Extending The Thue-Morse Sequence

Olivia Appleton-Crocker
TMW Center for Early Learning + Public Health
University of Chicago
Chicago, Illinois, United States
ORCID: 0009-0004-2296-7033

***Abstract*—In this paper, we discuss various ways to extend the Thue-Morse Sequence [1] when used as a fair-share sequence. Included are 20 definitions of the original sequence, 9 extensions to n players, for a total of 29 definitions. Also included are proofs of equality for all definitions, as well as an examination of several properties of the Thue-Morse Sequence and their presence in the Extended Thue-Morse Sequence. In the appendix are several complexity analyses for both time and space of each definition.**

***Index Terms*—Combinatorics, Generating Functions, Thue-Morse, Prouhet-Thue-Morse, Formal Languages, Number Theory, Periodicity, Aperiodicity, Rotation, Concatenation**

I. Introduction

This section is too fluffy. Probably needs to be rewritten entirely

The Thue-Morse Sequence is a fundamental object in combinatorics and theoretical computer science, widely studied for its remarkable properties and diverse applications. It is often presented as an infinite sequence starting with 0, as shown below:

$$T=\langle 0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,\ldots\rangle$$

This sequence arises in various fields, including automata theory, formal languages, number theory, and signal processing, due to its connection to binary operations, periodicity, and complexity. Its structure has been examined in relation to notions of minimality and non-repetitiveness, making it a natural object of study in mathematical logic and computational theory.

Over the past two centuries, the Thue-Morse Sequence has garnered attention for its role in constructing infinite words with specific properties (such as non-repetition over large substrings) and for its use in the design of error-correcting codes, pseudorandom number generators, and tiling problems. More recently, its applications have extended to the analysis of dynamical systems, coding theory, and the study of complexity within algorithms.

This survey primarily aims to explore the various definitions of the Thue-Morse Sequence found in the literature, with an emphasis on identifying distinct formulations and categorizing them. Additionally, this work seeks to extend these definitions from the binary alphabet $\{0,1\}$ to larger alphabets $\{0,1,\ldots,n-1\}$, often referred to as "integer bases." This work aims to explore how properties such as non-repetition, periodicity, and complexity are preserved under these extensions. By systematically analyzing these extensions and their associated properties, this overview hopes to present a deeper understanding of the Thue-Morse Sequence's generalizations and their potential applications in broader contexts.

This paper is divided into 8 sections. In Section 1, we introduce the concepts built upon in this paper. In Section 2 we present each of the 20 definitions of the standard Thue-Morse Sequence as seen in the literature today, divided by category of definition. We first look at numeric methods, then operations on ordered collections of numbers. We next examine definitions that relate to other sequences of integers, followed by a pair of generating functions, a hypergeometric definition, and one based on cellular automatons.

In Section 3 we prove their equivalence with each other in the standard base-2 domain. In Section 4 we present 9 definitions that extend into larger domains. Most of these can only construct sequences with integer value elements, though a select few can be extended into even broader domains, such as rational bases. In Section 5 we prove that the extensions are equivalent to each other in the domain of positive integer bases. In Section 6 we examine the preservation (or failure) of properties for the standard Thue-Morse Sequence when extended to larger bases. Sections 7 and 8 are acknowledgments and the appendix respectively. The appendix contains complexity analyses (both time and space) for the different definitions. These show which may be the most computationally efficient ways to calculate the Thue-Morse Sequence or Extended Thue-Morse Sequence, given your available resources.

I want to reorder these definitions by generation method

II. The Original Sequence

A. *Definition 1 of 20 — Parity of Hamming Weight*

This definition appears in [1–4].

The Hamming Weight, as typically defined, is the digit sum of a binary number. In other words, it is a count of the high bits in a given number. A common way to generate the Thue-Morse Sequence is to take the parity[1] of the Hamming Weight[2] for each natural number. We can define that as follows:

$$\begin{aligned} p(0) &= 0 \\ p(n) &= (n \mod 2) + p\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ T_{2,1}(n) &= p(n) \mod 2 \end{aligned} \tag{1}$$

[1]Whether a number is odd or even

[2]The count of 1s in the binary representation of a number

The subscript indicates that we are using 2 players (writing in base 2) and that we are using the first definition laid out in this paper. Note that when we extend to n players, the T function will get a second parameter for the number of players, so it will look like $T_{n,d}(x, s)$, where s is the size of the player pool, and therefore the base we use to define the sequence.

B. *Definition 2 of 20 — Powers of Negative One*

Should I split this into 2 definitions (yes)? I've done so for others that were this dissimilar while sharing a common root idea. It feels odd that this is the only one with an a and b version.

This appears in [5].

Another way to implement the modularity of the previous definition is to use the appropriate root of unity. In this paper, we use $\omega_s = \exp\left(\frac{2i\pi}{s}\right)$ to represent the primitive root of unity in base s, such that $(\omega_s)^s = 1$. Because of this, extracting the power of $(\omega_s)^x$ will give you the same value as $x \bmod s$.

There are two ways this can be approached. In the first we work with these output values directly, mapping $\{1, -1\} \to \{0, 1\}$:

$$\begin{aligned} T_{2,2a}(n) &= \frac{1 - (\omega_2)^{p(n)}}{2} \\ &= \frac{1 - (-1)^{p(n)}}{2} \end{aligned} \tag{2}$$

C. *Definition 3 of 20 — Root of Unity*

And in the second we extract the exponent:

$$\begin{aligned} T_{2,3b}(n) &= \frac{\log\left(\omega_2^{p(n)}\right)}{\log\left(\omega_2\right)} \\ &= \frac{\log\left((-1)^{p(n)}\right)}{\log(-1)} \\ &= \frac{(p(n) \bmod 2) \cdot \log(-1)}{\log(-1)} \\ &= \frac{(p(n) \bmod 2) \cdot i\pi}{i\pi} \\ &= p(n) \bmod 2 \end{aligned} \tag{3}$$

D. *Definition 4 of 20 — Recursion*

This definition appears in [1, 4, 6].

$$\begin{aligned} T_{2,4}(0) &= 0 \\ T_{2,4}(2n) &= T_{2,4}(n) \\ T_{2,4}(2n+1) &= 1 - T_{2,4}(n) \end{aligned} \tag{4}$$

E. *Definition 5 of 20 — Floor-Ceiling Difference*

This definition appears in [1]. Some of the info that's in the proof of equivalence should be lifted up into here

$$\begin{aligned} b(n) &= \begin{cases} n & \text{if } n \leq 1 \\ b\left(\left\lceil \frac{n}{2} \right\rceil\right) - b\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{otherwise} \end{cases} \\ T_{2,5}(n-1) &= \frac{1 - b(2n-1)}{2} \pmod 2 \end{aligned} \tag{5}$$

This seems very similar to the highest bit difference definition, and I think it may be what that was derived from

F. *Definition 6 of 20 — Highest Bit Difference*

This definition appears in [7].

The text below is from Wiki and needs to be entirely rewritten. I was able to derive the formula on my own from translating their code. This method leads to a fast method for computing the Thue-Morse Sequence: start with t0 = 0, and then, for each n, find the highest-order bit in the binary representation of n that is different from the same bit in the representation of n - 1. If this bit is at an even index, tn differs from tn - 1, and otherwise it is the same as tn - 1.

```
from itertools import count

def p2_d07():
    value = 1
    for n in count():
        # Assumes that (-1).bit_length() == 1
        x = (n ^ (n - 1)).bit_length() + 1
        if x & 1 == 0:
            # Bit index even, so toggle value
            value = 1 - value
        yield value
```

Should there be a +1 inside that call to log2?

$$\begin{aligned} T_{2,6}(0) &= 0 \\ T_{2,6}(n) &= \begin{array}{l} \lfloor \log_2(n \oplus (n-1)) \rfloor \\ + T_{2,6}(n-1) \pm 1 \end{array} \pmod 2 \end{aligned} \tag{6}$$

G. *Definition 7 of 20 — Hamming-Weight Complement*

This definition appears in [5]. The Hamming Weight Complement (defined as $\overline{p}(n)$) is the count of 0s in the minimal binary representation of a number, and A054429 is the sequence defined by reversing the order of integers between 2^n and $2^{n+1} - 1$ (ex: $0, 1, 3, 2, 7, 6, 5, 4, \ldots$).

$$\begin{aligned} \text{A054429}(n) &= \begin{cases} 0 & \text{if } n = 0 \\ 3 \cdot 2^{\lfloor \log_2(n) \rfloor} - n - 1 & \text{otherwise} \end{cases} \\ \overline{p}(x) &= \lceil \log_2(x+1) \rceil - p(x) \\ T_{2,7}(n) &= 1 - \overline{p}(\text{A054429}(n)) \bmod 2 \end{aligned} \tag{7}$$

H. *Definition 8 of 20 — Invert and Extend*

This definition appears in [1, 4, 8].

This definition is more natural to think about as extending a tuple that contains the sequence. We will give a recurrence relation below, but to build an intuition we will work in this framework first.

Let $t(n)$ be the first 2^n elements of the Thue-Morse Sequence. Given this, we can define[3]:

[3]Using $\|$ to mean concatenation, so $\langle 0 \rangle \| \langle 1 \rangle = \langle 0, 1 \rangle$. Likewise: $\mathop{\Big\|}_{i=a}^{b} f(i) = f(a) \| f(a+1) \| f(a+2) \| \cdots \| f(b)$

$$\begin{aligned}
\mathrm{inv}(\mathbf{x}) &= \begin{cases} 0, & \text{if } x_i = 1 \\ 1, & \text{if } x_i = 0 \end{cases} \\
&\qquad \text{for } \mathbf{x} = \langle x_0, x_1, \ldots, x_{(|\mathbf{x}|-1)} \rangle \\
t(0) &= \langle 0 \rangle \\
t(n) &= t(n-1) \parallel \mathrm{inv}(t(n-1))
\end{aligned} \tag{8}$$

Given the above, we can define a recurrence relation that will give us individual elements. It will be less efficient to compute, but will allow proofs of equivalence to be easier.

Should the below have a +1 inside the log?

$$\begin{aligned}
T_{2,8}(0) &= 0 \\
T_{2,8}(n) &= T_{2,8}\left(n - 2^{\lfloor \log_2(n) \rfloor}\right) + 1 \pmod 2
\end{aligned} \tag{9}$$

I. Definition 9 of 20 — Substitute and Flatten

This definition appears in [1, 2, 4, 6]. It is one of the most commonly used because of its simplicity. Define the morhpism μ on the alphabet $\{0, 1\}$ by $\mu(0) = 01, \mu(1) = 10$. T is then the unique fixed point of μ that begins with 0. In other words, $T = \lim_{n\to\infty} \mu^n(0)$.

$$\begin{aligned}
s(n) &= \begin{cases} \langle 0, 1 \rangle, & \text{if } n = 0 \\ \langle 1, 0 \rangle, & \text{if } n = 1 \end{cases} \\
t(0) &= \langle 0 \rangle \\
t(n) &= \mathop{\Big\Vert}_{i=0}^{2^{n-1}-1} s(t(n-1)_i) \\
T_{2,9}(n) &= t(\lceil \log_2(n+1) \rceil)_n
\end{aligned} \tag{10}$$

So for example, calculating $T_{2,4}(3)$ would look like:

$$\begin{aligned}
t(0) &= \langle 0 \rangle \\
t(1) &= \mathop{\Big\Vert}_{i=0}^{0} s(t(0)_i) = \langle 0, 1 \rangle \\
t(2) &= \mathop{\Big\Vert}_{i=0}^{1} s(t(1)_i) = \langle 0, 1, 1, 0 \rangle \\
T_{2,9}(3) &= t(\lceil \log_2(3+1) \rceil)_3 \\
&= t(2)_3 \\
&= \langle 0, 1, 1, 0 \rangle_3 \\
&= 0
\end{aligned} \tag{11}$$

J. Definition 10 of 20 — Recursive Rotation

Another way to phrase the above definition is as recursive rotation. To the best of our knowledge, this definition is original to this paper. If we decompose s, we can instead represent it as:

$$\begin{aligned}
r(\mathbf{x}, i) &= \langle x_{0+i \bmod |\mathbf{x}|}, x_{1+i \bmod |\mathbf{x}|}, \ldots \rangle \\
&\quad \text{for } \mathbf{x} = \langle x_0, x_1, \ldots, x_{(|\mathbf{x}|-1)} \rangle \\
t(0) &= \langle 0 \rangle \\
t(1) &= \langle 0, 1 \rangle \\
t(n) &= \mathop{\Big\Vert}_{i=0}^{1} r\left(t(n-1), i \cdot 2^{n-2}\right) \\
T_{2,10}(n) &= t(\lceil \log_2(n+1) \rceil)_n
\end{aligned} \tag{12}$$

K. Definition 11 of 20 — Odious Number Derivation

This definition appears in [1].

Another way to generate the Thue-Morse Sequence is to take the sequence of Odious Numbers [9] mod 2. Odious numbers are those with an odd number of 1s in their binary representation. Note that the player numbers in this derivation are swapped, so when generating this for testing and extension, we add 1 to the result. Some simple generating code [10] for this is as follows:

```python
from itertools import count

def seq_p2_d09():
    for i in count():
        if i.bit_count() & 1:
            yield (i + 1) & 1
```

In mathematical terms, this can be translated to:

$$\begin{aligned}
no(n) &= \begin{cases} n & \text{if } p(n) \bmod 2 = 1 \\ no(n+1) & \text{otherwise} \end{cases} \\
o(n) &= \begin{cases} 1 & \text{if } n = 0 \\ no(o(n-1)+1) & \text{if } n > 0 \end{cases} \\
T_{2,11}(n) &= o(n) + 1 \pmod 2
\end{aligned} \tag{13}$$

Aren't Odious Numbers exactly the numbers where the parity of the hamming weight is 1? So doesn't that mean that the Thue-Morse Sequence selects which numbers are Odious? From cursory testing, it seems to. There's something to be had there.

A possible way to extend this would be to reinterpret this as where the digit sum is not n-even

A related definition on OEIS [1] is

$$\begin{aligned}
T(n) + \text{Odious}(n-1) + 1 &= 2n \text{ for } n \geq 1 \\
T(n) &= 2n - \text{Odious}(n-1) - 1
\end{aligned}$$

L. Definition 12 of 20 — Evil Numbers Derivation 1

This definition appears in [1]. More text

The Evil Numbers [11] are those who have an even number of 1s in their binary representation. Note that this is the opposite of the Odious Numbers referenced above.

```python
from itertools import count

def evil():
    for i in count():
        if i.bit_count() & 1 == 0:
            yield i

def p2_d10():
    for n, i in enumerate(evil()):
        yield (i - 2 * n) & 1
```

In mathematical terms, this can be translated to:

$$
\begin{aligned}
ne(n) &= \begin{cases} n & \text{if } p(n) \bmod 2 = 1 \\ ne(n+1) & \text{otherwise} \end{cases} \\
e(n) &= \begin{cases} 1 & \text{if } n = 0 \\ ne(e(n-1)+1) & \text{if } n > 0 \end{cases} \\
T_{2,12}(n) &= e(n) - 2n \pmod 2
\end{aligned} \tag{14}
$$

M. Definition 13 of 20 — Evil Numbers Derivation 2

This definition ppears in [5]. More text

A second, more-efficient derivation from the Evil Numbers is as follows, where $ce()$ is the count of Evil Numbers less than n [12], and $p()$ is the function defined in Equation 1.

$$
\begin{aligned}
ce(n) &= \left\lfloor \frac{n+1}{2} \right\rfloor + p(n+1) \cdot (n+1 \bmod 2) \\
T_{2,13}(n) &= 1 - ce(n+1) + ce(n)
\end{aligned} \tag{15}
$$

N. Definition 14 of 20 — Odious & Evil Numbers Derivation

This definition appears in [5]. More text as to why. Also, figure out why

A second, more-efficient derivation from the Evil Numbers is as follows, where $ce()$ is the count of Evil Numbers less than n [12], and $p()$ is the function defined in Equation 1.

$$
\begin{aligned}
oe(n) &= \begin{cases} o\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{if } n \bmod 2 = 0 \\ e\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{if } n \bmod 2 = 1 \end{cases} \\
T_{2,14}(n) &= 1 - oe(n) \pmod 2
\end{aligned} \tag{16}
$$

O. Definition 15 of 20 — Gould's Sequence Derivation

This definition appears in [1].

Gould's Sequence [13] are the number of odd entries in a given row of Pascal's Triangle. Note that this is by far one of the least computationally efficient definition in this paper (see Tables XXV & XXVI).

Why mod 3? Everything else is mod 2. This definition is unlikely to be extendable, and the obvious routes fail. I don't get why this works, and I think I need to. I *think* $\text{Gould}(x)$ always returns $2k + \{0, 1\}$

$$
\begin{aligned}
T_{2,15}(n) &= \text{Gould}(n) - 1 \bmod 3 \\
&= \left(\sum_{k=0}^{n} \left(\binom{n}{k} \bmod 2\right)\right) - 1 \bmod 3
\end{aligned} \tag{17}
$$

P. Definition 16 of 20 — Derivation from Blue Code

This definition appears in [1].

In the OEIS, this sequence [14] is defined as the "binary coding of a polynomial over GF(2), substitute x+1 for x". There are a number of ways to generate it. One of the more computationally-accessible ones is:

$$
\begin{aligned}
\text{A001317}(n) &= \sum_{k=0}^{n} \left(\binom{n}{k} \bmod 2\right) \cdot 2^k \\
f(n,i) &= \left\lfloor \frac{n}{2^i} \bmod 2 \right\rfloor \\
\text{A193231}(n) &= \bigoplus_{i=0}^{\lceil \log_2(n) \rceil} (\text{A001317}(i \cdot f(n,i)) \cdot f(n,i)) \\
T_{2,16}(n) &= \text{A193231}(n) \pmod 2
\end{aligned} \tag{18}
$$

Translated into words, this function computes the value of Sierpiński's triangle for the index of each high bit, then takes the bitwise exclusive or[4] of all such resulting values. Note that since $f(n,i) = 0$ if and only if the ith bit of n is low, each low bit can be simplified out when calculating.

It seems to me that this might be extendable by using GF(n) instead of GF(2), though I don't know of a way to efficiently compute or prove such a result

Q. Definition 17 of 20 — Generating Function 1

This generating function[5] definition appears in [1, 3].

$$
\begin{aligned}
G(x) &= \mathcal{G}.\mathcal{F}. \; \frac{\frac{1}{1-x} - \prod_{k \geq 0}^{\infty} (1 - x^{2^k})}{2} \\
&= \mathcal{G}.\mathcal{F}. \; \frac{\sum_{k \geq 0}^{\infty} x^k - \prod_{k \geq 0}^{\infty} (1 - x^{2^k})}{2} \\
T_{2,17}(n) &= [x^n] G(x)
\end{aligned} \tag{19}
$$

The logic of this construction is fairly clear. The component that is $\frac{1}{1-x}$ will generate coefficients of all 1s. This is then subtracted by a sequence that counts the high bits of a given index, evaluating to -1 if the number of bits is even, and 1 if it is odd. This means that the only possible outputs of the top part are 0 or 2, so the divisor corrects for that.

This means that it is equivalent to $T_{2,1}$ (prove it).

In practice, to get the coefficients up to the nth term, one needs to only expand the product to $k_{\max} = \lfloor \log_2(n+1) \rfloor - 1$.

[4] Represented in this paper as $\oplus$.

$$x \oplus y = \sum_{i=0}^{\lfloor \log_2(\max(x,y)) \rfloor} 2^i \cdot \left(\left\lfloor \frac{x}{2^i} \right\rfloor + \left\lfloor \frac{y}{2^i} \right\rfloor \bmod 2\right)$$

$$\bigoplus_{i=a}^{b} f(i) = f(a) \oplus f(a+1) \oplus f(a+2) \oplus \cdots \oplus f(b)$$

[5] A generating function is a polynomial such that the coefficient of each term is equal to the value in a sequence. $[x^n]G(f)$ indicates "the coefficient of x^n in the function $G(x)$." So $[x^2]\left(1 + 2x + 3x^2\right) = 3$

Note that this and the next definition are by far the least computationally efficient definitions in this paper (see Figure 2 and Tables XXIX, XXX, XXXI, XXXII).[6]

R. Definition 18 of 20 — Generating Function 2

This definition appears in [17, 18]. The coefficients of this generating function are given in OEIS Sequence A309303 [19]. It appears that coefficients are grabbed using a Flint Series

$$\begin{aligned} G(x) &= \mathcal{G}.\mathcal{F}.\frac{\sqrt{x+1}-\sqrt{1-3x}}{2\cdot(x+1)^{\frac{3}{2}}} \\ T_{2,18}(n) &= \frac{1-(-1)^{[x^n]G(x)}}{2} \end{aligned} \tag{20}$$

S. Definition 19 of 20 — Hypergeometry

I'm a little unclear why this works, but it certainly seems to

This definition appears in [17, 18].

$$T_{2,19}(n) = \begin{pmatrix} 1+\dfrac{(-1)^n}{2}+ \\ \sqrt{\pi}\cdot(-3)^n\cdot\dfrac{{}_2F_1(\frac{3}{2},-n;\frac{3}{2}-n;\frac{-1}{3})}{4\cdot n!} \end{pmatrix} \bmod 2 \tag{21}$$

where ${}_2F_1(a_1,a_2;b_1;z)$ is the generalized, regularized hypergeometric function given by:

$${}_2F_1(a_1,a_2;b_1;z) = \sum_{k=0}^{\infty}\frac{\frac{(a_1)_k(a_2)_k}{(b_1)_k}\cdot\frac{z^k}{k!}}{\Gamma(b_1)}, \tag{22}$$

and $(a)_k$ is the Pochhammer symbol defined as:

$$(a)_k = \left(\prod_{i=0}^{k-1}(a+i),\ \ (a)_0=1\right) \approx \frac{(a+k-1)!}{(a-1)!} \tag{23}$$

T. Definition 20 of 20 — Cellular Automaton

This definition appears in [17, 18].

Using Mathematica-like notation,

$$\begin{aligned} T_{2,20} = \lim_{k\to\infty} 1 - \text{Flatten}[&\text{CellularAutomaton}[\\ &\{69540422, 2, 2\}, \\ &\{\{1\}, 0\}, \\ &2^k - 1, \\ &\{\text{All}, 0\}]] \end{aligned} \tag{24}$$

U. Summary

Of the 20 we discussed above:

- 17 were initially found on the OEIS [1, 5, 18, 20, 21] ($T_{2,1...2}, T_{2,4...5}, T_{2,9}, T_{2,11...20}$)
- 1 was found in a textbook [7] ($T_{2,6}$)
- 2 are original to this paper ($T_{2,3}, T_{2,10}$)

- 9 utilize recursion ($T_{2,1...4}, T_{2,6...10}$)
- 6 reference other integer sequences ($T_{2,11...16}$)
- 7 utilize floor-division ($T_{2,1...3}, T_{2,5}, T_{2,13...14}, T_{2,16}$)
- 3 use operations on strings, not integers ($T_{2,8...10}$)
- 2 use combinatoric functions ($T_{2,15...16}$)
- 2 utilizes a generating function ($T_{2,17...18}$)
- 1 is hypergeometric ($T_{2,19}$)
- 1 is based on a cellular automaton ($T_{2,20}$)
- 0 have closed form solutions

[6]Our implementation [10] heavily utilizes [15, 16] for these calculations

III. Proving Equivalence Between Standard Definitions

A. Correlating Definition 1 and Definition 2

Proof.

Observation 1: For all integers n, $(-1)^n \in \{1,-1\}$

Observation 2: For $n=\{1,-1\}$, $\frac{1-n}{2}=\{0,1\}$

Observation 3: For all $n=2k$, $(-1)^n=1$

Observation 4: For all $n=2k+1$, $(-1)^n=-1$

Inference 1: $\frac{1-(-1)^n}{2} = n \mod 2$

Conclusion:

$$T_{2,2}(n) = \frac{1-(-1)^{p(n)}}{2} = p(n) \bmod 2 = T_{2,1}(n) \tag{25}$$

□

B. Correlating Definition 1 and Definition 4

Note that in Equation 4 where we define $T_{2,4}$, we are working in mod 2, where $+1$ and -1 are logically equivalent. We can therefore simplify its definition to be:

$$\begin{aligned} T_{2,4}(0) &= 0 \\ T_{2,4}(n) &= n + T_{2,4}\left(\left\lfloor\frac{n}{2}\right\rfloor\right) \pmod 2 \end{aligned} \tag{26}$$

This is identical to Equation 1, where we define $T_{2,1}$.

C. Correlating Definition 1 and Definition 8

Proof.

Observation 1: $0 \le n < 2^k \implies t(k+1)_{2^k+n} \equiv t(k)_n + 1$ This is because at each step in the process, you are inverting and extending. Inversion is equivalent to $x+1 \mod 2$, and each entry keeps a relative position by adding a power of 2.

Inference 1:

$$\begin{aligned} T_{2,8}(0) &= 0 \\ T_{2,8}(n) &= T_{2,3}(n-2^{\lfloor\log_2(n)\rfloor}) + 1 \bmod 2 \end{aligned} \tag{27}$$

Observation 2: By subtracting a power of 2, you reduce the Hamming Weight by 1

Inference 2: Since you are adding 1 for each time you reduce the Hamming Weight by 1, this means $T_{2,8}$ is recursively computing the parity of the Hamming Weight.

Observation 3: $T_{2,1}=p(n)$, and $p(n)$ computes the parity of the Hamming Weight of n.

Conclusion: $T_{2,1}(n) = T_{2,8}(n)$ □

D. Correlating Definition 2 and Definition 5

Proof. **Observation 1**: If n is even:

$$\begin{aligned} b(n = 2k) &= b(\lceil \tfrac{2k}{2} \rceil) - b(\lfloor \tfrac{2k}{2} \rfloor) \\ &= b(k) - b(k) \\ &= 0 \end{aligned} \tag{28}$$

Inference 1: Therefore,

$$\begin{aligned} b(n = 2k+1) &= b\left(\left\lceil \frac{2k+1}{2} \right\rceil\right) - b\left(\left\lfloor \frac{2k+1}{2} \right\rfloor\right) \\ &= b(k+1) - b(k) \\ &= \begin{cases} b(k+1) & \text{if } k \text{ is even} \\ -b(k) & \text{if } k \text{ is odd} \end{cases} \end{aligned} \tag{29}$$

Hypothesis 1: $b(2k+1) = (-1)^{p(k)}$

Assumption 1: Assume $b(2k+1) \neq (-1)^{p(k)}$

Assumption 2:

$$\begin{aligned} T_{2,2}(n) = \tfrac{1-(-1)^{p(n)}}{2} &= \tfrac{1-b(2n+1)}{2} = T_{2,5}(n) \\ 1 - (-1)^{p(n)} &= 1 - b(2n+1) \\ (-1)^{p(n)} &= b(2n+1) \end{aligned} \tag{30}$$

Contradiction! The only way for these definitions to be equal is for $b(2n+1) = (-1)^{p(n)}$

I don't think this proof is 100% solid □

E. Correlating Definition 9 and Definition 10

Proof.

Observation 1: $r(\mathbf{x}, i) = r(\mathbf{x}, i + k \cdot |\mathbf{x}|)$

Inference 1: $r(\mathbf{x}, i) = r(\mathbf{x}, i \mod |\mathbf{x}|)$

Observation 2: $s(0) = \langle 0, 1 \rangle = r(s(0), 0)$

Observation 3: $s(1) = \langle 1, 0 \rangle = r(s(0), 1)$

Inference 2: $s(n) = r(s(0), n)$

This is where I run into trouble. I know I can keep going from here, but I'm not quite sure how.

Conclusion: $T_{2,9}(n) = T_{2,10}(n)$ □

IV. The Extensions

$$\begin{tikzcd}
T_{2,1} \arrow[r] \arrow[d] & T_{n,1} \\
T_{2,3} \arrow[r] & T_{n,2} \\
T_{2,4} \arrow[r] & T_{n,3} \\
T_{2,6} \arrow[r] & T_{n,4} \\
T_{2,8} \arrow[r] & T_{n,5} \\
T_{2,9} \arrow[r] \arrow[d] \arrow[ddr, dashed] & T_{n,6} \\
T_{2,10} \arrow[r] & T_{n,7} \\
 & T_{n,8} \\
T_{2,17} \arrow[r] & T_{n,9}
\end{tikzcd}$$

Fig. 1. Map of standard definitions to their extensions

While the Thue-Morse Sequence has a well-established binary form, its extension to larger alphabets introduces interpretive choices. In this work, we adopt a consistent methodology that extends all definitions equivalently. This approach ensures coherence across all extended definitions and preserves the underlying structure of the sequence. However, we acknowledge alternative constructions (such as the morphism $0 \to 012, 1 \to 02, 2 \to 1$ [4, 22–26]) which deviate from these extensions. While these may yield sequences of interest, they do not align with the equivalence criteria established in our framework.

A. Extension 1 of 9 — Modular Digit Sums

This definition appears in [20, 27–34].

To extend definition 1 from 2 to n players, we must first map our concept of parity to base n. We can do this by taking the parity equation defined above and replacing the 2s with n, for $n \in \mathbb{Z}_{\geq 2}$.

$$\begin{aligned} p_n(0) &= 0 \\ p_n(x) &= x + p_n\left(\left\lfloor \frac{x}{n} \right\rfloor\right) \pmod n \end{aligned} \tag{31}$$

Under this definition, you can construct the Thue-Morse Sequence using the following, starting at 0:

$$T_{n,1}(x, s) = p_s(x) \tag{32}$$

Note that this definition is trivially extensible to non-integer bases by redefining $p_n()$, though that is beyond the scope of this paper. This has been done in [20, 28]. It has also been extended to negative integer bases [29–31].

Some other works present a more generalized version, where

$$t_{b,m}(n) = p_b(n) \bmod m \tag{33}$$

This allows for increased flexibility, especially when using fractional bases. In this notation, for negative integer bases, $T_{n,1}(x,s) = p_s(x) \mod |s|$

1) Proof of Equivalence with Original Definition 1:

Proof. It is clear from visual inspection that p_2 is identical to our original definition of p.

$$\begin{aligned} p_2(x) &= p(x) \\ x + p_2\left(\left\lfloor \frac{x}{2} \right\rfloor\right) &= x + p\left(\left\lfloor \frac{x}{2} \right\rfloor\right) \\ x + \left\lfloor \frac{x}{2} \right\rfloor + p_2\left(\left\lfloor \frac{x}{2^2} \right\rfloor\right) &= x + \left\lfloor \frac{x}{2} \right\rfloor + p\left(\left\lfloor \frac{x}{2^2} \right\rfloor\right) \\ x + \left\lfloor \frac{x}{2} \right\rfloor + \left\lfloor \frac{x}{2^2} \right\rfloor + \dots &= x + \left\lfloor \frac{x}{2} \right\rfloor + \left\lfloor \frac{x}{2^2} \right\rfloor + \dots \end{aligned} \tag{34}$$

□

This definition is also trivially extended to negative integer bases.

B. Extended Definition 2 of 9 — Roots of Unity

$$T_{n,2}(x,s) = \frac{\log\left(\omega_s^{p_s(x)}\right)}{\log(\omega_s)} \tag{35}$$

1) Proof of Equivalence with Original Definition 3:

Proof. Let's start by substituting s for 2:

$$\begin{aligned} T_{n,2}(x,2) &= \frac{\log\left(\omega_2^{p_2(x)}\right)}{\log(\omega_2)} \\ &= \frac{\log\left((-1)^{p_2(x)}\right)}{\log(-1)} \\ &= \frac{(p_2(x) \bmod 2) \cdot \log(-1)}{\log(-1)} \\ &= \frac{(p_2(x) \bmod 2) \cdot i\pi}{i\pi} \\ &= p_2(x) \bmod 2 \end{aligned} \tag{36}$$

This is identical to $T_{2,1}$, which we earlier proved is equivalent to $T_{2,2}$. □

C. Extended Definition 3 of 9 — Recursion

$$\begin{aligned} T_{n,6}(0,s) &= 0 \\ T_{n,6}(s \cdot x, s) &= T_{n,6}(x,s) \\ T_{n,6}(s \cdot x + k, s) &= k + T_{n,6}(x,s) \pmod s \end{aligned} \tag{37}$$

1) Proof of Equivalence with Original Definition 4:

Proof. Let's begin by substituting s for 2:

$$\begin{aligned} T_{n,3}(0,2) &= 0 \\ T_{n,3}(2 \cdot x, 2) &= T_{n,3}(x,2) \\ T_{n,3}(2 \cdot x + k, 2) &= k + T_{n,3}(x,2) \pmod 2 \end{aligned} \tag{38}$$

Note that that only values for k that fit in this definition are 0 and 1. This means we can further simplify to:

$$T_{n,3}(2 \cdot x + 1, 2) = 1 + T_{n,3}(x,2) \pmod 2 \tag{39}$$

This is very similar to the definition found in equation 4, except that one is adding and the other subtracting. Fortunately, we know that the only values that $T_{2,4}$ will return are 0 and 1, which means that these operations will be completely equivalent.

$$\begin{aligned} 1 - 0 \bmod 2 &= 1 + 0 \bmod 2 \\ 1 - 1 \bmod 2 &= 1 + 1 \bmod 2 \end{aligned}$$

□

D. Extended Definition 4 of 9 — Highest Digit Difference

$$\begin{aligned} \mathrm{XOR}_n(a,b) &= \sum_{i=0}^{\lceil \log_n(\max(a,b)+1) \rceil} n^i \left(\left\lfloor \frac{a}{n^i} \right\rfloor - \left\lfloor \frac{b}{n^i} \right\rfloor \bmod n \right) \\ T_{n,4}(0,s) &= 0 \\ T_{n,4}(x,s) &= \begin{matrix} \lfloor \log_s(\mathrm{XOR}_s(x, x-1)) \rfloor \\ + T_{n,4}(x-1,s) + 1 \end{matrix} \pmod s \end{aligned} \tag{40}$$

Substitute n for 2, then simplify, plus a bit

1) Proof of Equivalence with Original Definition 6: ...

E. Extended Definition 5 of 9 — Increment and Extend

[35] gives a good example on how to possibly adapt inversion to incrementing

In the original version of this definition, we inverted the elements. In base 2, this is the same thing as adding 1 (mod 2). Given that, let $t(x,n)$ be the first n^x elements of the Extended Thue-Morse Sequence, for $n \in \mathbb{Z}_{\geq 2}$.

$$\mathrm{inc}(\mathbf{x}, n) = \begin{matrix} x_i + 1 \pmod n \\ \text{for } \mathbf{x} = (x_0, x_1, \dots, x_{(|\mathbf{x}|-1)}) \end{matrix} \tag{41}$$

$$\begin{aligned} t(0,n) &= \langle 0 \rangle \\ t(1,n) &= \langle 0, 1, \dots, n-1 \rangle \\ t(x,n) &= t(x-1,n) \cdot \mathrm{inc}(t(x-1,n),n) \end{aligned} \tag{42}$$

Given the above, we can define a recurrence relation that will give us individual elements. It will be less efficient to compute, but will allow proofs of equivalence to be easier.

$$\begin{aligned} T_{n,5}(0,s) &= 0 \\ T_{n,5}(x,s) &= T_{n,5}\left(x - s^{\lfloor \log_s(x) \rfloor}, s\right) + 1 \pmod s \end{aligned} \tag{43}$$

1) Proof of Equivalence with Original Definition 8: ...

F. Extended Definition 6 of 9 — Substitute and Flatten

This definition appears in [36]

There's a bit of a leap here, since we have to explain why the rotation is equivalent to the binary choice presented in the original. There also might be a better syntax to define the rotation, perhaps using the format used in inv and inc.

$$\begin{aligned} b(s) &= \langle 0, 1, \cdots, s-2, s-1 \rangle \\ r(\mathbf{x}, i) &= \begin{array}{l} \langle x_{0+i \bmod |\mathbf{x}|}, x_{1+i \bmod |\mathbf{x}|}, \dots \rangle \\ \text{for } \mathbf{x} = \langle x_0, x_1, \dots, x_{(|\mathbf{x}|-1)} \rangle \end{array} \\ s(x, s) &= r(b(s), x) \\ t(0) &= \langle 0 \rangle \\ t(x, s) &= \mathop{\Big|\Big|}_{i=0}^{2^{x-1}-1} s(t(x-1)_i, s) \\ T_{n,6}(x, s) &= t(\lceil \log_s(x+1) \rceil, s)_x \end{aligned} \tag{44}$$

1) Proof of Equivalence with Original Definition 9: ...

G. Extended Definition 7 of 9 — Recursive Rotation

$$\begin{aligned} r(\mathbf{x}, i) &= \begin{array}{l} \langle x_{0+i \bmod |\mathbf{x}|}, x_{1+i \bmod |\mathbf{x}|}, \dots \rangle \\ \text{for } \mathbf{x} = \langle x_0, x_1, \dots, x_{(|\mathbf{x}|-1)} \rangle \end{array} \\ t(0, s) &= \langle 0 \rangle \\ t(1, s) &= \langle 0, 1, \dots, s-1 \rangle \\ t(x, s) &= \mathop{\Big|\Big|}_{i=0}^{s-1} r\left(t(x-1, s), i \cdot s^{x-2}\right) \\ T_{n,7}(x, s) &= t(\lceil \log_s(x+1) \rceil, s)_x \end{aligned} \tag{45}$$

1) Proof of Equivalence with Original Definition 10: ...

H. Extended Definition 8 of 9 — Latin Square Constructions

This definition appears in [8], and seemingly only there. It is very clearly equivalent to our Extended Definition 4, but I will need to actually prove that.

Let $L(n)$ be the reduced-form Latin Square with a first row of $\langle 0, 1, \dots, n-1 \rangle$, and where each row progresses from one entry to the next as $L(n)_{a,x+1} \equiv L(n)_{a,x} + 1 \pmod n$. For each iteration t_n, substitute each entry x for the string $L(n)_{x,*}$.

Needs more explanation, largely copying from std def 8

$$L(N) = \begin{pmatrix} 0 & 1 & 2 & \dots & N{-}1 \\ 1 & 2 & \ddots & N{-}1 & 0 \\ 2 & \ddots & N{-}1 & 0 & 1 \\ \vdots & N{-}1 & 0 & 1 & \ddots \\ N{-}1 & 0 & 1 & \ddots & \ddots \end{pmatrix} \tag{46}$$

1) Proof of Equivalence with Original Definition 9:

I. Extended Definition 9 of 9 — Generating Functions

$$\begin{aligned} G_s(x) &= \mathcal{G.F.} \prod_{k \geq 0}^{\infty} \sum_{i=0}^{s-1} \omega_s^i \cdot x^{i \cdot s^k} \\ \omega_s^{T_{n,9}(j,s)} &= [x^j] G_s(x) \\ T_{n,9}(j, s) &= \frac{\log\left([x^j] G_s(x)\right)}{\log(\omega_s)} \\ &= \frac{\log\left([x^j] G_x(x)\right) \cdot s}{2i\pi} \end{aligned} \tag{47}$$

A similar definition to the below is found in [37] for $x \in \mathbf{Q}((x^{-1}))$. While that definition is equivalent to T_2, it does not match the values for the other generalizations in this paper. For a specific example:

$$\begin{aligned} T_3 &= \langle 0, 1, 2, 1, 2, 0, 2, 0, 1, \dots \rangle \\ T_3' &= \langle 0, 2, 0, 2, 1, 0, 0, 0, 0, \dots \rangle \pmod 3 \end{aligned} \tag{48}$$

Above needs checking. I am only 80% confident in my analysis here.

1) Proof of Equivalence with Original Definition 17:

Proof. **Observation 1**: to start, let us rephrase definition 17 slightly

$$\begin{aligned} T_{2,17}(n) &= [x^n] G(x) \\ &= [x^n] \left(\frac{\sum_{k \geq 0}^{\infty} x^k - \prod_{k \geq 0}^{\infty} (1 - x^{2^k})}{2} \right) \\ &= \frac{[x^n] \left(\sum_{k \geq 0}^{\infty} x^k - \prod_{k \geq 0}^{\infty} (1 - x^{2^k}) \right)}{2} \\ &= \frac{1 - [x^n] \prod_{k \geq 0}^{\infty} (1 - x^{2^k})}{2} \end{aligned} \tag{49}$$

Observation 2: the aparatus around the infinite product exists entirely to translate $\{1, -1\} \rightarrow \{0, 1\}$. Another way to do that is to take the complex log of this output: for $x = \{1, -1\} : \frac{\log(x)}{\log(-1)} = \{0, 1\}$

Inference 1:

$$\frac{1 - [x^n] \prod_{k \geq 0}^{\infty} (1 - x^{2^k})}{2} = \frac{\log\left([x^n] \prod_{k \geq 0}^{\infty} (1 - x^{2^k})\right)}{\log(-1)} \tag{50}$$

Observation 3: $\omega_2 = -1$

Observation 4: If we take $T_{n,9}(x, s)$ for $s = 2$, we get

$$\begin{aligned} G_2(x) &= \mathcal{G.F.} \prod_{k\geq 0}^{\infty} \sum_{i=0}^{2-1} \omega_2^i \cdot x^{i\cdot 2^k} \\ &= \mathcal{G.F.} \prod_{k\geq 0}^{\infty} \omega_2^0 \cdot x^{0\cdot 2^k} + \omega_2^i \cdot x^{i\cdot 2^k} \\ &= \mathcal{G.F.} \prod_{k\geq 0}^{\infty} 1 + (-1) \cdot x^{2^k} \\ &= \mathcal{G.F.} \prod_{k\geq 0}^{\infty} 1 - x^{2^k} \end{aligned} \tag{51}$$

This is identical to the product found in $T_{2,17}$.

Conclusion: $T_{n,9}(x, s) = T_{2,17}(x)$ □

J. Summary

Of the 9 we discussed above:

- 1 was found on the OEIS ($T_{n,1}$)
- 2 were found in another paper ($T_{n,6}, T_{n,8}$)
- 6 are original to this paper ($T_{n,2\ldots5}, T_{n,7}, T_{n,9}$)

- 8 utilize recursion ($T_{n,1\ldots8}$)
- 4 utilize floor-division ($T_{n,1\ldots4}$)
- 4 use operations on strings, not integers ($T_{2,5\ldots8}$)
- 0 have closed form solutions

V. Proving Equivalence Between Extended Definitions

A. Correlating Definition 1 and Definition 2

Proof.

Observation 1: For all integers n, $\omega_s^n = \omega_s^{n \bmod s}$

Inference 1: $\dfrac{\log(\omega_s^n)}{\log(\omega_s)} = n \bmod s$

Conclusion:

$$\begin{aligned} T_{n,2}(x, s) &= \frac{\log(\omega_s^{p_s(x)})}{\log(\omega_s)} \\ &= \frac{(p_s(x) \mod s) \cdot \log(\omega_s)}{\log(\omega_s)} \\ &= \frac{(p_s(x) \mod s) \cdot 2i\pi s^{-1}}{2i\pi s^{-1}} \\ &= p_s(x) \mod s \\ &= T_{n,1}(x, s) \end{aligned} \tag{52}$$

□

B. Correlating Definition 4 and Definition 8

Proof.

Observation 1: For any given row of $L(N)$, it will start with the index of the row

Observation 2: For any given row of $L(N)$, it will end with 1 less than the index of the row $\pmod N$

Inference 1: $L(N)_{x,*} = r(b(N), x)$

Observation 3: $T_{n,4}$ is defined as substituting $r(b(N), x)$ for each element x in the previous iteration

Conclusion: $T_{n,4} = T_{n,8}$ □

C. Summary

VI. Proving Persistence (Or Lack Thereof) of Original Properties

A. Use as a Fair-Share Sequence

Goal: show that for a variety of value functions, greedy algorithms given this turn order will always minimize inequality. They should at least do so more than the standard turn order. This is going to look like setting up an equation to show

$$\begin{aligned} eq(x, y) &= \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \\ f(v, p, s) &= \lim_{n\to\infty} \sum_{i=0}^{n} v(i) \cdot eq(T_n(i, s), p) \\ &\forall p : p \in \{1, \ldots, s-1\} \\ f(v, 0, s) + \epsilon &< f(v, p, s) < f(v, 0, s) - \epsilon \end{aligned} \tag{53}$$

This is for $v(i)$ being a value function and ϵ being an arbitrarily small number. $f()$ is therefore the sum of total value that they will be receiving. For example, one could model a board game as $v(i) = \frac{1}{2^i}$, where $v()$ models the amount each turn contributes to your probability of victory. Note that this may be very hard to show for versions of $v()$ which shrink too quickly, such as $v(i) = \frac{1}{i!}$, so for those cases we must show that it's better than the standard turn order

1) On the value function of 1: It is well known [35] for the Standard Thue-Morse Sequence that

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{T_2(i)}{n+1} = \frac{1}{2} \tag{54}$$

Does this generalize to: ?

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{T_n(i, s)}{n+1} = \sum_{i=0}^{s-1} \frac{i}{s} = \frac{n-1}{2} \tag{55}$$

and

$$\begin{aligned} &\forall x \mid 0 \leq x < s \text{ and } x \in \mathbf{Z} \\ &\lim_{n\to\infty} \sum_{i=0}^{n} \frac{eq\left(T_n(i, s), x\right)}{n+1} = \frac{1}{s} \end{aligned} \tag{56}$$

2) On more general value functions:

B. Palindrome

This seems trivially violated for $n > 2$. Example: $t_3(2) = \langle 0, 1, 2, 1, 2, 0, 2, 0, 1\rangle$

This doesn't even work for power-of-2 bases, like $t_4(2) = \langle 0, 1, 2, 3, 1, 2, 3, 0, 2, 3, 0, 1, 3, 0, 1, 2\rangle$

I am unable to think of any base other than 2 where this property could even conceivably hold true. Probably need to do a proof by contradiction, assuming base ≥ 3

C. Uniform Recurrence

The Thue-Morse Sequence is a uniformly recurrent word: given any finite string X in the sequence, there is some length nX (often much longer than the length of X) such that X appears in every block of length nX. Tackle this by using $T_{n,4}$

D. Cube Free

Given a non-empty word X, $(X \parallel X \parallel X) \notin T_n$. Pretty sure that Square Free implies Cube Free

E. Overlap Free

Given a non-empty word X, $(X \parallel X \parallel X_0) \notin T_n$. For T_2, this is shown in [4]. Note additionally that this apparently is equivalent to 7/3-power-free [38]

F. Square Free

Given a non-empty word X, $(X \parallel X) \notin T_n$

G. Aperiodicity

Not totally sure how this should go, but I think a good approximation would be to show that the distance between the nearest two appearances of a substring grows to infinity much faster than the length of those substrings grows

I seem to remember coming across a paper claiming that (square-free + cube-free) = aperiodic

VII. Acknowledgment

We thank Dan Rowe for helping with the initial work on this paper, as well as Randy Appleton and Lydia Crocker who helped greatly with proofreading and editing.

VIII. Appendix 1 — General Performance Results

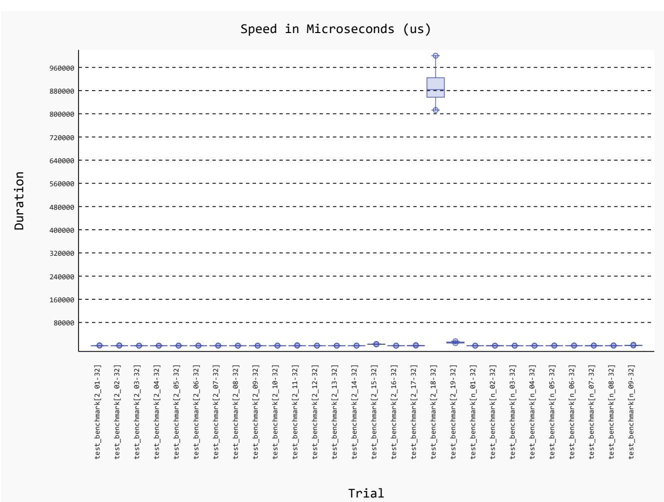

Fig. 2. Benchmark results up to seconds.

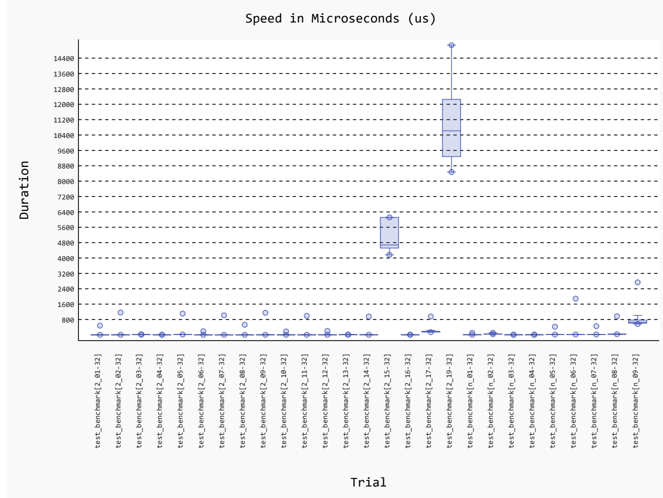

Fig. 3. Benchmark results up to milliseconds.

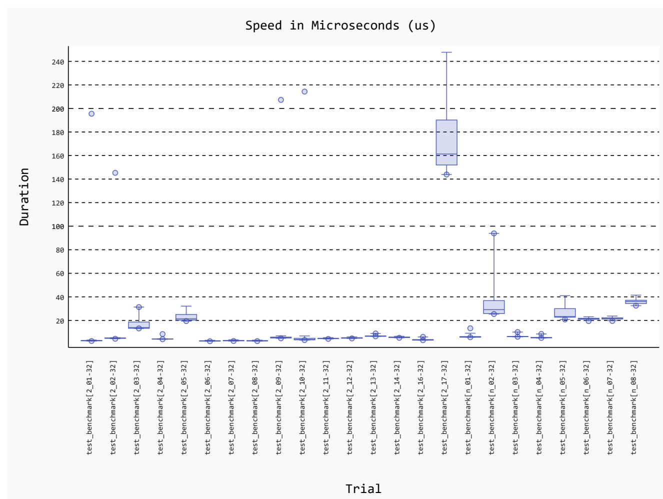

Fig. 4. Benchmark results up to microseconds.

These figures are temporary and will be replaced with something prettier when things are finalized

IX. Appendix 2 — Complexity of Orig. Definitions

A. Complexity of Original Def. 1 — Parity of Hamming Weight

1) Time Complexity: In an idealized case, this definition will simplify to:

$$T_{2,1}(n) = \left(\sum_{i=0}^{\lceil \log_2(n+1) \rceil} \left\lfloor \frac{n}{2^i} \bmod 2 \right\rfloor \right) \bmod 2 \tag{57}$$

This is pretty explicitly $O(\log(n))$ operations. This means that generating the first n entries will take $O(n \log(n))$ operations.

In languages with dynamically sized integers, this can be slightly more complicated. In the above, we perform $\log(n)$ bit shifts, multiplications, moduli, and additions. Since a bit shift is constant time, calculation will be dominated by multiplication, division, and moduli. Each of these take $O(\log(n) \cdot \log(\log(n)))$, where n is the largest number involved. This means that in such languages, we can expect it to take $O(\log(n)^2 \cdot \log(\log(n)))$ operations per element, for $O(n \cdot \log(n)^2 \cdot \log(\log(n)))$ in total.

TABLE I
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 1

	Fixed Size	**Arbitrary Size**
Per Element	$O(\log(n))$	$O(\log(n)^2 \cdot \log(\log(n)))$
In Total	$O(n \cdot \log(n))$	$O(n \cdot \log(n)^2 \cdot \log(\log(n)))$

2) Space Complexity: This is one of the more space-efficient implementations. Each element takes at most the same size as the passed integer. In languages that use Fixed Size integers, that means it will take $O(1)$ space. In languages like Python that use Arbitrary Size integers, it would take $O(\log(n))$ space, where n is the largest element you intend to calculate. If you intend to store all n elements, it will therefore take $O(n)$ or $O(n \cdot \log(n))$ space.

TABLE II
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 1

	Fixed Size	**Arbitrary Size**
Per Element	$O(1)$	$O(\log(n))$
In Total	$O(n)$	$O(n \cdot \log(n))$

B. Complexity of Original Definition 2 — Powers of Negative One

1) Time Complexity: ...

TABLE III
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 2

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE IV
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 2

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

C. Complexity of Original Definition 3 — Root of Unity

1) Time Complexity: ...

TABLE V
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 3

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE VI
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 3

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

D. Complexity of Original Definition 4 — Recursion

1) Time Complexity: At each step in calculation, the value of n passed to the next recursion is halved. This means that it will take $O(\log_2(n))$ recursive steps. Each recursion involves at maximum 2 subtractions and a bit shift. In most languages with Fixed Size integers, this will take constant time. However, in languages with Arbitrary Size integers these subtractions will typically take $O(\log(n))$, where n is the largest integer in the operation. This means we can expect it to take $O(\log(n)^2$ operations.

TABLE VII
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 4

	Fixed Size	**Arbitrary Size**
Per Element	$O(\log(n))$	$O(\log(n)^2)$
In Total	$O(n \cdot \log(n))$	$O(n \cdot \log(n)^2)$

2) Space Complexity: This is one of the more space-efficient implementations. Each element takes at most the same size as the passed integer. In languages that use Fixed Size integers, that means it will take $O(1)$ space. In languages like Python that use Arbitrary Size integers, it would take $O(\log(n))$ space, where n is the largest element you intend to calculate. If you intend to store all n elements, it will therefore take $O(n)$ or $O(n \cdot \log(n))$ space.

TABLE VIII
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 4

	Fixed Size	**Arbitrary Size**
Per Element	$O(1)$	$O(\log(n))$
In Total	$O(n)$	$O(n \cdot \log(n))$

E. Complexity of Original Def. 5 — Floor-Ceiling Difference

1) Time Complexity: memoization doesn't seem to help in the worst case of $1 \dots 1_2$, so you should still end up calculating the value of $b()$ for every positive number less than n

TABLE IX
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 5

	Fixed Size	**Arbitrary Size**
Per Element	$O(n)$	$O(n \cdot \log(n) \cdot \log(\log(n)))$
In Total	$O\left(n^2\right)$	$O\left(n^2 \cdot \log(n) \cdot \log(\log(n))\right)$

2) Space Complexity: There are two ways to implement this algorithm in terms of space complexity. They both have equal worst-case time complexity. The first is to take the recursive approach, and the second is to use dynamic programming.

In a recursive approach, you will end up descending $O(\log(n))$ stack frames, each of which will contain at minimum 1 integer. In the dynamic approach, you will keep a table of all the values of $b()$ from 0 through n. The biggest difference between these approaches is that in the recursive approach you may need to repeat calculations.

TABLE X
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 5

		Fixed Size	Arbitrary Size
Recursive	**Per Element**	$O(\log(n))$	$O(n \cdot \log(n))$
	In Total	$O(n \cdot \log(n))$	$O\left(n^2 \cdot \log(n)\right)$
Dynamic	**Per Element**	$O(n)$	$O(n \cdot \log(n))$
	In Total	$O\left(n^2\right)$	$O\left(n^2 \cdot \log(n)\right)$

F. Complexity of Original Def. 6 — Highest Bit Difference

1) Time Complexity: Since this algorithm works sequentially, and cannot perform computation of an arbitrary element without recursing to the base case, the time is equal on a per-element and in-total basis

TABLE XI
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 6

	Fixed Size	Arbitrary Size
Per Element	$O(n \cdot \log(n))$	$O\left(n \cdot \log(n)^2\right)$
In Total	$O(n \cdot \log(n))$	$O\left(n \cdot \log(n)^2\right)$

2) Space Complexity: ...

TABLE XII
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 6

	Fixed Size	Arbitrary Size
Per Element	$O(1)$	$O(\log(n))$
In Total	$O(n)$	$O(n \cdot \log(n))$

G. Complexity of Original Def. 7 — Hamming Weight Complement

1) Time Complexity: ...

TABLE XIII
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 7

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XIV
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 7

	Fixed Size	Arbitrary Size
Per Element		
In Total		

H. Complexity of Original Definition 8 — Invert and Extend

1) Time Complexity: ...

TABLE XV
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 8

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XVI
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 8

	Fixed Size	Arbitrary Size
Per Element		
In Total		

I. Complexity of Original Def. 9 — Substitute and Flatten

1) Time Complexity: ...

TABLE XVII
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 9

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XVIII
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 9

	Fixed Size	Arbitrary Size
Per Element		
In Total		

J. Complexity of Original Definition 10 — Recursive Rotation

1) Time Complexity: ...

TABLE XIX
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 10

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XX
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 10

	Fixed Size	Arbitrary Size
Per Element		
In Total		

K. Complexity of Orig. Def. 11 — Odious Number Derivation

1) Time Complexity: ...

TABLE XXI
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 11

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXII
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 11

	Fixed Size	Arbitrary Size
Per Element		
In Total		

L. Complexity of Orig. Def. 12 — Evil Numbers Derivation 1

1) Time Complexity: ...

TABLE XXIII
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 12

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXIV
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 12

	Fixed Size	Arbitrary Size
Per Element		
In Total		

M. Complexity of Orig. Def. 13 — Evil Numbers Derivation 2

1) Time Complexity: ...

TABLE XXV
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 13

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXVI
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 13

	Fixed Size	Arbitrary Size
Per Element		
In Total		

N. Complexity of Orig. Def. 14 — Odious & Evil Derivation

1) Time Complexity: ...

TABLE XXVII
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 14

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXVIII
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 14

	Fixed Size	Arbitrary Size
Per Element		
In Total		

O. Complexity of Orig. Def. 15 — Gould's Seq. Derivation

1) Time Complexity: There are two ways one could reasonably calculate this. The first is by building each row of Pascal's Triangle iteratively. This allows you to avoid multiplication whenever possible, and lets you apply a bitmask or modulus operation to take the parity of each entry. The downside is that this version is not parallelizable. Using the bit mask approach, this means that each entry will take $O(n)$ time.

The other is to take advantage of the relation $\binom{n}{k} = \binom{n}{k-1} \cdot \dfrac{n-(k-1)}{k}$. This allows you to calculate each row independently, using $\frac{n}{2}$ moduli, multiplications, and divisions. This means that each entry will take $O(n)$ operations, each of which take $O(\log(n) \cdot \log(\log(n)))$ if with arbitrary sized integers, totaling $O(n)$ or $O(n \cdot \log(n) \cdot \log(\log(n)))$.

TABLE XXIX
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 15

		Fixed Size	Arbitrary Size
Serial	**Per Element**	$O(n)$	$O(n)$
	In Total	$O\left(n^2\right)$	$O\left(n^2\right)$
Parallel	**Per Element**	$O(n)$	$O(n \cdot \log(n) \cdot \log(\log(n)))$
	In Total	$O\left(n^2\right)$	$O\left(n^2 \cdot \log(n) \cdot \log(\log(n))\right)$

2) Space Complexity: ...

TABLE XXX
SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 15

		Fixed Size	Arbitrary Size
Serial	**Per Element**	$O(n)$	$O\left(n \cdot \log(n)\right)$
	In Total	$O\left(n^2\right)$	$O\left(n^2 \cdot \log(n)\right)$
Parallel	**Per Element**	$O(1)$	$O\left(\log(n)\right)$
	In Total	$O(n)$	$O\left(n \cdot \log(n)^2\right)$

P. Complexity of Orig. Def. 16 — Derivation from Blue Code

1) Time Complexity: ...

TABLE XXXI
TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 16

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXXII

SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 16

	Fixed Size	Arbitrary Size
Per Element		
In Total		

Q. Complexity of Original Def. 17 — Generating Function 1

1) Time Complexity: Time complexity of multiplying polynomials via Karatsuba method is $O(n^{\log_2(3)})$. I think this means that, since we need to apply it $\log_2(n)$ times to get n terms, the overall complexity is $O(n^{\log_2(3)} \cdot \log(n))$

TABLE XXXIII

TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 17

	Fixed Size	Arbitrary Size
Per Element	$O\left(n^{\log_2(3)} \cdot \log(n)\right)$?	
In Total	$O\left(n^{\log_2(3)} \cdot \log(n)\right)$	

2) Space Complexity: ...

TABLE XXXIV

SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 17

	Fixed Size	Arbitrary Size
Per Element	$O(n)$?	$O(n\log(n))$?
In Total	$O(n)$?	$O(n\log(n))$?

R. Complexity of Original Def. 18 — Generating Function 2

1) Time Complexity: ...

TABLE XXXV

TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 18

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXXVI

SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 18

	Fixed Size	Arbitrary Size
Per Element		
In Total		

S. Complexity of Original Def. 19 — Hypergeometry

1) Time Complexity: ...

TABLE XXXVII

TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 19

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XXXVIII

SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 19

	Fixed Size	Arbitrary Size
Per Element		
In Total		

T. Complexity of Original Def. 20 — Cellular Automaton

1) Time Complexity: ...

TABLE XXXIX

TIME COMPLEXITY SUMMARY OF STANDARD DEFINITION 20

	Fixed Size	Arbitrary Size
Per Element		
In Total		

2) Space Complexity: ...

TABLE XL

SPACE COMPLEXITY SUMMARY OF STANDARD DEFINITION 20

	Fixed Size	Arbitrary Size
Per Element		
In Total		

X. APPENDIX 3 — COMPLEXITY OF EXT. DEFINITIONS

A. Complexity of Extension Def. 1 — Modular Digit Sums

In an idealized case, this definition will simplify to:

$$T_{n,1}(x,s) = \left(\sum_{i=0}^{\lceil \log_s(x+1) \rceil} \left\lfloor \frac{x}{s^i} \mod s \right\rfloor \right) \mod s \tag{58}$$

This is pretty explicitly $O(\log(n))$ operations. This means that generating the first n entries will take $O(n\log(n))$ operations.

In languages with dynamically sized integers, this can be slightly more complicated. In the above, we perform $\log(n)$ multiplications, moduli, and additions. Since additions are simpler, calculation will be dominated by multiplication, division, and moduli. Each of these take $O(\log(n) \cdot \log(\log(n)))$, where n is the largest number involved. This means that in such languages, we can expect it to take $O(\log(n)^2 \cdot \log(\log(n)))$ operations per element, for $O(n \cdot \log(n)^2 \cdot \log(\log(n)))$ in total.

TABLE XLI

TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 1

	Fixed Size	Arbitrary Size
Per Element	$O(\log(n))$	$O(\log(n)^2 \cdot \log(\log(n)))$
In Total	$O(n \cdot \log(n))$	$O(n \cdot \log(n)^2 \cdot \log(\log(n)))$

1) Space Complexity: This is one of the more space-efficient implementation. Each element takes at most the same size as the passed integer. In languages that use Fixed Size integers, that means it will take $O(1)$ space. In languages like Python that use Arbitrary Size integers, it would take $O(\log(n))$ space, where n is the largest element you intend to calculate. If you intend to store all n elements, it will therefore take $O(n)$ or $O(n \cdot \log(n))$ space.

TABLE XLII
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 1

	Fixed Size	**Arbitrary Size**
Per Element	$O(1)$	$O(\log(n))$
In Total	$O(n)$	$O(n \cdot \log(n))$

B. Complexity of Extension Definition 2 — Roots of Unity

1) Time Complexity: ...

TABLE XLIII
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 2

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE XLIV
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 2

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

C. Complexity of Extension Definition 3 — Recursion

1) Time Complexity: At each step in calculation, the value of n passed to the next recursion is divided by s (the selected base). This means that it will take $O(\log_s(n))$ recursive steps. Each recursion involves at maximum 2 subtractions and a bit shift. In most languages with Fixed Size integers, this will take constant time. However, in languages with Arbitrary Size integers these subtractions will typically take $O(\log(n))$, where n is the largest integer in the operation. This means we can expect it to take $O(\log(n)^2$ operations.

TABLE XLV
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 3

	Fixed Size	**Arbitrary Size**
Per Element	$O(\log(n))$	$O(\log(n)^2)$
In Total	$O(n \cdot \log(n))$	$O(n \cdot \log(n)^2)$

2) Space Complexity: This is one of the more space-efficient implementations. Each element takes at most the same size as the passed integer. In languages that use Fixed Size integers, that means it will take $O(1)$ space. In languages like Python that use Arbitrary Size integers, it would take $O(\log(n))$ space, where n is the largest element you intend to calculate. If you intend to store all n elements, it will therefore take $O(n)$ or $O(n \cdot \log(n))$ space.

TABLE XLVI
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 3

	Fixed Size	**Arbitrary Size**
Per Element	$O(1)$	$O(\log(n))$
In Total	$O(n)$	$O(n \cdot \log(n))$

D. Complexity of Extension Def. 4 — Highest Digit Difference

1) Time Complexity: ...

TABLE XLVII
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 4

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE XLVIII
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 4

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

E. Complexity of Extension Def. 5 — Increment and Extend

1) Time Complexity: ...

TABLE XLIX
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 5

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE L
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 5

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

F. Complexity of Extension Def. 6 — Substitute and Flatten

1) Time Complexity: ...

TABLE LI
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 6

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE LII
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 6

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

G. Complexity of Extension Definition 7 — Recursive Rotation

1) Time Complexity: ...

TABLE LIII
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 7

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE LIV
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 7

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

H. Complexity of Ext. Def. 8 — Latin Square Constructions

1) Time Complexity: ...

TABLE LV
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 8

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

2) Space Complexity: ...

TABLE LVI
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 8

	Fixed Size	**Arbitrary Size**
Per Element		
In Total		

I. Complexity of Extension Def. 9 — Generating Functions

1) Time Complexity: Time complexity of multiplying polynomials via Karatsuba method is $O(n^{\log_2(3)})$. I think this means that, since we need to apply it $\log_s(n)$ times to get n terms, the overall complexity is $O(b^{\log_2(3)} \cdot \log(n))$. I'm a *little* uncertain how the base scales things.

TABLE LVII
TIME COMPLEXITY SUMMARY OF EXTENDED DEFINITION 9

	Fixed Size	**Arbitrary Size**
Per Element	$O(b^{\log_2(3)} \cdot \log(n))$?	
In Total	$O(b^{\log_2(3)} \cdot \log(n))$?	

2) Space Complexity: ...

TABLE LVIII
SPACE COMPLEXITY SUMMARY OF EXTENDED DEFINITION 9

	Fixed Size	**Arbitrary Size**
Per Element	$O(n)$?	$O(n \cdot \log(n))$?
In Total	$O(n)$?	$O(n \cdot \log(n))$?

REFERENCES

[1] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A010060. The Thue-Morse Sequence.

[2] Lukas Spiegelhofer. The level of distribution of the thue–morse sequence. *Compositio Mathematica*, 156(12):2560–2587, 2020.

[3] Jean-Paul Allouche and Jeffrey Shallit. The ubiquitous prouhet-thue-morse sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and their Applications*, pages 1–16, London, 1999. Springer London.

[4] Diyath Pannipitiya. To symbolic dynamics through the thue-morse sequence, 2024.

[5] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A010059. The inversion of the Thue-Morse Sequence.

[6] M. Kolář, M. K. Ali, and Franco Nori. Generalized thue-morse chains and their physical properties. *Phys. Rev. B*, 43:1034–1047, Jan 1991.

[7] Jorg Arndt. *Matters computational*. Springer, Berlin, Germany, October 2010.

[8] Ethan D. Bolker, Carl Offner, Robert Richman, and Catalin Zara. The prouhet–tarry–escott problem and generalized thue–morse sequences. *Journal of Combinatorics*, 7(1):117–133, 2016.

[9] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A000069. The Odious Numbers.

[10] Olivia Appleton-Crocker. Extending the thue-morse sequence source code. Available at: https://github.com/LivInTheLookingGlass/Thue-Morse, https://thue.oliviappleton.com, 2024. This repository contains the source code and data for the research presented in this paper.

[11] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A001969. The Evil Numbers.

[12] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A159481. Number of evil numbers ¡= n.

[13] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A001316. The Gould Sequence.

[14] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A193231. Blue code for n.

[15] Jialin Ma, Isuru Fernando, and Xin Chen. *symengine: Interface to the 'SymEngine' Library*, 2022. Python package version 0.13.0.

[16] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[17] Eric W. Weisstein. Thue-morse sequence. https://mathworld.wolfram.com/Thue-MorseSequence.html, 2024. Accessed: 2024-12-02.

[18] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A106400. The Thue-Morse Sequence, substituting $1 \to -1$ and $0 \to 1$.

[19] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A309303. Expansion of g.f. $(\sqrt{x+1} - \sqrt{1-3x})/(2(x+1)^{3/2})$.

[20] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A357448. The Extended Thue-Morse Sequence in Base 3/2.

[21] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A001285. The inversion of the Thue-Morse Sequence plus one.

[22] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A007413. A squarefree (or Thue-Morse) ternary sequence: closed under $1 \to 123, 2 \to 13, 3 \to 2$. Start with 1.

[23] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A036577. Ternary Thue-Morse sequence: closed under $a \to abc, b \to ac, c \to b$.

[24] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A036585. Ternary Thue-Morse sequence: closed under $a \to abc, b \to ac, c \to b$.

[25] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A036580. Ternary Thue-Morse sequence: closed under $a \to abc, b \to ac, c \to b$.

[26] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A005679. Ternary Thue-Morse sequence: closed under $a \to abc, b \to ac, c \to b$.

[27] Ricardo Astudillo. On a class of thue-morse type sequences. *Journal of Integer Sequences*, 6, 2003.

[28] F. M. Dekking. The thue-morse sequence in base 3/2. *Journal of Integer Sequences*, 26, 2023.

[29] Jeffrey Shallit, Sonja Linghui Shan, and Kai Hsiang Yang. Automatic sequences in negative bases and proofs of some conjectures of shevelev, 2022.

[30] Vladimir Shevelev. Two analogs of thue-morse sequence, 2017.

[31] OEIS Foundation Inc. The on-line encyclopedia of integer sequences (oeis). Accessed: 2024-10-24, 2024. OEIS Sequence ID: A269027. The Extended Thue-Morse Sequence in base -2.

[32] Štěpán Starosta. Generalized thue-morse words and palindromic richness, 2011.

[33] Olga G. Parshina. *On Arithmetic Index in the Generalized Thue-Morse Word*, page 121–131. Springer International Publishing, 2017.

[34] Gerardo González Robert. Generalized thue-morse continued fractions, 2013.

[35] Yi Cai and Vilmos Komornik. Difference of cantor sets and frequencies in thue–morse type sequences, 2020.

[36] Jin Chen and Zhi-Xiong Wen. On the abelian complexity of generalized thue-morse sequences. *Theoretical Computer Science*, 780:66–73, August 2019.

[37] Dzmitry Badziahin and Evgeny Zorin. On generalized thue-morse functions and their values, 2015.

[38] Narad Rampersad. Words avoiding 7/3-powers and the thue-morse morphism, 2003.

XI. Task Sheet

TABLE LIX
Comparison Matrix of the Standard Definitions to Their Extensions

1	3	4	6	8	9	10	11	17
X	X	X						X
1	2	3	4	5	6	7	8	9

TABLE LX
Comparison Matrix of the Standard Definitions

X = done, S = started, O = target

1																			
X	2																		
	O	3																	
X			4																
	S			5															
					6														
						7													
X							8												
								9											
								S	10										
										11									
											12								
												13							
													14						
														15					
															16				
																17			
																	18		
																		19	
																			20

TABLE LXI
COMPARISON MATRIX OF THE EXTENDED DEFINITIONS

X = done, S = started, O = target

1								
X	2							
		3						
			4					
				5				
					6			
						7		
			X				8	
								9

TABLE LXII
LIST OF POSSIBLY PRESERVED PROPERTIES

X = Preserved, N = Not Preserved, L = Likely Modifier

1	Use as a Fair-Share Sequence	LX
2	Palindrome	LN
3	Uniform Recurrence	
4	Cube Free	
5	Overlap Free	
6	Square Free	
7	Aperiodic	

TABLE LXIII
ATTEMPT TO EXTEND

X = Done, N = Not Possible, L = Likely Modifier

1	X
2	LN
3	X
4	X
5	
6	X
7	
8	X
9	X
10	X
11	
12	
13	
14	
15	LN
16	
17	X
18	
19	
20	

XII. NOTES

```
Definitions From OEIS
For n >= 0, a(A004760(n+1)) = 1 - a(n). -
    Vladimir Shevelev, Apr 25 2009
Is this just true of numbers != 10..., or is
    this true of a class of prefixes?

a(A160217(n)) = 1 - a(n). - Vladimir Shevelev,
     May 05 2009

G.f. A(x) satisfies: A(x) = x / (1 - x^2) + (1
     - x) * A(x^2). - Ilya Gutkovskiy

From Bernard Schott, Jan 21 2022: (Start)
a(n) = a(n*2^k) for k >= 0.
a((2^m-1)^2) = (1-(-1)^m)/2 (see Hassan
    Tarfaoui link, Concours General 1990). (
    End)

A004760: numbers not beginning with 10_2
A160217: min incr.ing seq. w/ a(1)=3 + that a(
    n) & n are both in or not in A003159
A003159: Numbers whose binary representation
    ends in an even number of zeros.
-----------------------------------------------
Definitions from OEIS (n -> 1 - 2 * Thue-Morse
    (n): PNNPNPPN)
G.f. A(x) makes 0 = f(A(x), A(x^2), A(x^4))
    where f(u, v, w) = v^3 - 2*u*v*w + u^2*w

G.f. A(x) satisfies 0 = f(A(x), A(x^2), A(x^3)
    , A(x^6)) where f(u1, u2, u3, u6) = u6*u1
```

```
    ^3 - 3*u6*u2*u1^2 + 3*u6*u2^2*u1 - u3*u2
    ^3.

a(n) = B_n(-A038712(1)*0!, ..., -A038712(n)*(n
    -1)!)/n!, where B_n(x_1, ..., x_n) is the
    n-th complete Bell polynomial. See the
    Wikipedia link for complete Bell
    polynomials , and A036040 for the
    coefficients of these partition
    polynomials. - Gevorg Hmayakyan, Jul 10
    2016 (edited by - Wolfdieter Lang, Aug 31
    2016)

a(n) = A008836(A005940(1+n)). [Analogous to
    Liouville's lambda] - Antti Karttunen

A038712: Let k be the exp. of highest pwr of 2
     dividing n (A007814); a(n) = 2^(k+1)-1
A008836: Liouville's function lambda(n) = (-1)
    ^k, where k is number of primes dividing n
     (counted with multiplicity).
A005940: The Doudna sequence
-----------------------------------------
Definitions from OEIS (inverse: 10010110)

If A(n)=(a(0),a(2),...,a(2^n-1)), then A(n+1)
    =(A(n),1-A(n)). - Arie Bos, Jul 27 2022
```