frac{1}{(1-\epsilon)^{2^k}}-1\right)\\
&\quad -\epsilon(1-\epsilon)^{2^k}\left(\frac{2^{m2^k}}{F_k}\right)\left((\frac{1}{(1-\epsilon)2^m})^{2^k+1} - (\frac{1}{(1-\epsilon)2^m})^{2^k-h}\right)\left(\frac{(1-\epsilon)2^m}{1-(1-\epsilon)2^m}\right)\\
&= 1-\epsilon(1-\epsilon)^{2^k}\left(\frac{2^{m2^k}}{F_k}\right)(\frac{1}{(1-\epsilon)2^m})^{2^k-h}\left((\frac{1}{(1-\epsilon)2^m})^{h+1}-1\right)\left(\frac{(1-\epsilon)2^m}{1-(1-\epsilon)2^m}\right)\\
&= 1-\epsilon(1-\epsilon)^{h+1}\left(\frac{2^{m(h+1)}}{F_k}\right)\left((\frac{1}{(1-\epsilon)2^m})^{h+1}-1\right)\left(\frac{1}{1-(1-\epsilon)2^m}\right)\\
&= 1-\left(\frac{\epsilon}{F_k}\right)\left(\frac{1-((1-\epsilon)2^m)^{h+1}}{1-(1-\epsilon)2^m}\right).
\end{aligned}
$$

Since there are no infeasible answers $p_i(F_k) = 0$ and $p_f(F_k) = 1 - p_c(F_k)$, which completes the proof. ■

We have to show robustness for this case. However, as in earlier examples, we notice that since $p_i(F_k) = 0$ implies this algorithm is not robust, and we must combine two independent executions of this algorithm to satisfy the conditions for robustness.

We are ready to prove the second main result on robustness for Pepin's test. That is, we prove the analogue of Theorem 4.4.7 for the composite case.

**Theorem 4.5.12** *Consider Pepin's algorithm (Algorithm 4.1). If the input $F_k$ given to the algorithm is composite, and if the probability of an error occurring is $\epsilon < \frac{1}{2^k}$, the combination of two independent trials of Pepin's algorithm are robust.*

**Proof:** We must show two things: $\frac{p_i^*(F_k)}{p_i^*(F_k)+p_f^*(F_k)} > 0.5$ and $p_c^*(F_k) > b > 0$ for some constant $b$, where $p_c^*(F_k)$, $p_f^*(F_k)$ and $p_i^*(F_k)$ represent the probabilities of correctness, incorrect but feasible and infeasible under this combined execution model.

First, we will show that $p_c^*(F_k)$ is bounded below by a constant larger than 0.

We provide a very loose, but usable, lower bound on $p_c^*(F_k)$. Notice that if there is no error that occurs in either independent trial, the output will be correct. The probability of this occurring is $((1-\epsilon)^{2^k})^2$, which is a lower bound on $p_c^*(F_k)$. However, if $\epsilon < \frac{1}{2^k}$ then

$$p_c^*(F_k) \geq ((1-\epsilon)^{2^k})^2 \geq e^{-2} > 0.$$

We will rearrange the inequality $\frac{p_i^*(F_k)}{p_i^*(F_k)+p_f^*(F_k)} > 0.5$ to be $p_f^*(F_k) < p_i^*(F_k)$.

Notice that if the independent executions of the yield different results (i.e., one indicates the input was prime, while the other indicates the input was composite), that will be an infeasible result. Thus, $p_i^*(F_k) \geq 2p_c(F_k)p_f(F_k)$, since the right-hand side of this inequality is exactly the probability that the two independent executions yield different outcomes.

Since the only feasible output is when the algorithm outputs "prime" on composite input, both independent executions must yield prime. Thus, $p_f^*(F_k) = (p_f(F_k))^2$.

Thus, to show $p_f^*(F_k) < p_i^*(F_k)$, we only need to show $(p_f(F_k))^2 < 2p_c(F_k)p_f(F_k)$, which is equivalent to showing $p_f(F_k) < 2p_c(F_k)$. However, since $p_f(F_k) = 1 - p_c(F_k)$, we can simplify $p_f(F_k) < 2p_c(F_k)$ to $p_f(F_k) < 2(1 - p_f(F_k))$, which is equivalent to showing $p_f(F_k) < \frac{2}{3}$.

We have

$$\begin{aligned} p_f(F_k) &= \frac{\epsilon}{F_k}\frac{(1-\epsilon)^{h+1}2^{m(h+1)}-1}{(1-\epsilon)2^m-1} \\ &< \frac{\epsilon}{F_k}\frac{(1-\epsilon)^{h+1}2^{m(h+1)}}{(1-\epsilon)2^m-1} \\ &= \epsilon(1-\epsilon)^{h+1}\frac{1}{(1-\epsilon)2^m-1}\frac{2^{m(h+1)}}{F_k} \\ &< \frac{1}{2^k}\cdot 1\cdot 1\frac{2^{m(h+1)}}{F_k} \\ &\leq \frac{1}{2^k} \\ &< \frac{2}{3}, \end{aligned}$$

since the number of elements in the $2^m$-ary tree of height $h$ that has root $-1$ in $G_{F_k,0}$ is bounded above by $2^{m(h+1)}$, which is in turn, bounded above by $F_k$, since the number of elements in $G_{F_k,0}$ is $F_k$. ■

We now turn attention back towards some empirical results on Fermat primes, and how they apply to the results outlined in this chapter.

Notice that Theorem 4.5.10 relies on the factorization of $F_k$. This reliance seems counter-intuitive: knowing the factorization of $F_k$ helps us determine the robustness of the algorithm to determine if $F_k$ is prime. It may be more beneficial if robustness arguments could be made relying only on $k$, rather than the factorization of $F_k$. Moreover, we would like to bound the parameters $m$ (the number of factors of $F_k$) and $h$ ($= \min\{\nu_2(p_i-1)-1 : 1 \leq i \leq m\}$, the height of the complete $2^m$-ary tree) in order to give a reasonable and practical probability result.

In order to make such arguments, we recall the work of Keller [48]. It is well-known that if $F_k$ is composite, it has a factor $c \cdot 2^n + 1$, where $n \geq k+2$, $c$ odd. Heuristically, it appears as though the difference $n-k$ is an exponentially distributed function: see Table 4.1 for current empirical results derived from Keller [49].

| $n-k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Frequency | 117 | 60 | 31 | 5 | 7 | 5 | 4 | 2 | 3 |
| Freq/Total | 0.500 | 0.256 | 0.132 | 0.0214 | 0.030 | 0.021 | 0.017 | 0.001 | 0.012 |

Table 4.1: Values of $n-k$ where $c \cdot 2^n + 1 \,|\, F_k$ for 234 composite Fermat numbers

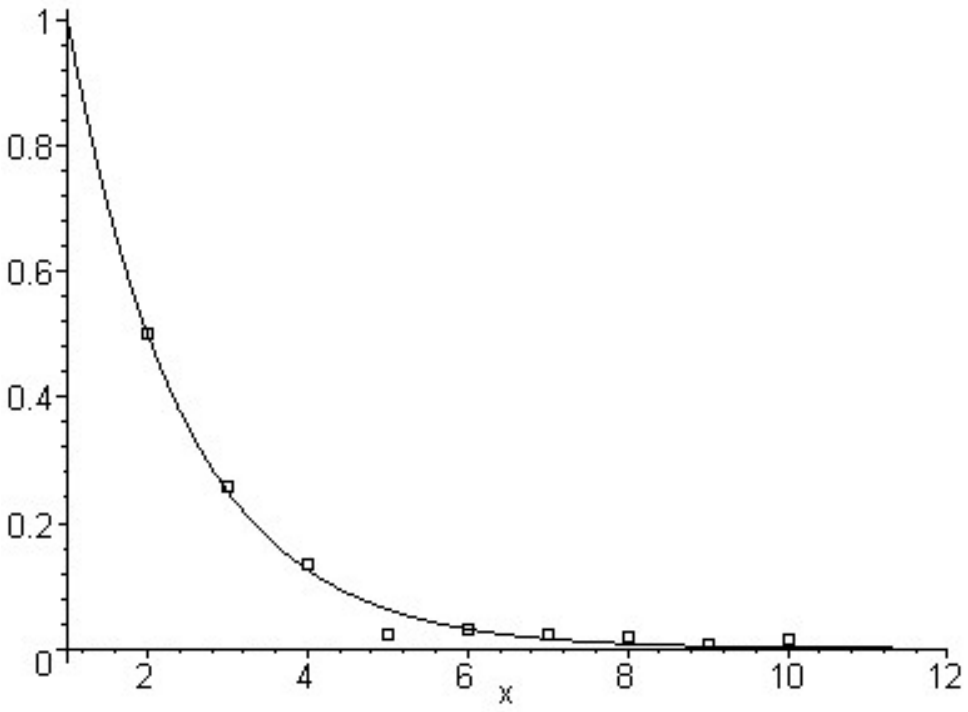


Figure 4.5: Plot of $f(x) = 2^{-(x-1)}$ and Freq/Total values from Table 4.1

As further evidence that $n - k$ is exponentially distributed, we can graph these empirical points against an exponential curve $f(x) = 2^{-(x-1)}$ as in Figure 4.5.

These empirical results yield the following conjecture.

**Conjecture 4.5.13** *Suppose $F_k$ is composite for some $k$ in the range $[1..N]$. Consider a prime divisor of $F_k$: it must have the form $c \cdot 2^n + 1$ where $n \geq k + 2$ and $c$ is odd. As $N \to \infty$, the fraction of all the differences $n - k$ that equal $l$ is about*

$$2^{-(l-1)}.$$

We use Conjecture 4.5.13 to determine the expected value of $h$ (the height of the $2^m$-ary complete tree in $G_{k,0}$).

**Theorem 4.5.14** *Assume Conjecture 4.5.13 holds. Consider a composite $F_k$ in the range $[1..N]$. As $N \to \infty$, the expected value of $h = \min\{\nu_2(p_i - 1) - 1 : 1 \leq i \leq m\}$ is*

$$k + 1 + \frac{1}{2^m - 1}.$$

**Proof:** Since $F_k$ is composite, we can write

$$F_k = \Pi_{i=1}^{m}(c_i \cdot 2^{k+2+l_i} + 1)$$

for some $l_i \geq 0$ and odd $c_i$, $1 \leq i \leq m$. Recall, by the definition of $h$, we have

$$\begin{aligned} h &= \min\{\nu_2(c_i \cdot 2^{k+2+l_i}) - 1 : 1 \leq i \leq m\} \\ &= \min\{k + 1 + l_i : 1 \leq i \leq m\} \\ &= k + 1 + \min\{l_i : 1 \leq i \leq m\}. \end{aligned}$$

Using the definition of expected value ($E(X) = \sum_{X=i} i \cdot \mathbf{P}(i)$), we have

$$E(h) = k + 1 + \sum_{j \geq 0} j \cdot \mathbf{P}(\min\{l_i : 1 \leq i \leq m\} = j).$$

Our task now is to evaluate $\mathbf{P}(\min\{l_i : 1 \leq i \leq m\} = j)$. Again, using basic probability theory, we know

$$\begin{aligned} \mathbf{P}(\min\{l_i : 1 \leq i \leq m\} = j) &= \mathbf{P}(l_i \geq j \text{ for all } i) \cdot \mathbf{P}(l_i = j \text{ for some } i | l_i \geq j \text{ for all } i) \\ &= \mathbf{P}(l_i \geq j \text{ for all } i) \cdot \mathbf{P}(l_i = 0 \text{ for some } i) \\ &= \mathbf{P}(l_i \geq j \text{ for all } i) \cdot (1 - \mathbf{P}(l_i > 0 \text{ for all } i)) \\ &= \mathbf{P}(l_i \geq j \text{ for all } i) \cdot (1 - \frac{1}{2^m}), \end{aligned}$$

since we know the distribution of these $l_i$ is governed by $\frac{1}{2^{(l_i+1)}}$ from Conjecture 4.5.13, where we have shifted the distribution curve 2 units to the left. Continuing, we have

$$\begin{aligned} \mathbf{P}(\min\{l_i : 1 \leq i \leq m\} = j) &= \mathbf{P}(l_i \geq j \text{ for all } i) \cdot (1 - \frac{1}{2^m}) \\ &= (\sum_{t=0}^{\infty} \frac{1}{2^{j+t+1}})^m \cdot (\frac{2^m - 1}{2^m}) \\ &= (\frac{1}{2^j})^m \cdot (\frac{2^m - 1}{2^m}) \\ &= (2^m - 1) \cdot (\frac{1}{2^m})^{j+1}. \end{aligned}$$

We now return to determining the expected value for $h$. Consider

$$\begin{aligned} E(h) &= k+1+\sum_{j\geq 0} j\cdot \mathbf{P}(\min\{l_i : 1\leq i\leq m\}=j) \\ &= k+1+(2^m-1)\sum_{j\geq 0} j\cdot(\frac{1}{2^m})^{j+1}. \end{aligned}$$

We know

$$\sum_{j\geq 0} x^j = \frac{1}{1-x},$$

which, after differentiating once and multiplying by $x^2$, yields

$$\sum_{j\geq 0} jx^{j+1} = \frac{x^2}{(1-x)^2} = \frac{1}{(\frac{1}{x}-1)^2}.$$

Letting $x=\frac{1}{2^m}$ we have

$$\begin{aligned} E(h) &= k+1+(2^m-1)\sum_{j\geq 0} j\cdot(\frac{1}{2^m})^{j+1} \\ &= k+1+(2^m-1)\cdot\frac{1}{(2^m-1)^2} \\ &= k+1+\frac{1}{2^m-1}. \end{aligned}$$

■

## 4.6 The iteration $x \to x^2$ on Mersenne primes

As a bridge between this chapter and Chapter 5, we analyze the iteration $x\to x^2$ on Mersenne primes, which are primes of the form $p=2^q-1$. We provide a complete characterization in the following theorem.

**Theorem 4.6.1** *When $p=2^q-1$, a Mersenne prime, the digraph $G_{p,0}$ consists of cycles whose length divides $q-1$. Off each element in these cycles there hangs a single element with tail length 1.*

**Proof:** We have $p-1=2(2^{q-1}-1)$, so $\tau=1$ and $\rho=2^{q-1}-1$. It follows that the divisors of $p-1$ are of the form $2^f j$, where $j\mid 2^{q-1}-1$ and $f\in\{0,1\}$. The cycle

length for any element is therefore given by $\text{ord}_j\ 2$ for some $j$ a divisor of $2^{q-1}-1$. Now $\text{ord}_j\ 2 \,|\, q-1$, and so the cycle length for every element is a divisor of $q-1$ which is approximately $\log_2 p$. ■

The result of Theorem 4.6.1 can be contrasted with the average cycle length of approximately $\sqrt{p}$ in the case of a random map [44].

Figure 4.2 illustrates Theorem 4.6.1 in the case where $q=5$, $p=31$. Specifically, the cycles lengths are 4, 2, 1 and the tails all have length 1.

Further work on Mersenne primes under $x \to x^2$ was developed in an internet forum (see [83]).

## 4.7 Statistical measures of $x \to x^2$

We will now consider some statistics about the tail and cycle lengths for a given prime $p$ over the iteration $x \to x^2$.

Recall the statistical measures from Section 2.4.

For example, $TC(29,0)=3$, $T_0(29,0)=7$, $AC(29,0)=7/3$, $C(29,0)=19/7$, and $T(29,0)=5/4$. The following theorem, which is one of the main results of this chapter, gives formulas for these quantities.

**Theorem 4.7.1** *Let $p-1=2^\tau \cdot \rho$ with $\rho$ odd. With respect to the iteration $x \to x^2 \pmod p$ we have*

*(a)* $TC(p,0)=\sum_{d\,|\,\rho} \frac{\varphi(d)}{\text{ord}_d\ 2}$;

*(b)* $T_0(p,0)=\rho$;

*(c)* $AC(p,0)=\frac{\rho}{TC(p,0)}$;

*(d)* $C(p,0)=\frac{1}{\rho}\sum_{d\,|\,\rho}\varphi(d)\text{ord}_d\ 2$;

*(e)* $T(p,0)=\frac{1}{p-1}\sum_{d\,|\,p-1}\varphi(d)\nu_2(d)=\tau-1+2^{-\tau}$.

**Proof:** Parts (a)-(d) follow directly from Corollary 4.4.3. For part (e) we have

$$\begin{aligned}
T(p,0) &= \frac{1}{p-1}\sum_{1\le a\le p-1} t_p(a) \\
&= \frac{1}{p-1}\sum_{d\,|\,p-1} \varphi(d)\nu_2(d) \\
&= \frac{1}{p-1}\sum_{d\,|\,\rho}\sum_{0\le i\le\tau} \varphi(d\cdot 2^i)\nu_2(d\cdot 2^i) \\
&= \frac{1}{p-1}\sum_{d\,|\,\rho}\varphi(d)\sum_{1\le i\le\tau} \varphi(2^i)\cdot i \\
&= \frac{1}{p-1}\sum_{d\,|\,\rho}\varphi(d)\sum_{1\le i\le\tau} i\cdot 2^{i-1} \\
&= \frac{1}{p-1}\sum_{d\,|\,\rho}\varphi(d)((\tau-1)2^\tau+1) \\
&= \frac{1}{p-1}\rho((\tau-1)2^\tau+1) \\
&= \tau-1+2^{-\tau}.
\end{aligned}$$

■

We now examine the average behavior of some of these quantities over all odd primes $p \le M$, which we capture in two main theorems.

We can obtain good asymptotic estimates for these quantities, assuming the Extended Riemann Hypothesis (ERH). We will need the following lemmas.

**Lemma 4.7.2** *Assume the ERH. Then if the logarithmic integral* $\mathrm{li}(x)$ *is defined by* $\mathrm{li}(x) = \int_2^x \frac{1}{\ln t}dt$ *and if* $\gcd(k,l) = 1$ *then*

$$\pi(x;l,k) = \frac{\mathrm{li}(x)}{\varphi(l)} + O(\sqrt{x}(\ln x + 2\ln l)).$$

**Proof:** See, for example, [5, pp. 217, 235]. ■

It should be noted that without the assumption of the ERH, we would not have such a tight bound on the $O$ term. Specifically, without the ERH, we would

have (using results from [5, p. 215]) that there is a constant $c > 0$ such that if $\gcd(k, l) = 1$ then

$$\pi(x; l, k) = \frac{\mathrm{li}(x)}{\varphi(l)} + O(xe^{-c(\ln x)^{3/5}(\ln\ln x)^{-1/5}}).$$

This bound is not strong enough for our purposes. Therefore, we assume the ERH and use the tighter bound in our analysis.

**Lemma 4.7.3** *Assume the ERH. Let $k, l$ be integers with $\gcd(k, l) = 1$. Then*

$$\sum_{\substack{p \le x \\ p \equiv k \pmod{l}}} p = \frac{1}{\varphi(l)} \left( \frac{x^2}{2 \ln x} \right) \left( 1 + O(\frac{1}{\ln x}) \right) + O(x^{3/2}(\ln x + 2 \ln l)).$$

**Proof:** By Lemma 4.7.2 we have

$$\pi(x; l, k) = \frac{\mathrm{li}(x)}{\varphi(l)} + O(\sqrt{x}(\ln x + 2 \ln l)).$$

Now, by Stieltjes integration (see, e.g., [5, pp. 28-29]), we have

$$\sum_{\substack{p \le x \\ p \equiv k \pmod{l}}} p = \frac{1}{\varphi(l)} \int_2^x \frac{t}{\ln t} dt + O(x^{3/2}(\ln x + 2 \ln l)). \tag{4.1}$$

On the other hand, by asymptotic integration (see, e.g., [5, pp. 27-28]), we have

$$\int_2^x \frac{t}{\ln t} dt = \frac{x^2}{2 \ln x} + O\left( \frac{x^2}{(\ln x)^2} \right). \tag{4.2}$$

The result comes from combining Eqs. (4.1) and (4.2). ∎

Now we are ready to estimate $ST_0(M)$.

**Theorem 4.7.4** *Assume the ERH. Then $ST_0(M) \sim \frac{M^2}{6 \ln M}$.*

**Proof:** We have, using Lemma 4.7.3, that

$$\begin{aligned}\sum_{p\le M}\frac{p-1}{2^{\nu_2(p-1)}} &= \sum_{1\le i\le \log_2 M}\ \sum_{\substack{p\le M\\ p\equiv 2^i+1 \pmod{2^{i+1}}}}\frac{p-1}{2^i}\\ &= \frac{M^2}{2\ln M}(1+O(\frac{1}{\ln M}))\sum_{1\le i\le \log_2 M}\frac{1}{4^i}\\ &= \frac{M^2}{2\ln M}(1+O(\frac{1}{\ln M}))\frac{1}{3}(1+O(\frac{1}{M})).\end{aligned}$$

■

We now turn to $ST(M)$.

**Theorem 4.7.5** *Assume the ERH. Then*

$$ST(M)\sim\frac{2}{3}\cdot\frac{M^2}{\ln M}.$$

**Proof:** We have

$$\begin{aligned}\sum_{p\le M}\sum_{1\le x\le p-1}t_p(x) &= \sum_{p\le M}(p-1)(\nu_2(p-1)-1+2^{-\nu_2(p-1)})\\ &= \sum_{p\le M}\nu_2(p-1)p-\sum_{p\le M}\nu_2(p-1)-\sum_{p\le M}p+\sum_{p\le M}1+\sum_{p\le M}\frac{p-1}{2^{\nu_2(p-1)}}.\end{aligned}$$

We start by evaluating $\sum_{p\le M}\nu_2(p-1)p$. We have

$$\begin{aligned}\sum_{p\le M}\nu_2(p-1)p &= \sum_{1\le i\le \log_2 M}\ \sum_{\substack{p\le M\\ p\equiv 1 \pmod{2^i}}}p\\ &= \frac{M^2}{\ln M}\left(1+O(\frac{1}{\ln M})\right)\left(1+O(\frac{1}{M})\right),\end{aligned}$$

where we have used Lemma 4.7.3.

Next we have, using Lemma 4.7.2, that

$$\begin{aligned}\sum_{p\le M}\nu_2(p-1) &= \sum_{1\le i\le \log_2 M}\pi(M;2^i,1)\\ &= \sum_{1\le i\le \log_2 M}\left(\frac{\mathrm{li}(M)}{2^{i-1}}+O(\sqrt{M}\ln M)\right)\\ &= \mathrm{li}(M)(2+O(\frac{1}{M}))+O(\sqrt{M}(\ln M)^2),\end{aligned}$$

It is well known that $\sum_{p \le M} p \sim \frac{M^2}{2 \ln M}$; see, for example, [5, p. 28-29].

By the prime number theorem, $\sum_{p \le M} 1 \sim \frac{M}{\ln M}$.

Putting all these estimates together with Theorem 4.7.4, and the well-known estimate $\mathrm{li}(x) = \frac{x}{\ln x}(1 + O(\frac{1}{\ln x}))$, we obtain the desired result. ∎

It is worth noting that Chou and Shparlinski [24] have removed the ERH condition on Theorems 4.7.4 and 4.7.5. This improvement occurred after the publication of Theorems 4.7.4 and 4.7.5 in Vasiga and Shallit [94]. Specifically, Chou and Shparlinski refine Lemma 4.7.3 using three mathematical tools. The first tool is the the *Page Bound* (as outlined in Davenport [28]) which states that

$$\pi(x; k, a) = \frac{x}{\varphi(k) \ln x} + O(x \ln^{-2} x),$$

for all integers $k \le \ln^{\frac{3}{2}} x$ and $\gcd(a, k) = 1$.

The second tool used by Chou and Shparlinski is the use of $\mathscr{I}$-units. Given a set $\mathscr{I} = \{p_1, p_2, ..., p_n\}$, we say that integer $q$ is an $\mathscr{I}$-unit if all prime divisors of $q$ belong in $\mathscr{I}$.

Finally, the previous two tools are combined together using inclusion-exclusion to obtain an ERH-free bound.

We now compare the estimates in Theorem 4.7.4 and 4.7.5 with empirical data:

| $M$ | $ST_0(M)$ | $M^2/(6 \ln M)$ | $ST(M)$ | $2M^2/(3 \ln M)$ |
|---|---|---|---|---|
| 10 | 5 | 7.24 | 9 | 28.95 |
| $10^2$ | 342 | 361.91 | 1366 | 1447.65 |
| $10^3$ | 25875 | 24127.47 | 99383 | 96509.88 |
| $10^4$ | 1922532 | 1809560.34 | 7481452 | 7238241.36 |
| $10^5$ | 151468221 | 144764827.30 | 605859857 | 579059309.20 |
| $10^6$ | 12531875547 | 12063735608.42 | 49994218943 | 48254942433.69 |

Table 2: Comparing $ST_0(M)$ and $ST(M)$ to asymptotic estimates

It is harder to estimate the average behavior of $c(x)$. A reasonable conjecture is that there are infinitely many primes $p$ such that (a) $p' := (p-1)/2$ is also prime and (b) 2 is a primitive root (mod $p'$). The first few such primes are

$$7, 11, 23, 59, 107, 167, 263, 347, 359, 587, 839, 887, 983, 1019, 1307, 1319, 2039, 2459,$$

$$2903, 2999, 3467, 3803, 3863, 3947, 4139, 4283, 4679, \cdots$$

For these primes $p$ we have

$$\sum_{1\le x<p} c(x) = 2 \sum_{d\,|\,(p-1)/2} \varphi(d)\mathrm{ord}_d\ 2 = 2(1+(p'-1)(p'-1)) = \Omega(p^2).$$

If $p$ is a Fermat prime, then $p-1 = 2^{2^k}$ for some $k$. Using Theorem 4.7.1, we have $\rho = 1$ and so

$$\sum_{1\le x<p} c(x) = \frac{p-1}{\rho} \sum_{d\,|\,\rho} \varphi(d)\mathrm{ord}_d\ 2 = 2^{2^k} = p-1.$$

However, few believe there are infinitely many Fermat primes.

If $p$ is a Mersenne prime, say $p = 2^q - 1$, then

$$\sum_{1\le x<p} c(x) \le (q-1)(2^{q-1}-1) = O(p\ln p).$$

Most people believe there are infinitely many Mersenne primes, but of course no proof currently exists.

Assuming a conjecture of Wagstaff [98] on the distribution of the least prime in an arithmetic progression, we now show there are infinitely many primes $p$ for which

$$\sum_{1\le x<p} c(x) = O(p(\ln p)^2).$$

To observe this, for each integer $\tau \ge 1$ consider the least prime $p$ with $p \equiv 1 \pmod{2^\tau}$. Now write

$$p - 1 = 2^{\tau+c}\cdot\rho \tag{4.3}$$

for some non-negative integer $c$ and odd integer $\rho$. Then $\varphi(p) = p-1 = 2^{\tau+c}\cdot\rho$. Wagstaff's conjecture states that the least prime $p \equiv x \pmod n$, when $\gcd(x,n) = 1$, is $O(\varphi(n)(\ln n)(\ln\varphi(n)))$. Letting $n = 2^\tau$, we find

$$\begin{aligned} p &= O(\varphi(2^\tau)(\ln 2^\tau)(\ln\varphi(2^\tau))) \\ &= O(2^{\tau-1}\tau(\ln 2)(\tau-1)(\ln 2)) \\ &= O(\tau^2 2^\tau). \end{aligned}$$

Dividing this last result by (4.3), we get $\rho = O(\tau^2)$. Also, $p = O(\tau^2 2^\tau)$ gives $\tau = \Theta(\ln p)$.

Using Theorem 4.7.1, we have

$$\begin{aligned}
\sum_{1 \le x < p} c(x) &= \frac{p-1}{\rho} \sum_{d \,|\, \rho} \varphi(d) \mathrm{ord}_d\ 2 \\
&= 2^{\tau+c} \sum_{d \,|\, \rho} \varphi(d) \mathrm{ord}_d\ 2 \\
&\le 2^{\tau+c} \cdot \rho \sum_{d \,|\, \rho} \varphi(d) \\
&= 2^{\tau+c} \cdot \rho^2 \\
&= O(\rho p).
\end{aligned}$$

Combining this result with the previous fact that $\rho = O(\tau^2) = O((\ln p)^2)$, we have

$$\sum_{1 \le x < p} c(x) = O(p(\ln p)^2),$$

as desired.

## 4.8 Conclusion

We have shown that two independent trials of Algorithm 4.1 are sufficient to meet the definition of robustness. In proving this result, we have analyzed the digraph formed by $x \to x^2$ obtaining interesting statistical measures of cycle length, tail length, number of cycles, etc., for this digraph.

Next, we apply this form of analysis to the Lucas-Lehmer test for primality, which will require slightly more sophisticated mathematical machinery in order to obtain bounds on robustness.

# Chapter 5

# Robustness of the Lucas-Lehmer primality test

## 5.1 Introduction

Rogers [84] stated,

> "The family of nonlinear maps given by $f(x) = x^2 + c,\ c \in \mathbb{F}_p$, for nonzero values of the parameter $c \in \mathbb{F}_p$, produces graphs whose tree structure (graphically, the transients leading down to the cycles) seems beyond description; in general the trees attached to the cycles are of variable height, and even those trees attached to the same cycle vary."

However, as we will see in this chapter, Rogers' statement is not true for the case $c = -2$, as recognized by Pollard [77]. In describing the tree structure of the iteration $x \to x^2 - 2$, and thus graphically demonstrating Pollard's caution, we will also prove robustness results for the Lucas-Lehmer primality test. The majority of the work in this chapter was published in Vasiga and Shallit [94].

## 5.2 Lucas-Lehmer primality test

The Lucas-Lehmer test determines the primality of numbers of the form $2^d - 1$. (See the original paper by Lucas [62] or the generalization by Lehmer, for example, in

Bach and Shallit [5].) Specifically, the Lucas-Lehmer primality test takes as input an integer $N = 2^d - 1, d > 2$, $d$ prime, and computes the sequence

$$\begin{aligned} S_1 &= 4, \\ S_k &= (S_{k-1}^2 - 2) \bmod N \qquad (1 < k < d). \end{aligned}$$

Then $N$ is prime iff $S_{d-1} = 0$. The key concept that will be used in this chapter is the recurrence $S_k = S_{k-1}^2 - 2 \pmod{n}$. We can consider the Lucas-Lehmer test can be expressed in pseudocode in the following manner:

```
x ← 4
for i from 1 to d − 1 do
   x ← x^2 − 2 (mod N)
end for
if x = 0 then
   print "N is prime"
else
   print "N is composite"
end if
```

Algorithm 5.1: Lucas-Lehmer Primality Algorithm

Although the algebraic structure on which Lucas-Lehmer is based is well-known, our observation about the particular elements contained in the binary tree seems to be new.

For the remainder of this chapter, let $h(x) = x^2 - 2$.

## 5.3 Previous work

It is worth noting that Dickson polynomials (see Lidl, Mullen and Turnwald [58]) can be used to describe this iteration. In particular, Dickson polynomials (of the first kind) can be defined recursively as follows:

$$\begin{aligned} D_0(x, a) &= 2, \\ D_1(x, a) &= x, \\ D_n(x, a) &= xD_{n-1}(x, a) - aD_{n-2}(x, a), \qquad \text{for } n \geq 2. \end{aligned}$$

where $x$ is an indeterminate and $a$ is an element from a commutative ring. From this, one can derive that $h^n(x) = D_{2^n}(x, 1)$. It is worth noting that Morton [67],

Vivaldi and Hatjispyros [95] and Batra and Morton [7, 8] (as well as many others) examined the more general context of periods of maps described by irreducible polynomials over finite fields. Dickson polynomials with $a = 1$ have been studied to some depth [70], but, as Lidl, Mullen and Turnwald [58, p. 90] point out,

> The computations and arguments for determining the fixed point formulas for the cases $a = 1$ and $a = -1$ are quite detailed and lengthy (some twenty pages for each case)...

Our techniques can be used to obtain these results for the case of prime moduli. Furthermore, we obtain much more detailed results (e.g., Theorem 5.4.5 and Corollary 5.4.6).

More recently, Peinado, Montoya, Munõz and Yuste [72] have proven upper bounds on the cycle lengths for $x \to x^2 + c$ over $\mathbb{F}_q$, where $q$ is a prime power.

Additionally, Gilbert, Kolesar, Reiter, and Storey [38] obtained similar results, but in an ad hoc manner. One of our contributions is a general algebraic framework for understanding the iteration $x \to x^2 - 2$, which shows that it is quite analogous to the (well-understood) map $x \to x^2$.

As well, the study of this iteration leads to interesting open problems (see [85, Exercise 40S]).

### 5.3.1 Viewing the iterations as a digraph

To build intuition, let us look at some examples. For the map $x \to x^2 - 2 \pmod N$, we illustrate four examples: one for Fermat primes (i.e., primes of the form $2^{2^k} + 1$), one for Mersenne primes (i.e., primes of the form $2^p - 1$, $p$ prime), and two examples of primes that are not of this form ($p = 29$ and $p = 191$). These are shown in Figures 5.1–5.4 respectively.

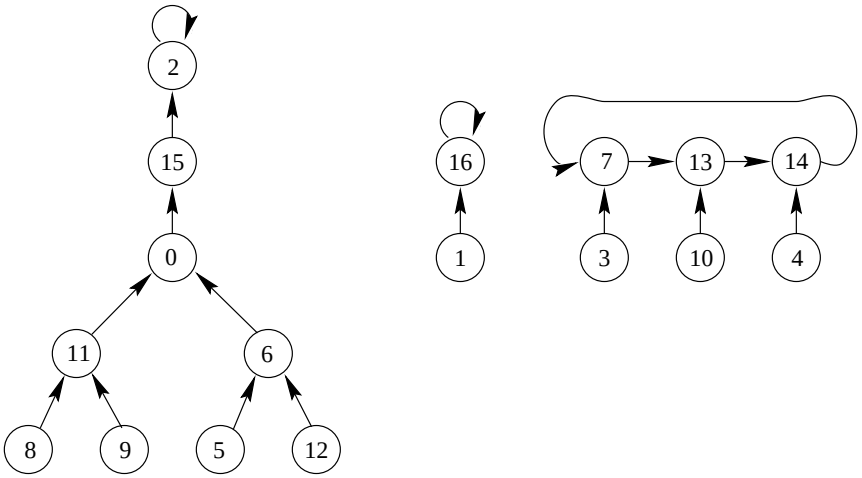


Figure 5.1: The topology of $G_{17,2}$

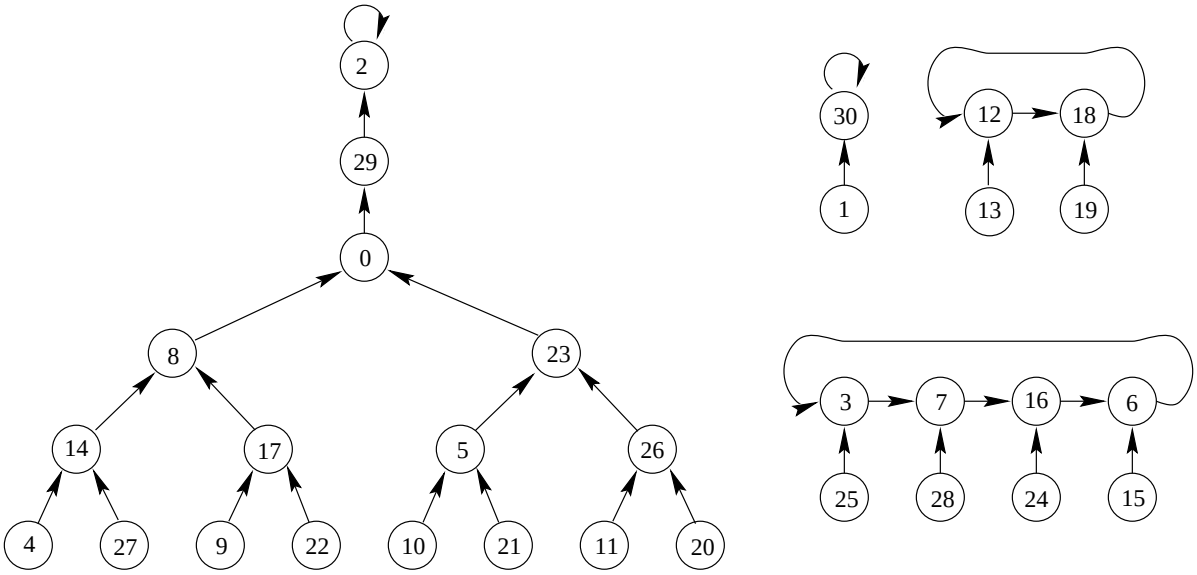


Figure 5.2: The topology of $G_{31,2}$

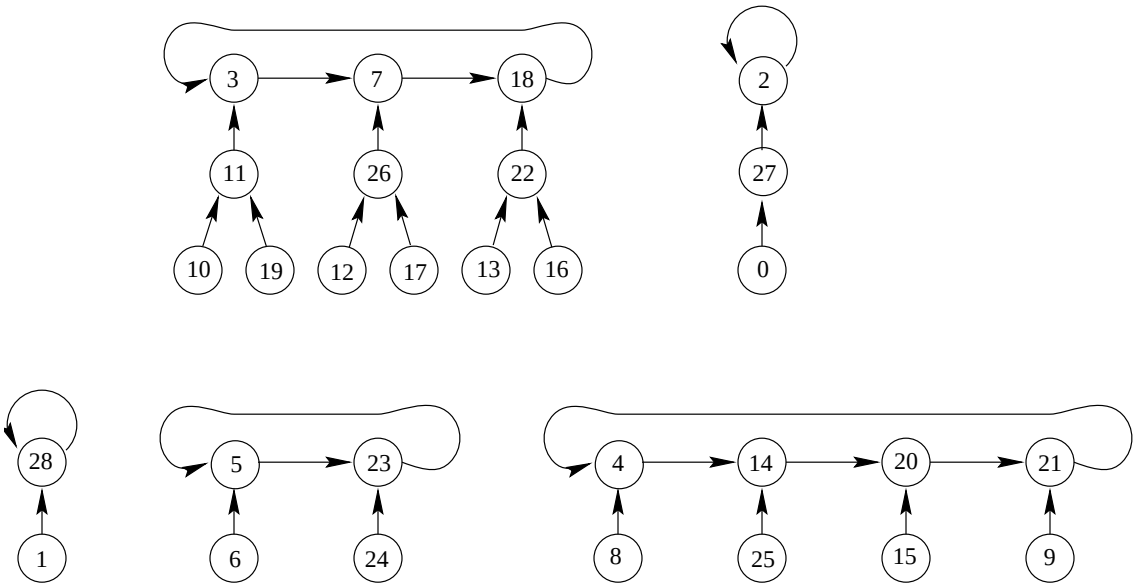


Figure 5.3: The topology of $G_{29,2}$

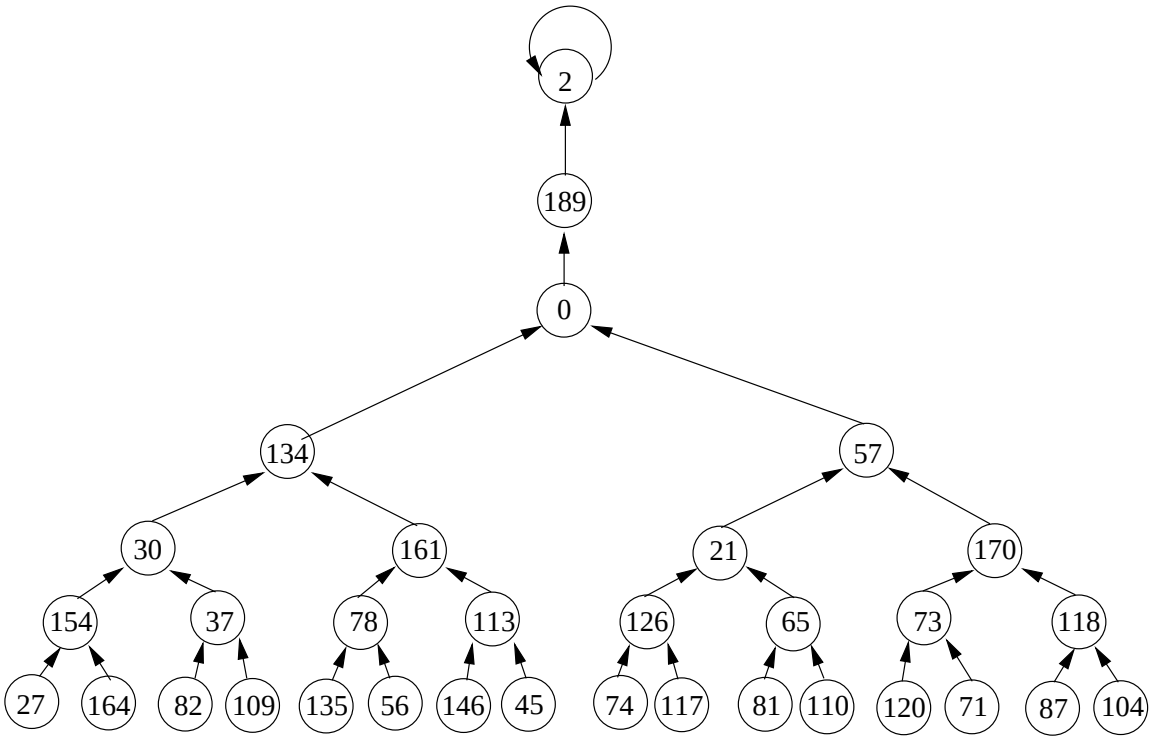


Figure 5.4: A component of the digraph $G_{191,2}$

## 5.4 General topology results for $x \to x^2 - 2$

In this section, we outline the general topology results for the iterated function $x \to x^2 - 2 \pmod p$, which will be used in proving robustness results for the Lucas-Lehmer primality test.

Given $a \in GF(p)$, let us define the polynomial $u(X) = X^2 - aX + 1$. Let $\alpha$ and $\beta$ be the roots of $u$ in $GF(p^2)$. Note that $\alpha + \beta = a$ and $\alpha\beta = 1$.

**Proposition 5.4.1** *We have $h^n(a) = \alpha^{2^n} + \beta^{2^n}$ for $n \geq 0$.*

**Proof:** By induction on $n$. For $n = 0$ we have $h^0(a) = a = \alpha + \beta$. Now assume the result is true for $n$; we prove it for $n + 1$. We have

$$\alpha^{2^{n+1}} + \beta^{2^{n+1}} = (\alpha^{2^n} + \beta^{2^n})^2 - 2\alpha^{2^n}\beta^{2^n} = h^n(a)^2 - 2.$$

■

We will also require the following small lemma.

**Lemma 5.4.2** *Suppose $F$ is a field and $a, b \in F - \{0\}$ with $a + a^{-1} = b + b^{-1}$. Then $a = b$ or $a = b^{-1}$.*

**Proof:** Notice that we can multiply both sides of the equation $a + a^{-1} = b + b^{-1}$ by $a$ to get

$$a^2 + 1 = ab + ab^{-1}.$$

Rearranging this equation we have

$$(a - b^{-1})(a - b) = 0,$$

and therefore, either $a = b$ or $a = b^{-1}$. ■

**Theorem 5.4.3** *Let $a \in GF(p)$, and suppose that iterating $h$, starting with $a$, results in a tail of length $t = t(a)$ and a cycle of length $c = c(a)$. Then $t$ and $c$ can be computed as follows. Let $\alpha$ and $\beta$ be the roots of $u(X) = X^2 - aX + 1$ over $GF(p^2)$. Let $\mathrm{ord}_{GF(p^2)^*}\ \alpha = 2^e \cdot l$, where $l$ is odd. Then $e = t$ and $c$ is the least integer $i \geq 1$ such that $2^i \equiv \pm 1 \pmod l$.*

**Proof:** We have $h^{t+c}(a) = h^t(a)$ and $t \geq 0$, $c \geq 1$ are as small as possible. Then by Proposition 5.4.1 this is equivalent to

$$\alpha^{2^{t+c}} + \alpha^{-2^{t+c}} = \alpha^{2^t} + \alpha^{-2^t}.$$

By Lemma 5.4.2, we know that $\alpha^{2^{t+c}} = \alpha^{2^t}$ or $\alpha^{2^{t+c}} = \alpha^{-2^t}$, which holds iff $\alpha^{2^t(2^c-1)} = 1$ or $\alpha^{2^t(2^c+1)} = 1$. If $\mathrm{ord}_{GF(p^2)^*}\ \alpha = 2^e \cdot l$, where $l$ is odd, then $2^e \cdot l \,|\, 2^t(2^c - 1)$ or $2^e \cdot l \,|\, 2^t(2^c + 1)$. The desired result now follows. ∎

It follows that $c = \mathrm{ord}_l\ 2$ or $(\mathrm{ord}_l\ 2)/2$.

From the previous result we see that $t(a)$ and $c(a)$ depend on $\mathrm{ord}_{GF(p^2)^*}\ \alpha$, where $\alpha, \beta$ are the roots of $X^2 - aX + 1 = 0$. (Note that $\mathrm{ord}_{GF(p^2)^*}\ \alpha = \mathrm{ord}_{GF(p^2)^*}\ \beta$.) The following theorem characterizes these orders.

**Theorem 5.4.4** *(a) For each divisor $d$ of $p-1$, $d \neq 1, 2$ there are $\varphi(d)/2$ elements $a \in GF(p)$ for which the corresponding $\alpha$ has $\mathrm{ord}_{GF(p^2)^*}\ \alpha = d$;*

*(b) For each divisor $d'$ of $p+1$, $d' \neq 1, 2$ there are $\varphi(d')/2$ elements $a \in GF(p)$ for which the corresponding $\alpha$ has $\mathrm{ord}_{GF(p^2)^*}\ \alpha = d'$;*

*(c) For $a = 2$ we have $\alpha = \beta = 1$ and $\mathrm{ord}_{GF(p^2)^*}\ \alpha = 1$;*

*(d) For $a = -2$ we have $\alpha = \beta = -1$ and $\mathrm{ord}_{GF(p^2)^*}\ \alpha = 2$.*

**Proof:** Consider the polynomial $u(X) = X^2 - aX + 1$ over $GF(p)$. This polynomial is reducible if and only if it can be written in the form $(X-b)(X-b^{-1})$ where $a = b + b^{-1}$. By symmetry, this occurs for $(p+1)/2$ distinct values of $a$. The roots $b, b^{-1}$ are identical iff $b^2 = 1$, that is, if $b = \pm 1$. For the remaining $(p-3)/2$ values of $a$ the roots are distinct. This proves parts (a), (c), and (d).

Otherwise the polynomial $u(X)$ is irreducible over $GF(p)$ with distinct zeroes $\alpha$, $\beta$. We claim that the equation

$$\theta^{p+1} = 1 \tag{5.1}$$

has $p+1$ roots in $GF(p^2)$: namely 1, $-1$, and the $p-1$ roots $\alpha, \beta$ of the irreducible $u(X)$. To see this, note that $\alpha^{p+1} = \alpha \cdot \alpha^p = \alpha\beta = 1$. Since the roots of Eq. (5.1) form a cyclic group, for each $d' \,|\, p+1$ there are $\varphi(d')$ roots of order $d'$. Now each $a$ corresponding to an irreducible $u$ has two roots, so there are $\varphi(d')/2$ different $a$'s corresponding to $\alpha$ of order $d'$. ∎

We now prove the analogue of Theorem 4.4.2.

**Theorem 5.4.5** *Let $p$ be an odd prime. Let $\delta$ be a generator for $GF(p^2)^*$ and define $\theta = \delta^{p-1}$, so that $\theta$ is a generator of the subgroup of $(p+1)$'th roots of unity in $GF(p^2)$. Let $\gamma = \delta^{p+1}$, so that $\gamma$ generates $GF(p)^*$.*

*If $p \equiv 1 \pmod 4$ then*

*(a)*

$$\{a \in GF(p) \ : \ t(a) = 0\} = \{\theta^i + \theta^{-i} \ : \ 1 \le i \le (p-1)/2 \text{ and } \nu_2(i) \ge \nu_2(p+1)\} \cup \{\gamma^j + \gamma^{-j} \ : \ 0 \le j \le (p-1)/2 \text{ and } \nu_2(j) \ge \nu_2(p-1)\}; \quad (5.2)$$

*(b) For $1 \le k \le \nu_2(p-1)$ we have*

$$\{a \in GF(p) \ : \ t(a) = k\} = \{\theta^i + \theta^{-i} \ : \ 1 \le i \le (p-1)/2 \text{ and } \nu_2(i) = \nu_2(p+1) - k\} \cup \{\gamma^j + \gamma^{-j} \ : \ 0 \le j \le (p-1)/2 \text{ and } \nu_2(j) = \nu_2(p-1) - k\}. \quad (5.3)$$

*If $p \equiv 3 \pmod 4$ then*

*(c)*

$$\{a \in GF(p) \ : \ t(a) = 0\} = \{\theta^i + \theta^{-i} \ : \ 0 \le i \le (p+1)/2 \text{ and } \nu_2(i) \ge \nu_2(p+1)\} \cup \{\gamma^j + \gamma^{-j} \ : \ 1 \le j \le (p-3)/2 \text{ and } \nu_2(j) \ge \nu_2(p-1)\}; \quad (5.4)$$

*(d) For $1 \le k \le \nu_2(p+1)$ we have*

$$\{a \in GF(p) \ : \ t(a) = k\} = \{\theta^i + \theta^{-i} \ : \ 0 \le i \le (p+1)/2 \text{ and } \nu_2(i) = \nu_2(p+1) - k\} \cup \{\gamma^j + \gamma^{-j} \ : \ 1 \le j \le (p-3)/2 \text{ and } \nu_2(j) = \nu_2(p-1) - k\}. \quad (5.5)$$

*Furthermore, all these unions are distinct.*

**Proof:** We begin by proving case (a) and (b). For case (a), assume $p \equiv 1 \pmod 4$. Write $p + 1 = 2^{\tau'} \cdot \rho'$, where $\rho'$ is odd. Note that $\tau' = 1$ since $p + 1 \equiv 2 \pmod 4$.

By Theorem 5.4.3 we have that

$$t(a) = 0 \text{ iff there exists } c \ge 1 \text{ such that } \alpha^{2^c - 1} = 1 \text{ or } \alpha^{2^c + 1} = 1, \quad (5.6)$$

where $\alpha$ is a zero of $u(X) = X^2 - aX + 1$. (Note: $a = \alpha + \alpha^{-1}$.) There are two cases to consider: (i) $u$ is irreducible over $GF(p)$ or (ii) $u$ is reducible.

(i) If $u$ is irreducible, then $\alpha = \theta^i$ for some $i$ with $1 \leq i \leq p$, $i \neq (p+1)/2$. (Note that $\theta^0 = \theta^{p+1} = 1$ and therefore $\theta^{(p+1)/2} = -1$.) Restating (5.6), we have $t(a) = 0$ iff there exists $c \geq 1$ such that $\theta^{i(2^c-1)} = 1$ or $\theta^{i(2^c+1)} = 1$, iff there exists $c \geq 1$ with $p+1 \,|\, i(2^c-1)$ or $p+1 \,|\, i(2^c+1)$, iff there exists $c \geq 1$ with $\nu_2(i) \geq \tau'$ and $\rho' \,|\, 2^c - 1$, or $\nu_2(i) \geq \tau'$ and $\rho' \,|\, 2^c + 1$. We know there does exist a $c$ that satisfies the condition $\rho' \,|\, 2^c - 1$: that is, pick $c = \mathrm{ord}_{\rho'} 2$. Therefore, $t(a) = 0$ iff $a = \theta^i + \theta^{-i}$ for some $i$ with $1 \leq i \leq p$, $i \neq (p+1)/2$, $\nu_2(i) \geq \tau'$. But $\theta^{p+1} = 1$, so $\theta^i + \theta^{-i} = \theta^{p+1-i} + \theta^{-(p+1-i)}$, so we may eliminate duplicates by dividing our range for $i$ by one-half. To summarize this case, we have $t(a) = 0$ iff $a = \theta^i + \theta^{-i}$ with $1 \leq i \leq (p-1)/2$ and $\nu_2(i) \geq \nu_2(p+1)$.

(ii) If $u$ is reducible, then $\alpha = \gamma^j$ for some $j$ with $0 \leq j \leq p-2$. Write $p - 1 = 2^\tau \cdot \rho$. From the proof of Theorem 5.4.3 we have $t(a) = 0$ iff there exists a $c \geq 1$ such that $\alpha^{2^c-1} = 1$ or $\alpha^{2^c+1} = 1$, iff $\gamma^{j(2^c-1)} = 1$ or $\gamma^{j(2^c+1)} = 1$, iff $2^\tau \cdot \rho \,|\, j(2^c-1)$ or $2^\tau \cdot \rho \,|\, j(2^c+1)$. That is, $t(a) = 0$ iff $\tau \leq \nu_2(j)$ and either $\rho \,|\, j(2^c-1)$ or $\rho \,|\, j(2^c+1)$. Again, as in the earlier case, we picking $c = \mathrm{ord}_\rho 2$ yields $\rho \,|\, (2^c-1)$. As well, notice that $\gamma^{p-1} = 1$, so $\gamma^j + \gamma^{-j} = \gamma^{p-1-j} + \gamma^{-(p-1-j)}$, so we need only consider one-half of the range of possible values for $j$. Thus, $t(a) = 0$ iff $a = \gamma^j + \gamma^{-j}$ with $0 \leq j \leq (p-1)/2$ and $\nu_2(j) \geq \nu_2(p-1)$.

We now show that the quantities $\theta^i + \theta^{-i}$, $1 \leq i \leq (p-1)/2$ and $\gamma^j + \gamma^{-j}$, $0 \leq j \leq (p-1)/2$ are all distinct.

If $\theta^i + \theta^{-i} = \theta^{i'} + \theta^{-i'}$ for $1 \leq i, i' \leq (p-1)/2$ we know from Lemma 5.4.2 that $\theta^i = \theta^{i'}$ or $\theta^i = \theta^{-i'}$. The case $i = -i'$ is impossible since $i \geq 1$, and thus $i = i'$.

A similar argument applies if $\gamma^j + \gamma^{-j} = \gamma^{j'} + \gamma^{-j'}$.

Finally, suppose $\theta^i + \theta^{-i} = \gamma^j + \gamma^{-j}$ where $1 \leq i \leq (p-1)/2$ and $0 \leq j \leq (p-1)/2$. We know from Lemma 5.4.2 that either $\theta^i = \gamma^j$ Now $\theta = \delta^{p-1}$ and $\gamma = \delta^{p+1}$, where $\delta$ is a generator for $GF(p^2)^*$. Thus $\delta^{(p-1)i+(p+1)j} = 1$ or $\delta^{(p-1)i-(p+1)j} = 1$. Since $\mathrm{ord}_{GF(p^2)^*} \delta = p^2-1$, it follows that $p^2-1 \,|\, (p-1)i+(p+1)j$ or $p^2-1 \,|\, (p-1)i-(p+1)j$. Therefore, since $p$ is odd, we get that there exists $k$ such that either

$$\frac{p-1}{2} i = -\frac{p+1}{2} j + k \frac{p^2-1}{2} \tag{5.7}$$

or

$$\frac{p-1}{2} i = \frac{p+1}{2} j + k \frac{p^2-1}{2}. \tag{5.8}$$

In both cases, $\frac{p+1}{2}$ divides both terms of the right-hand side, and hence must divide the left-hand side. But $\gcd(\frac{p-1}{2}, \frac{p+1}{2}) = 1$, so $\frac{p+1}{2} \,|\, i$, a contradiction. This concludes the proof of case (a).

Now let us look at case (b). By Theorem 5.4.3 we have

$$t(a) = k \text{ iff } \mathrm{ord}_{GF(p^2)}\ \alpha = 2^k \cdot l, \tag{5.9}$$

where $l$ is odd and $\alpha$ is a zero of $u(X) = X^2 - aX + 1$. Once again we break up the argument into two cases: (i) $u$ is irreducible and (ii) $u$ is reducible.

(i) If $u$ is irreducible, then $\alpha = \theta^i$, for some $i$ with $1 \leq i \leq p$, $i \neq (p+1)/k$. Restating (5.9), we have $t(a) = k$ iff $\theta^{i2^k l} = 1$ and $\theta^{i2^{k-1}l} \neq 1$, iff $p+1 \,|\, i2^k l$ and $p+1 \nmid i2^{k-1}l$, iff $\nu_2(p+1) = \nu_2(i) + k$.

Case (ii) is similarly proven.

We now indicate the minor changes needed to prove (c) and (d). We need only remark that the different ranges for the exponents arise because of two reasons: first, the polynomial $X^2+1$ is irreducible if $p \equiv 3 \pmod 4$ and reducible if $p \equiv 1 \pmod 4$. Second, $t(-2) = 1$ and must be treated as a special case depending on $p \pmod 4$.

■

For $l$ odd define $\mathrm{ord}'_l\ 2$ to be the least $e$ such that $2^e \equiv \pm 1 \pmod l$.

**Corollary 5.4.6** *Let $p$ be an odd prime with $p-1 = 2^\tau \cdot \rho$, $p+1 = 2^{\tau'} \cdot \rho'$, and $\rho, \rho'$ odd. For each divisor $d > 1$ of $\rho$, $G = G_{p,2}$ contains $\varphi(d)/(2\ \mathrm{ord}'_d\ 2)$ cycles of length $\mathrm{ord}'_d\ 2$. There are $\rho$ elements in all these cycles, and off each element in these cycles there hang reversed complete binary trees of height $\tau - 1$ containing $2^\tau - 1$ elements.*

*Similarly, for each divisor $d' > 1$ of $\rho'$ there exists $\varphi(d')/(2\ \mathrm{ord}'_{d'}\ 2)$ cycles of length $\mathrm{ord}'_{d'}\ 2$, and off each element in these cycles there hang reversed complete binary trees of height $\tau' - 1$ containing $2^{\tau'} - 1$ elements.*

*Finally, the element 0 is the root of a complete binary tree of height $\tau - 2$ (respectively $\tau' - 2$) when $p \equiv 1 \pmod 4$ (respectively $p \equiv 3 \pmod 4$), and $G$ also contains the directed edges $(0,-2)$, $(-2,2)$, $(2,2)$.*

**Proof:** Exactly like that in Corollary 4.4.3. ■

For $p = 29$ we have the structure in Figure 5.3 and the data in Table 3.

| $d$ | $\varphi(d)$ | $a$ with $\alpha$ of order $d$ | $t = \nu_2(d)$ | $l = d/2^t$ | $c = \text{ord}'_l\ 2$ |
|---|---|---|---|---|---|
| 1 | 1 | $\{2\}$ | 0 | 1 | 1 |
| 2 | 1 | $\{27\}$ | 1 | 1 | 1 |
| 4 | 2 | $\{0\}$ | 2 | 1 | 1 |
| 7 | 6 | $\{3, 7, 18\}$ | 0 | 7 | 3 |
| 14 | 6 | $\{11, 22, 26\}$ | 1 | 7 | 3 |
| 28 | 12 | $\{10, 12, 13, 16, 17, 19\}$ | 2 | 7 | 3 |
| 3 | 2 | $\{28\}$ | 0 | 3 | 1 |
| 5 | 4 | $\{5, 23\}$ | 0 | 5 | 2 |
| 6 | 2 | $\{1\}$ | 1 | 3 | 1 |
| 10 | 4 | $\{6, 24\}$ | 1 | 5 | 2 |
| 15 | 8 | $\{4, 14, 20, 21\}$ | 0 | 15 | 4 |
| 30 | 8 | $\{8, 9, 15, 25\}$ | 1 | 15 | 4 |

Table 3: The structure of $G_{29,2}$

There are two special cases where we can give more detailed information about $G_{p,2}$. The first is when $p = 2^{2^k} + 1$, a Fermat prime.

**Theorem 5.4.7** *The structure of the digraph $G_{p,2}$ when $p = 2^{2^k} + 1$, a Fermat prime is as follows:*

*(i) A reversed complete binary tree of height $2^k - 2$ with root 0, attached to the node $-2$, attached to the node 2 with a cycle of length 1 on this node. The elements in this component are of the form $3^j + 3^{-j}$ for $0 \leq j \leq 2^{2^k-1}$.*

*(ii) A set of cycles of length dividing $2^k - 1$. Off each element in these cycles there hangs a single element with tail length 1.*

**Proof:** Part (i) follows immediately from Theorem 5.4.5 and the fact that 3 is a primitive root (mod $p$).

Part (ii) follows from the fact that $p + 1 = 2(2^{2^k-1} + 1)$. ∎

For $p = 2^{2^2} + 1 = 17$ we have the structure in Figure 5.1.

## 5.5 Robustness results for Mersenne primes

The second case where we can describe $G_{p,2}$ more precisely is when $p = 2^q - 1$, a Mersenne prime. Here $q$ is an odd prime.

**Theorem 5.5.1** *When $p = 2^q - 1$, a Mersenne prime, the digraph $G_{p,2}$ consists of*

*(i) A reversed complete binary tree of height $q - 1$ with root 0, attached to the node $-2$, that is attached to the node 2 with a cycle of length 1 on this node. The nodes in this tree are given by $\theta^n + \theta^{-n}$, $0 \leq n \leq 2^{q-1}$, where $\theta$ is a zero of $X^2 - 4X + 1$.*

*(ii) A set of cycles of length dividing $q-1$. Off each element in these cycles there hangs a single element with tail length 1. The nodes in these cycles are given by $\gamma^n + \gamma^{-n}$, $1 \leq n \leq 2^{q-1} - 2$, where $\gamma$ is a generator of $GF(p)^*$.*

**Proof:** Use Corollary 5.4.6. ∎

(It should be noted that the Theorem 17(ii) stated in [94] is actually incorrect and differs from the statement above. As well, this structure of the cycles has been more thoroughly explored in an internet discussion forum: see [83].)

For $p = 2^5 - 1 = 31$ we have the structure in Figure 5.2.

We can use Theorem 5.5.1 to prove the main robustness result for Mersenne primes. We begin by proving a basic lemma that we will refine and extend in subsequent theorems.

**Lemma 5.5.2** *Suppose the input $p$ ($= 2^q - 1$, $q$ an odd prime) to Algorithm 5.2 is prime. If the last error occurs before iteration $i$ ($1 \leq i < q$) of the algorithm, then with probability*

$$\frac{2^{q-i-1}}{2^q - 1}$$

*the output of the algorithm is correct (that is, the output is "prime").*

**Proof:** By Theorem 5.5.1, there is a complete binary tree of height $q - 1$ with root 0. At step $i$ of our algorithm, we are $i$ levels away from the root (with value 0) of this tree. There are $2^{q-1-i}$ elements in the complete binary tree that are $i$ levels away from the root. Thus, so long as an error causes the algorithm to be

at the correct level in the tree, the output will be correct. The probability of this occurring is

$$\frac{2^{q-1-i}}{2^q-1} = \frac{2^q}{2^q-1}\frac{1}{2^{i+1}}.$$

■

We extend this result to create the analogous version of Lemma 4.4.6.

**Lemma 5.5.3** *Consider the Lucas-Lehmer algorithm (Algorithm 5.2) on input* $2^q-1$ *which is prime. If the probability of an error occurring at any step of the algorithm is* $\epsilon$*, we have*

1. $p_c(2^q-1) = (1-\epsilon)^{q-1} + \frac{2^q}{2^q-1}\frac{\epsilon}{2(1-2\epsilon)}\left((1-\epsilon)^{q-1} - \frac{1}{2^{q-1}}\right)$
2. $p_f(2^q-1) = 1 - p_c(F_k)$*, and*
3. $p_i(2^q-1) = 0$.

**Proof:** We follow the same reasoning as in Lemma 4.4.6. If no errors occur, the output Algorithm 5.2 will be correct. The probability of no error occurring in a single step is $(1-\epsilon)^{q-1}$. If there is an error that occurs, we need only consider the last error of the algorithm, sum up all possible steps where this last error can occur and use Lemma 5.5.2.

We have

$$\begin{aligned}
p_c(2^q-1) &= (1-\epsilon)^{q-1} + \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{1}{2^{i+1}}\frac{2^q}{2^q-1} \\
&= (1-\epsilon)^{q-1} + \epsilon(1-\epsilon)^{q-1}\frac{1}{2}\frac{2^q}{2^q-1}\sum_{i=1}^{q-1}\left(\frac{1}{2(1-\epsilon)}\right)^i \\
&= (1-\epsilon)^{q-1} + \epsilon(1-\epsilon)^{q-1}\left(\frac{1}{2}\right)\frac{2^q}{2^q-1}\frac{1}{2(1-\epsilon)}\frac{(\frac{1}{2(1-\epsilon)})^{q-1}-1}{\frac{1}{2(1-\epsilon)}-1} \\
&= (1-\epsilon)^{q-1} + \frac{\epsilon}{2(1-2\epsilon)}\frac{2^q}{2^q-1}\left((1-\epsilon)^{q-1} - \frac{1}{2^{q-1}}\right).
\end{aligned}$$

Since the only other output from the Lucas-Lehmer test will be a valid integer in the range $0,\ldots,2^q-2$, we have $p_f = 1-p_c$ and $p_i = 0$. That is, there are no infeasible outputs. ■

As in the case of Pepin's test, the Lucas-Lehmer test is not robust, since $p_i = 0$. However, combining two independent executions of the algorithm will yield a robust algorithm.

We capture bounds, though not explicit formulas, for $p_c^*(2^q - 1)$, $p_f^*(2^q - 1)$, and $p_i^*(2^q - 1)$ in the following lemma.

**Lemma 5.5.4** *Suppose the input $p$ $(= 2^q - 1)$ for Algorithm 5.2 is prime. Consider two independent executions of the algorithm on the same input. If the probability of an error occurring is $\epsilon$, we have bounds on the probability measures for robustness as follows:*

- *the probability of correctness is*

$$p_c^*(2^q - 1) = (p_c(2^q - 1))^2$$

,

- *the probability of feasible but incorrect (i.e., "composite") output is*

$$p_f^*(2^q - 1) \leq p_f(2^q - 1) \left( \frac{1}{2}\epsilon(1-\epsilon)^{q-1} \sum_{i=1}^{q-1} \frac{2^{q-1-i}}{2^q - 1} \frac{1}{(1-\epsilon)^i} + \frac{1}{2}\epsilon(1-\epsilon)^{q-1} \sum_{i=1}^{q-1} \frac{2}{2^q - 1} \frac{1}{(1-\epsilon)^i} \right),$$

*and*

- *the probability of infeasible output is*

$$p_i^*(2^q - 1) \geq 2p_c(2^q - 1)p_f(2^q - 1).$$

**Proof:** We begin by considering correctness. The only way that the combined execution outputs a correct answer is if both executions output a correct answer (of "prime"). That is, the combined probability of correctness is $p_c^*(2^q - 1) = (p_c(2^q - 1))^2$.

For feasible but incorrect output, we note that the first execution must yield a non-zero result, say $R$, and the second execution must result in the same result $R$. That is, the second execution must have an error which results in the same location at the termination of the algorithm. There are two cases to consider, based on the observations in Theorem 5.5.1: either $R$ is an element of the binary tree or it is an element contained in a cycle component. There are $2^{q-1} - 2$ elements in the cycles, and the remaining elements are in the binary tree.

We first discuss the case where $R$ is an element in the binary tree component of $G_{p,2}$.

Notice that non-zero elements in the binary tree other than $-2$ have two children in the tree. These elements each have two children (again, other than $-2$) and so on. Considering only the second execution of the algorithm, if the last error happens $k$ steps before termination of the algorithm, there are at most $2^k$ elements that would result in $R$ being the output. We say "at most" since $R$ may be an element that is less than $k$ levels from the bottom of the tree, in which case an error with $k$ steps remaining cannot produce element $R$.

We now consider the case where $R$ is part of a cycle. In this case, either $R$ has two "children" (elements $x, y$ such that $x^2 - 2 = y^2 - 2 \equiv R \pmod{p}$) or no children (if $R$ is an element outside of the cycle). Moreover, if the last error occurs $k$ steps before termination of the algorithm, there are at most 2 elements that would result in $R$ being the output, since there is no exponential branching in the cyclic components.

Thus, we can provide an upper bound on the probability of feasible but incorrect output by considering all $q-1$ steps where the last error may occur. First, we note that we must have a feasible but incorrect output in the first execution of the algorithm. Considering the second execution of the algorithm, the probability that the last error occurs at step $j$ $(1 \leq j \leq q-1)$ is $\epsilon(1-\epsilon)^{j-1}$. If the last error occurs at step $j$, then there are either at most 2 or at most $2^{q-1-j}$ possible elements that will yield the desired output, depending on whether the desired output is in a cycle or binary tree, respectively. We will combine these observations to yield an upper bound on $p_f^*(2^q - 1)$ of

$$p_f(2^q-1)\left(\frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2^{q-1-i}}{2^q-1}\frac{1}{(1-\epsilon)^i} + \frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2}{2^q-1}\frac{1}{(1-\epsilon)^i}\right).$$

For incorrect and infeasible output, $p_i^*(2^q - 1)$ is bounded below by

$$2p_c(2^q-1)p_f(2^q-1),$$

because, at a minimum, if the two independent executions yield different answers (i.e., one yields "composite" while the other yields "prime"), which taken together is infeasible. Since there are two orderings of output (i.e., (composite, prime) and (prime, composite)), we have the constant factor 2 in the bound. ■

Having established these bounds, we are now prepared to prove robustness for the Lucas-Lehmer test on prime input.

**Theorem 5.5.5** *Consider prime $p = 2^q - 1$. When the Lucas-Lehmer algorithm (Algorithm 5.2) is executed twice on input p and the results are combined, and if the probability of an error occurring is $\epsilon < \frac{1}{q-1}$, the combined execution is robust.*

**Proof:** We will have to show that $p_c^*(2^q - 1) > b > 0$ for some constant $b$ and $p_i^*(2^q - 1) > p_f^*(2^q - 1)$.

We begin with providing a constant lower bound on $p_c^*$.

We have

$$\begin{aligned} p_c^*(2^q - 1) &= (p_c(2^q - 1))^2 \\ &\geq ((1-\epsilon)^{q-1})^2 \\ &> \left(\frac{1}{e}\right)^2 \\ &> 0. \end{aligned}$$

We now prove $p_i^*(2^q - 1) > p_f^*(2^q - 1)$.

From Lemma 5.5.4, we know that

$$p_i^*(2^q - 1) \geq 2p_c(2^q - 1)p_f(2^q - 1)$$

and

$$p_f^*(2^q{-}1) \leq p_f(2^q{-}1)\left(\frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2^{q-1-i}}{2^q-1}\frac{1}{(1-\epsilon)^i} + \frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2}{2^q-1}\frac{1}{(1-\epsilon)^i}\right).$$

Notice that the previous two inequalities have a common term of $p_f(2^q - 1)$ on the right hand sides. Let $A = 2p_c(2^q - 1)$ and let

$$B = \left(\frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2^{q-1-i}}{2^q-1}\frac{1}{(1-\epsilon)^i} + \frac{1}{2}\epsilon(1-\epsilon)^{q-1}\sum_{i=1}^{q-1}\frac{2}{2^q-1}\frac{1}{(1-\epsilon)^i}\right).$$

Therefore, to show $p_i^*(2^q - 1) > p_f^*(2^q - 1)$, we will show

$$p_i^*(2^q - 1) \geq p_f(2^q - 1)A > p_f(2^q - 1)B \geq p_f^*(2^q - 1),$$

which will require demonstrating $A > B$, as all the other inequalities have been shown above.

Write $B = B_1 + B_2$ where

$$B_1 = \frac{1}{2}\epsilon \sum_{i=1}^{q-1} \frac{2^{q-1-i}}{2^q - 1}(1-\epsilon)^{q-1-i}$$

and

$$B_2 = \epsilon \sum_{i=1}^{q-1} \frac{1}{2^q - 1}(1-\epsilon)^{q-1-i}.$$

Since $A = 2p_c(2^q - 1)$, we will show that $p_c(2^q - 1) > B_1$ and $p_c(2^q - 1) > B_2$, which will prove $A = p_c(2^q - 1) + p_c(2^q - 1) > B_1 + B_2 = B$.

Recall from Lemma 5.5.3, we have

$$p_c(2^q - 1) = (1-\epsilon)^{q-1} + \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{1}{2^{i+1}}.$$

To show that $A/2 > B_1$, first observe that since $q > 1$, know $2^q < 2(2^q - 1)$. Using this fact, we have

$$\begin{aligned}
A/2 &= p_c(2^q - 1) \\
&\geq \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{1}{2^{i+1}} \\
&= \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{1}{2^{i+1}}\frac{2^{q-1-i}}{2^{q-1-i}} \\
&= \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{2^{q-1-i}}{2^q} \\
&> \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{2^{q-1-i}}{2^q}\frac{2^q}{2(2^q-1)} \\
&= \sum_{i=1}^{q-1}(1-\epsilon)^{q-1-i}\epsilon\frac{2^{q-1-i}}{2(2^q-1)} \\
&= B_1.
\end{aligned}$$

To show $A/2 > B_2$, we will use the facts that $\epsilon < \frac{1}{q-1}$ and $1 \leq \frac{1}{1-\epsilon} \leq 2$. Specifically, we have

$$\begin{aligned} A/2 &= p_c(2^q - 1) \\ &\geq (1-\epsilon)^{q-1} \\ &> (1-\epsilon)^{q-1}\epsilon(q-1) \\ &> (1-\epsilon)^{q-1}\epsilon(q-1)\frac{1}{2^q-1}2^{q-1} \\ &> (1-\epsilon)^{q-1}\epsilon\frac{1}{2^q-1}(q-1)\left(\frac{1}{1-\epsilon}\right)^{q-1} \\ &\geq (1-\epsilon)^{q-1}\epsilon\frac{1}{2^q-1}\sum_{i=1}^{q-1}(1-\epsilon)^{-i} \\ &= B_2. \end{aligned}$$

To see the last step, notice the third last line has $q-1$ terms of size $\left(\frac{1}{1-\epsilon}\right)^{q-1}$. The second last line has $q-1$ terms, where each term (of the form $(1-\epsilon)^i$) is bounded above by $(1-\epsilon)^{q-1}$.

Since we have shown $p_c$ satisfies the necessary inequalities, it follows that $p_i^*(2^q - 1) > p_f^*(2^q - 1)$. Therefore, the combined execution of the Lucas-Lehmer algorithm is robust on prime input. ■

## 5.6 Robustness results for Mersenne composites

We now consider the case where the input to the Lucas-Lehmer test is composite. That is, we consider $2^q - 1 = p_1^{e_1}p_2^{e_2}\cdots p_m^{e_m}$, where the $p_i$ are distinct primes and $e_i \geq 1$ for $1 \leq i \leq m$.

We note that any factor of a Mersenne number must be equivalent to 1 or 7 mod 8. To observe this, if prime $p \mid 2^q - 1$, then $2^q \equiv 1 \pmod{p}$, and therefore 2 has order $q \pmod{p}$. But 2 has order dividing $p-1$, so $p - 1 = 2kq$ for some integer $k > 1$. Thus

$$2^{\frac{p-1}{2}} = 2^{kq} \equiv 1 \pmod{p},$$

which implies 2 is a quadratic residue of $p$. Therefore, $p \equiv \pm 1 \pmod{8}$.

**Theorem 5.6.1** *Suppose the input to the Lucas-Lehmer test is composite; that is, $2^q - 1 = p_1^{e_1}p_2^{e_2}\cdots p_m^{e_m}$, where $p_i$ is prime and $e_i \geq 1$ for $1 \leq i \leq m$.*

*Since all odd prime factors of* $2^q - 1$ *are congruent to* 1 *or* 7 (mod 8)*, define*

$$h_1 = \min\{\nu_2(p_i - 1) : 1 \leq i \leq m, p_i \equiv 1 \pmod 8\},$$

*and*

$$h_2 = \min\{\nu_2(p_i + 1) : 1 \leq i \leq m, p_i \equiv 7 \pmod 8\}.$$

*Let* $h = \min\{h_1, h_2\} - 2$*. The output is correct with probability* 1 *if the last error occurs at or before the* $(q-1-h-1)$*th squaring step. If the last error occurs at the* $j$*th squaring step, where* $q - 1 - h \leq j \leq q - 1$*, the probability the output is correct is*

$$1 - \frac{(2^m)^{q-1-j}}{2^q - 1}.$$

**Proof:** First, notice that all $p_i$ are odd since $\nu_2(2^q - 1) = 0$. So, we may apply Lemma 4.5.2 to the function $f(x) = x^2 - 2 - k$, where $k \in \mathbb{Z}/(2^q - 1)$ to conclude that we need only consider solutions to each of $x^2 - 2 \equiv k \pmod{p_i^{e_i}}$. Using the Chinese Remainder Theorem, we may get to an element $k \in \mathbb{Z}/(2^q - 1)$ by an iteration of $x \to x^2 - 2$ by simultaneously satisfying

$$\begin{aligned} x^2 - 2 &\equiv k \pmod{p_1^{e_1}} \\ x^2 - 2 &\equiv k \pmod{p_2^{e_2}} \\ &\vdots \\ x^2 - 2 &\equiv k \pmod{p_m^{e_m}}. \end{aligned}$$

Since all prime divisors of $2^q - 1$ are congruent to 1 or 7 (mod 8), we can apply Corollary 5.4.6 on each of these prime factors. If the last error occurs at or before the $(q-1-h-1)$th squaring step, there are at least $h+1$ squaring steps remaining, but the minimum height of the complete binary tree in any $p_i$ subgroup is $h$, and thus, the algorithm cannot complete with value 0. Therefore, the algorithm will output "composite" in this case.

If the last error occurs at step $j$, $q - 1 - h \leq j \leq q - 1$, then the only way the output will be "prime" (i.e., the incorrect output) is if the error causes the element to be at the appropriate depth $(q - 1 - j)$ in these complete binary trees (with respect to each prime divisor). Since there are a total of $m$ prime divisors, and each solution to $x^2 - 2 = k \pmod{p_i}$ has either 0 or 2 solutions by Lemma 4.5.3, there are a total of $(2^m)^{q-1-j}$ elements at the appropriate depth. Thus, the output will be correct (i.e., "composite") with probability

$$1 - \frac{(2^m)^{q-1-j}}{2^q - 1}.$$

■

Thus, we have shown that Powers' statement that Lucas' method of determining primality "is free from any uncertainty as to the accuracy of the conclusion that the number under consideration is prime...since an error in calculating any term of the series would have the effect of preventing the appearance of the residue 0 [which indicates primality]" [78] requires some explicit quantification and is not (in an absolute sense) true.

It is worth noting that the previous theorem relies on the factors of the Mersenne number, and much work has been done on providing various bounds for the number or magnitude of Mersenne factors. A suitable starting point in this analysis of Mersenne factors is found in Stewart [90], which also contains similar analysis of Fermat factors. This work has been extended and improved more recently by Ford, Luca and Shparlinski [36].

We now turn our attention to proving robustness results for the Lucas-Lehmer test on composite input. As in the case of prime input, we begin by computing basic probability measures for correct output, incorrect but feasible output and infeasible output. Unlike the prime input case, however, our probabilities will be divided into two cases, based on the division used in Theorem 5.6.1.

**Lemma 5.6.2** *Suppose the input to the Lucas-Lehmer test is composite; that is,* $2^q - 1 = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$*, where* $p_i$ *is prime and* $e_i \geq 1$ *for* $1 \leq i \leq m$*. Suppose the probability of an error occurring at any given step is* $\epsilon$*.*

*Since all odd prime factors of* $2^q - 1$ *are congruent to* 1 *or* 7 $\pmod 8$*, define*

$$h_1 = \min\{\nu_2(p_i - 1) : 1 \leq i \leq m, p_i \equiv 1 \pmod 8\},$$

*and*

$$h_2 = \min\{\nu_2(p_i + 1) : 1 \leq i \leq m, p_i \equiv 7 \pmod 8\}.$$

*Let* $h = \min\{h_1, h_2\} - 2$*. We have the following results:*

*(a) the probability of correct output is*

$$p_c(2^q-1) = (1-\epsilon)^{q-1} + \sum_{i=1}^{q-1-h-1} \epsilon(1-\epsilon)^{q-1-i} + \sum_{i=q-1-h}^{q-1} \epsilon(1-\epsilon)^{q-1-i} \left(1 - \frac{(2^m)^{q-1-i}}{2^q - 1}\right),$$

*(b) the probability of incorrect but feasible output is* $p_f(2^q - 1) = 1 - p_c(2^q - 1)$*, and*

*(c) the probability of incorrect and infeasible output is* $p_i(2^q - 1) = 0$.

**Proof:** To begin, we prove the results for $p_i(2^q - 1)$ and $p_f(2^q - 1)$, which are very simple. For $p_i(2^q - 1)$, since the final numerical value computed by Algorithm 5.2 is in the range $0 \ldots 2^q - 2$, there are no infeasible outputs, and thus $p_i(2^q - 1) = 0$. Combining this fact with the fact that $p_i(2^q - 1) + p_f(2^q - 1) + p_c(2^q - 1) = 1$, we have $p_f(2^q - 1) = 1 - p_c(2^q - 1)$.

The remainder of the proof deals with the probability of correctness.

We turn to computing the probability of correct output in the case where all factors of $2^q - 1$ are equivalent to $1, 7 \pmod 8$. We use an argument similar to the above.

Notice that correct output can occur several ways in this case. If there are no errors introduced into the algorithm, the output will be correct. The probability that no errors occur is $(1 - \epsilon)^{q-1}$. From Theorem 5.6.1, we know the output will be correct if the last error occurs at or before the $(q - 1 - h - 1)$th step, and we also know the output will be correct with probability

$$1 - \frac{(2^m)^{q-1-i}}{2^q - 1}$$

if the last error occurs at or after step $q - 1 - h$. This implies the probability of correctness, taken over all possible steps where the last error occurs is

$$\sum_{i=1}^{q-1-h-1} \epsilon(1-\epsilon)^{q-1-i} + \sum_{i=q-1-h}^{q-1} \epsilon(1-\epsilon)^{q-1-i} \left(1 - \frac{(2^m)^{q-1-i}}{2^q - 1}\right),$$

which when combined with the probability of no error occurring, yields the stated value of $p_c(2^q - 1)$. ∎

As in the case for prime inputs into the Lucas-Lehmer test, the algorithm is not robust, since $p_f(2^q - 1) = 0$. To improve the robustness, we must combine two independent trials of the algorithm, as we have done in the prime case. As before, we let $p_c^*(2^q - 1), p_f^*(2^q - 1)$ and $p_i^*(2^q - 1)$ represent the probabilities of correct output, incorrect but feasible output and incorrect and infeasible output (respectively) under this combined independent execution model.

We give bounds on these values in the following lemma.

**Lemma 5.6.3** *Suppose the input* $p(= 2^q - 1)$ *for Algorithm 5.2 is composite. Consider two independent executions of the algorithm on the same input. If the probability of an error occurring is* $\epsilon$*, we have the following relationships on probabilities:*

- *the probability of correct output is* $p_c^*(2^q-1) \leq p_c(2^q-1)$,
- *the probability of incorrect but feasible output is* $p_f^*(2^q-1) = (p_f(2^q-1))^2$,
- *the probability of incorrect but infeasible output is*

$$p_i^*(2^q-1) \geq 2p_c(2^q-1)p_f(2^q-1).$$

**Proof:** In order for the combined execution to output an incorrect but feasible answer, both executions must output "prime." The probability of this occurring is exactly $(p_f(2^q-1))^2$.

For incorrect but infeasible output, a lower bound would be when one execution of the algorithm outputs "prime" while the other outputs "composite". This can happen one of two ways: either the first execution outputs "prime" while the second outputs "composite" or vice versa. To summarize in this case,

$$p_i^*(2^q-1) \geq 2p_c(2^q-1)p_f(2^q-1).$$

Finally, to bound the probability of correct output, we notice that $p_c(2^q-1) + p_f(2^q-1) = 1$, which follows from Lemma 5.6.2, we can square both sides to derive

$$(p_c(2^q-1))^2 + (p_f(2^q-1))^2 + 2p_c(2^q-1)p_f(2^q-1) = 1.$$

Rearranging this equation and applying the bounds for $p_i^*(2^q-1)$ and $p_f^*(2^q-1)$ yields $p_c^*(2^q-1) \leq (p_c(2^q-1))^2$. ■

Having these bounds is enough information to prove that the combining two independent executions of Algorithm 5.2 satisfies the conditions for robustness.

**Theorem 5.6.4** *Suppose the input* $p(= 2^q-1)$ *for the Lucas-Lehmer test (Algorithm 5.2) is composite. When Algorithm 5.2 is executed twice on input* $p$ *and the results are combined, and if the probability of an error occurring is* $\epsilon < \frac{1}{q-1}$*, the combined execution is robust.*

**Proof:** We will have to show that $p_c^*(2^q-1) > b > 0$ for some constant $b$ and $p_f^*(2^q-1) > p_i^*(2^q-1)$.

We begin by proving lower bounds on $p_c^*(2^q-1)$. Notice that the output will be correct if no error occurs in either one of the independent executions. The probability that no error occurs in either execution is $((1-\epsilon)^{q-1})^2$, which is a lower bound on $p_c^*(2^q-1)$. Since $\epsilon < \frac{1}{q-1}$, we have

$$p_c^*(2^q-1) \geq ((1-\epsilon)^{q-1})^2 \geq e^{-2} > 0.$$

The second condition that must be satisfied to show robustness is to prove $p_f^*(2^q-1) < p_i^*(2^q-1)$.

From Lemma 5.6.3, we know that $p_i^*(2^q-1) \geq 2p_c(2^q-1)p_f(2^q-1)$ and $p_f^*(2^q-1) = (p_f(2^q-1))^2$. We will prove that $2p_c(2^q-1)p_f(2^q-1) > p_f^*(2^q-1)$, which will demonstrate the desired condition of robustness.

Manipulating the previous inequality, we can substitute $p_f^*(2^q-1) = p_f(2^q-1)(1-p_c(2^q-1))$ into the right hand to prove $2p_c(2^q-1)p_f(2^q-1) > p_f(2^q-1)(1-p_c(2^q-1))$. We can cancel out $p_f(2^q-1)$ from both sides, yielding $2p_c(2^q-1) > 1-p_c(2^q-1)$. Thus, we must show $p_c(2^q-1) > \frac{1}{3}$. However, we know $p_c(2^q-1) \geq (1-\epsilon)^{q-1}$ due to the fact that it may be the case that no error occurs, and we also know that $\epsilon < \frac{1}{q-1}$. Therefore, combining this fact as we did in showing a lower-bound for $p_c^*(2^q-1)$, we have that $p_c(2^q-1) > e^{-1} > \frac{1}{3}$, which proves the result. ■

We combine the previous theorem on composite input with the theorem on prime input (Theorem 5.5.5) to make a statement on robustness in all cases for the Lucas-Lehmer test.

**Corollary 5.6.5** *When the Lucas-Lehmer test (Algorithm 5.2) is executed twice on input $p = (2^q-1)$ and the results are combined, and if the probability of error occurring is $\epsilon < \frac{1}{q-1}$, the combined execution is robust.*

**Proof:** Combine Theorems 5.5.5 and 5.6.4. ■

## 5.7 Statistical measures for $x \rightarrow x^2 - 2$

In this section, we will apply similar statistical measures as was done in Section 4.7.

Recall the definitions of Section 2.4 concerning $TC(p,2)$, $T_0(p,2)$, etc. We use these definitions to prove the main result for statistical measures for the Lucas-Lehmer algorithm.

**Corollary 5.7.1** *Let $p$ be prime. Let $p-1 = 2^\tau \cdot \rho$ and $p+1 = 2^{\tau'} \cdot \rho'$ with $\rho, \rho'$ odd. With respect to the iteration $x \rightarrow x^2 - 2$ (mod $p$), we have*

*(a)* $TC(p,2) = \frac{1}{2}\left(\sum_{d \,|\, \rho} \frac{\varphi(d)}{\mathrm{ord}'_d\ 2} \ + \ \sum_{d' \,|\, \rho'} \frac{\varphi(d')}{\mathrm{ord}'_{d'}\ 2}\right)$;

*(b)* $T_0(p,2) = (\rho + \rho')/2$;

*(c)* $AC(p,2) = T_0(p,2)/TC(p,2)$;

*(d)* $C(p,2) = \frac{1}{2p}\left(2^{\tau}\sum_{d\,|\,\rho}\varphi(d)\mathrm{ord}'_d\ 2\ +\ 2^{\tau'}\sum_{d'\,|\,\rho'}\varphi(d')\mathrm{ord}'_{d'}\ 2\right)$;

*(e)* $T(p,2) = \frac{1}{2p}\left(\sum_{d\,|\,p-1}\varphi(d)\nu_2(d)\ +\ \sum_{d'\,|\,p+1}\varphi(d')\nu_2(d')\right) = \frac{\tau+\tau'}{2} + \frac{\tau'-\tau+\rho+\rho'}{2p} - 1$.

**Proof:** Parts (a)-(d) rely on a direct application of Corollary 5.4.6. Part (e) requires some explanation. We have

$$\begin{aligned}
T(p,2) &= \frac{1}{2p}\left(\sum_{d\,|\,p-1}\varphi(d)\nu_2(d)\ +\ \sum_{d'\,|\,p+1}\varphi(d')\nu_2(d')\right)\\
&= \frac{1}{2p}\left(\sum_{d\,|\,\rho}\varphi(d)((\tau-1)2^{\tau}+1)\ +\ \sum_{d'\,|\,\rho'}\varphi(d')((\tau'-1)2^{\tau'}+1)\right)\\
&= \frac{1}{2p}\left(\rho((\tau-1)2^{\tau}+1)\ +\ \rho'((\tau'-1)2^{\tau'}+1)\right)\\
&= \frac{1}{2p}\left((\tau-1)(p-1)+\rho+(\tau'-1)(p+1)+\rho'\right)\\
&= \frac{\tau+\tau'}{2} + \frac{\tau'-\tau+\rho+\rho'}{2p} - 1.
\end{aligned}$$

■

As an example, we have $TC(29,2) = 5$; $T_0(29,2) = 11$; $AC(29,2) = 11/5$; $C(29,2) = 81/29$; and $T(29,2) = 25/29$.

In the next two theorems, we prove the main results concerning the asymptotics of the Lucas-Lehmer test. First, we give a result analogous to Theorem 4.7.4.

**Theorem 5.7.2** *Assume the ERH. Then with respect to the iteration $x \to x^2 - 2$ (mod p) we have $ST_0(N) \sim \frac{N^2}{6\ln N}$.*

**Proof:** Exactly like that for Theorem 4.7.4. ■

It is interesting to note that we can obtain a slightly weaker result without any unproved hypotheses. Indeed, since

$$\frac{p+1}{2} \le \rho + \rho' \le \frac{3p+1}{4}$$

we immediately obtain $T_0(p,2) = \Theta(p)$ and hence $ST_0(N) = \Theta(N^2/(\ln N))$.

Next, we prove a result analogous to Theorem 4.7.5.

**Theorem 5.7.3** *Assume the ERH. Then with respect to the iteration $x \to x^2 - 2$ (mod p) we have $ST(N) \sim \frac{2}{3} \cdot \frac{N^2}{\ln N}$.*

**Proof:** By Theorem 5.7.1 (e) we have

$$\begin{aligned} ST(N) &= \sum_{2<p\le N} p\left(\frac{\nu_2(p-1)+\nu_2(p+1)}{2} + \frac{\tau' - \tau + \rho + \rho'}{2p} - 1\right) \\ &= \frac{1}{2}\sum_{2<p\le N} \nu_2(p-1)(p-1) + \frac{1}{2}\sum_{2<p\le N} \nu_2(p+1)(p+1) + \frac{1}{2}\sum_{2<p\le N} \frac{p-1}{\nu_2(p-1)} \\ &\quad + \frac{1}{2}\sum_{2<p\le N} \frac{p+1}{\nu_2(p+1)} - \sum_{2<p\le N} p. \end{aligned}$$

Using exactly the same techniques as in the proof of Theorem 4.7.5, we obtain the desired result. ∎

It should be noted that unconditional results of Theorems 5.7.2 and 5.7.3 have been proved by Chou and Shparlinski [24].

Table 4 compares the asymptotic estimates to empirical data.

| $N$ | $ST_0(N)$ | $N^2/(6\ln N)$ | $ST(N)$ | $2N^2/(3\ln N)$ |
|---|---|---|---|---|
| 10 | 5 | 7.24 | 17 | 28.95 |
| $10^2$ | 350 | 361.91 | 1368 | 1447.65 |
| $10^3$ | 25484 | 24127.47 | 98718 | 96509.88 |
| $10^4$ | 1918051 | 1809560.34 | 7548493 | 7238241.36 |
| $10^5$ | 151494654 | 144764827.30 | 605787238 | 579059309.20 |
| $10^6$ | 12516198017 | 12063735608.42 | 50108219545 | 48254942433.69 |

Table 4: Comparing $ST_0'(N)$ and $ST(N)$ to asymptotic estimates

## 5.8 Pollard's factoring method

Pollard's factoring method is based on the fact that iterating a random quadratic map, modulo $p$, seems to produce tails and cycles that average $O(\sqrt{p})$ in size. Is this true for the iteration $x \to x^2 - 2$? As we have seen in Theorem 5.7.3,

$$\sum_{2<p\le N} \sum_{0\le a<p} t_p(a) \sim \frac{2}{3} \cdot \frac{N^2}{\ln N},$$

while

$$\sum_{2<p\le N}\sum_{0\le a<p} 1 \sim \frac{N^2}{2\ln N}.$$

One way to interpret this is to say that, on average, iterating the map $x \to x^2 - 2$ produces a tail of size $4/3$ — which is quite short.

However, we do not know any good asymptotic estimate for

$$SC(N) := \sum_{2<p\le N}\sum_{0\le a<p} c_p(a).$$

If $p$ is a Mersenne prime, say $p = 2^q - 1$, then

$$\begin{aligned}\sum_{0\le a<p} c_p(a) &= \frac{2^\tau \sum_{d\,|\,\rho}\varphi(d)\mathrm{ord}'_d\ 2\ +\ 2^{\tau'}\sum_{d'\,|\,\rho'}\varphi(d')\mathrm{ord}'_{d'}\ 2}{2} \\ &\le \frac{2(2^{q-1}-1)(q-1)+2^q}{2} = O(p\ln p).\end{aligned}$$

However, for certain primes $p$, such as those for which (a) $p' := (p-1)/2$ is prime and (b) 2 is a primitive root (mod $p'$), we have

$$\begin{aligned}\sum_{0\le a<p} c_p(a) &= \frac{2^\tau \sum_{d\,|\,\rho}\varphi(d)\mathrm{ord}'_d\ 2\ +\ 2^{\tau'}\sum_{d'\,|\,\rho'}\varphi(d')\mathrm{ord}'_{d'}\ 2}{2} \\ &\ge (p'-1)\frac{p'-1}{2} = \Omega(p^2).\end{aligned}$$

We expect there to be infinitely many such primes; indeed, heuristics such as Artin's conjecture on primitive roots suggest there are about $cN/(\ln N)^2$ such primes $\le N$. This suggests that $SC(N)$ might well be $\Omega(N^3/(\ln N)^2)$ and hence the "average" element will have cycle length at least $c'N/(\ln N)^2$. This suggests it is indeed wise to avoid the iteration $x \to x^2 - 2$, as Pollard suggested.

We did some computations on this question, which are summarized in Table 5.

| $N$ | $SC(N)$ |
|---|---:|
| 10 | 15 |
| $10^2$ | 6106 |
| $10^3$ | 3292717 |
| $10^4$ | 1896148462 |
| $10^5$ | 1269905340415 |
| $10^6$ | 902615197142485 |

Table 5: Some selected values of $SC(N)$

These computations suggest that perhaps $SC(N) \sim c'' \frac{N^3}{(\ln N)^2}$, where $c'' \doteq .17$.

## 5.9 Conclusions

In this chapter, we have fully described the topology of the digraph formed by $x \to x^2 - 2$. To do this, we used polynomials over Galois fields and statistical measurements for this digraph. We used the description of the topology of the digraph formed by $x \to x^2$ to prove that the Lucas-Lehmer test requires two independent executions to satisfy the definition of robustness.

In the following chapter, we use Dickson polynomials and Lucas functions to prove one of the main results of this chapter, Corollary 5.4.6.

# Chapter 6

# Alternative proofs concerning the Lucas-Lehmer test

## 6.1 Introduction and motivation

In this section, we prove the main results of the previous chapter using Dickson polynomials and Lucas functions.

## 6.2 Main result

We prove a simplified version of Corollary 5.4.6, which demonstrates the existence (and height) of the complete binary tree subgraph of $G^R_{p,2}$ for any prime $p$. This main theorem presents the same result as Corollary 5.4.6 but in a slightly different formulation.

**Theorem 6.2.1** *For any prime $p = k \cdot 2^d \pm 1$, where $k \geq 1$ is odd and $d \geq 3$, the digraph induced by the iterated function $x \to x^2 - 2 \pmod p$ contains a connected component which is a complete binary tree of height $d - 1$ with a "tail" of two additional vertices attached to the root.*

*In particular, there is a map $V : \mathbb{Z} \to GF(p)$ such that $V(i)$ and $V(j)$ are distinct for $0 \leq i < j \leq 2^d - 1$, and the digraph $G^R_{p,2}$ contains*

*(a) a complete binary tree with root 0 (attached to a tail consisting of vertices $V(2^{d-1}) = -2$ and $V(0) = 2$);*

*(b) leaves* $V(1), V(3), \ldots, V(2^{d-1}-1)$*; and*

*(c) internal vertices* $V(2), V(4), \ldots, V(2^{d-1}-2)$*. Each internal vertex has in-edges from* $V(i)$ *and* $V(2^{d-1}-i)$*.*

Our proof is broken down into several cases.

### 6.2.1 The case $p \equiv 3, 5 \pmod 8$

In this case, we know that $\left(\frac{2}{p}\right) = -1$ from the basic properties of the Legendre symbol; see, for example, Bach & Shallit [5]. Thus, $x^2 - 2 = 0$ has no solution in $GF(p)$, and so our "binary tree" has height 0 in this case.

### 6.2.2 The case $p \equiv 1, 7 \pmod 8$

Let $p$ be a prime such that $p = k \cdot 2^d \pm 1$, where $k \geq 1$, $k$ odd, $d \geq 3$.

Pick any primitive polynomial $f(x)$ over $GF(p)[x]$ of degree 2. It is well known that $f(x)$ exists. (For proof of this fact, see Cor. 2.11 of Lidl & Niederreiter [59].)

Let $\gamma, \delta$ be the roots of $f(x)$. Then $\gamma, \delta \in GF(p^2)$ and hence $\gamma^{p^2-1} = 1$. Since $f(x)$ is primitive, then $\gamma, \delta$ must be generators of the multiplicative subgroup $GF(p^2)^*$. By definition of a generator $\gamma^{\frac{p^2-1}{2}} = -1$.

Define $\alpha = \gamma^{\frac{(p-1)(p+1)}{2^d}}$ and $\beta = \alpha^{-1}$. Thus $\alpha\beta = 1$. It is worth noting that

$$\alpha = \begin{cases} \gamma^{(p+1)k}, & \text{if } p \equiv 1 \pmod 8; \\ \gamma^{(p-1)k}, & \text{if } p \equiv 7 \pmod 8. \end{cases}$$

**Lemma 6.2.2** *We have*

*(a)* $\alpha^p = \alpha$ *and* $\beta^p = \beta$ *if* $p \equiv 1 \pmod 8$.

*(b)* $\alpha^p = \beta$ *and* $\beta^p = \alpha$ *if* $p \equiv 7 \pmod 8$.

**Proof:**

(a) If $p \equiv 1 \pmod 8$, we know $\alpha = \gamma^{(p+1)k}$. Thus, we have

$$\alpha^p = \gamma^{(p+1)kp} = \gamma^{(p+1)kp}\gamma^{-(p^2-1)k} = \gamma^{kp^2+kp-kp^2+k} = \gamma^{k(1+p)} = \alpha.$$

Since $\beta = \alpha^{-1}$, we have $\beta^p = (\alpha^{-1})^p = (\alpha^p)^{-1} = \alpha^{-1} = \beta$.

(b) If $p \equiv 7 \pmod 8$, we have

$$\alpha^p = \gamma^{(p-1)kp} = \gamma^{kp^2-kp-kp^2+k} = \gamma^{(1-p)k} = \alpha^{-1} = \beta.$$

Since $\alpha \in GF(p^2)$, we have $\alpha^{p^2} = \alpha$. Thus, $\alpha^{p^2} = \beta^p = \alpha$. ■

**Lemma 6.2.3** *For $\alpha$ and $\beta$ defined as above, $\alpha + \beta \in GF(p)$.*

**Proof:** We know that if $x \in GF(p^2)$ where $p$ is prime, then $x \in GF(p)$ if and only $x^p = x$. For the case $p \equiv 7 \pmod 8$, we have

$$(\alpha + \beta)^p = \alpha^p + \beta^p = \beta + \alpha,$$

by Lemma 6.2.2.

For the case $p \equiv 1 \pmod 8$, we know that $\alpha^p = \alpha$ and $\beta^p = \beta$ by Lemma 6.2.2. Thus, $\alpha, \beta \in GF(p)$, which implies $\alpha + \beta \in GF(p)$. ■

It is clear that $\alpha+\beta+2 \in GF(p)$. We now prove a result regarding the Legendre symbol for $\alpha + \beta + 2$.

**Lemma 6.2.4** *We have*

$$\left(\frac{\alpha + \beta + 2}{p}\right) = -1.$$

**Proof:** From the definition of $\alpha$ and $\beta$, we know that $\alpha = \gamma^{(p-1)k}$ for $p \equiv 7 \pmod 8$ and $\alpha = \gamma^{(p+1)k}$ for $p \equiv 1 \pmod 8$. Notice that in the case $p \equiv 7 \pmod 8$ we have

$$\begin{aligned} \alpha + \beta + 2 &= \gamma^{(p-1)k} + \gamma^{(1-p)k} + 2 \\ &= (\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}})^2 \end{aligned}$$

and in the case $p \equiv 1 \pmod 8$ we have

$$\alpha + \beta + 2 = (\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}})^2.$$

It is worth noting that since $p$ is odd, the exponent $\frac{(p\pm1)k}{2} \in \mathbb{Z}$.

In order to prove the desired result, we need to show that $(\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}})^2 \notin GF(p)$, for $p \equiv 7 \pmod 8$ and $(\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}})^2 \notin GF(p)$, for $p \equiv 1 \pmod 8$ which is done in a manner similar to Lemma 6.2.3.

We will demonstrate that if $p \equiv 7 \pmod 8$ then

$$(\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}})^p = -(\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}})$$

and if $p \equiv 1 \pmod 8$ then

$$(\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}})^p = -(\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}}).$$

To prove this, first note that $\gamma^{\frac{(p^2-1)k}{2}} = -1$, since $\gamma$ has order $p^2 - 1$ and $k$ is odd. Based on this fact, if $p \equiv 7 \pmod 8$, we have

$$\begin{aligned}
(\gamma^{\frac{(p-1)k}{2}})^p &= \gamma^{\frac{(p^2-p)k}{2}} \\
&= \gamma^{\frac{-(p^2-1)k}{2}} \gamma^{\frac{-(p^2-1)k}{2}} \gamma^{\frac{(p^2-p)k}{2}} \\
&= -\gamma^{\frac{(p^2-p-p^2+1)k}{2}} \\
&= -\gamma^{\frac{(1-p)k}{2}}.
\end{aligned}$$

Similarly, if $p \equiv 1 \pmod 8$, we have

$$(\gamma^{\frac{(p+1)k}{2}})^p = -\gamma^{\frac{(p+1)k}{2}}.$$

In the case $p \equiv 7 \pmod 8$, we use the identity $(\gamma^{\frac{(p-1)k}{2}})^p = -\gamma^{\frac{(1-p)k}{2}}$ to derive the following:

$$\begin{aligned}
(\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}})^p &= \gamma^{\frac{(p^2-p)k}{2}} + \gamma^{\frac{(p-p^2)k}{2}} \\
&= -\gamma^{\frac{(1-p)k}{2}} - \gamma^{\frac{(p-1)k}{2}} \\
&= -(\gamma^{\frac{(p-1)k}{2}} + \gamma^{\frac{(1-p)k}{2}}).
\end{aligned}$$

In the case $p \equiv 1 \pmod 8$, we use the identity $(\gamma^{\frac{(p+1)k}{2}})^p = -\gamma^{\frac{(p+1)k}{2}}$ to deduce that

$$\begin{aligned}
(\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}})^p &= \gamma^{\frac{(p^2+p)k}{2}} + \gamma^{\frac{(-p-p^2)k}{2}} \\
&= -\gamma^{\frac{(p+1)k}{2}} - \gamma^{\frac{(-p-1)k}{2}} \\
&= -(\gamma^{\frac{(p+1)k}{2}} + \gamma^{\frac{(-1-p)k}{2}}).
\end{aligned}$$

Since there is no element $x \in GF(p)$ such that $x^2 = \alpha + \beta + 2$, it must be the case that $\left(\frac{\alpha+\beta+2}{p}\right) = -1$. ∎

Define $V(i) = \alpha^i + \beta^i$ for $i \geq 0$. We prove a simple relationship for $V(i)$ and then use this relationship to prove that $V(i) \in GF(p)$.

**Lemma 6.2.5** *For integers $i, j$ such that $i \geq j \geq 0$,*

$$V(i+j) = V(i) \cdot V(j) - V(i-j).$$

**Proof:** From the definition of $V(i)$ we have

$$\begin{aligned} V(i) \cdot V(j) &= (\alpha^i + \beta^i)(\alpha^j + \beta^j) \\ &= \alpha^{i+j} + \beta^{i+j} + \alpha^i \beta^j + \alpha^j \beta^i \\ &= V(i+j) + \alpha^{i-j}(\alpha\beta)^j + \beta^{i-j}(\alpha\beta)^j \\ &= V(i+j) + V(i-j). \end{aligned}$$

■

As an immediate consequence to the previous lemma, we have the following corollary.

**Corollary 6.2.6** *For all integers $i \geq 0$, $V(2i) = V(i)^2 - 2$.*

We now show that all $V(i)$ are elements of $GF(p)$.

**Lemma 6.2.7** *For $i \geq 0$, $V(i) \in GF(p)$.*

**Proof:** We prove that $V(i) \in GF(p)$ by showing that $V(i)^p = V(i)$. Consider

$$V(i)^p = (\alpha^i + \beta^i)^p = (\alpha^p)^i + (\beta^p)^i.$$

If $p \equiv 7 \pmod 8$, we know $\alpha^p = \beta$ and $\beta^p = \alpha$ by Lemma 6.2.2. Thus, $V(i)^p = \beta^i + \alpha^i = V(i)$.

If $p \equiv 1 \pmod 8$, we know $\alpha^p = \alpha$ and $\beta^p = \beta$ by Lemma 6.2.2. Thus $V(i)^p = \alpha^i + \beta^i = V(i)$. ■

Next we prove our claim about the distinctness of the $V(i)$:

**Lemma 6.2.8** *We have $V(i) \neq V(j)$ for $0 \leq i < j \leq 2^{d-1}$.*

**Proof:** Suppose $V(i) = V(j)$. Then $\alpha^i + \beta^i = \alpha^j + \beta^j$. Since $\beta = \alpha^{-1}$, we have $\alpha^i + \alpha^{-i} = \alpha^j + \alpha^{-j}$. Multiplying by $\alpha^j$ and rearranging, we get $\alpha^{2j} - \alpha^{j+i} - \alpha j - i + 1 = 0$. Factoring, we get $(\alpha^{j+i} - 1)(\alpha^{j-i} - 1) = 0$. Thus either $\alpha^{j-i} = 1$ or $\alpha^{j+i} = 1$.

Since $\alpha$ is of order $2^d$ in $GF(p)$, it follows that either $2^d \mid (j-i)$ or $2^d \mid (j+i)$. But this is impossible since $0 \leq i < j \leq 2^{d-1}$. ■

Now define the following recurrence relation:

$$\begin{aligned} U(0) &= 0 \\ U(1) &= 1 \\ U(n+1) &= (\alpha+\beta)U(n) - U(n-1) \text{ (for } n > 0) \end{aligned}$$

It is known (see Williams [99]) that the sequence $U(n)$ has the following closed form:

$$U(n) = \frac{\alpha^n - \beta^n}{\alpha - \beta}.$$

From the definitions of $V(n)$ and $U(n)$, we have the following results, which are proved in Williams [99, Chapter 4]

**Lemma 6.2.9** *For all $n \geq 0$, the following hold*

*(a)* $V(n) = 2U(n+1) - (\alpha+\beta)U(n)$;

*(b)* $U(2n) = 2U(n+1)U(n) - (\alpha+\beta)U(n)^2$;

*(c)* $U(2n+1) = U(n+1)^2 - U(n)^2$;

*(d)* $U(n+1)^2 - U(n)U(n+2) = 1$.

**Proof:** We will prove only part (a). It is worth noting that $\alpha \neq \beta$ since if $\alpha = \beta$, then $\alpha$ is of order 1 or 2; but $\alpha$ is of order $2^d \geq 8$ by definition.

$$\begin{aligned} V(n) &= \alpha^n + \beta^n \\ &= \frac{(\alpha^n + \beta^n)(\alpha - \beta)}{\alpha - \beta} \\ &= \frac{\alpha^{n+1} - \beta^{n+1} + \alpha^{n-1} - \beta^{n-1}}{\alpha - \beta} \\ &= \frac{2(\alpha^{n+1} - \beta^{n+1}) - (\alpha+\beta)(\alpha^n - \beta^n)}{\alpha - \beta} \\ &= 2U(n+1) - (\alpha+\beta)U(n). \end{aligned}$$

The remaining relationships (b)–(d) can be proved in a similar manner. ■

We now describe another relationship between $V(i)$ and $U(i)$ that allows us to prove our main theorem for this section.

**Lemma 6.2.10** *For $i \geq 0$, $V(2i+1)+2 = (\alpha+\beta+2)(U(i+1)-U(i))^2$.*

**Proof:** We have

$$\begin{aligned}
V(2i+1) &= 2U(2i+2) - (\alpha+\beta)U(2i+1) \\
&= 2(2U(i+2)U(i+1) - (\alpha+\beta)U(i+1)^2) - (\alpha+\beta)U(2i+1) \\
&= 4U(i+2)U(i+1) - 2(\alpha+\beta)U(i+1)^2 - (\alpha+\beta)(U(i+1)^2 - U(i)^2) \\
&= 4((\alpha+\beta)U(i+1) - U(i))U(i+1) - 3(\alpha+\beta)U(i+1)^2 + (\alpha+\beta)U(i)^2 \\
&= (\alpha+\beta)U(i+1)^2 - 4U(i)U(i+1) + (\alpha+\beta)U(i)^2 \\
&= (\alpha+\beta)U(i+1)^2 - 4U(i)U(i+1) + (\alpha+\beta)U(i)^2 + \\
&\quad 2U(i+1)^2 - 2U(i)U(i+2) - 2 \\
&= (\alpha+\beta)U(i+1)^2 - 4U(i)U(i+1) + (\alpha+\beta)U(i)^2 + \\
&\quad 2U(i+1)^2 - 2U(i)((\alpha+\beta)U(i+1) - U(i) - 2 \\
&= (\alpha+\beta+2)(U(i+1)^2 - 2U(i)U(i+1) + U(i)^2) - 2 \\
&= (\alpha+\beta+2)(U(i+1) - U(i))^2 - 2.
\end{aligned}$$

■

The following lemma follows directly from Lemmas 6.2.4 and 6.2.10.

**Lemma 6.2.11** *For all $i \geq 0$, there does not exist $x \in GF(p)$ such that $x^2 - 2 = V(2i+1)$.*

We now turn our attention to $V(i)$ for even values of $i$.

**Lemma 6.2.12** $V(2^{d-1}) = -2$.

**Proof:**

$$\begin{aligned}
V(2^{d-1}) &= \gamma^{(p-1)\frac{(p+1)}{2^d}2^{d-1}} + \gamma^{(1-p)\frac{(p+1)}{2^d}2^{d-1}} \\
&= \gamma^{(p-1)\frac{(p+1)}{2}} + \gamma^{(1-p)\frac{(p+1)}{2}} \\
&= \gamma^{\frac{p^2-1}{2}} + \gamma^{\frac{1-p^2}{2}} \\
&= -1 + -1 \\
&= -2.
\end{aligned}$$

■

From the previous lemma, the following corollary is immediate.

**Corollary 6.2.13** $V(2^{d-2}) = 0$.

We know by Corollary 6.2.6 that $V(2i) = V(i)^2 - 2$. Thus, in the digraph $G^R_{p,2}$, we view $V(i)$ as a "child" of $V(2i)$ in a binary tree sense. That is, for $1 \leq i \leq 2^{d-2}$ all $V(2i)$ have children. We now prove that $V(2i)$ has a second child in $G^R_{p,2}$.

**Lemma 6.2.14** $V(2i) = V(2^{d-1} - i)^2 - 2$.

**Proof:** We have

$$\begin{aligned} V(2^{d-1} - i)^2 &= (\alpha^{2^{d-1}-i} + \beta^{2^{d-1}-i})^2 \\ &= \alpha^{2^d-2i} + \beta^{2^d-2i} + 2 \\ &= \gamma^{p^2-1}\alpha^{-2i} + \gamma^{-(p^2-1)}\beta^{-2i} \\ &= V(2i) + 2. \end{aligned}$$

■

By Lemma 6.2.11, we know that all $V(k)$ where $k$ is odd have no children. Therefore, $V(2i+1)$ forms the leaves of the binary tree with height $2^{d-2}$ which has a root vertex of $V(2^{d-2}) = 0$.

In summary, by Lemmas 6.2.11, 6.2.14, 6.2.8 and Corollary 6.2.13, we have proven Theorem 6.2.1. ■

It should be noted that this result does not resolve the D. H. Lehmer's query about the sign in $S_{p-2} \equiv \pm 2^{(p+1)/2} \pmod{2^p - 1}$ (see Guy [43, p. 9]), since our result does not classify which "side" of the tree the (starting) element 4 resides on. Rather, we give a closed form for the elements of the tree, but the actual computation of $S_{p-2}$ cannot be short-circuited using our results.

We have the following Corollary that extends this result to other quadratic maps.

**Corollary 6.2.15** *If we have the map* $x \to Ax^2 + Bx + C$ *where*

$$\frac{B}{2} - \frac{B^2 - 4AC}{4} = -2,$$

*then we have the same binary tree structure in the directed graph formed by this map over* $GF(p)$ *as in Theorem 6.2.1.*

**Proof:** See Exercise 9.5 from Holmgren [46] to see that the map $x \to x^2 - 2$ has the conjugate function given above. ■

As an example application of the previous Corollary, the digraph induced with the map $4x^2 + 12x + 7$ over $GF(p)$ will contain a binary tree component as outlined above.

To conclude this section, we present an example which traces through all of the steps in the proof of Theorem 6.2.1.

Let $p = 191 = 3 \cdot 2^6 - 1$. Take the primitive polynomial over $GF(191)[x]$ to be $f(x) = x^2 + 9x + 21$. We have as roots $\gamma = x$ and $\delta = 190x + 182$. Since $p \equiv 7 \pmod 8$, we take $\alpha = \gamma^{k(p-1)} = x^{3 \cdot 190} = 123x + 136$. It follows that $\beta = 68x + 175$. Therefore $V(1) = \alpha + \beta = 120$. The remaining $V(i)$ values are illustrated in Figure 6.1. It is worth noting that Figure 6.1 corresponds to the same graph as in Figure 5.4.

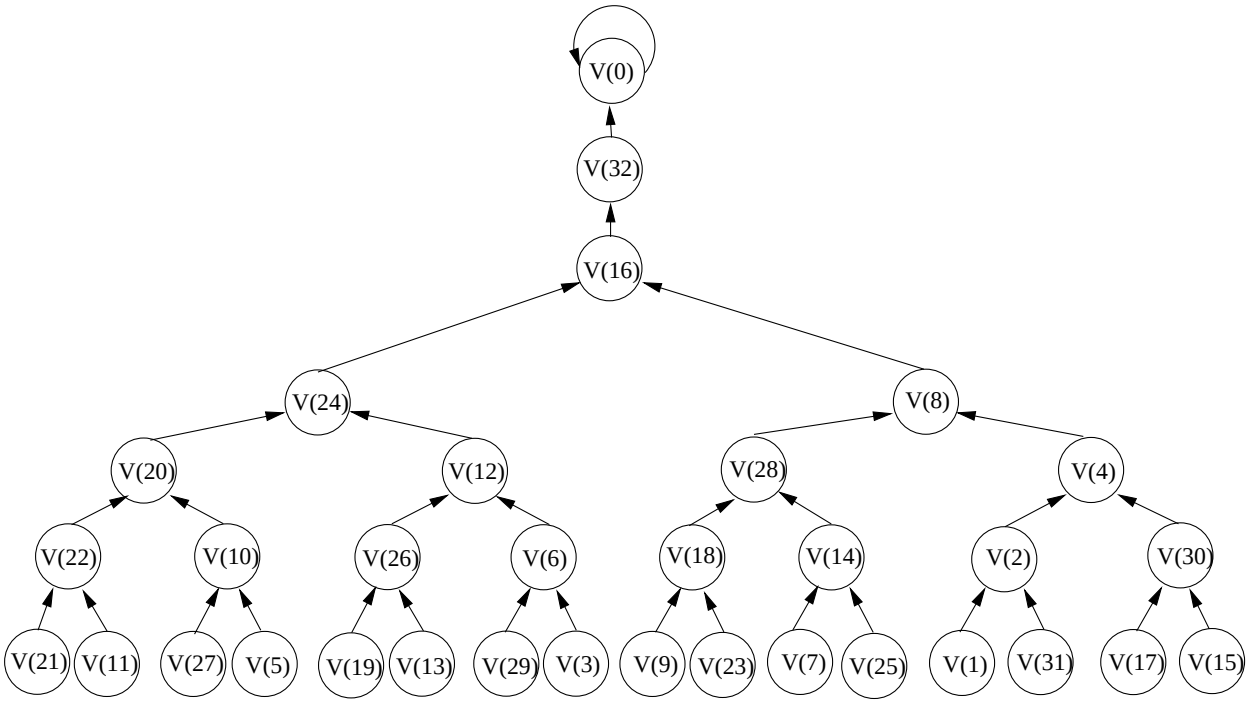


Figure 6.1: The digraph $G_{p,2}$ with $p = 191$ and $f(x) = x^2 + 9x + 21$

## 6.3 Conclusions

We have provided an alternative version of the underlying structure of the digraph formed by the iteration $x^2 \to x^2 - 2$.

We now move from the robustness analysis of primality tests to the robustness results for an algorithm that computes the order of an element in a group.

# Chapter 7

# Robustness of computing order of an element in a group

## 7.1 Introduction

In this chapter, we consider the robustness of algorithms for computing the order of an element in a group under a black-box model. We begin by formalizing the problem of computing the order of an element in a group, and then proceed to prove robustness results using Markov chains.

Suppose we have an additive group $G$ and an element $x \in G$. We wish to find the order of $x$: that is, we wish to find the least positive integer $k$ such that $x * k = e$, where $e$ is the identity element of group $G$. The algorithm to do this (Algorithm 7.1) will repeatedly add the element $x$ to itself, terminating at step $k$ iff $\sum_{i=1}^{k} x = e$. We will assume $|G| = m$.

As outlined in previous chapters, we will assume that an error can occur to the value $t$ during execution. That is, at each step, the value $t$ in Algorithm 7.1 can change uniformly to any value in group $G$ with probability $\epsilon$. We will quantify the probability that the algorithm computes the correct element order under this error model.

We will assume $G = \mathbb{Z}/(m)$, though the results in this chapter can be generalized to any additive group of order $m$.

We consider the values that $t$ may contain during execution of Algorithm 7.1. If we view each value of $t$) as "states" that Algorithm 7.1 can be in, we can model the algorithm execution by way of a Markov chain. Specifically, we consider the

**Input:** $x \in G$
$t \leftarrow e$
$i \leftarrow 0$
**repeat**
  $t \leftarrow t + x$
  $i \leftarrow i + 1$
**until** $t = e$
**print** "Order of $x$ is $i$"

Algorithm 7.1: Simple Group Order Algorithm

following matrix $M$ that is of size $(m+1) \times (m+1)$ (with the index of each position shown):

$$\begin{array}{c|c|ccccccc|} & 0 & 1 & 2 & 3 & \cdots & m-1 & m \\ 0 & 0 & 1-\frac{m-1}{m}\epsilon & \frac{\epsilon}{m} & \frac{\epsilon}{m} & \cdots & \frac{\epsilon}{m} & \frac{\epsilon}{m} \\ 1 & 0 & \frac{\epsilon}{m} & 1-\frac{m-1}{m}\epsilon & \frac{\epsilon}{m} & \cdots & \frac{\epsilon}{m} & \frac{\epsilon}{m} \\ 2 & 0 & \frac{\epsilon}{m} & \frac{\epsilon}{m} & 1-\frac{m-1}{m}\epsilon & \cdots & \frac{\epsilon}{m} & \frac{\epsilon}{m} \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ m-1 & 0 & \frac{\epsilon}{m} & \frac{\epsilon}{m} & \frac{\epsilon}{m} & \cdots & \frac{\epsilon}{m} & 1-\frac{m-1}{m}\epsilon \\ m & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ \end{array}.$$

We now describe the details of what this matrix actually represents and how we will use it in computing robustness results.

The crucial definition is that the element at position $M[i, j]$ $(0 \leq i, j < m)$ is the probability that variable $t$ had its value of $i$ changed to the value $j$ after one iteration of the loop of Algorithm 7.1. The elements in position $M[i, m]$ $(0 \leq i < m)$ represent the probability that variable $t$ had its value of $i$ changed to the loop-exit condition value 0. The elements in position $M[m, j]$ $(0 \leq j \leq m)$ represent the probability that variable $t$ contains the value 0, which is the starting condition. In other words, the last row (row $m$) represents starting in the "stopped state." That is, the algorithm has terminated on an earlier step due to its internal total $t$ being element $e$. In terms of Markov chain terminology, state $m$ is an *absorbing* state.

We now make a few observations regarding the elements of matrix $M$. One key point to observe is that there are $m + 1$ states. We create the initial state 0, which is represented by column 0 in the matrix $M$, with the other columns representing elements in the group, and specifically, column $m$ representing element $e$. The values in column 0 are 0, since we guarantee that this special start state is never entered again.

Notice that the row sums are 1, since we are guaranteed to be in *some* non-starting state after each step of the algorithm.

In terms of robustness results, we wish to determine the probability that Algorithm 7.1 terminates at step $m$ and not before, when the order of the element is $m$. We will assume the input element is 1 (which has order $m$): if any other element which has order $m$ is chosen, the same arithmetic derivations can yield the result. If the element we wish to consider has order less than $m$, our computations would need to be modified to examine all possible divisors of $m$.

In order to determine the probability that Algorithm 7.1 terminates at exactly step $m$, we first make a few general observations.

First, notice that $M^k[i,j]$ is the probability that we start in state $i$ and end in state $j$ after exactly $k$ steps, provided the input state $i \neq m$. We wish to determine the probability that we start in the starting state (corresponding to row 0) and end in state $m$ (i.e., at the identity element) in *exactly* $k$ steps. Our task is to determine

$$(M^k - M^{k-1})[0,m]$$

which can be factored as

$$\left(M^{k-1}(M-I)\right)[0,m].$$

(If an element other than 1, say $x \in G$, which has order $k$, is given as input, the previous formula changes to $\left(M^{k-1}(M-I)\right)[x,m]$.)

We prove the following lemma, which is one of the main results of this chapter.

**Lemma 7.1.1** *The entry in row 0, column $j$ $(0 \leq j \leq m)$ of the matrix $M^k$ $(2 \leq k \leq m)$ is*

$$M^k[0,j] = \begin{cases} 0, & \text{if } j = 0; \\ \frac{j\epsilon}{m} - \frac{j^3+3(m-k)j^2+(3k^2-3m-1)j}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3), & \text{if } 1 \leq j < k; \\ 1 - \frac{k(m-1)}{m}\epsilon + \frac{k(k-1)(m^2-m-1)/2-k(k-1)(k-2)/6}{m^2}\epsilon^2 + O(m^2\epsilon^3), & \text{if } j = k; \\ \frac{k\epsilon}{m} - \frac{k^3+3(m-k)k^2+(3k^2-3m-1)k}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3), & \text{if } k < j < m; \\ \frac{k(k+1)}{2m}\epsilon - \frac{k^4+(4m+2)k^3-k^2-(4m+2)k}{24}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3), & \text{if } j = m. \end{cases}$$

**Proof:** By induction on $k$. One can easily verify that the base case $k = 2$ is satisfied.

Suppose that the lemma holds for $k = q$, $q \geq 2$. To show the lemma holds for $k = q+1$, we break this into five cases:

**Case 1:** Position 0 to position $q$ in matrix $M^{q+1}$.
Clearly, position 0 is always 0. We consider position $i$, where $1 \leq j \leq q$.

Notice that column $j$ ($2 \leq j \leq q$) in matrix $M$ has $\frac{\epsilon}{m}$ in positions 0 through $j-2$, $1-\frac{m-1}{m}\epsilon$ in position $j-1$, $\frac{\epsilon}{m}$ in positions $j$ to $m-1$, and 0 in position $m$.

Computing the dot product of row 0 of matrix $M^q$ with column $j$ in matrix $M$ yields the sum $A+B+C+D+E$, where

$$\begin{aligned}
A &= \sum_{c=1}^{j-2}\left(\frac{c\epsilon}{m} - \frac{c^3+3(m-q)c^2+(3q^2-3m-1)c}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3)\right)\frac{\epsilon}{m},\\
B &= \frac{(j-1)\epsilon}{m} - \frac{(j-1)^3+3(m-q)(j-1)^2+(3q^2-3m-1)(j-1)}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3)\\
&\quad -\frac{(m-1)\epsilon}{m}\left(\frac{(j-1)\epsilon}{m} - \frac{(j-1)^3+3(m-q)(j-1)^2+(3q^2-3m-1)(j-1)}{6}\frac{\epsilon^2}{m^2}\right),\\
C &= \sum_{c=j}^{q-1}\left(\frac{c\epsilon}{m} - \frac{c^3+3(m-q)c^2+(3q^2-3m-1)c}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3)\right)\frac{\epsilon}{m},\\
D &= \left(1 - \frac{q(m-1)}{m}\epsilon + \frac{q(q-1)(m^2-m-1)/2 - q(q-1)(q-2)/6}{m^2}\epsilon^2 + O(m^2\epsilon^3)\right)\frac{\epsilon}{m},\\
E &= \sum_{c=q+1}^{m-1}\left(\frac{q\epsilon}{m} - \frac{q^3+3(m-q)q^2+(3q^2-3m-1)q}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3)\right)\frac{\epsilon}{m}.
\end{aligned}$$

It is worth noting that if $j=q$, the value of $C$ is 0.

Computing these sums (using Maple), we get the desired result for the $\frac{\epsilon}{m}$ and $\frac{\epsilon^2}{m}$ terms. For the $\epsilon^3$ terms, we consider each of $A, B, C, D$ and $E$ in turn.

For $A$, since $j-2 \in O(m)$, the dominant $\epsilon^3$ term is $\sum_{c=1}^{j-2} c^3\epsilon^3/m^3$, which is $O(m^4\epsilon^3/m^3) = O(m\epsilon^3)$ which is $O(m^2\epsilon^3)$.

For $B$, we note that $j-1 \in O(m)$ and $q \in O(m)$, so the dominant $\epsilon^3$ term is $(j-1)^3\frac{\epsilon(m-1)}{m}\frac{\epsilon^2}{m^2} \in O(m\epsilon^3)$ which is $O(m^2\epsilon^3)$.

For $C$, we have the same result as $A$, since $q-1 \in O(m)$.

For $D$, we have $q \in O(m)$ and thus the dominant $\epsilon^3$ term is $q(q-1)m^2\epsilon^3/m^3 \in O(m\epsilon^3)$ which is $O(m^2\epsilon^3)$.

For $E$, we note that each term in the sum does not depend on $c$, and thus we have the dominant $\epsilon^3$ term being $(m-q-2)mq^2\epsilon^3/m^3 \in O(m\epsilon^3)$, which is $O(m^2\epsilon^3)$.

**Case 2:** Position $q+1$ in matrix $M^{q+1}$.
As in the previous case, notice that column $q+1$ in matrix $M$ has $\frac{\epsilon}{m}$ in positions 0 to $q-1$, $1-\frac{m-1}{m}\epsilon$ in position $q$, $\frac{\epsilon}{m}$ in positions $q+1$ to $m-1$, and 0 in position $m$.

We compute the dot product of row 0 of matrix $M^q$ with column $q$ in matrix $M$, to yield the sum $A+B+C$ where

$$\begin{aligned}
A &= \sum_{c=1}^{q-1}\left(\frac{c\epsilon}{m}-\frac{c^3+3(m-q)c^2+(3q^2-3m-1)c}{6}\frac{\epsilon^2}{m^2}+O(m^2\epsilon^3)\right)\left(\frac{\epsilon}{m}\right),\\
B &= \left(1-\frac{q(m-1)m}{m}\epsilon+\frac{q(q-1)(m^2-m-1)/2-q(q-1)(q-2)/6}{m^2}\epsilon^2+O(m^2\epsilon^3)\right)\\
&\quad\cdot\left(1-\frac{m-1}{m}\epsilon\right),\\
C &= \sum_{c=q+1}^{m-1}\left(\frac{q\epsilon}{m}-\frac{q^3+3(m-q)q^2+(3q^2-3m-1)q}{6}\frac{\epsilon^2}{m^2}+O(m^2\epsilon^3)\right)\frac{\epsilon}{m}.
\end{aligned}$$

The coefficients of $\epsilon^0$, $\epsilon$ and $\epsilon^2$ can be verified correct by hand.

For the coefficient of $\epsilon^3$, we perform case analysis as we performed in Case 1 above.

The value of $A$ in this case is identical to the value for $A$ in Case 1, and thus, the relevant bounding term is $O(m^2\epsilon^3)$.

Looking at $B$, we have the $\epsilon^3$ term is dominated by $q(q-1)m^2(m-1)\epsilon^3/m^3$, and since $q\in O(m)$, this expression is $O(m^2\epsilon^3)$. (It is worth noting that this is the first time the bound actually required $m^2\epsilon^3$ as the bounding term.)

For $C$, this formula is analogous to the formula for $E$ in case 1, which had the $\epsilon^3$ term bounded by $O(m^2\epsilon^3)$.

Thus, in the sum $A+B+C$, the $\epsilon^3$ term is bounded by $O(m^2\epsilon^3)$.

**Case 3:** Position $q+2$ in matrix $M^{q+1}$.

In matrix $M$, column $q+2$, there is $\frac{\epsilon}{m}$ in positions 0 to $q$, $1-\frac{m-1}{m}\epsilon$ in position $q+1$, $\frac{\epsilon}{m}$ in positions $q+2$ to $m-1$, and 0 in position $m$.

As in the previous cases, we compute the dot product of row 0 in matrix $M^q$

and column $q+2$ in matrix $M$. The result sum can be written as

$$\begin{aligned}
A &= \sum_{c=1}^{q-1} \left( \frac{c\epsilon}{m} - \frac{c^3 + 3(m-q)c^2 + (3q^2 - 3m - 1)c}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}, \\
B &= (1 - \frac{q(m-1)}{m}\epsilon + \frac{q(q-1)(m^2 - m - 1)/2 - q(q-1)(q-2)/6}{m^2}\epsilon^2 + O(m^2\epsilon^3))\frac{\epsilon}{m}, \\
C &= \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \left( 1 - \frac{m-1}{m}\epsilon \right), \\
D &= \sum_{c=q+2}^{m-1} \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}.
\end{aligned}$$

The $\epsilon$ and $\epsilon^2$ terms can be verified correct by hand.

For the $\epsilon^3$ term, $A$ contributes $\sum_{c=1}^{q-1} c^3\epsilon^3/m^3$, and since $q - 1 \in O(m)$, this $A$ term contributes $O(m\epsilon^3)$.

$B$ contributes $q(q-1)m^2\epsilon^3/m^3$, and again, since $q \in O(m)$, the $B$ term provides $O(m\epsilon^3)$.

The $C$ term contributes $q^3(m-1)\epsilon^3/m^3 \in O(m^2\epsilon^3)$.

The $D$ term contributes $\sum_{c=q+2}^{m-1} q^3\epsilon^3/m^3$ and since $m - 1 - (q+2) \in O(m)$, we can bound the $\epsilon^3$ term by $O(mq^3\epsilon^3/m^2) \in O(m^2\epsilon^3)$.

Thus, the sum $A + B + C + D$ contains $O(m^2\epsilon^3)$ as the $\epsilon^3$ term, as required.

**Case 4:** Position $q+3$ to $m-1$ in matrix $M^{q+1}$.

In matrix $M$, column $j$ ($q + 3 \leq j \leq m - 1$), there is $\frac{\epsilon}{m}$ in positions 0 to $j - 2$, $1 - \frac{m-1}{m}\epsilon$ in position $j - 1$, $\frac{\epsilon}{m}$ in positions $j$ to $m - 1$, and 0 in position $m$.

As in the previous cases, we compute the dot product of row 0 of matrix $M^q$

with column $j$ ($q+2 \leq j \leq m-1$), to yield the sum $A+B+C+D$, where

$$\begin{aligned}
A &= \sum_{c=1}^{q-1} \left( \frac{c\epsilon}{m} - \frac{c^3 + 3(m-q)c^2 + (3q^2 - 3m - 1)c}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \left( \frac{\epsilon}{m} \right), \\
B &= (1 - \frac{q(m-1)}{m}\epsilon + \frac{q(q-1)(m^2 - m - 1)/2 - q(q-1)(q-2)/6}{m^2}\epsilon^3 + O(m^2\epsilon^3))\frac{\epsilon}{m}, \\
C &= \sum_{c=q+1}^{j-2} \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}, \\
D &= \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \left( 1 - \frac{m-1}{m}\epsilon \right), \\
E &= \sum_{c=j}^{m-1} \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}.
\end{aligned}$$

The $\epsilon$ and $\epsilon^2$ terms can be verified by hand.

To bound the $\epsilon^3$ term, notice that $A$ and $B$ in this case are identical to $A$ and $B$ in case 3. Thus, those terms are bounded by $O(m\epsilon^3)$.

Since $j \in O(m)$ and $q \in O(m)$, it follows that $C$ contributes $O(mq^3\epsilon^3/m^3) \in O(m\epsilon^3)$.

The $D$ term contributes $q^3(m-1)\epsilon^3/m^3 \in O(m\epsilon^3)$.

For the $E$ term, we have $(m-1-j)q^3\epsilon^3/m^3 \in O(m\epsilon^3)$.

Thus, the sum $A+B+C+D+E$, we can bound the $\epsilon^3$ term by $O(m^2\epsilon^3)$ (though it should be noted that $O(m\epsilon^3)$ is sufficient in this case).

**Case 5:** Position $m$.
Consider column $m$ in matrix $M$. This column has $\frac{\epsilon}{m}$ in positions 0 to $m-2$, $1 - \frac{m-1}{m}\epsilon$ at position $m-1$ and 1 in position $m$.

$$
\begin{aligned}
A &= \sum_{c=1}^{q-1} \left( \frac{c\epsilon}{m} - \frac{c^3 + 3(m-q)c^2 + (3q^2 - 3m - 1)c}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}, \\
B &= (1 - \frac{q(m-1)}{m}\epsilon + \frac{q(q-1)(m^2 - m - 1)/2 - q(q-1)(q-2)/6}{m^2}\epsilon^2 + O(m^2\epsilon^3))\frac{\epsilon}{m}, \\
C &= \sum_{c=q+1}^{m-2} \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \frac{\epsilon}{m}, \\
D &= \left( \frac{q\epsilon}{m} - \frac{q^3 + 3(m-q)q^2 + (3q^2 - 3m - 1)q}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) \right) \left( 1 - \frac{m-1}{m}\epsilon \right), \\
E &= \frac{q(q+1)}{2m}\epsilon - \frac{q^4 + (4m+2)q^3 - q^2 - (4m+2)q}{24} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3).
\end{aligned}
$$

This sum yields the correct coefficients for both the $\epsilon$ and $\epsilon^2$ terms, which can be verified by hand.

For the $\epsilon^3$ term, the $A$, $B$, $C$ and $D$ terms where analyzed (with minor variation) in case 4, and all these terms provided an $O(m^2\epsilon^3)$ factor. For the $E$ term, there is only a $O(m^2\epsilon^3)$ term. Thus, in sum we have an $O(m^2\epsilon^3)$ bound, which proves the lemma. ■

From Lemma 7.1.1, we can see that each element of the first row of $M^{m-1}$ can be described as $M^{m-1}[0, j]$ with values

$$
\begin{cases}
0 & \text{if } j = 0, \\
\frac{j\epsilon}{m} - \frac{j^3 + 3j^2 + (3(m-1)^2 - 3m - 1)j}{6} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) & \text{if } 1 \le j < m-1, \\
1 - \frac{(m-1)^2}{m}\epsilon + \frac{(m-1)(m-2)(m^2 - m - 1)/2 - (m-1)(m-2)(m-3)/6}{m^2}\epsilon^2 + O(m^2\epsilon^3) & \text{if } j = m-1, \\
\frac{(m-1)m}{2m}\epsilon - \frac{(m-1)^4 + (4m+2)(m-1)^3 - (m-1)^2 - (4m+2)(m-1)}{24} \frac{\epsilon^2}{m^2} + O(m^2\epsilon^3) & \text{if } j = m.
\end{cases}
$$

Now, when we take the product of $M^{m-1}$ and $M - I$, we only concern ourselves with the element in row 0, column $m$. Thus, multiply the first row of $M^{m-1}$ (described above) by the last column of $M - I$ to obtain $(M^m - M^{m-1})[0, m]$.

We get

$$\begin{aligned}(M^m - M^{m-1})[0,m] &= \sum_{j=1}^{m-2}\left(\frac{j\epsilon}{m} - \frac{j^3+3j^2+(3(m-1)^2-3m-1)j}{6}\frac{\epsilon^2}{m^2} + O(m^2\epsilon^3)\right)\frac{\epsilon}{m} + \\ &\quad \left(1-\frac{m-1}{m}\epsilon\right)\cdot \\ &\quad \left(1-\frac{(m-1)^2}{m}\epsilon\right. \\ &\quad + \frac{(m-1)(m-2)(m^2-m-1)/2-(m-1)(m-2)(m-3)/6}{m^2}\epsilon^2 \\ &\quad \left. + O(m^2\epsilon^3)\right) \\ &= 1-(m-1)\epsilon + \frac{3m^3-7m^2+3m+1}{6m}\epsilon^2 + O(m^2\epsilon^3).\end{aligned}$$

We summarize this result in the following theorem, which handles the "correct output" case of robustness for Algorithm 7.1.

**Theorem 7.1.2** *Let $G$ be a group of order $m$. Suppose $x \in G$ where the order of $x$ is $|G|$. Let $\epsilon$ be the probability that Algorithm 7.1 alters its stored element $t$ to a random element in $G$. Under this error model, the probability that the Algorithm 7.1 correctly computes the group order of $x$ is*

$$p_c(m) = 1-(m-1)\epsilon + (m^2/2 - 7m/6 + 1/2 + 1/(6m))\epsilon^2 + O(m^2\epsilon^3).$$

In terms of the robustness analysis for Algorithm 7.1, we have two other cases to consider, based on the definition of robustness outlined in Chapter 1.

Before we analyze the incorrect but feasible and incorrect and infeasible cases, we will need the following lemma.

**Lemma 7.1.3** *The entry in matrix $(M^k - M^{k-1})[0,m]$ $(2 \le k < m)$ is*

$$\frac{k}{m}\epsilon - \frac{k^3+3mk^2-3mk-k}{6m^2}\epsilon^2 + O(m^2\epsilon^3).$$

**Proof:** Apply Lemma 7.1.1 twice. ∎

We are now ready to prove the final two cases for the robustness analysis.

We will define an incorrect and infeasible output of Algorithm 7.1 to be a group order which does not divide $|G|$. Similarly, we define an incorrect but feasible output of Algorithm 7.1 to be a group order which divides $|G|$ but is strictly less than $|G|$.

We will use these definitions in the following two theorems.

**Theorem 7.1.4** *Let $G$ be a group of order $m$. Suppose $x \in G$ where the order of $x$ is $|G|$. Let $\epsilon$ be the probability that Algorithm 7.1 alters its stored element $t$ to a random element in $G$. Under this error model, the probability that Algorithm 7.1 computes an incorrect but feasible group order of $x$ is*

$$p_f(m) = (\sigma(m) - m)\frac{\epsilon}{m} + O(m\epsilon^2).$$

**Proof:** First, we require some number theoretic results. We use the standard notation of

$$\sigma_i(m) = \sum_{\substack{1 \le k \le m \\ k|m}} k^i,$$

and we use $\sigma(m)$ to denote $\sigma_1(m)$.

From Lemma 7.1.3, we have

$$\begin{aligned} p_f(m) &= \sum_{\substack{1 \le k < m \\ k|m}} \left( \frac{k}{m}\epsilon - \frac{k^3 + 3mk^2 - 3mk - k}{6m^2}\epsilon^2 + O(m^2\epsilon^3) \right) \\ &= (\sigma_1(m) - m)\frac{\epsilon}{m} \\ &\quad + \frac{(\sigma_3(m) - m^3 + 3m(\sigma_2(m) - m^2) - (3m+1)(\sigma_1(m) - m)}{6m^2}\epsilon^2 + O(m^2\epsilon^3). \end{aligned}$$

From Gronwall [41], we know that $\sigma_2(m) \in O(m^2)$ and $\sigma_3(m) \in O(m^3)$. Thus we can simplify the above expression to write $p_f(m) = (\sigma_1(m) - m)\frac{\epsilon}{m} + O(m\epsilon^2)$, which proves the result. ■

Next, we complete the quantification of the error under this algorithm.

**Theorem 7.1.5** *Let $G$ be a group of order $m$. Suppose $x \in G$ where the order of $x$ is $|G|$. Let $\epsilon$ be the probability that Algorithm 7.1 alters its stored element $t$ to a*

*random element in $G$. Under this error model, the probability that Algorithm 7.1 computes an incorrect and infeasible group order of $x$ is*

$$p_i(m) = (m^2 - \sigma(m))\frac{\epsilon}{m} + O(m\epsilon^2).$$

**Proof:** Use the results of Theorems 7.1.2 and 7.1.4 and the fact that $p_i(m) = 1 - p_c(m) - p_f(m)$. ∎

We now prove robustness results for Algorithm 7.1, which is the second main result of this chapter.

**Theorem 7.1.6** *Let $G$ be a group of order $m$. Suppose $x \in G$ where $\gcd(|x|, m) = 1$. Let $\epsilon$ be the probability that the internal black box used in Algorithm 7.1 produces a random element from the range $1..m$. Under this error model, Algorithm 7.1 is robust, provided $\epsilon < \frac{b}{m}$ for a constant $0 < b < 1$ and $m$ sufficiently large.*

**Proof:** From Section 1.3, we know we must show two things to prove robustness. First, we must show that $p_c(m) > b$ for some constant $C > 0$. Second, we must show that $p_i(m) > p_f(m)$.

To show that $p_c(m) > C$, we have

$$\begin{aligned} p_c(m) &= 1 - (m-1)\epsilon + (m^2/2 - 7m/6 + 1/2 + 1/(6m))\epsilon^2 + O(m^2\epsilon^3) \\ &> 1 - (m-1)\epsilon \end{aligned}$$

if $m$ is sufficiently large, since the $O(m^2\epsilon^3)$ becomes vanishingly small. Continuing, we have

$$\begin{aligned} p_c(m) &> 1 - (m-1)\epsilon \\ &> 1 - \frac{b(m-1)}{m} \\ &= 1 - b + \frac{b}{m} \\ &> 1 - b \\ &> 0, \end{aligned}$$

since we know that the constant $0 < b < 1$. Thus $C = 1 - b$ and the result is shown in this case.

We now turn our attention to showing that $p_i(m) > p_f(m)$. We will ignore the difference in the lower order terms since $O(m\epsilon^2)$ becomes vanishing small for sufficiently large $m$, and thus we have

$$\begin{aligned} p_i(m) - p_f(m) &= (m^2 - \sigma(m))\frac{\epsilon}{m} - (\sigma(m) - m)\frac{\epsilon}{m} \\ &= (m^2 + m - 2\sigma(m))\frac{\epsilon}{m}. \end{aligned}$$

Notice that $\sigma(m) < m(m+1)/2$ if $m > 2$, and thus, $p_i(m) > p_f(m)$. ■

# Chapter 8

# Conclusions and Further Work

## 8.1 Conclusions

In this thesis, we have formulated an error model for computation that mimics the types of errors that may alter CPU performance. By applying this error model on various algorithms, we have new and interesting results due to two consequences. The first consequence is that probabilistic arguments allow us to consider the robustness of algorithms under various error assumptions, allowing some reassurance of the correctness of the output. A second consequence is that, in proving the probabilistic results concerning algorithms, surprising and interesting algebraic structures emerge.

## 8.2 Further work

There are three main areas of open research problems that are worth pursuing.

The first area is an open problem from number theory due to Lehmer (as outlined in Guy [43, page 9]). Lehmer notes that the $p-2$nd squaring step yields either a value $2^{((p+1)/2)} \bmod 2^p - 1$ or $-2^{((p+1)/2)} \bmod 2^p - 1$. Determining whether the value is the positive or negative value is still an open problem. Our digraph representation and algebraic definition of the elements in the digraph does not seem to help answer this problem. Determining a self-checking mechanism for the Lucas-Lehmer test or Mersenne test, seems even more difficult, since to do this, it would appear at each "level" in the binary tree in the digraph would need to be categorized in an easily computable and comparable manner.

A second area of further work would be to apply the analysis of Chapters 6 and 4 to other primality tests, such as the Solovay-Strassen or Miller-Rabin primality tests (see Bach and Shallit [5] for details about these algorithms).

A final area of further work would be finding the maximum and minimum value from a set of integers simultaneously, where queries on the set may be erroneous. Ravikumar, Ganesan and Lakshmanan [79] find the minimum number of queries with a fixed number of erroneous queries for an algorithm that determines the maximum value. Combining this result with the $\frac{3}{2}n - \frac{3}{2}$ algorithm for finding the maximum and minimum simultaneously (which is outlined in, for example, Basse [6]) is an unsolved problem.

# References


[1] G. M. Adel'son-Vel'skii and E. M. Landis, "An algorithm for the organization of information", *Soviet Math. Doklady* **3** (1962) 1259–1262. 3

[2] M. Adler et. al. "Selection in the presence of noise: The design of playoff schemes", *Proc. Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (1995) 564–572. 3

[3] L. Alonso et. al. "Quicksort with unreliable comparisons: a probabilistic analysis", *Combinatorics, Probability and Computing* **34** (2004), 419–449.

[4] J. A. Aslam and A. Dhagat, "Searching in the presence of linearly bounded errors", *Proc. Twenty-Third Annual ACM Symposium on Theory of Computing (STOC)* (1991) 486–493. 3

[5] E. Bach and J. Shallit, *Algorithmic Number Theory, Volume 1: Efficient Algorithms*, MIT Press, 1996. 25, 72, 73, 75, 79, 106, 128

[6] S. Baase, *Computer Algorithms: Introduction to Design and Analysis* (2nd ed.) Addison-Wesley, Boston, Mass., 1991. 128

[7] A. Batra and P. Morton, "Algebraic dynamics of polynomial maps on the algebraic closure of a finite field (I)", *Rocky Mt. J. Math* **24** (1994), 453–481. 80

[8] A. Batra and P. Morton, "Algebraic dynamics of polynomial maps on the algebraic closure of a finite field (II)", *Rocky Mt. J. Math* **24** (1994), 905–932. 80

[9] S. W. Bent, D.D. Sleator and R. E. Tarjan, "Biased search trees", *SIAM J. Comput.* **14** (1985), 545–568.

[10] E. L. Blanton, S.P. Hurd and J.S. McCranie, "On the digraph defined by squaring mod $m$, when $m$ has primitive roots", *Congress. Num.* **82** (1991), 167–177. 48, 53

[11] E.L. Blanton, S. P. Hurd and J.S. McCranie, "On a digraph defined by squaring modulo $n$", *Fib. Quart.* **30** (1992), 322–333. 48, 53

[12] L. Blum, M. Blum and M. Shub, "A simple unpredictable pseudo-random number generator", *SIAM J. Comput.* **15**, (1986) 364–381. 49

[13] M. Blum and S. Kannan, "Designing programs that check their work," *J. ACM* **42**, 269–291 (1995). 5

[14] M. Blum, M. Luby and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *J. Comp. Sys. Sci* **47**, 549–595 (1993). 5

[15] M. Blum and H. Wasserman, "Reflections on the Pentium division bug", *IEEE Trans. Comput.* **45** 385–393 (1996). 2

[16] R.S. Borgstrom and S.R. Kosaraju, "Comparison-based search in the presense of errors", *Proc. Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC)* (1993), 130–136. 3

[17] J.J. Brennan and B. Geist, "Analysis of iterated modular exponentiation: The orbits of $x^\alpha \pmod N$", *Designs, Codes and Cryptography* **30** (1998), 229–245. 49

[18] R.P. Brent, "An improved Monte Carlo Factorization Algorithm", *BIT* **20** (1980), 176–184. 20

[19] A. Z. Broder, *Weighted random mappings; properties and applications*. PhD thesis, Department of Computer Science, Stanford University, May 1985. Technical Report STAN-CS-85-1054.

[20] A. Cataldo, "IBM moves to protect DRAM from cosmic invaders", *EE Times* (1998). Available: `http://www.eetimes.com/news/98/1012news/ibm.html`. 2

[21] G. Chassé. *Applications d'un corps fini dans lui-même*, Vol. 149. Université de Rennes I U.E.R. de Mathématiques et Informatique, Rennes, 1984. Dissertation, Université de Rennes I, Rennes, 1984. 48

[22] G. Chassé. "Applications d'un corps fini dans lui-même", In *Algebra Colloquium (Rennes, 1985)*, Univ. Rennes I, Rennes (1985), 207–219. 48

[23] G. Chassé. "Combinatorial cycles of a polynomial map over a commutative field", *Discrete Math.* **61** (1986), 21–26. 48

[24] W.-S. Chou and I. Shparlinski, "On the cycle structure of repeated exponentiation modulo a prime", *J. Number Theory* **107** (2004), 345–356. 75, 102

[25] G. Cooper and B. Ravikumar, "Fibonacci search algorithm for finding extremum using unary predicates" *Foundations of Computer Science Conference* **2005** 59–63.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms* (2nd ed.), McGraw-Hill, Boston, Mass., 2001. 25

[27] A. Czumaj and C. Sohler, "Soft kinetic data structures", *Symposium on Discrete Algorithms* (2001), 865–872. 3

[28] H. Davenport, *Multiplicative Number Theory* (2nd ed.), Springer, New York, 1980. 75

[29] L. Devroye, "A note on the height of binary search trees", *Journal of the ACM* **33** (1986), 489–498.

[30] B. Delyon and O. Maler, "On the effects of noise and speed on computations", *Theor. Comp. Sci.* **129** (1994), 279–291. 3

[31] F. Ergün et. al., "Spot-checkers", *J. Comput. System Sci.* **60** (2000) , 717–751. 5

[32] U. Feige et al., "Computing with noisy information", *SIAM J. Computing* **23** (1990), 1001–1018. 3

[33] D. F. Ferguson and John W. Wrench, Jr. *Mathematical Tables and Other Aids to Computation*, Vol. 3, No. 21. (Jan., 1948), pp. 18–19. 1

[34] P. Flajolet and A. Odlyzko, "Random mapping statistics", in J.-J. Quisquater and J. Vandewalle, eds. *Proceedings of Advances in Cryptography – Eurocrypt '89, Lecture Notes in Computer Science #434*, Springer-Verlag (1989), 329–354.

[35] A. Flores. "Geometry of numeric iterations", *PRIMUS* **4** (1994), 29–38. 48

[36] K. Ford, F. Luca and I. E. Shparlinski, "On the largest prime factor of the Mersenne numbers", preprint. Available at: `http://www.math.uiuc.edu/ ford/wwwpapers/P2n-1.pdf` 97

[37] R. Fox, "Crash-free consortium", *Comm. of ACM*, **44** (2001), 9. 2

[38] C. L. Gilbert et. al., "Function digraphs of quadratic maps modulo $p$", *Fib. Quart.* **39** (2001), 32–49. 80

[39] C. C. Gillispie (ed.), *Dictionary of Scientific Biography*, (vol. 12), Scribners Sons, New York, 1975. 1

[40] D. Gottesman, "An introduction to quantum error correction", in *Quantum computation: A grand mathematical challenge for the twenty-first century and the millennium*, ed. S. J. Lomonaco, Jr, American Mathematical Society (2002).

[41] T. H. Gronwall, "Some asymptotic expressions in the theory of numbers", *Trans. Amer. Math. Soc.* **14** (1913), 113–122. 124

[42] R. Grübel and U. Rösler, "Asymptotic distribution theory for Hoare's selection algorithm", *Adv. Appl. Prob.* **28** (1996), 252–269.

[43] R. K. Guy, *Unsolved problems in number theory* (2nd ed.), Springer-Verlag, 1991. 112, 127

[44] B. Harris, "Probability distributions related to random mappings", *Annals of Math. Stat.* **31** (1960), 1045–1062. 19, 71

[45] W. Heidergott, "SEU Tolerant Device, Circut and Processor Design", *Proceedings of the 42nd Annual Conference on Design Automation (DAC)* (2005), 5–10. 2

[46] R. A. Holmgren, *A First Course in Discrete Dynamical Systems (Second Edition)*, Springer-Verlag, 1996. 113

[47] L. K. Hua, *Introduction to number theory*, Springer-Verlag, Berlin, 1982. 59, 60

[48] W. Keller, "Factors of Fermat numbers and large primes of the form $k \cdot 2^n + 1$", *Math. Comput.* **41** (1983), 661–673. 67

[49] W. Keller, "Fermat factoring status," World Wide Web, `http://www.prothsearch.net/fermat.html#Count`, January 2002. 67

[50] A. Khrennikov and M. Nilsson, "On the number of cycles of $p$-adic dynamical systems, *J. Number Theory* **90** (2001), 255–264. 49

[51] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms* vol. 2 (3rd ed.), Addison-Wesley, Boston, Mass., 1997. 20

[52] M. Kraitchik, *Théorie des Nombres, Tome II*, Gauthier-Villars, Paris, 1926. 46

[53] S. Kravitz, "The Lucas-Lehmer test for Mersenne numbers", *Fib. Quart.* **8** (1970), 1–3. 19

[54] K. B. Lakshmanan, B. Ravikumar and K. Ganesan, "Coping with erroneous information while sorting", *IEEE Trans. Comp.* **40** (1991), 1081–1084. 3

[55] D. H. Lehmer, "Tests for primality by the converse of Fermat's theorem", *Bull. Amer. Math. Soc.* **33** (1927), 327–340. 46

[56] D. H. Lehmer, "An extended theory of Lucas' functions", *Ann. Math.* **31** (1930), 419–448.

[57] H. R. Lewis and L. Denenberg. *Data Structures & Their Algorithms*. Harper-Collins, 1991. 3, 23

[58] G. L. Mullen R. Lidl and G. Turnwald, *Dickson Polynomials*. Pitman Monographs and Surveys in Pure and Applied Mathematics, Vol. 65, Longman Scientific, Essex, England, 1993. 79, 80

[59] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, 1994. 106

[60] P. M. Long, "Sorting and searching with a faulty comparison oracle", *Techincal Report UCSC-CRL-92-15* (1992). 4

[61] E. Lucas, *Théorie des Nombres*, Gauthier-Villars, Paris, 1891. Also available at `http://visualiseur.bnf.fr/CadresFenetre?O=NUMM-29021`. 1

[62] E. Lucas, "Théorie des fonctions numériques simplement périodiques", *Amer. J. Math.* **1** (1878), 184–240; 289–321. 78

[63] C. Lucheta, E. Miller, C. Reiter, "Digraphs from powers modulo $p$", *Fib. Quart.* **34** (1996), 226–239. 48

[64] H. P. Luhn, "Computer for Verifying Numbers", *US Patent 2,950,048* (1960). 15

[65] G. Martin and C. Pomerance, "The iterated Carmichael $\lambda$-function and the number of cycles of the power generator", *Acta Arith.* **118** (2005), 305–335. 49

[66] Mitsubishi Electronics, "Mitsubishi Electric develops high-frequency synchronous SRAM with dramatically reduced soft error rate", (1999). Available: `http://www.businesswire.com/webbox/bw.021599/1101534.htm`. 2

[67] P. Morton, "Period of maps on irreducible polynomials over finite fields", *Finite Fields Appl.* **3** (1997), 11–24. 79

[68] S. Muthukrishnan, "On optimal strategies for searching in the presence of errors", *Symposium on Discrete Algorithms (SODA)* **1994**, 680–689. 3

[69] NASA/Marshall Space Flight Center, "Solar Physics." Available: `http://science.nasa.gov/ssl/PAD/SOLAR/`. 2

[70] W. Nöbauer. Über die Fixpunte der Dickson-Permutationen. *Österreich. Akad. Wiss. Math.-Natur. Kl. Sitzungsber. II* **193** (1984), 115–133. 80

[71] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, New York, 1994. 21

[72] A. Peinado, F. Montoya, J. Muñoz, and A. J. Yuste. Maximal periods of $x^2+c$ in $\mathbb{F}_q$. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (Melbourne, 2001)*, Vol. 2227 of *Lecture Notes in Comput. Sci.*, pp. 219–228. Springer, 2001. 80

[73] A. Pelc, "Detecting errors in searching games" *J. Comb. Theory A* **54** (1989), 43–54. 3

[74] A. Pelc, "Searching with known error probability", *Theor. Comp. Sci.* **63** (1989), 185–202. 3

[75] T. Pepin, "Sur la formule $2^{2^n}+1$," *C. R. Acad. Sci. Paris* **85** (1877), 329–331. 46, 47

[76] T. Pepin, "Sur la formule $2^n-1$,", *C. R. Acad. Sci. Paris* **86** (1878), 307–310.

[77] J. M. Pollard, "A Monte Carlo method for factorization," *BIT* **15** (1975), 331–334. 20, 78

[78] R.E. Powers, "The tenth perfect number," *Amer. Math. Monthly* **XVIII**, 195–7 (1911). 1, 4, 97

[79] B. Ravikumar, K. Ganesan and K.B. Lakshmanan, "On selecting the largest element in spite of erroneous information", *Lecture Notes in Computer Science, Vol. 247. Fourth Symposium on Theoretical Aspects of Computer Science (STACS)* (1987), 88–99. 128

[80] B. Ravikumar and K.B. Lakshmanan, "Coping with known patterns of lies in a search game", *Theor. Comp. Sci* **33** (1984) 85–94. 3

[81] B. Ravikumar, "A fault-tolerant merge sorting algorithm", *Lecture Notes in Computer Science, Vol. 2387. Proceedings of the 8th Annual International Conference on Computing and Combinatorics.* (2002), 440–447. 4

[82] R. L. Rivest et al. "Coping with errors in binary search procedures", *J. Comp. Sys. Sci.* **20** (1980), 396–404. 3, 4

[83] T. Reix et. al. "Number of cycles in digraph under $x^2$ with a Mersenne prime", Internet forum discussion. `http://www.mathlinks.ro/Forum/viewtopic.php?t=61673` 71, 89

[84] T. D. Rogers, "The graph of the square mapping on the prime fields", *Discrete Math.* **148** (1996), 317–324. 48, 78

[85] D. Shanks, *Solved and Unsolved Problems in Number Theory*, Chelsea, New York (1985). 80

[86] W. Shanks, *Contributions to Mathematics Comprising Chiefly of the Rectification of the Circle to 607 Places of Decimals*, G. Bell: London (1853). 1

[87] L. Somer and M. Křížek, "On a connection of number theory and graph theory", *Czech. Math. J.* **54** (2004), 465–485. 49

[88] W. Stallings, *Computer Organization and Architecture: Principles of Structure and Function*, second edition, Macmillan, New York, 1990. 2

[89] D. Stanton and D. White, *Constructive Combinatorics*, Springer-Verlag, New York, 1986.

[90] C.L. Stewart, "On divisors of Fermat, Fibonacci, Lucas and Lehmer numbers", *Proc. London Math. Soc.* **35** (1977), 425–447. 97

[91] M. Szegedy and X. Chen, "Computing boolean functions with multiple faulty copies of input bits", in S. Rajsbaum, ed., *Proceedings of LATIN 2002, Lecture Notes in Computer Science #2286*, Springer-Verlag (2002), 539–553.

[92] E. Teske and H.C. Williams, "A note on Shanks' chains of primes" in W. Bosma, ed., *Proceedings of ANTS IV, Lecture Notes in Computer Science #1838*, Springer-Verlag (2000), 563–580. 19, 49

[93] H. S. Uhler, "A brief history of the investigations on the Mersenne numbers and the latest immense primes," *Scripta Mathematica* **181**, 122–131 (1952). 4

[94] T. Vasiga and J. Shallit, "On the iteration of certain quadratic maps over GF($p$)", *Discrete Math.*, **277**, 219–240 (2004). 75, 78, 89

[95] F. Vivaldi and S. Hatjispyros, "Galois theory of periodic orbits of rational maps", *Nonlinearity*, **5**, 961–978 (1992). 80

[96] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components", *Automata Studies* (ed. by C. E. Shannon and J. McCarthy) Princeton University Press, 43–98. 3

[97] H. Wasserman and M. Blum, "Software reliability via run-time result-checking", *J. ACM* **44** (1997), 826–849. 5

[98] S. S. Wagstaff, Jr. "Greatest of the least primes in arithmetic progressions having a given modulus," *Math. Comp.* **33** (1979), 1073–1080. 76

[99] H. C. Williams, *Édouard Lucas and Primality Testing*, Wiley, 1998. 110

[100] B. Wilson, "Power digraphs modulo $n$", *Fib. Quart.* **36** (1998), 229–239. 49, 59, 61

[101] J.F. Ziegler et al., "IBM experiments in soft fails in computer electronics (1978-1994)", *IBM J. Res. Develop.* **40**, 1–17 (1996). 2