Collatz Conjecture Proof

Kelly Cox

Introduction and Understandings

Key Points

- General Information
- Interesting Items
    - Understanding Binary Difference from + & *
    - Detailed Analysis of Number Evolution through Modular Arithmetic
    - Detailed Analysis of Odd Numbers Flow in Modular Arithmetic: A Flexible Starting Point
- Understanding Power Positions
- The Nx + 1 Rule and Its Exceptions: A Comprehensive Analysis
- Right side Tree from the number 1
    - With examples
- All Numbers Fall Below Themselves
- Conclusion
- Examples of Power Slot for 1023 and 27
- R Studio Code
    - Runs input number but skips steps
        - All even are skips
        - All middle odd numbers are skipped
    - Tree Location from the number 1
        - Multiple examples
    - All 3MOD(4) and the next 1MOD(4) and Steps
        - First 100 results
    - Power Slot Run for any number up to 2^750
        - Results for 27
    - Collatz Map Print out
        - Results of code
    - Prints 1000 numbers
        - 1st 1000 7MOD(8) numbers with Power Slots
- Log example of 27
- Collatz Map
- Patterns
- Summary

General Information

The Collatz conjecture, also known as the 3n + 1 conjecture, states that starting with any positive integer n:

- If n is odd, then multiply it by 3 and add 1: 3n+1.
- If n is even, then divide it by 2: n/2.

All odd numbers will give a result of even numbers, so we will just take two steps at once:

(3n+1)/2

An alternative way to express this formula uses the position location to find the same number:

(((n - 1) / 2) + 1) + n

(n + ((n + 1) /2))

If we want to get from even to its odd counterpart, such as from 4 to 1, we use:

(n – 1) / 3

This is applicable only for 1MOD(3) even numbers.

You Can Divide Odd Numbers

- When you divide an even number, you will find an odd number when you run out of powers of 2 (zeros). Thus, N/2 will find an odd number.
- When you apply (3n+1)/2, you will find an even number when you run out of powers of 2 (ones). Therefore, (3n+1)/2 will find an even number.

This proof shows that all numbers will fall below the start number in three steps for 5MOD(8) and 1MOD(8).

- All 3MOD(8) numbers will go to either 5MOD(8) or 1MOD(8).
- All 7MOD(8) will go to 3MOD(8).
- No odd number can reach 1 unless it is 5MOD(8).

This also explains the loop at 4 → 2 → 1, where 1 is the only 1MOD(8) that breaks the above rules.

Dividing an Odd Number in Half

- The question is: Can we divide an odd number in half?
- The answer is yes. By using (3n+1)/2, we have divided an odd number in half.
- The 4 → 2 → 1 loop occurs because you cannot divide a 2^0 number in half.

    - With 0, we get undefined
    - With 1, we get 4
- Thus, when we divide 1 in half, the result is 4 because it cannot be divided further.

Interchangeable

- 1MOD(4) is the same as 4k+1
- 3MOD(4) is the same as 4k+3
- 1MOD(3) is the same as 3k+1
- Any other examples can be changed to the other format as needed.

Now, let's explain these findings in detail.

Interesting Items

Color Code

I will use the following color code for all MOD(8) numbers:

- 1MOD(8) - Yellow
- 3MOD(8) - Green
- 5MOD(8) - Blue
- 7MOD(8) - Red

(n + ((n + 1) /2))

When we apply this formula to all odd numbers starting at 1, we see a pattern arises. For example, if we start at the number 1 and go to the number 71, we notice a recurring pattern based on the MOD value.

Here is the formula to help you look at odd numbers:

(n + ((n + 1) /2))

We are taking a remainder from the right and moving it to the power.

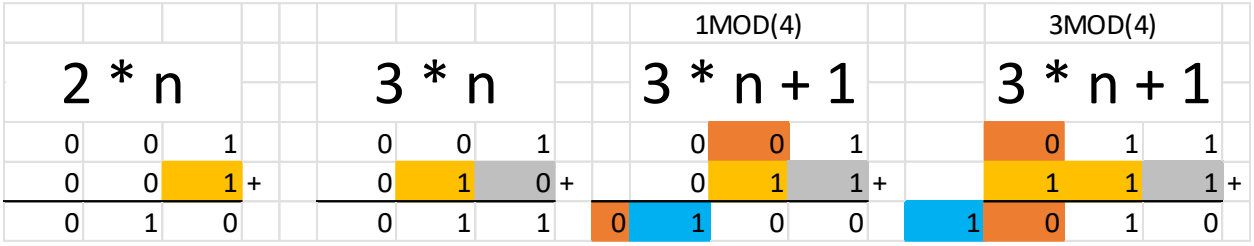
Number	Binary	MOD(8)	3N+1/2	Result	Binary	Diff R-N
1	0000 0001	1	(n + ((n + 1) /2))	2	0000 0010	1
3	0000 0011	3	(n + ((n + 1) /2))	5	0000 0101	2
5	0000 0101	5	(n + ((n + 1) /2))	8	0000 1000	3
7	0000 0111	7	(n + ((n + 1) /2))	11	0000 1011	4
9	0000 1001	1	(n + ((n + 1) /2))	14	0000 1110	5
11	0000 1011	3	(n + ((n + 1) /2))	17	0001 0001	6
13	0000 1101	5	(n + ((n + 1) /2))	20	0001 0100	7
15	0000 1111	7	(n + ((n + 1) /2))	23	0001 0111	8
17	0001 0001	1	(n + ((n + 1) /2))	26	0001 1010	9
19	0001 0011	3	(n + ((n + 1) /2))	29	0001 1101	10
21	0001 0101	5	(n + ((n + 1) /2))	32	0010 0000	11
23	0001 0111	7	(n + ((n + 1) /2))	35	0010 0011	12
25	0001 1001	1	(n + ((n + 1) /2))	38	0010 0110	13
27	0001 1011	3	(n + ((n + 1) /2))	41	0010 1001	14
29	0001 1101	5	(n + ((n + 1) /2))	44	0010 1100	15
31	0001 1111	7	(n + ((n + 1) /2))	47	0010 1111	16
33	0010 0001	1	(n + ((n + 1) /2))	50	0011 0010	17

35	0010 0011	3	(n + ((n + 1) /2))	53	0011 0101	18
37	0010 0101	5	(n + ((n + 1) /2))	56	0011 1000	19
39	0010 0111	7	(n + ((n + 1) /2))	59	0011 1011	20
41	0010 1001	1	(n + ((n + 1) /2))	62	0011 1110	21
43	0010 1011	3	(n + ((n + 1) /2))	65	0100 0001	22
45	0010 1101	5	(n + ((n + 1) /2))	68	0100 0100	23
47	0010 1111	7	(n + ((n + 1) /2))	71	0100 0111	24
49	0011 0001	1	(n + ((n + 1) /2))	74	0100 1010	25
51	0011 0011	3	(n + ((n + 1) /2))	77	0100 1101	26
53	0011 0101	5	(n + ((n + 1) /2))	80	0101 0000	27
55	0011 0111	7	(n + ((n + 1) /2))	83	0101 0011	28
57	0011 1001	1	(n + ((n + 1) /2))	86	0101 0110	29
59	0011 1011	3	(n + ((n + 1) /2))	89	0101 1001	30
61	0011 1101	5	(n + ((n + 1) /2))	92	0101 1100	31
63	0011 1111	7	(n + ((n + 1) /2))	95	0101 1111	32
65	0100 0001	1	(n + ((n + 1) /2))	98	0110 0010	33
67	0100 0011	3	(n + ((n + 1) /2))	101	0110 0101	34
69	0100 0101	5	(n + ((n + 1) /2))	104	0110 1000	35
71	0100 0111	7	(n + ((n + 1) /2))	107	0110 1011	36

Table Explanation:

- All 1MOD(4) numbers go from odd to even after two steps.
  - 1MOD(4) numbers are incapable of moving a 1 bit from the remainder to the power side of the number causing an even result after 2 steps.
- All 3MOD(4) numbers stay odd after two steps.
  - 3MOD(4) numbers move a 1 bit from the remainder to the power side every time.

Understanding Binary Difference from + & *



- We can see that when looking at the binary that 2*n is the same as adding the number to itself.
- When we look at 3*n, we take the binary number, add a 0, and shift to the left. We can now add that number to the original and get the results of multiplying by 3.
- When we look at 3∗+1, we are just replacing the 0 with a 1 and adding that to get the results.
    - 1MOD(4) we see that it cannot move the 1 bit to the left of the power slot
        - The power slot moves from 2^1 to 2^4 or greater
        - This creates the 4 2 1 loop or a greater loop
    - 3MOD(4) we see the power slot doesn't move but a 1 bit moves from the remainder to the power side of the binary number
        - The power slot stays at 2^2 in this case
        - If you look at the remainder of this number after we divide by 2 will take the remainder from 3 to 1 effectively dividing it by half.

All Numbers are Buildable from 8x+r Formula

8	*	0	+	1	=	1	0000 0000 0000 0000 0000 0000 0000 0001
8	*	1	+	1	=	9	0000 0000 0000 0000 0000 0000 0000 1001
8	*	2	+	1	=	17	0000 0000 0000 0000 0000 0000 0001 0001
8	*	3	+	1	=	25	0000 0000 0000 0000 0000 0000 0001 1001
8	*	4	+	1	=	33	0000 0000 0000 0000 0000 0000 0010 0001
8	*	5	+	1	=	41	0000 0000 0000 0000 0000 0000 0010 1001
8	*	6	+	1	=	49	0000 0000 0000 0000 0000 0000 0011 0001
8	*	7	+	1	=	57	0000 0000 0000 0000 0000 0000 0011 1001
8	*	8	+	1	=	65	0000 0000 0000 0000 0000 0000 0100 0001
8	*	9	+	1	=	73	0000 0000 0000 0000 0000 0000 0100 1001
8	*	10	+	1	=	81	0000 0000 0000 0000 0000 0000 0101 0001
8	*	0	+	3	=	3	0000 0000 0000 0000 0000 0000 0000 0011
8	*	1	+	3	=	11	0000 0000 0000 0000 0000 0000 0000 1011
8	*	2	+	3	=	19	0000 0000 0000 0000 0000 0000 0001 0011
8	*	3	+	3	=	27	0000 0000 0000 0000 0000 0000 0001 1011
8	*	4	+	3	=	35	0000 0000 0000 0000 0000 0000 0010 0011
8	*	5	+	3	=	43	0000 0000 0000 0000 0000 0000 0010 1011
8	*	6	+	3	=	51	0000 0000 0000 0000 0000 0000 0011 0011
8	*	7	+	3	=	59	0000 0000 0000 0000 0000 0000 0011 1011
8	*	8	+	3	=	67	0000 0000 0000 0000 0000 0000 0100 0011
8	*	9	+	3	=	75	0000 0000 0000 0000 0000 0000 0100 1011
8	*	10	+	3	=	83	0000 0000 0000 0000 0000 0000 0101 0011
8	*	0	+	5	=	5	0000 0000 0000 0000 0000 0000 0000 0101
8	*	1	+	5	=	13	0000 0000 0000 0000 0000 0000 0000 1101
8	*	2	+	5	=	21	0000 0000 0000 0000 0000 0000 0001 0101
8	*	3	+	5	=	29	0000 0000 0000 0000 0000 0000 0001 1101
8	*	4	+	5	=	37	0000 0000 0000 0000 0000 0000 0010 0101
8	*	5	+	5	=	45	0000 0000 0000 0000 0000 0000 0010 1101
8	*	6	+	5	=	53	0000 0000 0000 0000 0000 0000 0011 0101
8	*	7	+	5	=	61	0000 0000 0000 0000 0000 0000 0011 1101
8	*	8	+	5	=	69	0000 0000 0000 0000 0000 0000 0100 0101
8	*	9	+	5	=	77	0000 0000 0000 0000 0000 0000 0100 1101
8	*	10	+	5	=	85	0000 0000 0000 0000 0000 0000 0101 0101
8	*	0	+	7	=	7	0000 0000 0000 0000 0000 0000 0000 0111
8	*	1	+	7	=	15	0000 0000 0000 0000 0000 0000 0000 1111
8	*	2	+	7	=	23	0000 0000 0000 0000 0000 0000 0001 0111
8	*	3	+	7	=	31	0000 0000 0000 0000 0000 0000 0001 1111
8	*	4	+	7	=	39	0000 0000 0000 0000 0000 0000 0010 0111

8	*	5	+	7	=	47	0000 0000 0000 0000 0000 0000 0010 1111
8	*	6	+	7	=	55	0000 0000 0000 0000 0000 0000 0011 0111
8	*	7	+	7	=	63	0000 0000 0000 0000 0000 0000 0011 1111
8	*	8	+	7	=	71	0000 0000 0000 0000 0000 0000 0100 0111
8	*	9	+	7	=	79	0000 0000 0000 0000 0000 0000 0100 1111
8	*	10	+	7	=	87	0000 0000 0000 0000 0000 0000 0101 0111
8	*	11	+	7	=	95	0000 0000 0000 0000 0000 0000 0101 1111
8	*	12	+	7	=	103	0000 0000 0000 0000 0000 0000 0110 0111
8	*	13	+	7	=	111	0000 0000 0000 0000 0000 0000 0110 1111
8	*	14	+	7	=	119	0000 0000 0000 0000 0000 0000 0111 0111
8	*	15	+	7	=	127	0000 0000 0000 0000 0000 0000 0111 1111
8	*	31	+	7	=	255	0000 0000 0000 0000 0000 0000 1111 1111
8	*	63	+	7	=	511	0000 0000 0000 0000 0000 0001 1111 1111
8	*	127	+	7	=	1,023	0000 0000 0000 0000 0000 0011 1111 1111
8	*	255	+	7	=	2,047	0000 0000 0000 0000 0000 0111 1111 1111
8	*	511	+	7	=	4,095	0000 0000 0000 0000 0000 1111 1111 1111
8	*	1023	+	7	=	8,191	0000 0000 0000 0000 0001 1111 1111 1111
8	*	2047	+	7	=	16,383	0000 0000 0000 0000 0011 1111 1111 1111
8	*	4095	+	7	=	32,767	0000 0000 0000 0000 0111 1111 1111 1111
8	*	8191	+	7	=	65,535	0000 0000 0000 0000 1111 1111 1111 1111
8	*	16383	+	7	=	131,071	0000 0000 0000 0001 1111 1111 1111 1111
8	*	32767	+	7	=	262,143	0000 0000 0000 0011 1111 1111 1111 1111
8	*	65535	+	7	=	524,287	0000 0000 0000 0111 1111 1111 1111 1111
8	*	131071	+	7	=	1,048,575	0000 0000 0000 1111 1111 1111 1111 1111
8	*	262143	+	7	=	2,097,151	0000 0000 0001 1111 1111 1111 1111 1111
8	*	524287	+	7	=	4,194,303	0000 0000 0011 1111 1111 1111 1111 1111
8	*	1048575	+	7	=	8,388,607	0000 0000 0111 1111 1111 1111 1111 1111
8	*	2097151	+	7	=	16,777,215	0000 0000 1111 1111 1111 1111 1111 1111

Structure

- Each row represents a calculation of the form 8×n+r.
- Columns show the calculation, the result, and the binary representation of the result.
- The table is divided into four groups based on the value of r: 1, 3, 5, and 7.

Patterns and Observations

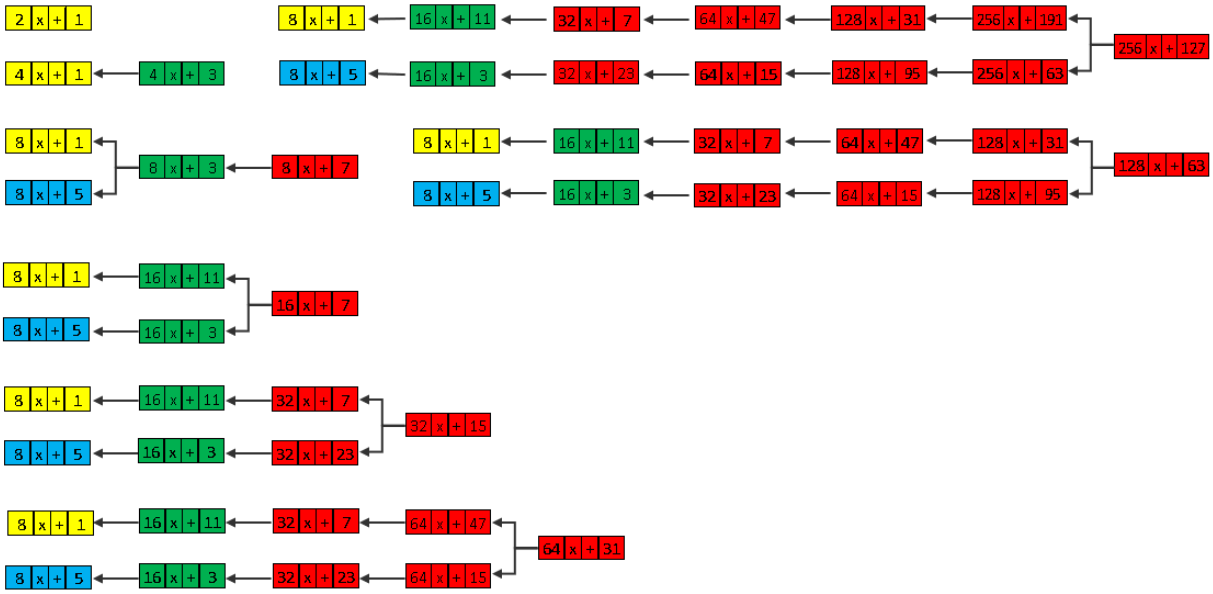
- **For r=1**:
  - The binary representation for numbers of the form 8n+1 ends with a '1' bit, indicating they are always odd.
- **For r=3**:
  - The binary representation for numbers of the form 8n+3 ends with '11', indicating they are always odd.

- **For r=5**:
    - The binary representation for numbers of the form 8n+5 ends with '101', indicating they are always odd.
- **For r=7**:
    - The binary representation for numbers of the form 8n+7 ends with '111', indicating they are always odd.
    - The Red show the next MOD level of changes
    - They are created by taking the previous n and doing 2x+1 on n
        - n = 0, 1, 3, 7, 15, 31 ....

Binary Analysis

- **Odd numbers**: All numbers generated with r=1,3,5,7 are odd. This is consistent with the fact that adding an odd number r to 8n (which is always even) results in an odd number.
- **Consistent pattern**: For each r, there is a consistent pattern in the binary representation which can be used to predict the number's properties and behavior in transformations.

All Numbers Follow a Specific Path



Detailed Analysis of Number Evolution through Modular Arithmetic

The provided flowchart illustrates the transformation paths of numbers based on their modular properties. This analysis will help in understanding how numbers evolve through different modular levels and how this can be useful in studying number theory and the Collatz Conjecture.

Initial Step: 2x+1

- **Starting Point**:
  - Numbers of the form 2x+1 are always odd.
  - In binary representation, they end with a '1' bit.

Second Step: 4x+1 and 4x+3

- **Splitting Paths**:
  - Numbers of the form 2x+1 split into two paths: 4x+1 and 4x+3.
  - Both paths maintain the odd property.
- **Binary Representation**:
  - 4x+1 ends in '001' (odd).
  - 4x+3 ends in '011' (odd).

Third Step: 8x+r

- **Further Splitting**:
  - Each number from the previous step further divides into four paths: 8x+1, 8x+3, 8x+5, and 8x+7.
  - These represent all possible odd numbers in the next modular level.
- **Binary Representation**:
  - 8x+1: ends in '0001'.
  - 8x+3: ends in '0011'.
  - 8x+5: ends in '0101'.
  - 8x+7: ends in '0111'.

Subsequent Steps: Higher Modular Levels

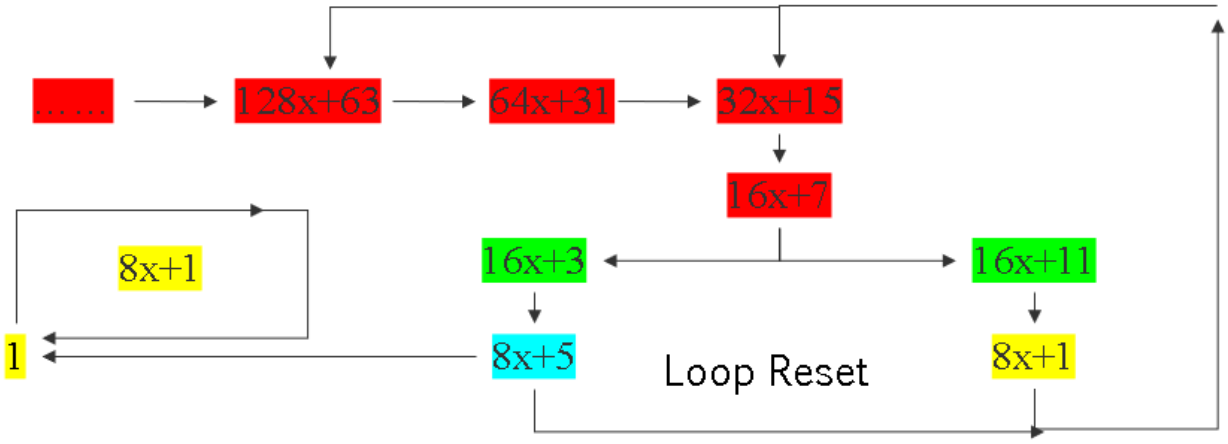
- **General Pattern**:
  - As numbers evolve, they continue to split into higher modular levels: 16x+r, 32x+r, 64x+r, etc.
  - Each step follows the rule of adding multiples of the current modular base to the remainder r.
- **Modular Patterns**:
  - Each higher-level shows how numbers are distributed among possible remainders.
  - This helps predict future states of numbers based on their current modular form.

Application to the Collatz Conjecture

- **Transformation Rule**:
  - Under the Collatz transformation, odd numbers transform via 3n+1 and subsequent division by 2.
  - Understanding the modular paths helps in predicting how numbers change and potentially converge.

Practical Implications

- **Predictive Power**:
  - Recognizing patterns in the modular evolution of numbers provides predictive power in number theory.
  - It helps identify which numbers may converge quickly or take longer under specific transformations.
- **Binary Shifts**:
  - Each modular step corresponds to specific shifts in binary representation.
  - Tracking these shifts helps understand the binary behavior of numbers under transformations.



Detailed Analysis of Odd Numbers Flow in Modular Arithmetic: A Flexible Starting Point

The diagram illustrates how odd numbers transform through various modular levels, emphasizing their cyclical behavior and eventual reduction to 1. This analysis will help understand the progression of numbers across different modular levels, aiding in the exploration of number theory and the Collatz Conjecture.

Flexible Starting Points

1. **Initial Modular Forms**:
   - Numbers can start from any modular form, such as 8x+1, 8x+3, 8x+5, or 8x+7.
   - Each modular form follows its transformation path through modular levels.
2. **Transformation Paths**:
   - Regardless of the starting point, numbers follow a specific sequence of modular transformations.
   - The paths include transformations such as 16x+r, 32x+r, 64x+r, and so on, where r is the remainder that defines the modular form.
3. **Cyclical Patterns**:
   - Numbers transform through modular levels, reducing to lower modular forms in a cyclical pattern.
   - This cyclical behavior ensures that numbers eventually reach a repeating loop involving 5MOD(8), leading to 1.

Flow Diagram Analysis with Flexible Starting Points

1. **From Any Modular Form (e.g., 8x+5)**:
   - **Initial Step**:
     - Start from 8x+5 (highlighted in blue).
     - Transform through the path: 128x+63 → 64x+31 → 32x+15 → 16x+7 → 16x+3 or 16x+11 → and so on.

    - **Cyclical Transformation**:
        - The transformations loop back through modular resets, always reducing to lower forms.
        - Eventually, numbers reach 5MOD(8) and follow a predictable path to 1.
2. **General Patterns**:
    - Each starting modular form follows a transformation path that reduces to a lower modular form.
    - The cyclical behavior is consistent across all starting points, ensuring eventual convergence to 1.

Practical Implications

1. **Collatz Conjecture**:
    - Understanding these transformation paths helps analyze the behavior of sequences under the Collatz Conjecture.
    - The flexible starting points show that any odd number will eventually reduce to 1 through predictable modular transformations.
2. **Binary Representation**:
    - Each modular form corresponds to specific binary representations.
    - For instance, 8x+5 corresponds to binary numbers ending in '0101', showing how bits shift through transformations.
3. **Predictive Power**:
    - Recognizing the modular transformation patterns allows for predictive analysis.
    - This helps determine how quickly numbers will reach 1 aiding in studying the Collatz Conjecture.

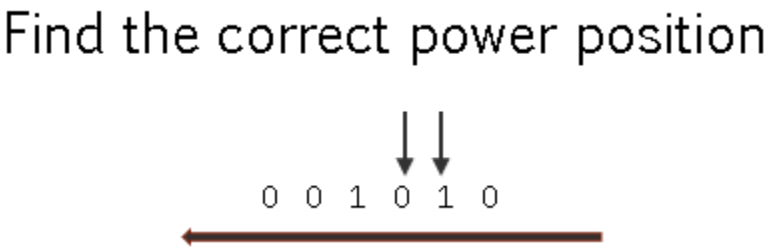
Conclusion

The flow diagram and analysis demonstrate that odd numbers can start from any modular form and follow a predictable transformation path. Modular transformations reveal a hidden pattern: numbers cyclical nature finally gives way to a steadfast finale of 1, offering a glimpse into their behavior under systematic changes.

Understanding Power Positions

Understanding the Correct Power Position in Binary Representation 2^(x=1) Format

This method explains how to determine the correct power position of a number in its binary form, using the 2^(x+1) format. Pinpointing the modular form relies on watching how numbers change shape under transformations.



Steps to Find the Correct Power Position in 2^(x+1) Format

1. **Read the Binary Representation Right to Left**:
   - Begin reading the binary digits from right to left.
2. **Identify the First Group of Ones**:
   - Locate the first group of consecutive '1' bits in the binary sequence.
3. **Find the First Zero After the Group of Ones**:
   - After identifying the group of '1' bits, look for the first '0' bit that follows this group.
4. **Determine the Power Position in 2^(x+1) Format**:
   - The position of this '0' bit, when counted from right to left, is referred to as the power position.
   - The power position is represented in 2^(x+1) format, where x is the position of the '0' bit.

Example Analysis

- **Binary Example**: 001010
  - Reading from right to left: 0, 0, 1, 0, 1, 0
  - First group of ones: '10'
  - First zero after the group: '0' (at the 3rd position from the right)
- **Power Position**:
  - The power position in this example is the 3rd position from the right.
  - In 2^(x+1) format, this is represented as 2^3.

Purpose and Ramifications

1. **Identifying Modular Forms**:
    - This method helps identify the modular form of a number, which is crucial for analyzing its behavior under various transformations.
2. **Predicting Number Transformations**:
    - Understanding the modular form allows for accurate predictions of how numbers will transform through modular arithmetic sequences.
    - Studying sequences, like those found in the Collatz Conjecture, becomes particularly fascinating with this approach.
3. **Binary Analysis**:
    - By understanding the binary of the numbers we can follow them through the properties and transformations of numbers.
    - It highlights how bit positions influence the modular form and subsequent behavior of numbers.

Conclusion

This method of finding the correct power position in binary representation, using the 2^(x+1) format, provides a systematic way to determine the modular form of numbers. By following the steps to read the binary digits, identify groups of ones, and locate the power position, one can accurately calculate the modular form.

Power Position with 4x+3 & 4x+1 numbers

- **4x+3 Numbers**
    - Whenever you multiply a number of the form 4x+3 by 3, it affects the power side of the number as 3x+1 happens before you add 1 to the remainder side.
        - 3MOD(8) will always to go 5MOD(8) numbers
        - 7MOD(8) will stay 7MOD(8) until the change to 3MOD(8)
            - This can be calculated by checking the power slot
            - The power slot will tell you how many steps until it changes to 3MOD(8)
- **4x+1 Numbers**
    - The numbers in the sequence 4x+1 cannot achieve this, and the addition of 1 does not shift the remainder to the power side. This results in the repetitive loop of 4, 2, 1.
    - The loop is happening because the power slot is being reset and moved and the power slot will be 2^4 or greater.
        - For 1MOD(8) it will be 2^4 every time
        - For 5MOD(8) it will be 2^5 or greater every time
            - You can calculate this by checking the power slot on even numbers after any 1MOD(4) odd number.

We can skip all even numbers using these formulas

- **1MOD(8)**
  - **2 Steps = (((((N - 1) / 2)) + ((N - (((N - 1) / 2))) / 2)) * 2) + 1**
  - **3 Steps = (((((N - 1) / 2)) - ((((N - 1) / 2)) / 4)) * 2) + 1**
- **5MOD(5)**
  - **2 Steps = (((((N - 1) / 2)) + ((N - (((N - 1) / 2))) / 2)) * 2) + 1**
  - **3 Steps = (((((N - 1) / 2)) - ((((N - 1) / 2)) / 4)) * 2) + 1**
- **7MOD(8)**
  - **2 Steps = (((((N - 1) / 2)) + ((N - (((N - 1) / 2))) / 2)) * 2) + 1**
- **3MOD(8)**
  - **2 Steps = (((((N - 1) / 2)) + ((N - (((N - 1) / 2))) / 2)) * 2) + 1**
  - **5 Steps =(((((N - 1) / 2)) + ((((N - (((N - 1) / 2))) / 2) + 1) / 4)) * 2) + 1**

- **Knowing these formulas plus understanding binary we can know with exact precision where the next odd numbers are.**
  - If I am at a 1MOD(8) I can use the 3 step to get to the next odd number
  - If I am at a 5MOD(8) I can look at the binary and know how far the next odd number is and use this formula
    - (3n+1)/B
    - B is where the first 0 before the first 1 bit, looking at the results of our 3n+1 number.
  - If I am at a 3MOD(8) I will just do 2 steps because it will be odd.
    - I can skip 5 steps if I know 2 steps is a 1MOD(8) but this requires us doing extra calculations that we can just go 2 steps and not have to worry about
  - If I am at a 7MOD(8) I can skip to the next 1MOD(4) by looking at the binary of the odd number and finding its power position and skipping all the odd numbers between them

The Nx + 1 Rule and Its Exceptions: A Comprehensive Analysis

In my exploration of the Collatz Conjecture, I observed a general pattern that applies when multiplying numbers together. The pattern involves a shift from the remainder to the power side of the power slot in binary form, following a straightforward rule known as the Nx+1 rule. Here, N is the constant multiplier, x is the power of the second number, and the formula is N·x+1. However, there are interesting exceptions, particularly when the multiplier is 3.

General Pattern: The Nx + 1 Rule

For most numbers, multiplying x by a constant N and then adding 1 produces the next number in the sequence. This rule demonstrates a shift from the remainder to the power side of the power slot. This can be summarized as follows:

N·x+1=Next Number

Specific Multipliers and Their Exceptions

Multiplier 4:
- **General Rule:** 4·x+1

Multiplier 5:
- **General Rule:** 5·x+1

Multiplier 6:
- **General Rule:** 6·x+1
- **Special Rule:** For x MOD 4 =3, use 6·x+2.

Multiplier 7:
- **General Rule:** 7·x+1
- **Special Rule 1:** For x MOD 8=3, use 7·x+2.
- **Special Rule 2:** For x MOD 8=7, use 7·x+3.

The Big Exception: Multiplier 3

With the number 3, we encounter a significant exception to the usual pattern. Here's a detailed breakdown:

Multiplier 3:
- **General Rule:** For any x MOD 4 ≠ 1:
  - 3·x+1 = Next Number
- **Special Rule:** For x MOD 4 = 1:
  - 3·x+0 = Next Number

Explanation of the Exception for Multiplier 3

When we multiply numbers and look at their binary representation in terms of power and remainder, N is our constant number, and x represents the power of the second number. For example, when multiplying 3 and 1:

- **Example Calculation:**
  - Multiply: $3 \cdot 1 = 3$
  - N (3) is our constant multiplier.
  - x (1) is the number whose power we need to determine.
  - In binary, the power of 1 is 4 (as in $2^{(0+1)}$), so the power/remainder relationship is: $4 \cdot 0 + 1 = 1$
  - Multiplying this by 3 gives: $3 \cdot 1 = 3$
  - The result is a power of 4 as well: $4 \cdot 0 + 3 = 3$

However, when x MOD 4=1 and we multiply by 3, it does not shift to the power side as expected. Instead, it creates a loop and resets the power slot. This unique behavior distinguishes it from other multipliers.

Why the Exception for Multiplier 3 is Significant

For x MOD 4 = 1, multiplying by 3 and adding 0 (instead of 1) means these numbers do not follow the usual transition from the remainder to the power side of the power slot. Instead, they create a loop, effectively resetting the power slot. This unique behavior sets them apart from other numbers and makes 3 a special case in the Collatz Conjecture.

Summary

While most numbers conform to the Nx+1 rule with a few exceptions, the number 3 introduces a significant change. For all other multipliers, the exceptions involve adding an extra value based on specific remainders. However, with 3, the x MOD 4 = 1 numbers do not transition in the same way, instead creating a loop that resets their power slot.

Visual Aids

Yellow is our constant number
Orange is or random number
Red is the result of multiplying
Green is our starting power
Blue is our result power

If you take yellow and multiply by green you will get blue under normal rules

- **(Y*G)+1 = B**
- **The above rules change this for some numbers**

For the number 3 there is only 1 rule change that causes the conjecture to work

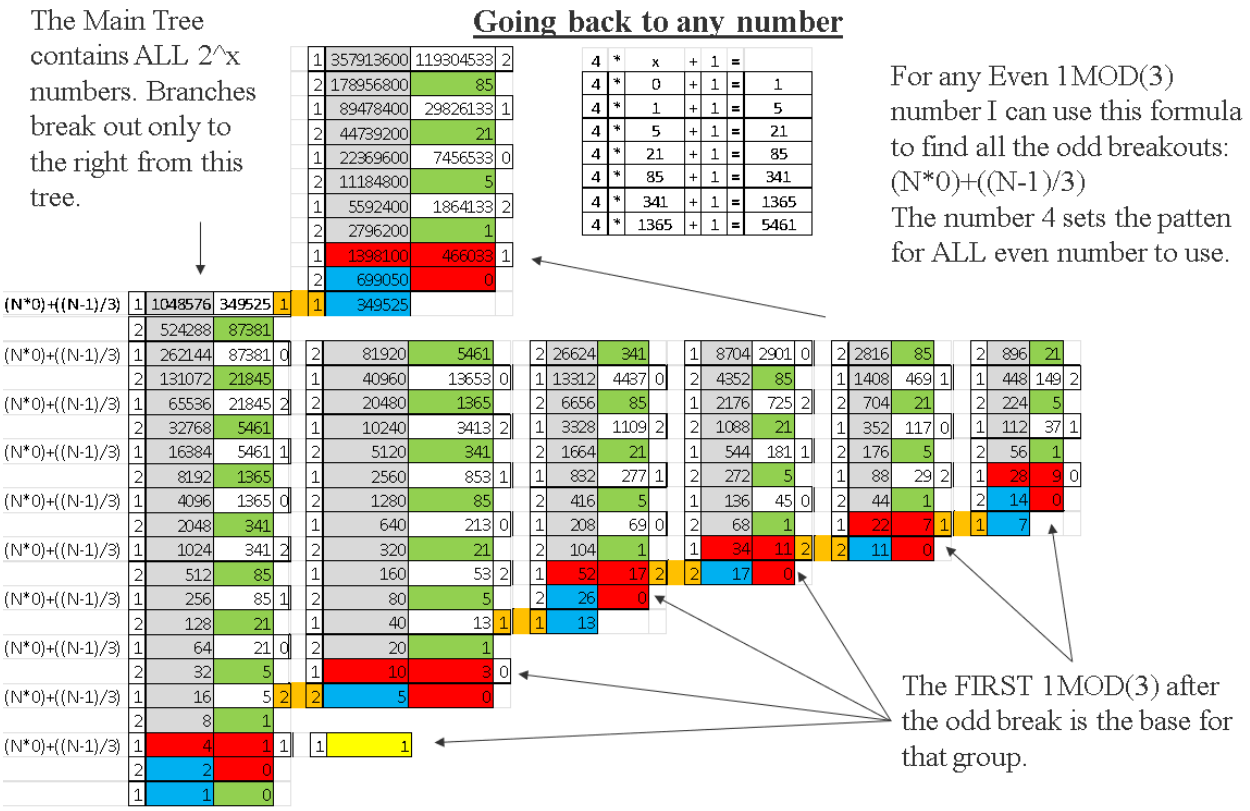
- **If our number is 3MOD(4) process as normal 3n+1**
- **If our number is 1MOD(4) do a loop and reset 3n+0**

																			MOD 4
1	0000 0000 0001	*	4	0000 0000 0100	4	*	0	+	1	=	4	0000 0000 0100	4	*	1	+	0	1
3	0000 0000 0011	*	4	0000 0000 0100	8	*	0	+	3	=	12	0000 0000 1100	8	*	1	+	4	3
5	0000 0000 0101	*	4	0000 0000 0100	4	*	1	+	1	=	20	0000 0001 0100	4	*	5	+	0	1
7	0000 0000 0111	*	4	0000 0000 0100	16	*	0	+	7	=	28	0000 0001 1100	16	*	1	+	12	3
9	0000 0000 1001	*	4	0000 0000 0100	4	*	2	+	1	=	36	0000 0010 0100	4	*	9	+	0	1
11	0000 0000 1011	*	4	0000 0000 0100	8	*	1	+	3	=	44	0000 0010 1100	8	*	5	+	4	3
13	0000 0000 1101	*	4	0000 0000 0100	4	*	3	+	1	=	52	0000 0011 0100	4	*	13	+	0	1
15	0000 0000 1111	*	4	0000 0000 0100	32	*	0	+	15	=	60	0000 0011 1100	32	*	1	+	28	3
17	0000 0001 0001	*	4	0000 0000 0100	4	*	4	+	1	=	68	0000 0100 0100	4	*	17	+	0	1
19	0000 0001 0011	*	4	0000 0000 0100	8	*	2	+	3	=	76	0000 0100 1100	8	*	9	+	4	3
21	0000 0001 0101	*	4	0000 0000 0100	4	*	5	+	1	=	84	0000 0101 0100	4	*	21	+	0	1
23	0000 0001 0111	*	4	0000 0000 0100	16	*	1	+	7	=	92	0000 0101 1100	16	*	5	+	12	3
25	0000 0001 1001	*	4	0000 0000 0100	4	*	6	+	1	=	100	0000 0110 0100	4	*	25	+	0	1
27	0000 0001 1011	*	4	0000 0000 0100	8	*	3	+	3	=	108	0000 0110 1100	8	*	13	+	4	3
29	0000 0001 1101	*	4	0000 0000 0100	4	*	7	+	1	=	116	0000 0111 0100	4	*	29	+	0	1
31	0000 0001 1111	*	4	0000 0000 0100	64	*	0	+	31	=	124	0000 0111 1100	64	*	1	+	60	3

																			MOD 4
1	0000 0000 0001	*	5	0000 0000 0101	4	*	0	+	1	=	5	0000 0000 0101	4	*	1	+	1	1
3	0000 0000 0011	*	5	0000 0000 0101	8	*	0	+	3	=	15	0000 0000 1111	8	*	1	+	7	3
5	0000 0000 0101	*	5	0000 0000 0101	4	*	1	+	1	=	25	0000 0001 1001	4	*	6	+	1	1
7	0000 0000 0111	*	5	0000 0000 0101	32	*	0	+	7	=	35	0000 0010 0011	32	*	1	+	3	3
9	0000 0000 1001	*	5	0000 0000 0101	4	*	2	+	1	=	45	0000 0010 1101	4	*	11	+	1	1
11	0000 0000 1011	*	5	0000 0000 0101	8	*	1	+	3	=	55	0000 0011 0111	8	*	6	+	7	3
13	0000 0000 1101	*	5	0000 0000 0101	4	*	3	+	1	=	65	0000 0100 0001	4	*	16	+	1	1
15	0000 0000 1111	*	5	0000 0000 0101	64	*	0	+	15	=	75	0000 0100 1011	64	*	1	+	11	3
17	0000 0001 0001	*	5	0000 0000 0101	4	*	4	+	1	=	85	0000 0101 0101	4	*	21	+	1	1
19	0000 0001 0011	*	5	0000 0000 0101	8	*	2	+	3	=	95	0000 0101 1111	8	*	11	+	7	3
21	0000 0001 0101	*	5	0000 0000 0101	4	*	5	+	1	=	105	0000 0110 1001	4	*	26	+	1	1
23	0000 0001 0111	*	5	0000 0000 0101	16	*	1	+	7	=	115	0000 0111 0011	16	*	7	+	3	3
25	0000 0001 1001	*	5	0000 0000 0101	4	*	6	+	1	=	125	0000 0111 1101	4	*	31	+	1	1
27	0000 0001 1011	*	5	0000 0000 0101	8	*	3	+	3	=	135	0000 1000 0111	8	*	16	+	7	3
29	0000 0001 1101	*	5	0000 0000 0101	4	*	7	+	1	=	145	0000 1001 0001	4	*	36	+	1	1
31	0000 0001 1111	*	5	0000 0000 0101	128	*	0	+	31	=	155	0000 1001 1011	128	*	1	+	27	3

																			MOD 4
1	0000 0000 0001	*	6	0000 0000 0110	4	*	0	+	1	=	6	0000 0000 0110	4	*	1	+	2	1
3	0000 0000 0011	*	6	0000 0000 0110	8	*	0	+	3	=	18	0000 0001 0010	8	*	2	+	2	3
5	0000 0000 0101	*	6	0000 0000 0110	4	*	1	+	1	=	30	0000 0001 1110	4	*	7	+	2	1
7	0000 0000 0111	*	6	0000 0000 0110	16	*	0	+	7	=	42	0000 0010 1010	16	*	2	+	10	3
9	0000 0000 1001	*	6	0000 0000 0110	4	*	2	+	1	=	54	0000 0011 0110	4	*	13	+	2	1
11	0000 0000 1011	*	6	0000 0000 0110	8	*	1	+	3	=	66	0000 0100 0010	8	*	8	+	2	3
13	0000 0000 1101	*	6	0000 0000 0110	4	*	3	+	1	=	78	0000 0100 1110	4	*	19	+	2	1
15	0000 0000 1111	*	6	0000 0000 0110	32	*	0	+	15	=	90	0000 0101 1010	32	*	2	+	26	3
17	0000 0001 0001	*	6	0000 0000 0110	4	*	4	+	1	=	102	0000 0110 0110	4	*	25	+	2	1
19	0000 0001 0011	*	6	0000 0000 0110	8	*	2	+	3	=	114	0000 0111 0010	8	*	14	+	2	3
21	0000 0001 0101	*	6	0000 0000 0110	4	*	5	+	1	=	126	0000 0111 1110	4	*	31	+	2	1
23	0000 0001 0111	*	6	0000 0000 0110	16	*	1	+	7	=	138	0000 1000 1010	16	*	8	+	10	3
25	0000 0001 1001	*	6	0000 0000 0110	4	*	6	+	1	=	150	0000 1001 0110	4	*	37	+	2	1
27	0000 0001 1011	*	6	0000 0000 0110	8	*	3	+	3	=	162	0000 1010 0010	8	*	20	+	2	3
29	0000 0001 1101	*	6	0000 0000 0110	4	*	7	+	1	=	174	0000 1010 1110	4	*	43	+	2	1
31	0000 0001 1111	*	6	0000 0000 0110	64	*	0	+	31	=	186	0000 1011 1010	64	*	2	+	58	3

																			MOD 8
1	0000 0000 0001	*	7	0000 0000 0111	4	*	0	+	1	=	7	0000 0000 0111	4	*	1	+	3	1
3	0000 0000 0011	*	7	0000 0000 0111	8	*	0	+	3	=	21	0000 0001 0101	8	*	2	+	5	3
5	0000 0000 0101	*	7	0000 0000 0111	4	*	1	+	1	=	35	0000 0010 0011	4	*	8	+	3	5
7	0000 0000 0111	*	7	0000 0000 0111	16	*	0	+	7	=	49	0000 0011 0001	16	*	3	+	1	7
9	0000 0000 1001	*	7	0000 0000 0111	4	*	2	+	1	=	63	0000 0011 1111	4	*	15	+	3	1
11	0000 0000 1011	*	7	0000 0000 0111	8	*	1	+	3	=	77	0000 0100 1101	8	*	9	+	5	3
13	0000 0000 1101	*	7	0000 0000 0111	4	*	3	+	1	=	91	0000 0101 1011	4	*	22	+	3	5
15	0000 0000 1111	*	7	0000 0000 0111	32	*	0	+	15	=	105	0000 0110 1001	32	*	3	+	9	7
17	0000 0001 0001	*	7	0000 0000 0111	4	*	4	+	1	=	119	0000 0111 0111	4	*	29	+	3	1
19	0000 0001 0011	*	7	0000 0000 0111	8	*	2	+	3	=	133	0000 1000 0101	8	*	16	+	5	3
21	0000 0001 0101	*	7	0000 0000 0111	4	*	5	+	1	=	147	0000 1001 0011	4	*	36	+	3	5
23	0000 0001 0111	*	7	0000 0000 0111	16	*	1	+	7	=	161	0000 1010 0001	16	*	10	+	1	7
25	0000 0001 1001	*	7	0000 0000 0111	4	*	6	+	1	=	175	0000 1010 1111	4	*	43	+	3	1
27	0000 0001 1011	*	7	0000 0000 0111	8	*	3	+	3	=	189	0000 1011 1101	8	*	23	+	5	3
29	0000 0001 1101	*	7	0000 0000 0111	4	*	7	+	1	=	203	0000 1100 1011	4	*	50	+	3	5
31	0000 0001 1111	*	7	0000 0000 0111	64	*	0	+	31	=	217	0000 1101 1001	64	*	3	+	25	7
																			MOD 4
1	0000 0000 0001	*	3	0000 0000 0011	4	*	0	+	1	=	3	0000 0000 0011	4	*	0	+	3	1
3	0000 0000 0011	*	3	0000 0000 0011	8	*	0	+	3	=	9	0000 0000 1001	8	*	1	+	1	3
5	0000 0000 0101	*	3	0000 0000 0011	4	*	1	+	1	=	15	0000 0000 1111	4	*	3	+	3	1
7	0000 0000 0111	*	3	0000 0000 0011	16	*	0	+	7	=	21	0000 0001 0101	16	*	1	+	5	3
9	0000 0000 1001	*	3	0000 0000 0011	4	*	2	+	1	=	27	0000 0001 1011	4	*	6	+	3	1
11	0000 0000 1011	*	3	0000 0000 0011	8	*	1	+	3	=	33	0000 0010 0001	8	*	4	+	1	3
13	0000 0000 1101	*	3	0000 0000 0011	4	*	3	+	1	=	39	0000 0010 0111	4	*	9	+	3	1
15	0000 0000 1111	*	3	0000 0000 0011	32	*	0	+	15	=	45	0000 0010 1101	32	*	1	+	13	3
17	0000 0001 0001	*	3	0000 0000 0011	4	*	4	+	1	=	51	0000 0011 0011	4	*	12	+	3	1
19	0000 0001 0011	*	3	0000 0000 0011	8	*	2	+	3	=	57	0000 0011 1001	8	*	7	+	1	3
21	0000 0001 0101	*	3	0000 0000 0011	4	*	5	+	1	=	63	0000 0011 1111	4	*	15	+	3	1
23	0000 0001 0111	*	3	0000 0000 0011	16	*	1	+	7	=	69	0000 0100 0101	16	*	4	+	5	3
25	0000 0001 1001	*	3	0000 0000 0011	4	*	6	+	1	=	75	0000 0100 1011	4	*	18	+	3	1
27	0000 0001 1011	*	3	0000 0000 0011	8	*	3	+	3	=	81	0000 0101 0001	8	*	10	+	1	3
29	0000 0001 1101	*	3	0000 0000 0011	4	*	7	+	1	=	87	0000 0101 0111	4	*	21	+	3	1
31	0000 0001 1111	*	3	0000 0000 0011	64	*	0	+	31	=	93	0000 0101 1101	64	*	1	+	29	3



All Numbers Can Be Calculated to a Specific Location on a Right-Side-Only Tree

Main Ideas:

1. **Right-Side-Only Tree Structure:**
   - Each number in the Collatz sequence can be traced back to a specific location on a structured right-side-only tree.
   - This tree helps visualize the sequence's progression and provides an organized way to calculate the position of any number.
2. **Base Calculation:**
   - The process of tracing numbers backward allows us to identify base numbers, which are crucial for maintaining the tree's structure.
   - These base numbers set the pattern for all subsequent branches in the tree.

The Base Tree 4x+1 Creates the Pattern for All Branches

Main Ideas:

1. **Base Tree Definition:**
   - The base tree is defined by the function 4x+1.
   - This function creates a repeatable and predictable pattern that forms the foundation of the tree structure.

2. **Pattern Formation:**
    - The pattern created by 4x+1 ensures that all branches follow a consistent framework.
    - Each branch is an extension of this base pattern, providing a systematic way to map out the tree.
3. **Role and Significance:**
    - The base tree's pattern is crucial for understanding the distribution and behavior of numbers within the Collatz sequence.
    - It ensures that the tree structure remains organized and predictable.

Going Back from 1

Main Ideas:

1. **Tracing Process:**
    - Tracing back from 1 involves identifying all possible predecessors that could lead to 1 according to the Collatz rules.
    - This process uses the formula $(N*0)+((N-1) / 3)$ to find the predecessors.
2. **Visual Representation:**
    - The tree structure visually represents how each number is connected back to 1.
    - This helps in understanding the reverse path from 1 to any number.

Shifts and Multiplication

Explain the Shifts and Multiplications Involved in This Process

Main Ideas:

1. **Operations Overview:**
    - The process involves two main operations: shifting and multiplying by powers of 2.
    - For even numbers that are 1MOD(3), shifts are used instead of direct multiplication. We only multiply and shift; we apply the formula $(N*0)+((N-1) / 3)$ for all 1MOD(3) numbers.
    - $(N*0)+((N-1) / 3)$ is used to match all odd numbers to an even number
        1. Applying 4 our first 1MOD(3) number in this equation results in 1
        2. 10 is the next and you get 3, 16 = 5, 22 = 7 and so on…
2. **Detailed Explanation:**
    - Shifting helps in navigating the tree and finding the correct paths.
    - Multiplying is used to trace the path back to any number from 1.
    - Applying the formula ensures proper structure and accurate tracing within the sequence.

Base Defined

Main Ideas:

1. **Definition:**
   - The base refers to the fundamental structure established by the base tree (e.g., 4x+).
   - Base numbers are critical for creating and maintaining the tree's pattern.
2. **Significance:**
   - Having a well-defined base ensures that the tree structure remains consistent and predictable.
   - It provides a foundation for mapping out the entire tree and understanding the behavior of the Collatz sequence.

Example of Base Calculation Using the Base Tree 4x+1

4	*	x	+	1	=		(4-1)/3 = 1					
4	*	0	+	1	=	1	1		4	*	0	+	1
4	*	1	+	1	=	5	101		4	*	1	+	1
4	*	5	+	1	=	21	10101		4	*	5	+	1
4	*	21	+	1	=	85	1010101		4	*	21	+	1
4	*	85	+	1	=	341	101010101		4	*	85	+	1
4	*	341	+	1	=	1365	10101010101		4	*	341	+	1
4	*	1365	+	1	=	5461	1010101010101		4	*	1365	+	1
10	*	x	+	3	=		(10-1)/3 = 3					
10	*	0	+	3	=	3	11		8	*	0	+	3
10	*	1	+	3	=	13	1101		4	*	3	+	1
10	*	5	+	3	=	53	110101		4	*	13	+	1
10	*	21	+	3	=	213	11010101		4	*	53	+	1
10	*	85	+	3	=	853	1101010101		4	*	213	+	1
10	*	341	+	3	=	3413	110101010101		4	*	853	+	1
10	*	1365	+	3	=	13653	11010101010101		4	*	3413	+	1
16	*	x	+	5	=		(16-1)/3 = 5					
16	*	0	+	5	=	5	101		16	*	0	+	5
16	*	1	+	5	=	21	10101		4	*	5	+	1
16	*	5	+	5	=	85	1010101		4	*	21	+	1
16	*	21	+	5	=	341	101010101		4	*	85	+	1
16	*	85	+	5	=	1365	10101010101		4	*	341	+	1
16	*	341	+	5	=	5461	1010101010101		4	*	1365	+	1
16	*	1365	+	5	=	21845	101010101010101		4	*	5461	+	1
22	*	x	+	7	=		(22-1)/3 = 7					
22	*	0	+	7	=	7	111		32	*	0	+	7
22	*	1	+	7	=	29	11101		4	*	7	+	1

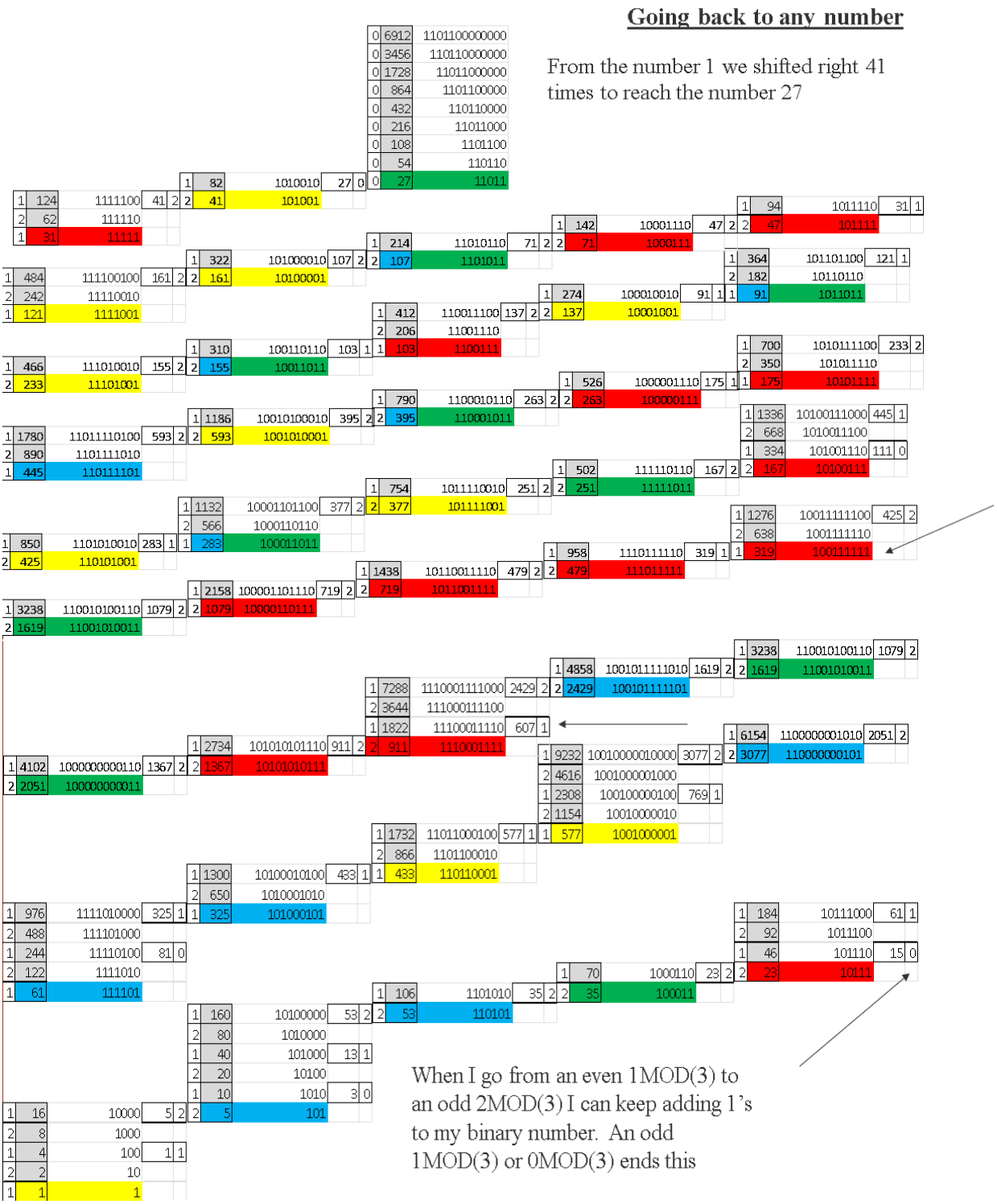
22	*	5	+	7	=	117	1110101		4	*	29	+	1
22	*	21	+	7	=	469	111010101		4	*	117	+	1
22	*	85	+	7	=	1877	11101010101		4	*	469	+	1
22	*	341	+	7	=	7509	1110101010101		4	*	1877	+	1
22	*	1365	+	7	=	30037	111010101010101		4	*	7509	+	1
28	*	x	+	9	=		(28-1)/3 = 9						
28	*	0	+	9	=	9	1001		32	*	0	+	9
28	*	1	+	9	=	37	100101		4	*	9	+	1
28	*	5	+	9	=	149	10010101		4	*	37	+	1
28	*	21	+	9	=	597	1001010101		4	*	149	+	1
28	*	85	+	9	=	2389	100101010101		4	*	597	+	1
28	*	341	+	9	=	9557	10010101010101		4	*	2389	+	1
28	*	1365	+	9	=	38229	1001010101010101		4	*	9557	+	1
34	*	x	+	11	=		(34-1)/3 = 11						
34	*	0	+	11	=	11	1011		32	*	0	+	11
34	*	1	+	11	=	45	101101		4	*	11	+	1
34	*	5	+	11	=	181	10110101		4	*	45	+	1
34	*	21	+	11	=	725	1011010101		4	*	181	+	1
34	*	85	+	11	=	2901	101101010101		4	*	725	+	1
34	*	341	+	11	=	11605	10110101010101		4	*	2901	+	1
34	*	1365	+	11	=	46421	1011010101010101		4	*	11605	+	1
40	*	x	+	13	=		(40-1)/3 = 13						
40	*	0	+	13	=	13	1101		32	*	0	+	13
40	*	1	+	13	=	53	110101		4	*	13	+	1
40	*	5	+	13	=	213	11010101		4	*	53	+	1
40	*	21	+	13	=	853	1101010101		4	*	213	+	1
40	*	85	+	13	=	3413	110101010101		4	*	853	+	1
40	*	341	+	13	=	13653	11010101010101		4	*	3413	+	1
40	*	1365	+	13	=	54613	1101010101010101		4	*	13653	+	1

Explanation:

- The table provided shows the calculation for the base tree 4x+1.
- Starting with $4*0+1=1$, each subsequent calculation follows the same pattern:
  - $4 * 1+1=5$
  - $4 * 5+1=21$
  - $4 * 21+1=85$
  - $4 * 85+1=341$
  - $4 * 341+1=1365$
  - $4 * 1365+1=5461$

- This pattern demonstrates how each result becomes the next value to be multiplied by 4 and added to 1, setting the standard for all other number sets.
- It is also the pattern that all other base numbers must follow.

Going Back from 1 to 27



Shifts and Divide

Num	Steps	Shifts	Div	Base	Base Odd
1	0	0	0	NO	4
2	1	0	1	NO	0
3	7	2	5	NO	10
4	2	0	2	YES	1
5	5	1	4	NO	16
6	8	2	6	NO	0
7	16	5	11	NO	22
8	3	0	3	NO	0
9	19	6	13	NO	28
10	6	1	5	YES	3
11	14	4	10	NO	34
12	9	2	7	NO	0
13	9	2	7	NO	40
14	17	5	12	NO	0
15	17	5	12	NO	46
16	4	0	4	NO	5
17	12	3	9	NO	52
18	20	6	14	NO	0
19	20	6	14	NO	58
20	7	1	6	NO	0
21	7	1	6	NO	64
22	15	4	11	YES	7
23	15	4	11	NO	70
24	10	2	8	NO	0
25	23	7	16	NO	76
26	10	2	8	NO	0
27	111	41	70	NO	82
28	18	5	13	YES	9
29	18	5	13	NO	88
30	18	5	13	NO	0
31	106	39	67	NO	94
32	5	0	5	NO	0
33	26	8	18	NO	100
34	13	3	10	YES	11

.

All Numbers Fall Below Themselves

Precision Out to 2^750 Numbers

We are going to talk about number way beyond the existing known Collatz numbers of about 2^71 range.

Highest Number Verified

95,560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,17395,560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,17395,560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,173

All numbers fall below themselves with these calculations

MOD Level	Initial Remainer	Steps to 1MOD(4)	Total Steps
8	1, 5	0	3
8	3	2	5
16	7	4	7
32	15	6	9
64	31	8	11
128	63	10	13
256	127	12	15
512	255	14	17
1,024	511	16	19
2,048	1,023	18	21
4,096	2,047	20	23
8,192	4,095	22	25
16,384	8,191	24	27
32,768	16,383	26	29

MOD Level	Initial Remainer	Steps to 1MOD(4)	Total Steps
65,536	32,767	28	31
131,072	65,535	30	33
262,144	131,071	32	35
524,288	262,143	34	37
1,048,576	524,287	36	39
2,097,152	1,048,575	38	41
4,194,304	2,097,151	40	43
8,388,608	4,194,303	42	45
16,777,216	8,388,607	44	47
33,554,432	16,777,215	46	49
67,108,864	33,554,431	48	51
134,217,728	67,108,863	50	53
268,435,456	134,217,727	52	55
536,870,912	268,435,455	54	57
1,073,741,824	536,870,911	56	59
2,147,483,648	1,073,741,823	58	61
4,294,967,296	2,147,483,647	60	63
8,589,934,592	4,294,967,295	62	65

MOD Level	Initial Remainer	Steps to 1MOD(4)	Total Steps
17,179,869,184	8,589,934,591	64	67
34,359,738,368	17,179,869,183	66	69
68,719,476,736	34,359,738,367	68	71
137,438,953,472	68,719,476,735	70	73
274,877,906,944	137,438,953,471	72	75
549,755,813,888	274,877,906,943	74	77
1,099,511,627,776	549,755,813,887	76	79
2,199,023,255,552	1,099,511,627,775	78	81
4,398,046,511,104	2,199,023,255,551	80	83
8,796,093,022,208	4,398,046,511,103	82	85
17,592,186,044,416	8,796,093,022,207	84	87
35,184,372,088,832	17,592,186,044,415	86	89
70,368,744,177,664	35,184,372,088,831	88	91
140,737,488,355,328	70,368,744,177,663	90	93
281,474,976,710,656	140,737,488,355,327	92	95
562,949,953,421,312	281,474,976,710,655	94	97
1,125,899,906,842,624	562,949,953,421,311	96	99
2,251,799,813,685,248	1,125,899,906,842,623	98	101
4,503,599,627,370,496	2,251,799,813,685,247	100	103

What this table shows

- **Power Slots**
  - Every number has a power slot that can be found with checking the number with MOD's
- **What the remainder for each power slot says**
  - All remainders are (2^x) – 1
- **Steps to first 1MOD(4)**
  - Knowing the Power Slot and the remainder, I can calculate how many steps to the next 1MOD(4) that is 3 steps from below itself
- **Total Step to below itself**
  - This means that at this step the starting number and the number 3 steps prior are below themselves.
    - NOTE: the original number will not be below itself, but it will have crossed the path of a number that will.
    - The 1MOD(4) will be of greater value and go below itself in 3 steps, meaning all numbers of lesser value go below themselves.

Conclusion of the Collatz Conjecture Proof

Understanding the Unique Behavior of the Number 1 in the Collatz Conjecture

1. Dividing Powers of 2:

- When we divide powers of 2, the result systematically decreases:
  - 2^4 / 2 = 2^3
  - 2^3 / 2 = 2^2
  - 2^2 / 2 = 2^1
  - 2^1 / 2 = 2^0
  - 2^0 / 2 is not applicable (N/A)

2. Applying 3x+1/2 Transformation:

- When we apply the 3x+1/2 operation to numbers, they reduce step by step:
  - 15 (binary 01111) becomes 7 (binary 0111)
  - 7 (binary 0111) becomes 3 (binary 011)
  - 3 (binary 011) becomes 1 (binary 01)
  - 1 (binary 001) becomes 4 (binary 0100), then to 2 (binary 010)
  - We are essentially dividing the remainder of the number by 1, highlighting the role of the first 0 bit (power slot).

3. Unique Role of Number 1:

- **Number 1 and the Divide by Zero Concept**:
  - The number 1 fails both operations in the Collatz Conjecture:
    - It cannot be divided in half or split into two parts.

        - Attempting to apply the Collatz operations to the number 1 leads to a unique loop.
    - **The Special Reset Mechanism**:
        - When we attempt to divide the number 1 by 2 within the context of the sequence:
            - Applying $3\times1+1=4$ results in an even number.
            - Dividing 4 by 2 twice brings us back to 1, effectively "resetting" the sequence.
        - This loop occurs because the number 1 cannot be reduced further within the rules of the Collatz Conjecture, similar to how dividing by zero is undefined in mathematics.
    - **1MOD(8) Special Case**:
        - The number 1 is the only 1MOD(8) number that connects back to itself in the loop.
        - Other numbers that fit this pattern include 5MOD(8):
            - 5, 21, 85, 341, 1365, 5461, 21845, 87381
        - No other numbers form a loop because they either:
            - Have a remainder.
            - Can move a remainder to the power side of the number.
            - Can be divided by 2.
        - The exception is the number 1.

4. Understanding the Power Slot:

- **Role of the Power Slot in the Collatz Conjecture**:
    - Knowing the power slot of all numbers allows us to predict when a number will encounter a 1MOD(4) number.
    - From there, it is three steps to fall below itself in the sequence.
    - This means we can calculate and predict that all numbers will eventually find a 1MOD(4) number that falls below themselves.
    - Consequently, this ensures that all numbers eventually fall below themselves in the Collatz sequence.

Conclusion:

- The unique behavior of the number 1 in the Collatz Conjecture, explained through the concept of dividing by zero, highlights why it forms a loop.
- The understanding of power slots and their roles in the Collatz sequence provides a predictive mechanism for the behavior of all numbers.
- This ensures that all numbers will eventually fall below themselves in the Collatz sequence, proving the conjecture.

More work completed in the process of working the solution

The numbers 1023 and 27 are ran through step by step showing the following:

1. Steps
2. Number
3. MOD(8)
4. Binary of the number
5. Power Slot
    a. Correct Power
    b. Power
    c. Remainder
6. Power Slot 2
    a. Correct Power
    b. What is happening to the power
    c. Remainder

The interesting thing to see is the remainder being divided in half every 2 steps as we move a remainder to the power side.

- The remainder counts down to 1 and then does the loop reset and then counts down to 1 and keeps repeating this until it reaches the number 1.
- 7mod(8) are the only odd numbers that repeat and they repeat until they are left with a remainder of 3 which turns them into 3MOD(8) which in 2 steps turns them into the 1MOD(4).
- The loop reset happens with every 1MOD(4) number or in the examples for every 1MOD(8) and 5MOD(8)

Given that, you can see the power side incrementing up at 3x+1 until it reaches the 1MOD(4) and then does a loop reset.

Steps	Number	MOD 8	131072	65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	Correct Power	*	Powers	+	Remainder	Correct Power	*	Powers	+	Remainder
	1023	7	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2048	*	0	+	1023	2048	*	x	+	1023
1	3070	6	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	0	2048	*	1	+	1022	2048	*	((3*x)+1)	+	1022
2	1535	7	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1024	*	1	+	511	1024	*	x	+	511
3	4606	6	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1024	*	4	+	510	1024	*	((3*x)+1)	+	510
4	2303	7	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	512	*	4	+	255	512	*	x	+	255
5	6910	6	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	0	512	*	13	+	254	512	*	((3*x)+1)	+	254
6	3455	7	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	256	*	13	+	127	256	*	x	+	127
7	10366	6	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	1	0	256	*	40	+	126	256	*	((3*x)+1)	+	126
8	5183	7	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	1	128	*	40	+	63	128	*	x	+	63
9	15550	6	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1	1	0	128	*	121	+	62	128	*	((3*x)+1)	+	62
10	7775	7	0	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1	1	64	*	121	+	31	64	*	x	+	31
11	23326	6	0	0	1	0	1	1	0	1	1	0	0	0	1	1	1	1	0	64	*	364	+	30	64	*	((3*x)+1)	+	30
12	11663	7	0	0	0	1	0	1	1	0	1	1	0	0	0	1	1	1	1	32	*	364	+	15	32	*	x	+	15
13	34990	6	0	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	0	32	*	1093	+	14	32	*	((3*x)+1)	+	14
14	17495	7	0	0	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	16	*	1093	+	7	16	*	x	+	7
15	52486	6	0	1	1	0	0	1	1	0	1	0	0	0	0	0	1	1	0	16	*	3280	+	6	16	*	((3*x)+1)	+	6
16	26243	3	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	1	1	8	*	3280	+	3	8	*	x	+	3
17	78730	2	1	0	0	1	1	0	0	1	1	1	0	0	0	1	0	1	0	8	*	9841	+	2	8	*	((3*x)+1)	+	2
18	39365	5	0	1	0	0	1	1	0	0	1	1	1	0	0	0	1	0	1	4	*	9841	+	1	4	*	x	+	1
19	118096	0	1	1	1	0	0	1	1	0	1	0	1	0	1	0	0	0	0	64	*	1845	+	16	64	*	R	+	16
20	59048	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	0	0	0	32	*	1845	+	8	32	*	x	+	8
21	29524	4	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	0	0	16	*	1845	+	4	16	*	x	+	4
22	14762	2	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	0	8	*	1845	+	2	8	*	x	+	2
23	7381	5	0	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	4	*	1845	+	1	4	*	x	+	1
24	22144	0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	512	*	43	+	128	512	*	R	+	128
25	11072	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	256	*	43	+	64	256	*	x	+	64
26	5536	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	0	128	*	43	+	32	128	*	x	+	32
27	2768	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	64	*	43	+	16	64	*	x	+	16
28	1384	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	32	*	43	+	8	32	*	x	+	8
29	692	4	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	16	*	43	+	4	16	*	x	+	4
30	346	2	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	8	*	43	+	2	8	*	x	+	2
31	173	5	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	4	*	43	+	1	4	*	x	+	1
32	520	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	32	*	16	+	8	32	*	R	+	8
33	260	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	16	*	16	+	4	16	*	x	+	4
34	130	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	8	*	16	+	2	8	*	x	+	2
35	65	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	4	*	16	+	1	4	*	x	+	1
36	196	4	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	16	*	12	+	4	16	*	R	+	4
37	98	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	8	*	12	+	2	8	*	x	+	2
38	49	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	4	*	12	+	1	4	*	x	+	1
39	148	4	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	16	*	9	+	4	16	*	R	+	4
40	74	2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	8	*	9	+	2	8	*	x	+	2
41	37	5	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	4	*	9	+	1	4	*	x	+	1
42	112	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	256	*	0	+	112	256	*	R	+	112
43	56	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	128	*	0	+	56	128	*	x	+	56
44	28	4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	64	*	0	+	28	64	*	x	+	28
45	14	6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	32	*	0	+	14	32	*	x	+	14
46	7	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	16	*	0	+	7	16	*	x	+	7
47	22	6	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	16	*	1	+	6	16	*	((3*x)+1)	+	6
48	11	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	8	*	1	+	3	8	*	x	+	3
49	34	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	8	*	4	+	2	8	*	((3*x)+1)	+	2
50	17	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	4	*	4	+	1	4	*	x	+	1
51	52	4	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	16	*	3	+	4	16	*	R	+	4
52	26	2	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	8	*	3	+	2	8	*	x	+	2
53	13	5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	4	*	3	+	1	4	*	x	+	1
54	40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	32	*	1	+	8	32	*	R	+	8
55	20	4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	16	*	1	+	4	16	*	x	+	4
56	10	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	8	*	1	+	2	8	*	x	+	2
57	5	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	4	*	1	+	1	4	*	x	+	1
58	16	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	64	*	0	+	16	64	*	R	+	16
59	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	32	*	0	+	8	32	*	x	+	8
60	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	16	*	x	+	4
61	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	8	*	x	+	2
62	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	4	*	x	+	1
	4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	16	*	R	+	4
	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	8	*	x	+	2
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	4	*	x	+	1

Steps	Number	MOD 8	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	Correct Power	*	Powers	+	Remainder	Correct Power	*	Powers	+	Remainder
	27	3	0	0	0	0	0	0	0	0	0	1	1	0	1	1	8	*	3	+	3	8	*	x	+	3
1	82	2	0	0	0	0	0	0	0	1	0	1	0	0	1	0	8	*	10	+	2	8	*	((3*x)+1)	+	2
2	41	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	4	*	10	+	1	4	*	x	+	1
3	124	4	0	0	0	0	0	0	0	1	1	1	1	1	0	0	256	*	0	+	124	256	*	R	+	124
4	62	6	0	0	0	0	0	0	0	0	1	1	1	1	1	0	128	*	0	+	62	128	*	x	+	62
5	31	7	0	0	0	0	0	0	0	0	0	1	1	1	1	1	64	*	0	+	31	64	*	x	+	31
6	94	6	0	0	0	0	0	0	0	1	0	1	1	1	1	0	64	*	1	+	30	64	*	((3*x)+1)	+	30
7	47	7	0	0	0	0	0	0	0	0	1	0	1	1	1	1	32	*	1	+	15	32	*	x	+	15
8	142	6	0	0	0	0	0	0	1	0	0	0	1	1	1	0	32	*	4	+	14	32	*	((3*x)+1)	+	14
9	71	7	0	0	0	0	0	0	0	1	0	0	0	1	1	1	16	*	4	+	7	16	*	x	+	7
10	214	6	0	0	0	0	0	0	1	1	0	1	0	1	1	0	16	*	13	+	6	16	*	((3*x)+1)	+	6
11	107	3	0	0	0	0	0	0	0	1	1	0	1	0	1	1	8	*	13	+	3	8	*	x	+	3
12	322	2	0	0	0	0	0	1	0	1	0	0	0	0	1	0	8	*	40	+	2	8	*	((3*x)+1)	+	2
13	161	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	4	*	40	+	1	4	*	x	+	1
14	484	4	0	0	0	0	0	1	1	1	1	0	0	1	0	0	16	*	30	+	4	16	*	R	+	4
15	242	2	0	0	0	0	0	0	1	1	1	1	0	0	1	0	8	*	30	+	2	8	*	x	+	2
16	121	1	0	0	0	0	0	0	0	1	1	1	1	0	0	1	4	*	30	+	1	4	*	x	+	1
17	364	4	0	0	0	0	0	1	0	1	1	0	1	1	0	0	32	*	11	+	12	32	*	R	+	12
18	182	6	0	0	0	0	0	0	1	0	1	1	0	1	1	0	16	*	11	+	6	16	*	x	+	6
19	91	3	0	0	0	0	0	0	0	1	0	1	1	0	1	1	8	*	11	+	3	8	*	x	+	3
20	274	2	0	0	0	0	0	1	0	0	0	1	0	0	1	0	8	*	34	+	2	8	*	((3*x)+1)	+	2
21	137	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	4	*	34	+	1	4	*	x	+	1
22	412	4	0	0	0	0	0	1	1	0	0	1	1	1	0	0	64	*	6	+	28	64	*	R	+	28
23	206	6	0	0	0	0	0	0	1	1	0	0	1	1	1	0	32	*	6	+	14	32	*	((3*x)+1)	+	14
24	103	7	0	0	0	0	0	0	0	1	1	0	0	1	1	1	16	*	6	+	7	16	*	x	+	7
25	310	6	0	0	0	0	0	1	0	0	1	1	0	1	1	0	16	*	19	+	6	16	*	((3*x)+1)	+	6
26	155	3	0	0	0	0	0	0	1	0	0	1	1	0	1	1	8	*	19	+	3	8	*	x	+	3
27	466	2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	8	*	58	+	2	8	*	((3*x)+1)	+	2
28	233	1	0	0	0	0	0	0	1	1	1	0	1	0	0	1	4	*	58	+	1	4	*	x	+	1
29	700	4	0	0	0	0	1	0	1	0	1	1	1	1	0	0	128	*	5	+	60	128	*	R	+	60
30	350	6	0	0	0	0	0	1	0	1	0	1	1	1	1	0	64	*	5	+	30	64	*	x	+	30
31	175	7	0	0	0	0	0	0	1	0	1	0	1	1	1	1	32	*	5	+	15	32	*	x	+	15
32	526	6	0	0	0	0	1	0	0	0	0	0	1	1	1	0	32	*	16	+	14	32	*	((3*x)+1)	+	14
33	263	7	0	0	0	0	0	1	0	0	0	0	0	1	1	1	16	*	16	+	7	16	*	x	+	7
34	790	6	0	0	0	0	1	1	0	0	0	1	0	1	1	0	16	*	49	+	6	16	*	((3*x)+1)	+	6
35	395	3	0	0	0	0	0	1	1	0	0	0	1	0	1	1	8	*	49	+	3	8	*	x	+	3
36	1,186	2	0	0	0	1	0	0	1	0	1	0	0	0	1	0	8	*	148	+	2	8	*	((3*x)+1)	+	2
37	593	1	0	0	0	0	1	0	0	1	0	1	0	0	0	1	4	*	148	+	1	4	*	x	+	1
38	1,780	4	0	0	0	1	1	0	1	1	1	1	0	1	0	0	16	*	111	+	4	16	*	R	+	4
39	890	2	0	0	0	0	1	1	0	1	1	1	1	0	1	0	8	*	111	+	2	8	*	x	+	2
40	445	5	0	0	0	0	0	1	1	0	1	1	1	1	0	1	4	*	111	+	1	4	*	x	+	1
41	1,336	0	0	0	0	1	0	1	0	0	1	1	1	0	0	0	128	*	10	+	56	128	*	R	+	56
42	668	4	0	0	0	0	1	0	1	0	0	1	1	1	0	0	64	*	10	+	28	64	*	x	+	28
43	334	6	0	0	0	0	0	1	0	1	0	0	1	1	1	0	32	*	10	+	14	32	*	x	+	14
44	167	7	0	0	0	0	0	0	1	0	1	0	0	1	1	1	16	*	10	+	7	16	*	x	+	7
45	502	6	0	0	0	0	0	1	1	1	1	1	0	1	1	0	16	*	31	+	6	16	*	((3*x)+1)	+	6
46	251	3	0	0	0	0	0	0	1	1	1	1	1	0	1	1	8	*	31	+	3	8	*	x	+	3
47	754	2	0	0	0	0	1	0	1	1	1	1	0	0	1	0	8	*	94	+	2	8	*	((3*x)+1)	+	2
48	377	1	0	0	0	0	0	1	0	1	1	1	1	0	0	1	4	*	94	+	1	4	*	x	+	1
49	1,132	4	0	0	0	1	0	0	0	1	1	0	1	1	0	0	32	*	35	+	12	32	*	R	+	12
50	566	6	0	0	0	0	1	0	0	0	1	1	0	1	1	0	16	*	35	+	6	16	*	x	+	6
51	283	3	0	0	0	0	0	1	0	0	0	1	1	0	1	1	8	*	35	+	3	8	*	x	+	3
52	850	2	0	0	0	0	1	1	0	1	0	1	0	0	1	0	8	*	106	+	2	8	*	((3*x)+1)	+	2
53	425	1	0	0	0	0	0	1	1	0	1	0	1	0	0	1	4	*	106	+	1	4	*	x	+	1
54	1,276	4	0	0	0	1	0	0	1	1	1	1	1	1	0	0	512	*	2	+	252	512	*	R	+	252
55	638	6	0	0	0	0	1	0	0	1	1	1	1	1	1	0	256	*	2	+	126	256	*	x	+	126
56	319	7	0	0	0	0	0	1	0	0	1	1	1	1	1	1	128	*	2	+	63	128	*	x	+	63

57	958	6	0	0	0	0	1	1	1	0	1	1	1	1	1	0	128	*	7	+	62	128	*	((3*x)+1)	+	62
58	479	7	0	0	0	0	0	1	1	1	0	1	1	1	1	1	64	*	7	+	31	64	*	x	+	31
59	1,438	6	0	0	0	1	0	1	1	0	0	1	1	1	1	0	64	*	22	+	30	64	*	((3*x)+1)	+	30
60	719	7	0	0	0	0	1	0	1	1	0	0	1	1	1	1	32	*	22	+	15	32	*	x	+	15
61	2,158	6	0	0	1	0	0	0	0	1	1	0	1	1	1	0	32	*	67	+	14	32	*	((3*x)+1)	+	14
62	1,079	7	0	0	0	1	0	0	0	0	1	1	0	1	1	1	16	*	67	+	7	16	*	x	+	7
63	3,238	6	0	0	1	1	0	0	1	0	1	0	0	1	1	0	16	*	202	+	6	16	*	((3*x)+1)	+	6
64	1,619	3	0	0	0	1	1	0	0	1	0	1	0	0	1	1	8	*	202	+	3	8	*	x	+	3
65	4,858	2	0	1	0	0	1	0	1	1	1	1	1	0	1	0	8	*	607	+	2	8	*	((3*x)+1)	+	2
66	2,429	5	0	0	1	0	0	1	0	1	1	1	1	1	0	1	4	*	607	+	1	4	*	x	+	1
67	7,288	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	256	*	28	+	120	256	*	R	+	120
68	3,644	4	0	0	1	1	1	0	0	0	1	1	1	1	0	0	128	*	28	+	60	128	*	x	+	60
69	1,822	6	0	0	0	1	1	1	0	0	0	1	1	1	1	0	64	*	28	+	30	64	*	x	+	30
70	911	7	0	0	0	0	1	1	1	0	0	0	1	1	1	1	32	*	28	+	15	32	*	x	+	15
71	2,734	6	0	0	1	0	1	0	1	0	1	0	1	1	1	0	32	*	85	+	14	32	*	((3*x)+1)	+	14
72	1,367	7	0	0	0	1	0	1	0	1	0	1	0	1	1	1	16	*	85	+	7	16	*	x	+	7
73	4,102	6	0	1	0	0	0	0	0	0	0	0	0	1	1	0	16	*	256	+	6	16	*	((3*x)+1)	+	6
74	2,051	3	0	0	1	0	0	0	0	0	0	0	0	0	1	1	8	*	256	+	3	8	*	x	+	3
75	6,154	2	0	1	1	0	0	0	0	0	0	0	1	0	1	0	8	*	769	+	2	8	*	((3*x)+1)	+	2
76	3,077	5	0	0	1	1	0	0	0	0	0	0	0	1	0	1	4	*	769	+	1	4	*	x	+	1
77	9,232	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	64	*	144	+	16	64	*	R	+	16
78	4,616	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	32	*	144	+	8	32	*	x	+	8
79	2,308	4	0	0	1	0	0	1	0	0	0	0	0	1	0	0	16	*	144	+	4	16	*	x	+	4
80	1,154	2	0	0	0	1	0	0	1	0	0	0	0	0	1	0	8	*	144	+	2	8	*	x	+	2
81	577	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	4	*	144	+	1	4	*	x	+	1
82	1,732	4	0	0	0	1	1	0	1	1	0	0	0	1	0	0	16	*	108	+	4	16	*	R	+	4
83	866	2	0	0	0	0	1	1	0	1	1	0	0	0	1	0	8	*	108	+	2	8	*	x	+	2
84	433	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	4	*	108	+	1	4	*	x	+	1
85	1,300	4	0	0	0	1	0	1	0	0	0	1	0	1	0	0	16	*	81	+	4	16	*	R	+	4
86	650	2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	8	*	81	+	2	8	*	x	+	2
87	325	5	0	0	0	0	0	1	0	1	0	0	0	1	0	1	4	*	81	+	1	4	*	x	+	1
88	976	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	64	*	15	+	16	64	*	R	+	16
89	488	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	32	*	15	+	8	32	*	x	+	8
90	244	4	0	0	0	0	0	0	1	1	1	1	0	1	0	0	16	*	15	+	4	16	*	x	+	4
91	122	2	0	0	0	0	0	0	0	1	1	1	1	0	1	0	8	*	15	+	2	8	*	x	+	2
92	61	5	0	0	0	0	0	0	0	0	1	1	1	1	0	1	4	*	15	+	1	4	*	x	+	1
93	184	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	128	*	1	+	56	128	*	R	+	56
94	92	4	0	0	0	0	0	0	0	1	0	1	1	1	0	0	64	*	1	+	28	64	*	x	+	28
95	46	6	0	0	0	0	0	0	0	0	1	0	1	1	1	0	32	*	1	+	14	32	*	x	+	14
96	23	7	0	0	0	0	0	0	0	0	0	1	0	1	1	1	16	*	1	+	7	16	*	x	+	7
97	70	6	0	0	0	0	0	0	0	1	0	0	0	1	1	0	16	*	4	+	6	16	*	((3*x)+1)	+	6
98	35	3	0	0	0	0	0	0	0	0	1	0	0	0	1	1	8	*	4	+	3	8	*	x	+	3
99	106	2	0	0	0	0	0	0	0	1	1	0	1	0	1	0	8	*	13	+	2	8	*	((3*x)+1)	+	2
100	53	5	0	0	0	0	0	0	0	0	1	1	0	1	0	1	4	*	13	+	1	4	*	x	+	1
101	160	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	128	*	1	+	32	128	*	R	+	32
102	80	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	64	*	1	+	16	64	*	x	+	16
103	40	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	32	*	1	+	8	32	*	x	+	8
104	20	4	0	0	0	0	0	0	0	0	0	1	0	1	0	0	16	*	1	+	4	16	*	x	+	4
105	10	2	0	0	0	0	0	0	0	0	0	0	1	0	1	0	8	*	1	+	2	8	*	x	+	2
106	5	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	4	*	1	+	1	4	*	x	+	1
107	16	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	64	*	0	+	16	64	*	R	+	16
108	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	32	*	0	+	8	32	*	x	+	8
109	4	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	16	*	x	+	4
110	2	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	8	*	x	+	2
111	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	4	*	x	+	1
	4	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	16	*	R	+	4
	2	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	8	*	x	+	2
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	4	*	x	+	1

RStudio Code

This code will start with the number you input and go to 1 but skip all even numbers and can skip middle numbers if there are more than 2 in a sequence.

Start R Code

```
==========================================================================
library(gmp)
library(Rmpfr)
#######################
#Skips all even numbers and middle odd numbers
#Works up to 2^750 numbers
#######################

collatz_conjecture <- function() {
  # Read and clean input to allow commas
  input <- readline(prompt = "Enter a number: ")
  input <- gsub(",", "", input)
  x <- as.bigz(input)

  # Initialize variables
  xx <- x    # Starting Number
  ST <- 0    # Total Steps
  SR <- 0    # Count of Shifts Right

  print(paste("Initial value:", as.character(x)))

  # Function to calculate next step for even number
  next_step_even <- function(x) {
    N <- x
    NF <- x / 2
    x <- x / 2
    if (as.integer(mod.bigz(x, 2)) == 0) {
      PO <- x / 2
    } else {
      PO <- (x - 1) / 2
    }
    print(paste("Even step:", as.character(N), "End:", as.character(NF), "Pos
ition:", as.character(PO)))
    return(x)
  }

  # Function to calculate next step for odd number
  next_step_odd <- function(x) {
    SR <<- SR + 1
    N <- x
    FN <- x
    PP <- (x - 1) / 2
    PP8 <- x
    MOD8 <- as.bigz(0)

    if (as.integer(mod.bigz(PP8, 8)) == 3) {
      # MOD 3
      NF <- FN + ((FN + 1) / 2)
      MOD8 <- 3
    } else if (as.integer(mod.bigz(PP8, 8)) == 7) {
      # MOD 7
      CP <- PP
      while (as.integer(mod.bigz(CP, 4)) == 3) {
        FN <- FN + ((FN + 1) / 2)
        CP <- (FN - 1) / 2
        SR <<- SR + 1
```

```
      }
      NF <- FN + ((FN + 1) / 2)
      MOD8 <- 7
    } else if (as.integer(mod.bigz(PP8, 8)) == 1) {
      # MOD 1
      NP <- PP - (PP / 4)
      NF <- (NP * 2) + 1
      MOD8 <- 1
    } else if (as.integer(mod.bigz(PP8, 8)) == 5) {
      # MOD 5
      TN <- N - 1
      MD <- 4
      if (N == 5) {
        NF <- ((N * 3) + 1) / 16
      } else {
        while (as.integer(mod.bigz(TN, MD)) == 0) {
          TN <- TN - MD
          MD <- (MD * 2) * 2
        }
        MDD <- MD / 2
        MO <- TN
        if (as.integer(mod.bigz(MO, MDD)) == 0) {
          DM <- mod.bigz(TN, MD)
          NF <- ((N * 3) + 1) / DM
        } else {
          DM <- mod.bigz(TN, MDD)
          NF <- ((N * 3) + 1) / DM
        }
      }
      MOD8 <- 5
    }

    x <- NF
    if (as.integer(mod.bigz(x, 2)) == 0) {
      PE <- x / 2
    } else {
      PE <- (x - 1) / 2
    }
    print(paste("Odd step:", as.character(N), "End:", as.character(NF), "NMOD
:", as.character(MOD8)))
    return(x)
  }

  # Main loop
  while (x != 1) {
    # print(paste("Current value:", as.character(x)))
    if (as.integer(mod.bigz(x, 2)) == 0) {
      x <- next_step_even(x)
    } else {
      x <- next_step_odd(x)
      ST <- ST + 1
    }
  }

  # Print results
  print(paste("Start:", as.character(xx)))
  print(paste("Steps:", as.character(ST)))
}

# Call the function
collatz_conjecture()
```

==========

End R Code

Results of Code

```
Enter a number: 3245246568679867856745645645645645
[1] "Initial value: 3245246568679867856745645645645645"
[1] "Odd step: 3245246568679867856745645645645645 End: 1216967463254950446279
617117117117 NMOD: 5"
[1] "Odd step: 121696746325495044627961711711711 7 End: 4563627987206064173548
56418918919 NMOD: 5"
[1] "Odd step: 456362798720606417354856418918919 End: 10268162971213644390484
26942567569 NMOD: 7"
[1] "Odd step: 1026816297121364439048426942567569 End: 7701122228410233292863
20206925677 NMOD: 1"
[1] "Odd step: 770112222841023329286320206925677 End: 28879208356538374848237
0077597129 NMOD: 5"
[1] "Odd step: 2887920835653837484823700775971 29 End: 21659406267403781136177
7558197847 NMOD: 1"
[1] "Odd step: 2165940626740378113617775581978 47 End: 48733664101658507556399
9505945157 NMOD: 7"
[1] "Odd step: 4873366410165850755639995059451 57 End: 91375620190609701668249
907364717 NMOD: 5"
[1] "Odd step: 9137562019060970166824990736471 7 End: 34265857571478638125593715
15261769 NMOD: 5"
[1] "Odd step: 3426585757147863812559371526176 9 End: 25699393178608978594195
86446327 NMOD: 1"
[1] "Odd step: 2569939317860897859419528644632 7 End: 57823634651870201836939
94504237 NMOD: 7"
[1] "Odd step: 5782363465187020183693939450423 7 End: 21683862994451325688852
72939089 NMOD: 5"
[1] "Odd step: 2168386299445132568885227293908 9 End: 16262897245838494266639
04704317 NMOD: 1"
[1] "Odd step: 1626289724583849426663920470431 7 End: 60985864671894353499897
1764119 NMOD: 5"
[1] "Odd step: 6098586467189435349989701764119 End: 13721819551176229537476828
8969269 NMOD: 7"
[1] "Odd step: 1372181955117622953747682896926 9 End: 12864205829227715191384
2715869 NMOD: 5"
[1] "Odd step: 1286420582922771519138452715869 End: 48240771859603931967691976
68451 NMOD: 5"
[1] "Odd step: 4824077185960393196769197684 51 End: 72361157789405897951537965
2677 NMOD: 3"
[1] "Odd step: 7236115778940589795153796526 77 End: 13567717085513605865913368
4877 NMOD: 5"
[1] "Odd step: 1356771708551360586591336848 77 End: 50878939070676021997175131
829 NMOD: 5"
[1] "Odd step: 5087893907067602199717513182 9 End: 476990053787587706223516860
9 NMOD: 5"
[1] "Odd step: 4769900537875877062235168609 End: 35774254034069077966763764 57
NMOD: 1"
[1] "Odd step: 3577425403406907796676376457 End: 26830690525551808475072823 43
NMOD: 1"
[1] "Odd step: 2683069052555180847507282343 End: 60369053682491569068913852 73
NMOD: 7"
[1] "Odd step: 6036905368249156906891385273 End: 45276790261868676801685389 55
NMOD: 1"
[1] "Odd step: 4527679026186867680168538955 End: 67915185392803015202528084 33
NMOD: 3"
[1] "Odd step: 6791518539280301520252808433 End: 50936389044602261401896063 25
NMOD: 1"
[1] "Odd step: 5093638904460226140189606325 End: 47752864729314620064277559 3
NMOD: 5"
[1] "Odd step: 4775286472931462006427755 93 End: 35814648546985965048208169 5 N
MOD: 1"
```

```
[1] "Odd step: 358146485469859650482081695 End: 1813116582691164480565538585
NMOD: 7"
[1] "Odd step: 1813116582691164480565538585 End: 135983743701837336042415393 9
NMOD: 1"
[1] "Odd step: 135983743701837336042415393 9 End: 203975615552756004063623090 9
NMOD: 3"
[1] "Odd step: 203975615552756004063623090 9 End: 764908558322835015238586591
NMOD: 5"
[1] "Odd step: 764908558322835015238586591 End: 387234957650935226464534462 1
NMOD: 7"
[1] "Odd step: 387234957650935226464534462 1 End: 145213109119100709924200423 3
NMOD: 5"
[1] "Odd step: 145213109119100709924200423 3 End: 108909831839325532443150317 5
NMOD: 1"
[1] "Odd step: 108909831839325532443150317 5 End: 245047121638482447997088214 5
NMOD: 7"
[1] "Odd step: 245047121638482447997088214 5 End: 183785341228861835997816160 9
NMOD: 1"
[1] "Odd step: 183785341228861835997816160 9 End: 137839005921646376998362120 7
NMOD: 1"
[1] "Odd step: 137839005921646376998362120 7 End: 310137763323704348246314771 7
NMOD: 7"
[1] "Odd step: 310137763323704348246314771 7 End: 581508306231945652961840197
NMOD: 5"
[1] "Odd step: 581508306231945652961840197 End: 109032807418489809930345037 N
MOD: 5"
[1] "Odd step: 109032807418489809930345037 End: 40887302781933678723879389 NM
OD: 5"
[1] "Odd step: 40887302781933678723879389 End: 15332738543225129521454771 NMO
D: 5"
[1] "Odd step: 15332738543225129521454771 End: 22999107814837694282182157 NMO
D: 3"
[1] "Odd step: 22999107814837694282182157 End: 86246654305641353558183 09 NMOD
: 5"
[1] "Odd step: 86246654305641353558183 09 End: 16171247682307753792159 33 NMOD:
5"
[1] "Odd step: 16171247682307753792159 33 End: 60642178808654076720597 5 NMOD:
5"
[1] "Odd step: 60642178808654076720597 5 End: 13644490231947167262134 45 NMOD:
7"
[1] "Odd step: 13644490231947167262134 45 End: 25583419184900938616502 1 NMOD:
5"
[1] "Odd step: 25583419184900938616502 1 End: 95937821943378519811883 NMOD: 5"
[1] "Odd step: 95937821943378519811883 End: 14390673291506777971782 5 NMOD: 3"
[1] "Odd step: 14390673291506777971782 5 End: 10793004968630083478836 9 NMOD: 1
"
[1] "Odd step: 10793004968630083478836 9 End: 8094753726472562609127 7 NMOD: 1"
[1] "Odd step: 8094753726472562609127 7 End: 30355326474272109784229 NMOD: 5"
[1] "Odd step: 30355326474272109784229 End: 5691623713926020584543 NMOD: 5"
[1] "Odd step: 5691623713926020584543 End: 2881384505175047920925 3 NMOD: 7"
[1] "Odd step: 2881384505175047920925 3 End: 5402595947203214851735 NMOD: 5"
[1] "Odd step: 5402595947203214851735 End: 1215584088120723341640 5 NMOD: 7"
[1] "Odd step: 1215584088120723341640 5 End: 284902520653294533197 NMOD: 5"
[1] "Odd step: 284902520653294533197 End: 106838445244985449949 NMOD: 5"
[1] "Odd step: 106838445244985449949 End: 40064416966869543731 NMOD: 5"
[1] "Odd step: 40064416966869543731 End: 60096625450304315597 NMOD: 3"
[1] "Odd step: 60096625450304315597 End: 22536234543864118349 NMOD: 5"
[1] "Odd step: 22536234543864118349 End: 8451087953949044381 NMOD: 5"
[1] "Odd step: 8451087953949044381 End: 3169157982730891643 NMOD: 5"
[1] "Odd step: 3169157982730891643 End: 4753736974096337465 NMOD: 3"
[1] "Odd step: 4753736974096337465 End: 3565302730572253099 NMOD: 1"
[1] "Odd step: 3565302730572253099 End: 5347954095858379649 NMOD: 3"
[1] "Odd step: 5347954095858379649 End: 4010965571893784737 NMOD: 1"
[1] "Odd step: 4010965571893784737 End: 3008224178920338553 NMOD: 1"
```

```
[1] "Odd step: 3008224178920338553 End: 2256168134190253915 NMOD: 1"
[1] "Odd step: 2256168134190253915 End: 3384252201285380873 NMOD: 3"
[1] "Odd step: 3384252201285380873 End: 2538189150964035655 NMOD: 1"
[1] "Odd step: 2538189150964035655 End: 5710925589669080225 NMOD: 7"
[1] "Odd step: 5710925589669080225 End: 4283194192251810169 NMOD: 1"
[1] "Odd step: 4283194192251810169 End: 3212395644188857627 NMOD: 1"
[1] "Odd step: 3212395644188857627 End: 4818593466283286441 NMOD: 3"
[1] "Odd step: 4818593466283286441 End: 3613945099712464831 NMOD: 1"
[1] "Odd step: 3613945099712464831 End: 27443395600941529817 NMOD: 7"
[1] "Odd step: 27443395600941529817 End: 20582546700706147363 NMOD: 1"
[1] "Odd step: 20582546700706147363 End: 30873820051059221045 NMOD: 3"
[1] "Odd step: 30873820051059221045 End: 2894420629786801973 NMOD: 5"
[1] "Odd step: 2894420629786801973 End: 271351934042512685 NMOD: 5"
[1] "Odd step: 271351934042512685 End: 101756975265942257 NMOD: 5"
[1] "Odd step: 101756975265942257 End: 76317731449456693 NMOD: 1"
[1] "Odd step: 76317731449456693 End: 7154787323386565 NMOD: 5"
[1] "Odd step: 7154787323386565 End: 1341522623134981 NMOD: 5"
[1] "Odd step: 1341522623134981 End: 251535491837809 NMOD: 5"
[1] "Odd step: 251535491837809 End: 188651618878357 NMOD: 1"
[1] "Odd step: 188651618878357 End: 8843044634923 NMOD: 5"
[1] "Odd step: 8843044634923 End: 13264566952385 NMOD: 3"
[1] "Odd step: 13264566952385 End: 9948425214289 NMOD: 1"
[1] "Odd step: 9948425214289 End: 7461318910717 NMOD: 1"
[1] "Odd step: 7461318910717 End: 2797994591519 NMOD: 5"
[1] "Odd step: 2797994591519 End: 14164847619569 NMOD: 7"
[1] "Odd step: 14164847619569 End: 10623635714677 NMOD: 1"
[1] "Odd step: 10623635714677 End: 995965848251 NMOD: 5"
[1] "Odd step: 995965848251 End: 1493948772377 NMOD: 3"
[1] "Odd step: 1493948772377 End: 1120461579283 NMOD: 1"
[1] "Odd step: 1120461579283 End: 1680692368925 NMOD: 3"
[1] "Odd step: 1680692368925 End: 630259638347 NMOD: 5"
[1] "Odd step: 630259638347 End: 945389457521 NMOD: 3"
[1] "Odd step: 945389457521 End: 709042093141 NMOD: 1"
[1] "Odd step: 709042093141 End: 8309087029 NMOD: 5"
[1] "Odd step: 8309087029 End: 778976909 NMOD: 5"
[1] "Odd step: 778976909 End: 292116341 NMOD: 5"
[1] "Odd step: 292116341 End: 27385907 NMOD: 5"
[1] "Odd step: 27385907 End: 41078861 NMOD: 3"
[1] "Odd step: 41078861 End: 15404573 NMOD: 5"
[1] "Odd step: 15404573 End: 5776715 NMOD: 5"
[1] "Odd step: 5776715 End: 8665073 NMOD: 3"
[1] "Odd step: 8665073 End: 6498805 NMOD: 1"
[1] "Odd step: 6498805 End: 609263 NMOD: 5"
[1] "Odd step: 609263 End: 2056265 NMOD: 7"
[1] "Odd step: 2056265 End: 1542199 NMOD: 1"
[1] "Odd step: 1542199 End: 3469949 NMOD: 7"
[1] "Odd step: 3469949 End: 1301231 NMOD: 5"
[1] "Odd step: 1301231 End: 4391657 NMOD: 7"
[1] "Odd step: 4391657 End: 3293743 NMOD: 1"
[1] "Odd step: 3293743 End: 11116385 NMOD: 7"
[1] "Odd step: 11116385 End: 8337289 NMOD: 1"
[1] "Odd step: 8337289 End: 6252967 NMOD: 1"
[1] "Odd step: 6252967 End: 14069177 NMOD: 7"
[1] "Odd step: 14069177 End: 10551883 NMOD: 1"
[1] "Odd step: 10551883 End: 15827825 NMOD: 3"
[1] "Odd step: 15827825 End: 11870869 NMOD: 1"
[1] "Odd step: 11870869 End: 556447 NMOD: 5"
[1] "Odd step: 556447 End: 2817017 NMOD: 7"
[1] "Odd step: 2817017 End: 2112763 NMOD: 1"
[1] "Odd step: 2112763 End: 3169145 NMOD: 3"
[1] "Odd step: 3169145 End: 2376859 NMOD: 1"
[1] "Odd step: 2376859 End: 3565289 NMOD: 3"
[1] "Odd step: 3565289 End: 2673967 NMOD: 1"
[1] "Odd step: 2673967 End: 9024641 NMOD: 7"
```

```
[1] "Odd step: 9024641 End: 6768481 NMOD: 1"
[1] "Odd step: 6768481 End: 5076361 NMOD: 1"
[1] "Odd step: 5076361 End: 3807271 NMOD: 1"
[1] "Odd step: 3807271 End: 8566361 NMOD: 7"
[1] "Odd step: 8566361 End: 6424771 NMOD: 1"
[1] "Odd step: 6424771 End: 9637157 NMOD: 3"
[1] "Odd step: 9637157 End: 1806967 NMOD: 5"
[1] "Odd step: 1806967 End: 4065677 NMOD: 7"
[1] "Odd step: 4065677 End: 1524629 NMOD: 5"
[1] "Odd step: 1524629 End: 71467 NMOD: 5"
[1] "Odd step: 71467 End: 107201 NMOD: 3"
[1] "Odd step: 107201 End: 80401 NMOD: 1"
[1] "Odd step: 80401 End: 60301 NMOD: 1"
[1] "Odd step: 60301 End: 22613 NMOD: 5"
[1] "Odd step: 22613 End: 265 NMOD: 5"
[1] "Odd step: 265 End: 199 NMOD: 1"
[1] "Odd step: 199 End: 449 NMOD: 7"
[1] "Odd step: 449 End: 337 NMOD: 1"
[1] "Odd step: 337 End: 253 NMOD: 1"
[1] "Odd step: 253 End: 95 NMOD: 5"
[1] "Odd step: 95 End: 485 NMOD: 7"
[1] "Odd step: 485 End: 91 NMOD: 5"
[1] "Odd step: 91 End: 137 NMOD: 3"
[1] "Odd step: 137 End: 103 NMOD: 1"
[1] "Odd step: 103 End: 233 NMOD: 7"
[1] "Odd step: 233 End: 175 NMOD: 1"
[1] "Odd step: 175 End: 593 NMOD: 7"
[1] "Odd step: 593 End: 445 NMOD: 1"
[1] "Odd step: 445 End: 167 NMOD: 5"
[1] "Odd step: 167 End: 377 NMOD: 7"
[1] "Odd step: 377 End: 283 NMOD: 1"
[1] "Odd step: 283 End: 425 NMOD: 3"
[1] "Odd step: 425 End: 319 NMOD: 1"
[1] "Odd step: 319 End: 2429 NMOD: 7"
[1] "Odd step: 2429 End: 911 NMOD: 5"
[1] "Odd step: 911 End: 3077 NMOD: 7"
[1] "Odd step: 3077 End: 577 NMOD: 5"
[1] "Odd step: 577 End: 433 NMOD: 1"
[1] "Odd step: 433 End: 325 NMOD: 1"
[1] "Odd step: 325 End: 61 NMOD: 5"
[1] "Odd step: 61 End: 23 NMOD: 5"
[1] "Odd step: 23 End: 53 NMOD: 7"
[1] "Odd step: 53 End: 5 NMOD: 5"
[1] "Odd step: 5 End: 1 NMOD: 5"
[1] "Start: 3245246568679867856745645645645645"
[1] "Steps: 179"

Enter a number: 27
[1] "Initial value: 27"
[1] "Odd step: 27 End: 41 NMOD: 3"
[1] "Odd step: 41 End: 31 NMOD: 1"
[1] "Odd step: 31 End: 161 NMOD: 7"
[1] "Odd step: 161 End: 121 NMOD: 1"
[1] "Odd step: 121 End: 91 NMOD: 1"
[1] "Odd step: 91 End: 137 NMOD: 3"
[1] "Odd step: 137 End: 103 NMOD: 1"
[1] "Odd step: 103 End: 233 NMOD: 7"
[1] "Odd step: 233 End: 175 NMOD: 1"
[1] "Odd step: 175 End: 593 NMOD: 7"
[1] "Odd step: 593 End: 445 NMOD: 1"
[1] "Odd step: 445 End: 167 NMOD: 5"
[1] "Odd step: 167 End: 377 NMOD: 7"
[1] "Odd step: 377 End: 283 NMOD: 1"
[1] "Odd step: 283 End: 425 NMOD: 3"
```

```
[1] "Odd step: 425 End: 319 NMOD: 1"
[1] "Odd step: 319 End: 2429 NMOD: 7"
[1] "Odd step: 2429 End: 911 NMOD: 5"
[1] "Odd step: 911 End: 3077 NMOD: 7"
[1] "Odd step: 3077 End: 577 NMOD: 5"
[1] "Odd step: 577 End: 433 NMOD: 1"
[1] "Odd step: 433 End: 325 NMOD: 1"
[1] "Odd step: 325 End: 61 NMOD: 5"
[1] "Odd step: 61 End: 23 NMOD: 5"
[1] "Odd step: 23 End: 53 NMOD: 7"
[1] "Odd step: 53 End: 5 NMOD: 5"
[1] "Odd step: 5 End: 1 NMOD: 5"
[1] "Start: 27"
[1] "Steps: 27"

Enter a number: 1023
[1] "Initial value: 1023"
[1] "Odd step: 1023 End: 39365 NMOD: 7"
[1] "Odd step: 39365 End: 7381 NMOD: 5"
[1] "Odd step: 7381 End: 173 NMOD: 5"
[1] "Odd step: 173 End: 65 NMOD: 5"
[1] "Odd step: 65 End: 49 NMOD: 1"
[1] "Odd step: 49 End: 37 NMOD: 1"
[1] "Odd step: 37 End: 7 NMOD: 5"
[1] "Odd step: 7 End: 17 NMOD: 7"
[1] "Odd step: 17 End: 13 NMOD: 1"
[1] "Odd step: 13 End: 5 NMOD: 5"
[1] "Odd step: 5 End: 1 NMOD: 5"
[1] "Start: 1023"
[1] "Steps: 11"
```

This code calculates the Tree location from 1

Start R Code

==========

```
library(gmp)
library(Rmpfr)

########################
# Calculates the Steps and how many shifts and divides and
# if the number is base and its matching base number
# Added branch and base checks
# Works up to 2^750 numbers
########################

# Function to check if a number is 1MOD3
is_1MOD3 <- function(n) {
  return((as.bigz(n) %% as.bigz(3)) == 1)
}

# Function to check if an even number divided by 4 is a base
is_base_check <- function(n) {
  div4 <- as.bigz(n) / as.bigz(4)
  return((div4 %% as.bigz(2)) == 0 && !is_1MOD3(div4))
```

```
}

# Define the main function for the Collatz calculation
run_collatz <- function() {
  # Prompt the user for input and remove commas
  input <- readline(prompt="Enter a number: ")
  input <- gsub(",", "", input)
  x <- as.bigz(input)

  xx <- x    # Starting Number
  ST <- 1    # Total Steps, start from 1
  STB <- 0   # Steps to Below Start
  STBB <- 0  # Step Count
  SD <- 0    # Search for MOD
  SD2 <- 0   # Divide by 2
  SR <- 0    # Count of Shifts Right
  MDD1 <- 0  #
  BB <- 0    # Base check
  BBC <- 3   # Base Count
  Branch <- "NO" # Is Branch
  BA <- "NO" # Is Base
  BC <- 0    # Base C
  BE <- as.bigz(0) # Even Base

  while (!is.na(x) && x != 1) {
    if ((x %% as.bigz(2)) == 0) {
      N <- x
      NF <- x / 2
      x <- as.bigz(NF)  # Ensure x remains a bigz integer
      SD2 <- SD2 + 1
      TN <- N
      MD <- as.bigz(2)
      MDD1 <- TN %% MD
      MDD8 <- TN %% as.bigz(8)

      while (SD != 1) {
        MDD2 <- MDD1
        MD <- MD * 2
        MDD1 <- TN %% MD
        if (MDD1 == MDD2 && MDD1 != 0) {
          SD <- 1
        }
      }
      SD <- 0
      R <- TN %% MD
      AP <- (N - R) / MD
```

```
  RR <- (MD * AP) + R
  if (BB == 1) {
    BBC <- BBC + 1
  }

  ST <- ST + 1

} else {
  if ((xx %% as.bigz(2)) == 0) {
    BB <- 2
    BE <- (xx * 3) + 1
  } else if (BB == 1 && BBC < 3) {
    BE <- (xx * 3) + 1
  } else {
    BE <- (xx * 3) + 1
  }
  if (BB == 1) {
    BB <- 2
  }
  if (BB == 0) {
    BB <- BB + 1
    BBC <- 0
  }

  N <- x
  FN <- x
  NF <- (3 * x) + 1
  TN <- N
  MD <- as.bigz(2)
  MDD1 <- TN %% MD
  MDD8 <- TN %% as.bigz(8)

  while (SD != 1) {
    MDD2 <- MDD1
    MD <- MD * 2
    MDD1 <- TN %% MD
    if (MDD1 == MDD2 && MDD1 != 0) {
      SD <- 1
    }
  }
  SD <- 0
  R <- TN %% MD
  AP <- (N - R) / MD
  RR <- (MD * AP) + R

  ST <- ST + 1
```

```
    if (x == 1) {

    } else {
      x <- as.bigz((3 * x) + 1)  # Ensure x remains a bigz integer
      SR <- SR + 1
    }
  }
}

if (xx %% as.bigz(2) == 1) {  # If the number is odd
  Branch <- "NO"
  BA <- "NO"
  BE <- (xx * 3) + 1  # Set E Base as 3x+1 for odd numbers
} else if (is_1MOD3(xx)) {  # Check if the even number is a branch
  Branch <- "YES"
  BE <- (xx - as.bigz(1)) / as.bigz(3)  # Use as.bigz to ensure precision
  BE <- as.bigz(BE)  # Ensure BE is treated as a bigz integer
  #cat("BE after calculation:", as.character(BE), "\n")  # Debug print
  # Perform the base check on xx
  div4 <- xx / as.bigz(4)
  div4 <- as.bigz(div4)  # Ensure div4 is treated as a bigz integer
  #cat("div4:", as.character(div4), "\n")  # Debug print
  if ((div4 %% as.bigz(2)) == 0) {  # If div4 is even
    #cat("div4 is even\n")
    if (!is_1MOD3(div4)) {  # If div4 is not 1MOD3, it is a base
      BA <- "YES"
    } else {
      BA <- "NO"
    }
  } else {  # If div4 is odd, it is a base
    #cat("div4 is odd\n")
    BA <- "YES"
  }
} else {  # Even number but not a branch
  Branch <- "NO"
  BA <- "NO"
  BE <- as.bigz(0)
}

BC <- SR - 1

cat("Start:", as.character(xx), "\n")
cat("Steps:", ST - 1, "\n")  # Subtract 1 to correct the final step count
cat("Shift:", SR, "\n")
cat("Divide:", SD2, "\n")
```

```
  cat("Branch:", Branch, "\n")
  cat("Base:", BA, "\n")
  # Print E Base directly to ensure proper formatting
  if (as.numeric(BE) == 0) {
    cat("E Base: 0\n")
  } else {
    cat("E Base:", as.character(BE), "\n")
  }
}

# Run the function
run_collatz()
```

==========

End R Code

Results of Code

```
Enter a number: 95,560,746,531,118,716,018,384,891,544,079,288,513,588,277,15
8,888,376,726,675,291,951,451,666,121,649,17395,560,746,531,118,716,018,384,8
91,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,17395,
560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,2
91,951,451,666,121,649,173
Start: 95560746531118716018384891544079288513588277158888376726675291951451
6121649173955607465311187160183848915440792885135882771588883767266752919514
1666121649173955607465311187160183848915440792885135882771588883767266752919
1451666121649173
Steps: 5851
Shift: 1955
Divide: 3896
Branch: NO
Base: NO
E Base: 2866822395933561480551546746322378655407648314766651301800258758543
5499836494752186682239593356148055154674632237865540764831476665130180025875
4354998364947521866822395933561480551546746322378655407648314766651301800258
7543549983649475218668223959335614805515467463223786554076483147666513018002
58754998364947520
```

```
Enter a number: 27
Start: 27
Steps: 111
Shift: 41
Divide: 70
Branch: NO
Base: NO
E Base: 82
```

```
Enter a number: 1023
Start: 1023
Steps: 62
Shift: 20
Divide: 42
Branch: NO
Base: NO
E Base: 3070
```

```
Enter a number: 29384739048572309458723948 57
Start: 29384739048572309458723948 57
Steps: 655
Shift: 218
Divide: 437
Branch: NO
Base: NO
E Base: 881542171457169283761718 4572

Enter a number: 4
Start: 4
Steps: 2
Shift: 0
Divide: 2
Branch: YES
Base: YES
E Base: 1

Enter a number: 10
Start: 10
Steps: 6
Shift: 1
Divide: 5
Branch: YES
Base: YES
E Base: 3

Enter a number: 16
Start: 16
Steps: 4
Shift: 0
Divide: 4
Branch: YES
Base: NO
E Base: 5
```

This code writes all Odd 3MOD(4) numbers to a file

- 3MOD(8) numbers are all MOD(8) and below themselves in 5 steps
- 7MOD(8) numbers follow a pattern starting at MOD(16) and slowly growing to larger MOD's

Start R Code

```
=====================================================================
library(gmp)
library(Rmpfr)
########################
#This code writes all the odd 3MOD(4) numbers to a file with there 1MOD(4) nu
mber
#that is 3 steps from below itself
#It also shows the power MOD slot for that number
#Works up to 2^750 numbers
########################

# Function to find the next 1MOD(4) number and the number that is 3 steps fro
m being below itself
find_1MOD4_and_steps <- function(start_num) {
  x <- mpfr(start_num, precBits = 128)

  # Check if the start number is already 1MOD(4)
  if (asNumeric(x %% 4) == 1) {
    return(NULL)
  }
```

```
  TN <- x
  MD <- 2
  MDD1 <- asNumeric(TN %% MD)
  SD <- 0
  start_mod <- MD

  while (SD != 1) {
    MDD2 <- MDD1
    MD <- MD * 2
    MDD1 <- asNumeric(TN %% MD)
    if (MDD1 == MDD2 && MDD1 != 0) {
      SD <- 1
      start_mod <- MD
    }
  }

  ST <- 0  # Total Steps

  while (TRUE) {
    if ((x %% 2) == 0) {
      x <- x / 2
      ST <- ST + 1
    } else {
      x <- (3 * x) + 1
      ST <- ST + 1
    }

    if (asNumeric(x %% 4) == 1) {
      # Found the next 1MOD(4) number
      steps_to_below_itself <- ST + 3
      return(data.frame(
        StartNumber = asNumeric(start_num),
        StartingMOD = paste0("MOD(", start_mod, ")"),
        Next1MOD4Number = asNumeric(x),
        StepsToBelowItself = steps_to_below_itself
      ))
    }
  }
}

# Create an empty data frame to store the results
results <- data.frame(StartNumber = numeric(), StartingMOD = character(), Nex
t1MOD4Number = numeric(), StepsToBelowItself = numeric(), stringsAsFactors =
FALSE)

# Initialize starting number
i <- 1

# Infinite loop to process odd numbers indefinitely
while (TRUE) {
  if (i %% 2 != 0) {  # Process only odd numbers
    result <- find_1MOD4_and_steps(i)
    if (!is.null(result)) {
      results <- rbind(results, result)

      # Periodically write the results to a CSV file to avoid data loss in ca
se of interruption
      if (nrow(results) %% 100 == 0) {  # Adjust the frequency as needed
        write.csv(results, "C:/3x+1/results.csv", row.names = FALSE)
        cat("Results written to C:/3x+1/results.csv up to number", i, "\n")
      }
    }
  }
  i <- i + 1
```

```
}
```

```
==========================================================================
```

End R Code

Results of Code

First 100 results

StartNumber	StartingMOD	Next1MOD4Number	StepsToBelowItself
3	MOD(8)	5	5
7	MOD(16)	17	7
11	MOD(8)	17	5
15	MOD(32)	53	9
19	MOD(8)	29	5
23	MOD(16)	53	7
27	MOD(8)	41	5
31	MOD(64)	161	11
35	MOD(8)	53	5
39	MOD(16)	89	7
43	MOD(8)	65	5
47	MOD(32)	161	9
51	MOD(8)	77	5
55	MOD(16)	125	7
59	MOD(8)	89	5
63	MOD(128)	485	13
67	MOD(8)	101	5
71	MOD(16)	161	7
75	MOD(8)	113	5
79	MOD(32)	269	9
83	MOD(8)	125	5
87	MOD(16)	197	7
91	MOD(8)	137	5
95	MOD(64)	485	11
99	MOD(8)	149	5
103	MOD(16)	233	7
107	MOD(8)	161	5
111	MOD(32)	377	9
115	MOD(8)	173	5
119	MOD(16)	269	7
123	MOD(8)	185	5
127	MOD(256)	1457	15
131	MOD(8)	197	5
135	MOD(16)	305	7
139	MOD(8)	209	5
143	MOD(32)	485	9

147	MOD(8)	221	5
151	MOD(16)	341	7
155	MOD(8)	233	5
159	MOD(64)	809	11
163	MOD(8)	245	5
167	MOD(16)	377	7
171	MOD(8)	257	5
175	MOD(32)	593	9
179	MOD(8)	269	5
183	MOD(16)	413	7
187	MOD(8)	281	5
191	MOD(128)	1457	13
195	MOD(8)	293	5
199	MOD(16)	449	7
203	MOD(8)	305	5
207	MOD(32)	701	9
211	MOD(8)	317	5
215	MOD(16)	485	7
219	MOD(8)	329	5
223	MOD(64)	1133	11
227	MOD(8)	341	5
231	MOD(16)	521	7
235	MOD(8)	353	5
239	MOD(32)	809	9
243	MOD(8)	365	5
247	MOD(16)	557	7
251	MOD(8)	377	5
255	MOD(512)	4373	17
259	MOD(8)	389	5
263	MOD(16)	593	7
267	MOD(8)	401	5
271	MOD(32)	917	9
275	MOD(8)	413	5
279	MOD(16)	629	7
283	MOD(8)	425	5
287	MOD(64)	1457	11
291	MOD(8)	437	5
295	MOD(16)	665	7
299	MOD(8)	449	5
303	MOD(32)	1025	9
307	MOD(8)	461	5
311	MOD(16)	701	7
315	MOD(8)	473	5

319	MOD(128)	2429	13
323	MOD(8)	485	5
327	MOD(16)	737	7
331	MOD(8)	497	5
335	MOD(32)	1133	9
339	MOD(8)	509	5
343	MOD(16)	773	7
347	MOD(8)	521	5
351	MOD(64)	1781	11
355	MOD(8)	533	5
359	MOD(16)	809	7
363	MOD(8)	545	5
367	MOD(32)	1241	9
371	MOD(8)	557	5
375	MOD(16)	845	7
379	MOD(8)	569	5
383	MOD(256)	4373	15
387	MOD(8)	581	5
391	MOD(16)	881	7
395	MOD(8)	593	5

This code prints a number and all the steps to a CSV file and adds all supporting power slot info and all the MOD(8) for each number

Start R Code

```
=====================================================================
rm(list = ls())
library(gmp)
library(Rmpfr)
########################
#This runs the steps and prints them to a file
#with supporting Power Slot info and MOD(8) info
#Works up to 2^750 numbers
########################

# Function to read a large integer input with commas
read_large_integer <- function(prompt) {
  as.bigz(gsub(",", "", readline(prompt = prompt)))
}

# Function to get the current date in YYYYMMDD format
get_current_date <- function() {
  format(Sys.Date(), "%Y%m%d")
}

collatz_conjecture <- function(output_folder) {
  # Ensure the output folder exists
  if (!dir.exists(output_folder)) {
    dir.create(output_folder, recursive = TRUE)
  }

  # Read the input number
  input_str <- readline(prompt = "Enter a number: ")
```

```
  x <- as.bigz(gsub(",", "", input_str))

  # Create filename based on current timestamp
  timestamp <- get_current_date()
  output_file <- file.path(output_folder, paste0("output_Run_", timestamp, ".
csv"))

  # Open a connection to the output file
  con <- file(output_file, open = "wt")
  on.exit(close(con))

  # Write CSV headers
  writeLines("Start,End,MOD Power,AP,R,RR,Step,MOD8", con)

  xx <- x    # Starting Number
  ST <- 1    # Total Steps
  STB <- 0   # Steps to Below Start
  STBB <- 0  # Step Count
  OS <- 0    # Trend Counter
  OSO <- 0   # Trend Counter
  OSH <- 0   # Trend Height
  SD <- 0    # Search for MOD
  SD2 <- 0   # Divide by 2
  SR <- 0    # Count of Shifts Right
  MDD1 <- as.bigz(0)  #
  BB <- 0    # Base check
  BBC <- 3   # Base Count
  BA <- "NO" # Is Base
  BC <- 0    # Base C
  BE <- 0    # Even Base
  OS <- 0    # Trend Counter
  OSO <- 0   # Trend Counter
  OSH <- 0   # Trend Height

  while (x != 1) {
    x <- as.bigz(x) # Ensure x is a bigz object
    if (x %% 2 == 0) {
      # Trend
      if (OSO == 0) {
        OS <- 0
        OSO <- 0
      } else {
        if (OSO == 1) {
          if (OS > OSH) {
            OSH <- OS
          }
        }
        OSO <- OSO + 1
      }

      N <- x
      NF <- x / 2
      x <- x / 2
      SD2 <- SD2 + 1
      TN <- N
      MD <- as.bigz(2)
      MDD1 <- TN %% MD
      MDD8 <- TN %% 8

      while (SD != 1) {
        MDD2 <- MDD1
        MD <- MD * 2
        MDD1 <- TN %% MD
        if (MDD1 == MDD2 && MDD1 != 0) {
```

```
          SD <- 1
        }
      }
      SD <- 0
      R <- TN %% MD
      AP <- (N - R) / MD
      RR <- (MD * AP) + R
      if (BB == 1) {
        BBC <- BBC + 1
      }

      writeLines(paste(as.character(N), as.character(NF), as.character(MD), a
s.character(AP), as.character(R), as.character(RR), ST, as.character(MDD8), s
ep = ","), con)

      R <- as.bigz(0)
      AP <- as.bigz(0)
      RR <- as.bigz(0)
      MD <- as.bigz(0)
      ST <- ST + 1

    } else if (x %% 2 == 1) {
      x <- as.bigz(x) # Ensure x is a bigz object
      # Trend
      if (OSO == 2) {
        OS <- OS + 1
        OSO <- 1
      } else {
        OS <- 1
        OSO <- 1
      }

      if (xx %% 2 == 0) {
        BB <- 2
        BE <- (xx * 3) + 1
      } else if (BB == 1 && BBC < 3) {
        BA <- "YES"
        BE <- (xx * 3) + 1
      } else {
        BE <- (xx * 3) + 1
      }
      if (BB == 1) {
        BB <- 2
      }
      if (BB == 0) {
        BB <- BB + 1
        BBC <- 0
      }

      N <- x
      FN <- x
      NF <- (3 * x) + 1
      TN <- N
      MD <- as.bigz(2)
      MDD1 <- TN %% MD
      MDD8 <- TN %% 8

      while (SD != 1) {
        MDD2 <- MDD1
        MD <- MD * 2
        MDD1 <- TN %% MD
        if (MDD1 == MDD2 && MDD1 != 0) {
          SD <- 1
        }
```

```
      }
      SD <- 0
      R <- TN %% MD
      AP <- (N - R) / MD
      RR <- (MD * AP) + R

      writeLines(paste(as.character(N), as.character(NF), as.character(MD), a
s.character(AP), as.character(R), as.character(RR), ST, as.character(MDD8), s
ep = ","), con)

      R <- as.bigz(0)
      AP <- as.bigz(0)
      RR <- as.bigz(0)
      MD <- as.bigz(0)

      ST <- ST + 1

      if (x != 1) {
        x <- (3 * x) + 1
        SR <- SR + 1
      }
    }
  }
  cat("Output written to", output_file, "\n")
}

# Call the function to execute the code and save the output to the specified
folder
collatz_conjecture("C:/3x+1")
```

==========

End R Code

Results of Code

Start	End	MOD Power	AP	R	RR	Step	MOD8
27	82	8	3	3	27	1	3
82	41	8	10	2	82	2	2
41	124	4	10	1	41	3	1
124	62	256	0	124	124	4	4
62	31	128	0	62	62	5	6
31	94	64	0	31	31	6	7
94	47	64	1	30	94	7	6
47	142	32	1	15	47	8	7
142	71	32	4	14	142	9	6
71	214	16	4	7	71	10	7
214	107	16	13	6	214	11	6
107	322	8	13	3	107	12	3
322	161	8	40	2	322	13	2
161	484	4	40	1	161	14	1
484	242	16	30	4	484	15	4
242	121	8	30	2	242	16	2
121	364	4	30	1	121	17	1
364	182	32	11	12	364	18	4

182	91	16	11	6	182	19	6
91	274	8	11	3	91	20	3
274	137	8	34	2	274	21	2
137	412	4	34	1	137	22	1
412	206	64	6	28	412	23	4
206	103	32	6	14	206	24	6
103	310	16	6	7	103	25	7
310	155	16	19	6	310	26	6
155	466	8	19	3	155	27	3
466	233	8	58	2	466	28	2
233	700	4	58	1	233	29	1
700	350	128	5	60	700	30	4
350	175	64	5	30	350	31	6
175	526	32	5	15	175	32	7
526	263	32	16	14	526	33	6
263	790	16	16	7	263	34	7
790	395	16	49	6	790	35	6
395	1186	8	49	3	395	36	3
1186	593	8	148	2	1186	37	2
593	1780	4	148	1	593	38	1
1780	890	16	111	4	1780	39	4
890	445	8	111	2	890	40	2
445	1336	4	111	1	445	41	5
1336	668	128	10	56	1336	42	0
668	334	64	10	28	668	43	4
334	167	32	10	14	334	44	6
167	502	16	10	7	167	45	7
502	251	16	31	6	502	46	6
251	754	8	31	3	251	47	3
754	377	8	94	2	754	48	2
377	1132	4	94	1	377	49	1
1132	566	32	35	12	1132	50	4
566	283	16	35	6	566	51	6
283	850	8	35	3	283	52	3
850	425	8	106	2	850	53	2
425	1276	4	106	1	425	54	1
1276	638	512	2	252	1276	55	4
638	319	256	2	126	638	56	6
319	958	128	2	63	319	57	7
958	479	128	7	62	958	58	6
479	1438	64	7	31	479	59	7
1438	719	64	22	30	1438	60	6
719	2158	32	22	15	719	61	7

2158	1079	32	67	14	2158	62	6
1079	3238	16	67	7	1079	63	7
3238	1619	16	202	6	3238	64	6
1619	4858	8	202	3	1619	65	3
4858	2429	8	607	2	4858	66	2
2429	7288	4	607	1	2429	67	5
7288	3644	256	28	120	7288	68	0
3644	1822	128	28	60	3644	69	4
1822	911	64	28	30	1822	70	6
911	2734	32	28	15	911	71	7
2734	1367	32	85	14	2734	72	6
1367	4102	16	85	7	1367	73	7
4102	2051	16	256	6	4102	74	6
2051	6154	8	256	3	2051	75	3
6154	3077	8	769	2	6154	76	2
3077	9232	4	769	1	3077	77	5
9232	4616	64	144	16	9232	78	0
4616	2308	32	144	8	4616	79	0
2308	1154	16	144	4	2308	80	4
1154	577	8	144	2	1154	81	2
577	1732	4	144	1	577	82	1
1732	866	16	108	4	1732	83	4
866	433	8	108	2	866	84	2
433	1300	4	108	1	433	85	1
1300	650	16	81	4	1300	86	4
650	325	8	81	2	650	87	2
325	976	4	81	1	325	88	5
976	488	64	15	16	976	89	0
488	244	32	15	8	488	90	0
244	122	16	15	4	244	91	4
122	61	8	15	2	122	92	2
61	184	4	15	1	61	93	5
184	92	128	1	56	184	94	0
92	46	64	1	28	92	95	4
46	23	32	1	14	46	96	6
23	70	16	1	7	23	97	7
70	35	16	4	6	70	98	6
35	106	8	4	3	35	99	3
106	53	8	13	2	106	100	2
53	160	4	13	1	53	101	5
160	80	128	1	32	160	102	0
80	40	64	1	16	80	103	0
40	20	32	1	8	40	104	0

20	10	16	1	4	20	105	4
10	5	8	1	2	10	106	2
5	16	4	1	1	5	107	5
16	8	64	0	16	16	108	0
8	4	32	0	8	8	109	0
4	2	16	0	4	4	110	4
2	1	8	0	2	2	111	2

Input a number and it will give you the number that is below itself and how many steps to get there

Start R Code

```
==========================================================================
library(gmp)

#######################
#Input a number and it will give you the number that is below itself
#and how many steps to get there
#works up to 2^750 numbers
#######################

# Function to find the next 1MOD(4) number and the number that is 3 steps fro
m being below itself
find_1MOD4_and_steps <- function(start_num) {
  x <- as.bigz(start_num)

  # Check if the start number is already 1MOD(4)
  if (as.integer(mod.bigz(x, 4)) == 1) {
    cat("Start number:", format(start_num, big.mark = ","), "is already 1MOD(
4)\n")
    return(NULL)
  }

  TN <- x
  MD <- as.bigz(2)
  MDD1 <- mod.bigz(TN, MD)
  SD <- 0
  start_mod <- MD

  while (SD != 1) {
    MDD2 <- MDD1
    MD <- MD * 2
    MDD1 <- mod.bigz(TN, MD)
    if (MDD1 == MDD2 && MDD1 != 0) {
      SD <- 1
      start_mod <- MD
    }
  }

  ST <- 0  # Total Steps

  while (TRUE) {
    if (as.integer(mod.bigz(x, 2)) == 0) {
      x <- x / 2
      ST <- ST + 1
    } else {
      x <- (3 * x) + 1
      ST <- ST + 1
    }
```

```
    if (as.integer(mod.bigz(x, 4)) == 1) {
      # Found the next 1MOD(4) number
      steps_to_below_itself <- ST + 3
      cat("Start Number:", format(start_num, big.mark = ","), "\n")
      cat("Starting MOD:", paste0("MOD(", format(start_mod, big.mark = ","),
")"), "\n")
      cat("Next 1MOD(4) Number:", format(x, big.mark = ","), "\n")
      cat("Steps to below itself:", steps_to_below_itself, "\n")
      return(invisible())
    }
  }
}

# Function to remove commas from the input number
remove_commas <- function(num_str) {
  return(gsub(",", "", num_str))
}

# Prompt the user for input
input_num <- readline(prompt = "Enter a number: ")
start_num <- as.bigz(remove_commas(input_num))

# Check if the input is valid
if (!is.na(start_num) && as.numeric(start_num) > 0) {
  find_1MOD4_and_steps(start_num)
} else {
  cat("Invalid input. Please enter a positive number.\n")
}
```
===

End R Code

Results of Code

```
Enter a number: 95,560,746,531,118,716,018,384,891,544,079,288,513,588,277,15
8,888,376,726,675,291,951,451,666,121,649,17395,560,746,531,118,716,018,384,8
91,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,17395,
560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,2
91,951,451,666,121,649,173

Start number: 955607465311187160183848915440792885135882771588883767266752919
51451666121649173955607465311187160183848915440792885135882771588883767266752
91951451666121649173955607465311187160183848915440792885135882771588883767266
75291951451666121649173 is already 1MOD(4)

Enter a number: 95,560,746,531,118,716,018,384,891,544,079,288,513,588,277,15
8,888,376,726,675,291,951,451,666,121,649,17395,560,746,531,118,716,018,384,8
91,544,079,288,513,588,277,158,888,376,726,675,291,951,451,666,121,649,17395,
560,746,531,118,716,018,384,891,544,079,288,513,588,277,158,888,376,726,675,2
91,951,451,666,121,649,171

Start Number: 955607465311187160183848915440792885135882771588883767266752919
51451666121649173955607465311187160183848915440792885135882771588883767266752
91951451666121649173955607465311187160183848915440792885135882771588883767266
75291951451666121649171
Starting MOD: MOD(8)
Next 1MOD(4) Number: 14334111979667807402757733731611893277038241573833256509
00129379271774991824737609334111979667807402757733731611893277038241573833256
50900129379271774991824737609334111979667807402757733731611893277038241573833
2565090012937927177499182473757
Steps to below itself: 5

Enter a number: 27
```

```
Start Number: 27
Starting MOD: MOD(8)
Next 1MOD(4) Number: 41
Steps to below itself: 5

Enter a number: 2,854,495,385,411,919,762,116,571,938,898,990,272,765,493,247

Start Number: 2854495385411919762116571938898990272765493247
Starting MOD: MOD(57089907708238395242331438777979805455309864 96)
Next 1MOD(4) Number: 7399769700702539458494015649033932883729462007794459476 30368810603496497
Steps to below itself: 303
```

This Code prints out the next 1000 number of your choice:

- All Numbers
- All Odd numbers
- All 7MOD(8) numbers

Start R Code

```
================================================================================
library(gmp)

########################
#Prints out the next 1000 numbers and their power info
#In one of 3 ways: All numbers, All Odd, Odd 7MOD(8)
#Works up to 2^750 numbers
########################

# Function to read a large integer input with commas
read_large_integer <- function(prompt) {
  as.bigz(gsub(",", "", readline(prompt = prompt)))
}

# Function to calculate the MOD Power structure
calculate_mod_power <- function(num) {
  TN <- num
  MD <- as.bigz(2)
  MDD1 <- TN %% MD
  SD <- 0

  while (SD != 1) {
    MDD2 <- MDD1
    MD <- MD * 2
    MDD1 <- TN %% MD
    if (MDD1 == MDD2 && MDD1 != 0) {
      SD <- 1
    }
  }
  R <- TN %% MD
  AP <- (num - R) / MD
  RR <- (MD * AP) + R
  list(MD = MD, AP = AP, R = R, RR = RR)
}

# Function to print all numbers
print_all_numbers <- function(x) {
  for (i in 0:999) {
    current_num <- x + i
    mod_power <- calculate_mod_power(current_num)
    mod8 <- current_num %% 8
    print(paste(
      as.character(current_num),
      "MOD Power:", as.character(mod_power$MD), "*", as.character(mod_power$A
P), "+", as.character(mod_power$R), "=", as.character(mod_power$RR),
```

```
      "MOD8:", as.character(mod8)
    ))
  }
}

# Function to print all odd numbers
print_odd_numbers <- function(x) {
  count <- 0
  while (count < 1000) {
    if (x %% 2 != 0) { # Check if the number is odd
      mod_power <- calculate_mod_power(x)
      mod8 <- x %% 8
      print(paste(
        as.character(x),
        "MOD Power:", as.character(mod_power$MD), "*", as.character(mod_power
$AP), "+", as.character(mod_power$R), "=", as.character(mod_power$RR),
        "MOD8:", as.character(mod8)
      ))
      count <- count + 1
    }
    x <- x + 1
  }
}

# Function to print numbers that are 7MOD(8)
print_mod7_numbers <- function(x) {
  count <- 0
  while (count < 1000) {
    if (x %% 8 == 7) { # Check if the number is 7MOD(8)
      mod_power <- calculate_mod_power(x)
      mod8 <- x %% 8
      print(paste(
        as.character(x),
        "MOD Power:", as.character(mod_power$MD), "*", as.character(mod_power
$AP), "+", as.character(mod_power$R), "=", as.character(mod_power$RR),
        "MOD8:", as.character(mod8)
      ))
      count <- count + 1
    }
    x <- x + 1
  }
}

# Main function to get user input and execute the chosen option
main <- function() {
  cat("Choose an option:\n")
  cat("1. Print all numbers\n")
  cat("2. Print all odd numbers\n")
  cat("3. Print numbers that are 7MOD(8)\n")
  choice <- as.integer(readline(prompt = "Enter your choice (1, 2, or 3): "))

  if (choice %in% 1:3) {
    x <- read_large_integer("Enter a starting number: ")

    if (choice == 1) {
      print_all_numbers(x)
    } else if (choice == 2) {
      print_odd_numbers(x)
    } else if (choice == 3) {
      print_mod7_numbers(x)
    }
  } else {
    cat("Invalid choice. Please run the script again and enter a valid option
.\n")
```

```
  }
}

# Call the main function to execute the code
main()
```

==========

End R Code

Results of Code

1st 1000 7MOD (8) numbers

```
[1] "7 MOD Power: 16 * 0 + 7 = 7 MOD8: 7"
[1] "15 MOD Power: 32 * 0 + 15 = 15 MOD8: 7"
[1] "23 MOD Power: 16 * 1 + 7 = 23 MOD8: 7"
[1] "31 MOD Power: 64 * 0 + 31 = 31 MOD8: 7"
[1] "39 MOD Power: 16 * 2 + 7 = 39 MOD8: 7"
[1] "47 MOD Power: 32 * 1 + 15 = 47 MOD8: 7"
[1] "55 MOD Power: 16 * 3 + 7 = 55 MOD8: 7"
[1] "63 MOD Power: 128 * 0 + 63 = 63 MOD8: 7"
[1] "71 MOD Power: 16 * 4 + 7 = 71 MOD8: 7"
[1] "79 MOD Power: 32 * 2 + 15 = 79 MOD8: 7"
[1] "87 MOD Power: 16 * 5 + 7 = 87 MOD8: 7"
[1] "95 MOD Power: 64 * 1 + 31 = 95 MOD8: 7"
[1] "103 MOD Power: 16 * 6 + 7 = 103 MOD8: 7"
[1] "111 MOD Power: 32 * 3 + 15 = 111 MOD8: 7"
[1] "119 MOD Power: 16 * 7 + 7 = 119 MOD8: 7"
[1] "127 MOD Power: 256 * 0 + 127 = 127 MOD8: 7"
[1] "135 MOD Power: 16 * 8 + 7 = 135 MOD8: 7"
[1] "143 MOD Power: 32 * 4 + 15 = 143 MOD8: 7"
[1] "151 MOD Power: 16 * 9 + 7 = 151 MOD8: 7"
[1] "159 MOD Power: 64 * 2 + 31 = 159 MOD8: 7"
[1] "167 MOD Power: 16 * 10 + 7 = 167 MOD8: 7"
[1] "175 MOD Power: 32 * 5 + 15 = 175 MOD8: 7"
[1] "183 MOD Power: 16 * 11 + 7 = 183 MOD8: 7"
[1] "191 MOD Power: 128 * 1 + 63 = 191 MOD8: 7"
[1] "199 MOD Power: 16 * 12 + 7 = 199 MOD8: 7"
[1] "207 MOD Power: 32 * 6 + 15 = 207 MOD8: 7"
[1] "215 MOD Power: 16 * 13 + 7 = 215 MOD8: 7"
[1] "223 MOD Power: 64 * 3 + 31 = 223 MOD8: 7"
[1] "231 MOD Power: 16 * 14 + 7 = 231 MOD8: 7"
[1] "239 MOD Power: 32 * 7 + 15 = 239 MOD8: 7"
[1] "247 MOD Power: 16 * 15 + 7 = 247 MOD8: 7"
[1] "255 MOD Power: 512 * 0 + 255 = 255 MOD8: 7"
[1] "263 MOD Power: 16 * 16 + 7 = 263 MOD8: 7"
[1] "271 MOD Power: 32 * 8 + 15 = 271 MOD8: 7"
[1] "279 MOD Power: 16 * 17 + 7 = 279 MOD8: 7"
[1] "287 MOD Power: 64 * 4 + 31 = 287 MOD8: 7"
[1] "295 MOD Power: 16 * 18 + 7 = 295 MOD8: 7"
[1] "303 MOD Power: 32 * 9 + 15 = 303 MOD8: 7"
[1] "311 MOD Power: 16 * 19 + 7 = 311 MOD8: 7"
[1] "319 MOD Power: 128 * 2 + 63 = 319 MOD8: 7"
[1] "327 MOD Power: 16 * 20 + 7 = 327 MOD8: 7"
[1] "335 MOD Power: 32 * 10 + 15 = 335 MOD8: 7"
[1] "343 MOD Power: 16 * 21 + 7 = 343 MOD8: 7"
[1] "351 MOD Power: 64 * 5 + 31 = 351 MOD8: 7"
[1] "359 MOD Power: 16 * 22 + 7 = 359 MOD8: 7"
[1] "367 MOD Power: 32 * 11 + 15 = 367 MOD8: 7"
[1] "375 MOD Power: 16 * 23 + 7 = 375 MOD8: 7"
```

```
[1] "383 MOD Power: 256 * 1 + 127 = 383 MOD8: 7"
[1] "391 MOD Power: 16 * 24 + 7 = 391 MOD8: 7"
[1] "399 MOD Power: 32 * 12 + 15 = 399 MOD8: 7"
[1] "407 MOD Power: 16 * 25 + 7 = 407 MOD8: 7"
[1] "415 MOD Power: 64 * 6 + 31 = 415 MOD8: 7"
[1] "423 MOD Power: 16 * 26 + 7 = 423 MOD8: 7"
[1] "431 MOD Power: 32 * 13 + 15 = 431 MOD8: 7"
[1] "439 MOD Power: 16 * 27 + 7 = 439 MOD8: 7"
[1] "447 MOD Power: 128 * 3 + 63 = 447 MOD8: 7"
[1] "455 MOD Power: 16 * 28 + 7 = 455 MOD8: 7"
[1] "463 MOD Power: 32 * 14 + 15 = 463 MOD8: 7"
[1] "471 MOD Power: 16 * 29 + 7 = 471 MOD8: 7"
[1] "479 MOD Power: 64 * 7 + 31 = 479 MOD8: 7"
[1] "487 MOD Power: 16 * 30 + 7 = 487 MOD8: 7"
[1] "495 MOD Power: 32 * 15 + 15 = 495 MOD8: 7"
[1] "503 MOD Power: 16 * 31 + 7 = 503 MOD8: 7"
[1] "511 MOD Power: 1024 * 0 + 511 = 511 MOD8: 7"
[1] "519 MOD Power: 16 * 32 + 7 = 519 MOD8: 7"
[1] "527 MOD Power: 32 * 16 + 15 = 527 MOD8: 7"
[1] "535 MOD Power: 16 * 33 + 7 = 535 MOD8: 7"
[1] "543 MOD Power: 64 * 8 + 31 = 543 MOD8: 7"
[1] "551 MOD Power: 16 * 34 + 7 = 551 MOD8: 7"
[1] "559 MOD Power: 32 * 17 + 15 = 559 MOD8: 7"
[1] "567 MOD Power: 16 * 35 + 7 = 567 MOD8: 7"
[1] "575 MOD Power: 128 * 4 + 63 = 575 MOD8: 7"
[1] "583 MOD Power: 16 * 36 + 7 = 583 MOD8: 7"
[1] "591 MOD Power: 32 * 18 + 15 = 591 MOD8: 7"
[1] "599 MOD Power: 16 * 37 + 7 = 599 MOD8: 7"
[1] "607 MOD Power: 64 * 9 + 31 = 607 MOD8: 7"
[1] "615 MOD Power: 16 * 38 + 7 = 615 MOD8: 7"
[1] "623 MOD Power: 32 * 19 + 15 = 623 MOD8: 7"
[1] "631 MOD Power: 16 * 39 + 7 = 631 MOD8: 7"
[1] "639 MOD Power: 256 * 2 + 127 = 639 MOD8: 7"
[1] "647 MOD Power: 16 * 40 + 7 = 647 MOD8: 7"
[1] "655 MOD Power: 32 * 20 + 15 = 655 MOD8: 7"
[1] "663 MOD Power: 16 * 41 + 7 = 663 MOD8: 7"
[1] "671 MOD Power: 64 * 10 + 31 = 671 MOD8: 7"
[1] "679 MOD Power: 16 * 42 + 7 = 679 MOD8: 7"
[1] "687 MOD Power: 32 * 21 + 15 = 687 MOD8: 7"
[1] "695 MOD Power: 16 * 43 + 7 = 695 MOD8: 7"
[1] "703 MOD Power: 128 * 5 + 63 = 703 MOD8: 7"
[1] "711 MOD Power: 16 * 44 + 7 = 711 MOD8: 7"
[1] "719 MOD Power: 32 * 22 + 15 = 719 MOD8: 7"
[1] "727 MOD Power: 16 * 45 + 7 = 727 MOD8: 7"
[1] "735 MOD Power: 64 * 11 + 31 = 735 MOD8: 7"
[1] "743 MOD Power: 16 * 46 + 7 = 743 MOD8: 7"
[1] "751 MOD Power: 32 * 23 + 15 = 751 MOD8: 7"
[1] "759 MOD Power: 16 * 47 + 7 = 759 MOD8: 7"
[1] "767 MOD Power: 512 * 1 + 255 = 767 MOD8: 7"
[1] "775 MOD Power: 16 * 48 + 7 = 775 MOD8: 7"
[1] "783 MOD Power: 32 * 24 + 15 = 783 MOD8: 7"
[1] "791 MOD Power: 16 * 49 + 7 = 791 MOD8: 7"
[1] "799 MOD Power: 64 * 12 + 31 = 799 MOD8: 7"
[1] "807 MOD Power: 16 * 50 + 7 = 807 MOD8: 7"
[1] "815 MOD Power: 32 * 25 + 15 = 815 MOD8: 7"
[1] "823 MOD Power: 16 * 51 + 7 = 823 MOD8: 7"
[1] "831 MOD Power: 128 * 6 + 63 = 831 MOD8: 7"
[1] "839 MOD Power: 16 * 52 + 7 = 839 MOD8: 7"
[1] "847 MOD Power: 32 * 26 + 15 = 847 MOD8: 7"
[1] "855 MOD Power: 16 * 53 + 7 = 855 MOD8: 7"
[1] "863 MOD Power: 64 * 13 + 31 = 863 MOD8: 7"
[1] "871 MOD Power: 16 * 54 + 7 = 871 MOD8: 7"
[1] "879 MOD Power: 32 * 27 + 15 = 879 MOD8: 7"
[1] "887 MOD Power: 16 * 55 + 7 = 887 MOD8: 7"
```

```
[1] "895 MOD Power: 256 * 3 + 127 = 895 MOD8: 7"
[1] "903 MOD Power: 16 * 56 + 7 = 903 MOD8: 7"
[1] "911 MOD Power: 32 * 28 + 15 = 911 MOD8: 7"
[1] "919 MOD Power: 16 * 57 + 7 = 919 MOD8: 7"
[1] "927 MOD Power: 64 * 14 + 31 = 927 MOD8: 7"
[1] "935 MOD Power: 16 * 58 + 7 = 935 MOD8: 7"
[1] "943 MOD Power: 32 * 29 + 15 = 943 MOD8: 7"
[1] "951 MOD Power: 16 * 59 + 7 = 951 MOD8: 7"
[1] "959 MOD Power: 128 * 7 + 63 = 959 MOD8: 7"
[1] "967 MOD Power: 16 * 60 + 7 = 967 MOD8: 7"
[1] "975 MOD Power: 32 * 30 + 15 = 975 MOD8: 7"
[1] "983 MOD Power: 16 * 61 + 7 = 983 MOD8: 7"
[1] "991 MOD Power: 64 * 15 + 31 = 991 MOD8: 7"
[1] "999 MOD Power: 16 * 62 + 7 = 999 MOD8: 7"
[1] "1007 MOD Power: 32 * 31 + 15 = 1007 MOD8: 7"
[1] "1015 MOD Power: 16 * 63 + 7 = 1015 MOD8: 7"
[1] "1023 MOD Power: 2048 * 0 + 1023 = 1023 MOD8: 7"
[1] "1031 MOD Power: 16 * 64 + 7 = 1031 MOD8: 7"
[1] "1039 MOD Power: 32 * 32 + 15 = 1039 MOD8: 7"
[1] "1047 MOD Power: 16 * 65 + 7 = 1047 MOD8: 7"
[1] "1055 MOD Power: 64 * 16 + 31 = 1055 MOD8: 7"
[1] "1063 MOD Power: 16 * 66 + 7 = 1063 MOD8: 7"
[1] "1071 MOD Power: 32 * 33 + 15 = 1071 MOD8: 7"
[1] "1079 MOD Power: 16 * 67 + 7 = 1079 MOD8: 7"
[1] "1087 MOD Power: 128 * 8 + 63 = 1087 MOD8: 7"
[1] "1095 MOD Power: 16 * 68 + 7 = 1095 MOD8: 7"
[1] "1103 MOD Power: 32 * 34 + 15 = 1103 MOD8: 7"
[1] "1111 MOD Power: 16 * 69 + 7 = 1111 MOD8: 7"
[1] "1119 MOD Power: 64 * 17 + 31 = 1119 MOD8: 7"
[1] "1127 MOD Power: 16 * 70 + 7 = 1127 MOD8: 7"
[1] "1135 MOD Power: 32 * 35 + 15 = 1135 MOD8: 7"
[1] "1143 MOD Power: 16 * 71 + 7 = 1143 MOD8: 7"
[1] "1151 MOD Power: 256 * 4 + 127 = 1151 MOD8: 7"
[1] "1159 MOD Power: 16 * 72 + 7 = 1159 MOD8: 7"
[1] "1167 MOD Power: 32 * 36 + 15 = 1167 MOD8: 7"
[1] "1175 MOD Power: 16 * 73 + 7 = 1175 MOD8: 7"
[1] "1183 MOD Power: 64 * 18 + 31 = 1183 MOD8: 7"
[1] "1191 MOD Power: 16 * 74 + 7 = 1191 MOD8: 7"
[1] "1199 MOD Power: 32 * 37 + 15 = 1199 MOD8: 7"
[1] "1207 MOD Power: 16 * 75 + 7 = 1207 MOD8: 7"
[1] "1215 MOD Power: 128 * 9 + 63 = 1215 MOD8: 7"
[1] "1223 MOD Power: 16 * 76 + 7 = 1223 MOD8: 7"
[1] "1231 MOD Power: 32 * 38 + 15 = 1231 MOD8: 7"
[1] "1239 MOD Power: 16 * 77 + 7 = 1239 MOD8: 7"
[1] "1247 MOD Power: 64 * 19 + 31 = 1247 MOD8: 7"
[1] "1255 MOD Power: 16 * 78 + 7 = 1255 MOD8: 7"
[1] "1263 MOD Power: 32 * 39 + 15 = 1263 MOD8: 7"
[1] "1271 MOD Power: 16 * 79 + 7 = 1271 MOD8: 7"
[1] "1279 MOD Power: 512 * 2 + 255 = 1279 MOD8: 7"
[1] "1287 MOD Power: 16 * 80 + 7 = 1287 MOD8: 7"
[1] "1295 MOD Power: 32 * 40 + 15 = 1295 MOD8: 7"
[1] "1303 MOD Power: 16 * 81 + 7 = 1303 MOD8: 7"
[1] "1311 MOD Power: 64 * 20 + 31 = 1311 MOD8: 7"
[1] "1319 MOD Power: 16 * 82 + 7 = 1319 MOD8: 7"
[1] "1327 MOD Power: 32 * 41 + 15 = 1327 MOD8: 7"
[1] "1335 MOD Power: 16 * 83 + 7 = 1335 MOD8: 7"
[1] "1343 MOD Power: 128 * 10 + 63 = 1343 MOD8: 7"
[1] "1351 MOD Power: 16 * 84 + 7 = 1351 MOD8: 7"
[1] "1359 MOD Power: 32 * 42 + 15 = 1359 MOD8: 7"
[1] "1367 MOD Power: 16 * 85 + 7 = 1367 MOD8: 7"
[1] "1375 MOD Power: 64 * 21 + 31 = 1375 MOD8: 7"
[1] "1383 MOD Power: 16 * 86 + 7 = 1383 MOD8: 7"
[1] "1391 MOD Power: 32 * 43 + 15 = 1391 MOD8: 7"
[1] "1399 MOD Power: 16 * 87 + 7 = 1399 MOD8: 7"
```

```
[1] "1407 MOD Power: 256 * 5 + 127 = 1407 MOD8: 7"
[1] "1415 MOD Power: 16 * 88 + 7 = 1415 MOD8: 7"
[1] "1423 MOD Power: 32 * 44 + 15 = 1423 MOD8: 7"
[1] "1431 MOD Power: 16 * 89 + 7 = 1431 MOD8: 7"
[1] "1439 MOD Power: 64 * 22 + 31 = 1439 MOD8: 7"
[1] "1447 MOD Power: 16 * 90 + 7 = 1447 MOD8: 7"
[1] "1455 MOD Power: 32 * 45 + 15 = 1455 MOD8: 7"
[1] "1463 MOD Power: 16 * 91 + 7 = 1463 MOD8: 7"
[1] "1471 MOD Power: 128 * 11 + 63 = 1471 MOD8: 7"
[1] "1479 MOD Power: 16 * 92 + 7 = 1479 MOD8: 7"
[1] "1487 MOD Power: 32 * 46 + 15 = 1487 MOD8: 7"
[1] "1495 MOD Power: 16 * 93 + 7 = 1495 MOD8: 7"
[1] "1503 MOD Power: 64 * 23 + 31 = 1503 MOD8: 7"
[1] "1511 MOD Power: 16 * 94 + 7 = 1511 MOD8: 7"
[1] "1519 MOD Power: 32 * 47 + 15 = 1519 MOD8: 7"
[1] "1527 MOD Power: 16 * 95 + 7 = 1527 MOD8: 7"
[1] "1535 MOD Power: 1024 * 1 + 511 = 1535 MOD8: 7"
[1] "1543 MOD Power: 16 * 96 + 7 = 1543 MOD8: 7"
[1] "1551 MOD Power: 32 * 48 + 15 = 1551 MOD8: 7"
[1] "1559 MOD Power: 16 * 97 + 7 = 1559 MOD8: 7"
[1] "1567 MOD Power: 64 * 24 + 31 = 1567 MOD8: 7"
[1] "1575 MOD Power: 16 * 98 + 7 = 1575 MOD8: 7"
[1] "1583 MOD Power: 32 * 49 + 15 = 1583 MOD8: 7"
[1] "1591 MOD Power: 16 * 99 + 7 = 1591 MOD8: 7"
[1] "1599 MOD Power: 128 * 12 + 63 = 1599 MOD8: 7"
[1] "1607 MOD Power: 16 * 100 + 7 = 1607 MOD8: 7"
[1] "1615 MOD Power: 32 * 50 + 15 = 1615 MOD8: 7"
[1] "1623 MOD Power: 16 * 101 + 7 = 1623 MOD8: 7"
[1] "1631 MOD Power: 64 * 25 + 31 = 1631 MOD8: 7"
[1] "1639 MOD Power: 16 * 102 + 7 = 1639 MOD8: 7"
[1] "1647 MOD Power: 32 * 51 + 15 = 1647 MOD8: 7"
[1] "1655 MOD Power: 16 * 103 + 7 = 1655 MOD8: 7"
[1] "1663 MOD Power: 256 * 6 + 127 = 1663 MOD8: 7"
[1] "1671 MOD Power: 16 * 104 + 7 = 1671 MOD8: 7"
[1] "1679 MOD Power: 32 * 52 + 15 = 1679 MOD8: 7"
[1] "1687 MOD Power: 16 * 105 + 7 = 1687 MOD8: 7"
[1] "1695 MOD Power: 64 * 26 + 31 = 1695 MOD8: 7"
[1] "1703 MOD Power: 16 * 106 + 7 = 1703 MOD8: 7"
[1] "1711 MOD Power: 32 * 53 + 15 = 1711 MOD8: 7"
[1] "1719 MOD Power: 16 * 107 + 7 = 1719 MOD8: 7"
[1] "1727 MOD Power: 128 * 13 + 63 = 1727 MOD8: 7"
[1] "1735 MOD Power: 16 * 108 + 7 = 1735 MOD8: 7"
[1] "1743 MOD Power: 32 * 54 + 15 = 1743 MOD8: 7"
[1] "1751 MOD Power: 16 * 109 + 7 = 1751 MOD8: 7"
[1] "1759 MOD Power: 64 * 27 + 31 = 1759 MOD8: 7"
[1] "1767 MOD Power: 16 * 110 + 7 = 1767 MOD8: 7"
[1] "1775 MOD Power: 32 * 55 + 15 = 1775 MOD8: 7"
[1] "1783 MOD Power: 16 * 111 + 7 = 1783 MOD8: 7"
[1] "1791 MOD Power: 512 * 3 + 255 = 1791 MOD8: 7"
[1] "1799 MOD Power: 16 * 112 + 7 = 1799 MOD8: 7"
[1] "1807 MOD Power: 32 * 56 + 15 = 1807 MOD8: 7"
[1] "1815 MOD Power: 16 * 113 + 7 = 1815 MOD8: 7"
[1] "1823 MOD Power: 64 * 28 + 31 = 1823 MOD8: 7"
[1] "1831 MOD Power: 16 * 114 + 7 = 1831 MOD8: 7"
[1] "1839 MOD Power: 32 * 57 + 15 = 1839 MOD8: 7"
[1] "1847 MOD Power: 16 * 115 + 7 = 1847 MOD8: 7"
[1] "1855 MOD Power: 128 * 14 + 63 = 1855 MOD8: 7"
[1] "1863 MOD Power: 16 * 116 + 7 = 1863 MOD8: 7"
[1] "1871 MOD Power: 32 * 58 + 15 = 1871 MOD8: 7"
[1] "1879 MOD Power: 16 * 117 + 7 = 1879 MOD8: 7"
[1] "1887 MOD Power: 64 * 29 + 31 = 1887 MOD8: 7"
[1] "1895 MOD Power: 16 * 118 + 7 = 1895 MOD8: 7"
[1] "1903 MOD Power: 32 * 59 + 15 = 1903 MOD8: 7"
[1] "1911 MOD Power: 16 * 119 + 7 = 1911 MOD8: 7"
```

```
[1] "1919 MOD Power: 256 * 7 + 127 = 1919 MOD8: 7"
[1] "1927 MOD Power: 16 * 120 + 7 = 1927 MOD8: 7"
[1] "1935 MOD Power: 32 * 60 + 15 = 1935 MOD8: 7"
[1] "1943 MOD Power: 16 * 121 + 7 = 1943 MOD8: 7"
[1] "1951 MOD Power: 64 * 30 + 31 = 1951 MOD8: 7"
[1] "1959 MOD Power: 16 * 122 + 7 = 1959 MOD8: 7"
[1] "1967 MOD Power: 32 * 61 + 15 = 1967 MOD8: 7"
[1] "1975 MOD Power: 16 * 123 + 7 = 1975 MOD8: 7"
[1] "1983 MOD Power: 128 * 15 + 63 = 1983 MOD8: 7"
[1] "1991 MOD Power: 16 * 124 + 7 = 1991 MOD8: 7"
[1] "1999 MOD Power: 32 * 62 + 15 = 1999 MOD8: 7"
[1] "2007 MOD Power: 16 * 125 + 7 = 2007 MOD8: 7"
[1] "2015 MOD Power: 64 * 31 + 31 = 2015 MOD8: 7"
[1] "2023 MOD Power: 16 * 126 + 7 = 2023 MOD8: 7"
[1] "2031 MOD Power: 32 * 63 + 15 = 2031 MOD8: 7"
[1] "2039 MOD Power: 16 * 127 + 7 = 2039 MOD8: 7"
[1] "2047 MOD Power: 4096 * 0 + 2047 = 2047 MOD8: 7"
[1] "2055 MOD Power: 16 * 128 + 7 = 2055 MOD8: 7"
[1] "2063 MOD Power: 32 * 64 + 15 = 2063 MOD8: 7"
[1] "2071 MOD Power: 16 * 129 + 7 = 2071 MOD8: 7"
[1] "2079 MOD Power: 64 * 32 + 31 = 2079 MOD8: 7"
[1] "2087 MOD Power: 16 * 130 + 7 = 2087 MOD8: 7"
[1] "2095 MOD Power: 32 * 65 + 15 = 2095 MOD8: 7"
[1] "2103 MOD Power: 16 * 131 + 7 = 2103 MOD8: 7"
[1] "2111 MOD Power: 128 * 16 + 63 = 2111 MOD8: 7"
[1] "2119 MOD Power: 16 * 132 + 7 = 2119 MOD8: 7"
[1] "2127 MOD Power: 32 * 66 + 15 = 2127 MOD8: 7"
[1] "2135 MOD Power: 16 * 133 + 7 = 2135 MOD8: 7"
[1] "2143 MOD Power: 64 * 33 + 31 = 2143 MOD8: 7"
[1] "2151 MOD Power: 16 * 134 + 7 = 2151 MOD8: 7"
[1] "2159 MOD Power: 32 * 67 + 15 = 2159 MOD8: 7"
[1] "2167 MOD Power: 16 * 135 + 7 = 2167 MOD8: 7"
[1] "2175 MOD Power: 256 * 8 + 127 = 2175 MOD8: 7"
[1] "2183 MOD Power: 16 * 136 + 7 = 2183 MOD8: 7"
[1] "2191 MOD Power: 32 * 68 + 15 = 2191 MOD8: 7"
[1] "2199 MOD Power: 16 * 137 + 7 = 2199 MOD8: 7"
[1] "2207 MOD Power: 64 * 34 + 31 = 2207 MOD8: 7"
[1] "2215 MOD Power: 16 * 138 + 7 = 2215 MOD8: 7"
[1] "2223 MOD Power: 32 * 69 + 15 = 2223 MOD8: 7"
[1] "2231 MOD Power: 16 * 139 + 7 = 2231 MOD8: 7"
[1] "2239 MOD Power: 128 * 17 + 63 = 2239 MOD8: 7"
[1] "2247 MOD Power: 16 * 140 + 7 = 2247 MOD8: 7"
[1] "2255 MOD Power: 32 * 70 + 15 = 2255 MOD8: 7"
[1] "2263 MOD Power: 16 * 141 + 7 = 2263 MOD8: 7"
[1] "2271 MOD Power: 64 * 35 + 31 = 2271 MOD8: 7"
[1] "2279 MOD Power: 16 * 142 + 7 = 2279 MOD8: 7"
[1] "2287 MOD Power: 32 * 71 + 15 = 2287 MOD8: 7"
[1] "2295 MOD Power: 16 * 143 + 7 = 2295 MOD8: 7"
[1] "2303 MOD Power: 512 * 4 + 255 = 2303 MOD8: 7"
[1] "2311 MOD Power: 16 * 144 + 7 = 2311 MOD8: 7"
[1] "2319 MOD Power: 32 * 72 + 15 = 2319 MOD8: 7"
[1] "2327 MOD Power: 16 * 145 + 7 = 2327 MOD8: 7"
[1] "2335 MOD Power: 64 * 36 + 31 = 2335 MOD8: 7"
[1] "2343 MOD Power: 16 * 146 + 7 = 2343 MOD8: 7"
[1] "2351 MOD Power: 32 * 73 + 15 = 2351 MOD8: 7"
[1] "2359 MOD Power: 16 * 147 + 7 = 2359 MOD8: 7"
[1] "2367 MOD Power: 128 * 18 + 63 = 2367 MOD8: 7"
[1] "2375 MOD Power: 16 * 148 + 7 = 2375 MOD8: 7"
[1] "2383 MOD Power: 32 * 74 + 15 = 2383 MOD8: 7"
[1] "2391 MOD Power: 16 * 149 + 7 = 2391 MOD8: 7"
[1] "2399 MOD Power: 64 * 37 + 31 = 2399 MOD8: 7"
[1] "2407 MOD Power: 16 * 150 + 7 = 2407 MOD8: 7"
[1] "2415 MOD Power: 32 * 75 + 15 = 2415 MOD8: 7"
[1] "2423 MOD Power: 16 * 151 + 7 = 2423 MOD8: 7"
```

```
[1] "2431 MOD Power: 256 * 9 + 127 = 2431 MOD8: 7"
[1] "2439 MOD Power: 16 * 152 + 7 = 2439 MOD8: 7"
[1] "2447 MOD Power: 32 * 76 + 15 = 2447 MOD8: 7"
[1] "2455 MOD Power: 16 * 153 + 7 = 2455 MOD8: 7"
[1] "2463 MOD Power: 64 * 38 + 31 = 2463 MOD8: 7"
[1] "2471 MOD Power: 16 * 154 + 7 = 2471 MOD8: 7"
[1] "2479 MOD Power: 32 * 77 + 15 = 2479 MOD8: 7"
[1] "2487 MOD Power: 16 * 155 + 7 = 2487 MOD8: 7"
[1] "2495 MOD Power: 128 * 19 + 63 = 2495 MOD8: 7"
[1] "2503 MOD Power: 16 * 156 + 7 = 2503 MOD8: 7"
[1] "2511 MOD Power: 32 * 78 + 15 = 2511 MOD8: 7"
[1] "2519 MOD Power: 16 * 157 + 7 = 2519 MOD8: 7"
[1] "2527 MOD Power: 64 * 39 + 31 = 2527 MOD8: 7"
[1] "2535 MOD Power: 16 * 158 + 7 = 2535 MOD8: 7"
[1] "2543 MOD Power: 32 * 79 + 15 = 2543 MOD8: 7"
[1] "2551 MOD Power: 16 * 159 + 7 = 2551 MOD8: 7"
[1] "2559 MOD Power: 1024 * 2 + 511 = 2559 MOD8: 7"
[1] "2567 MOD Power: 16 * 160 + 7 = 2567 MOD8: 7"
[1] "2575 MOD Power: 32 * 80 + 15 = 2575 MOD8: 7"
[1] "2583 MOD Power: 16 * 161 + 7 = 2583 MOD8: 7"
[1] "2591 MOD Power: 64 * 40 + 31 = 2591 MOD8: 7"
[1] "2599 MOD Power: 16 * 162 + 7 = 2599 MOD8: 7"
[1] "2607 MOD Power: 32 * 81 + 15 = 2607 MOD8: 7"
[1] "2615 MOD Power: 16 * 163 + 7 = 2615 MOD8: 7"
[1] "2623 MOD Power: 128 * 20 + 63 = 2623 MOD8: 7"
[1] "2631 MOD Power: 16 * 164 + 7 = 2631 MOD8: 7"
[1] "2639 MOD Power: 32 * 82 + 15 = 2639 MOD8: 7"
[1] "2647 MOD Power: 16 * 165 + 7 = 2647 MOD8: 7"
[1] "2655 MOD Power: 64 * 41 + 31 = 2655 MOD8: 7"
[1] "2663 MOD Power: 16 * 166 + 7 = 2663 MOD8: 7"
[1] "2671 MOD Power: 32 * 83 + 15 = 2671 MOD8: 7"
[1] "2679 MOD Power: 16 * 167 + 7 = 2679 MOD8: 7"
[1] "2687 MOD Power: 256 * 10 + 127 = 2687 MOD8: 7"
[1] "2695 MOD Power: 16 * 168 + 7 = 2695 MOD8: 7"
[1] "2703 MOD Power: 32 * 84 + 15 = 2703 MOD8: 7"
[1] "2711 MOD Power: 16 * 169 + 7 = 2711 MOD8: 7"
[1] "2719 MOD Power: 64 * 42 + 31 = 2719 MOD8: 7"
[1] "2727 MOD Power: 16 * 170 + 7 = 2727 MOD8: 7"
[1] "2735 MOD Power: 32 * 85 + 15 = 2735 MOD8: 7"
[1] "2743 MOD Power: 16 * 171 + 7 = 2743 MOD8: 7"
[1] "2751 MOD Power: 128 * 21 + 63 = 2751 MOD8: 7"
[1] "2759 MOD Power: 16 * 172 + 7 = 2759 MOD8: 7"
[1] "2767 MOD Power: 32 * 86 + 15 = 2767 MOD8: 7"
[1] "2775 MOD Power: 16 * 173 + 7 = 2775 MOD8: 7"
[1] "2783 MOD Power: 64 * 43 + 31 = 2783 MOD8: 7"
[1] "2791 MOD Power: 16 * 174 + 7 = 2791 MOD8: 7"
[1] "2799 MOD Power: 32 * 87 + 15 = 2799 MOD8: 7"
[1] "2807 MOD Power: 16 * 175 + 7 = 2807 MOD8: 7"
[1] "2815 MOD Power: 512 * 5 + 255 = 2815 MOD8: 7"
[1] "2823 MOD Power: 16 * 176 + 7 = 2823 MOD8: 7"
[1] "2831 MOD Power: 32 * 88 + 15 = 2831 MOD8: 7"
[1] "2839 MOD Power: 16 * 177 + 7 = 2839 MOD8: 7"
[1] "2847 MOD Power: 64 * 44 + 31 = 2847 MOD8: 7"
[1] "2855 MOD Power: 16 * 178 + 7 = 2855 MOD8: 7"
[1] "2863 MOD Power: 32 * 89 + 15 = 2863 MOD8: 7"
[1] "2871 MOD Power: 16 * 179 + 7 = 2871 MOD8: 7"
[1] "2879 MOD Power: 128 * 22 + 63 = 2879 MOD8: 7"
[1] "2887 MOD Power: 16 * 180 + 7 = 2887 MOD8: 7"
[1] "2895 MOD Power: 32 * 90 + 15 = 2895 MOD8: 7"
[1] "2903 MOD Power: 16 * 181 + 7 = 2903 MOD8: 7"
[1] "2911 MOD Power: 64 * 45 + 31 = 2911 MOD8: 7"
[1] "2919 MOD Power: 16 * 182 + 7 = 2919 MOD8: 7"
[1] "2927 MOD Power: 32 * 91 + 15 = 2927 MOD8: 7"
[1] "2935 MOD Power: 16 * 183 + 7 = 2935 MOD8: 7"
```

```
[1] "2943 MOD Power: 256 * 11 + 127 = 2943 MOD8: 7"
[1] "2951 MOD Power: 16 * 184 + 7 = 2951 MOD8: 7"
[1] "2959 MOD Power: 32 * 92 + 15 = 2959 MOD8: 7"
[1] "2967 MOD Power: 16 * 185 + 7 = 2967 MOD8: 7"
[1] "2975 MOD Power: 64 * 46 + 31 = 2975 MOD8: 7"
[1] "2983 MOD Power: 16 * 186 + 7 = 2983 MOD8: 7"
[1] "2991 MOD Power: 32 * 93 + 15 = 2991 MOD8: 7"
[1] "2999 MOD Power: 16 * 187 + 7 = 2999 MOD8: 7"
[1] "3007 MOD Power: 128 * 23 + 63 = 3007 MOD8: 7"
[1] "3015 MOD Power: 16 * 188 + 7 = 3015 MOD8: 7"
[1] "3023 MOD Power: 32 * 94 + 15 = 3023 MOD8: 7"
[1] "3031 MOD Power: 16 * 189 + 7 = 3031 MOD8: 7"
[1] "3039 MOD Power: 64 * 47 + 31 = 3039 MOD8: 7"
[1] "3047 MOD Power: 16 * 190 + 7 = 3047 MOD8: 7"
[1] "3055 MOD Power: 32 * 95 + 15 = 3055 MOD8: 7"
[1] "3063 MOD Power: 16 * 191 + 7 = 3063 MOD8: 7"
[1] "3071 MOD Power: 2048 * 1 + 1023 = 3071 MOD8: 7"
[1] "3079 MOD Power: 16 * 192 + 7 = 3079 MOD8: 7"
[1] "3087 MOD Power: 32 * 96 + 15 = 3087 MOD8: 7"
[1] "3095 MOD Power: 16 * 193 + 7 = 3095 MOD8: 7"
[1] "3103 MOD Power: 64 * 48 + 31 = 3103 MOD8: 7"
[1] "3111 MOD Power: 16 * 194 + 7 = 3111 MOD8: 7"
[1] "3119 MOD Power: 32 * 97 + 15 = 3119 MOD8: 7"
[1] "3127 MOD Power: 16 * 195 + 7 = 3127 MOD8: 7"
[1] "3135 MOD Power: 128 * 24 + 63 = 3135 MOD8: 7"
[1] "3143 MOD Power: 16 * 196 + 7 = 3143 MOD8: 7"
[1] "3151 MOD Power: 32 * 98 + 15 = 3151 MOD8: 7"
[1] "3159 MOD Power: 16 * 197 + 7 = 3159 MOD8: 7"
[1] "3167 MOD Power: 64 * 49 + 31 = 3167 MOD8: 7"
[1] "3175 MOD Power: 16 * 198 + 7 = 3175 MOD8: 7"
[1] "3183 MOD Power: 32 * 99 + 15 = 3183 MOD8: 7"
[1] "3191 MOD Power: 16 * 199 + 7 = 3191 MOD8: 7"
[1] "3199 MOD Power: 256 * 12 + 127 = 3199 MOD8: 7"
[1] "3207 MOD Power: 16 * 200 + 7 = 3207 MOD8: 7"
[1] "3215 MOD Power: 32 * 100 + 15 = 3215 MOD8: 7"
[1] "3223 MOD Power: 16 * 201 + 7 = 3223 MOD8: 7"
[1] "3231 MOD Power: 64 * 50 + 31 = 3231 MOD8: 7"
[1] "3239 MOD Power: 16 * 202 + 7 = 3239 MOD8: 7"
[1] "3247 MOD Power: 32 * 101 + 15 = 3247 MOD8: 7"
[1] "3255 MOD Power: 16 * 203 + 7 = 3255 MOD8: 7"
[1] "3263 MOD Power: 128 * 25 + 63 = 3263 MOD8: 7"
[1] "3271 MOD Power: 16 * 204 + 7 = 3271 MOD8: 7"
[1] "3279 MOD Power: 32 * 102 + 15 = 3279 MOD8: 7"
[1] "3287 MOD Power: 16 * 205 + 7 = 3287 MOD8: 7"
[1] "3295 MOD Power: 64 * 51 + 31 = 3295 MOD8: 7"
[1] "3303 MOD Power: 16 * 206 + 7 = 3303 MOD8: 7"
[1] "3311 MOD Power: 32 * 103 + 15 = 3311 MOD8: 7"
[1] "3319 MOD Power: 16 * 207 + 7 = 3319 MOD8: 7"
[1] "3327 MOD Power: 512 * 6 + 255 = 3327 MOD8: 7"
[1] "3335 MOD Power: 16 * 208 + 7 = 3335 MOD8: 7"
[1] "3343 MOD Power: 32 * 104 + 15 = 3343 MOD8: 7"
[1] "3351 MOD Power: 16 * 209 + 7 = 3351 MOD8: 7"
[1] "3359 MOD Power: 64 * 52 + 31 = 3359 MOD8: 7"
[1] "3367 MOD Power: 16 * 210 + 7 = 3367 MOD8: 7"
[1] "3375 MOD Power: 32 * 105 + 15 = 3375 MOD8: 7"
[1] "3383 MOD Power: 16 * 211 + 7 = 3383 MOD8: 7"
[1] "3391 MOD Power: 128 * 26 + 63 = 3391 MOD8: 7"
[1] "3399 MOD Power: 16 * 212 + 7 = 3399 MOD8: 7"
[1] "3407 MOD Power: 32 * 106 + 15 = 3407 MOD8: 7"
[1] "3415 MOD Power: 16 * 213 + 7 = 3415 MOD8: 7"
[1] "3423 MOD Power: 64 * 53 + 31 = 3423 MOD8: 7"
[1] "3431 MOD Power: 16 * 214 + 7 = 3431 MOD8: 7"
[1] "3439 MOD Power: 32 * 107 + 15 = 3439 MOD8: 7"
[1] "3447 MOD Power: 16 * 215 + 7 = 3447 MOD8: 7"
```

```
[1] "3455 MOD Power: 256 * 13 + 127 = 3455 MOD8: 7"
[1] "3463 MOD Power: 16 * 216 + 7 = 3463 MOD8: 7"
[1] "3471 MOD Power: 32 * 108 + 15 = 3471 MOD8: 7"
[1] "3479 MOD Power: 16 * 217 + 7 = 3479 MOD8: 7"
[1] "3487 MOD Power: 64 * 54 + 31 = 3487 MOD8: 7"
[1] "3495 MOD Power: 16 * 218 + 7 = 3495 MOD8: 7"
[1] "3503 MOD Power: 32 * 109 + 15 = 3503 MOD8: 7"
[1] "3511 MOD Power: 16 * 219 + 7 = 3511 MOD8: 7"
[1] "3519 MOD Power: 128 * 27 + 63 = 3519 MOD8: 7"
[1] "3527 MOD Power: 16 * 220 + 7 = 3527 MOD8: 7"
[1] "3535 MOD Power: 32 * 110 + 15 = 3535 MOD8: 7"
[1] "3543 MOD Power: 16 * 221 + 7 = 3543 MOD8: 7"
[1] "3551 MOD Power: 64 * 55 + 31 = 3551 MOD8: 7"
[1] "3559 MOD Power: 16 * 222 + 7 = 3559 MOD8: 7"
[1] "3567 MOD Power: 32 * 111 + 15 = 3567 MOD8: 7"
[1] "3575 MOD Power: 16 * 223 + 7 = 3575 MOD8: 7"
[1] "3583 MOD Power: 1024 * 3 + 511 = 3583 MOD8: 7"
[1] "3591 MOD Power: 16 * 224 + 7 = 3591 MOD8: 7"
[1] "3599 MOD Power: 32 * 112 + 15 = 3599 MOD8: 7"
[1] "3607 MOD Power: 16 * 225 + 7 = 3607 MOD8: 7"
[1] "3615 MOD Power: 64 * 56 + 31 = 3615 MOD8: 7"
[1] "3623 MOD Power: 16 * 226 + 7 = 3623 MOD8: 7"
[1] "3631 MOD Power: 32 * 113 + 15 = 3631 MOD8: 7"
[1] "3639 MOD Power: 16 * 227 + 7 = 3639 MOD8: 7"
[1] "3647 MOD Power: 128 * 28 + 63 = 3647 MOD8: 7"
[1] "3655 MOD Power: 16 * 228 + 7 = 3655 MOD8: 7"
[1] "3663 MOD Power: 32 * 114 + 15 = 3663 MOD8: 7"
[1] "3671 MOD Power: 16 * 229 + 7 = 3671 MOD8: 7"
[1] "3679 MOD Power: 64 * 57 + 31 = 3679 MOD8: 7"
[1] "3687 MOD Power: 16 * 230 + 7 = 3687 MOD8: 7"
[1] "3695 MOD Power: 32 * 115 + 15 = 3695 MOD8: 7"
[1] "3703 MOD Power: 16 * 231 + 7 = 3703 MOD8: 7"
[1] "3711 MOD Power: 256 * 14 + 127 = 3711 MOD8: 7"
[1] "3719 MOD Power: 16 * 232 + 7 = 3719 MOD8: 7"
[1] "3727 MOD Power: 32 * 116 + 15 = 3727 MOD8: 7"
[1] "3735 MOD Power: 16 * 233 + 7 = 3735 MOD8: 7"
[1] "3743 MOD Power: 64 * 58 + 31 = 3743 MOD8: 7"
[1] "3751 MOD Power: 16 * 234 + 7 = 3751 MOD8: 7"
[1] "3759 MOD Power: 32 * 117 + 15 = 3759 MOD8: 7"
[1] "3767 MOD Power: 16 * 235 + 7 = 3767 MOD8: 7"
[1] "3775 MOD Power: 128 * 29 + 63 = 3775 MOD8: 7"
[1] "3783 MOD Power: 16 * 236 + 7 = 3783 MOD8: 7"
[1] "3791 MOD Power: 32 * 118 + 15 = 3791 MOD8: 7"
[1] "3799 MOD Power: 16 * 237 + 7 = 3799 MOD8: 7"
[1] "3807 MOD Power: 64 * 59 + 31 = 3807 MOD8: 7"
[1] "3815 MOD Power: 16 * 238 + 7 = 3815 MOD8: 7"
[1] "3823 MOD Power: 32 * 119 + 15 = 3823 MOD8: 7"
[1] "3831 MOD Power: 16 * 239 + 7 = 3831 MOD8: 7"
[1] "3839 MOD Power: 512 * 7 + 255 = 3839 MOD8: 7"
[1] "3847 MOD Power: 16 * 240 + 7 = 3847 MOD8: 7"
[1] "3855 MOD Power: 32 * 120 + 15 = 3855 MOD8: 7"
[1] "3863 MOD Power: 16 * 241 + 7 = 3863 MOD8: 7"
[1] "3871 MOD Power: 64 * 60 + 31 = 3871 MOD8: 7"
[1] "3879 MOD Power: 16 * 242 + 7 = 3879 MOD8: 7"
[1] "3887 MOD Power: 32 * 121 + 15 = 3887 MOD8: 7"
[1] "3895 MOD Power: 16 * 243 + 7 = 3895 MOD8: 7"
[1] "3903 MOD Power: 128 * 30 + 63 = 3903 MOD8: 7"
[1] "3911 MOD Power: 16 * 244 + 7 = 3911 MOD8: 7"
[1] "3919 MOD Power: 32 * 122 + 15 = 3919 MOD8: 7"
[1] "3927 MOD Power: 16 * 245 + 7 = 3927 MOD8: 7"
[1] "3935 MOD Power: 64 * 61 + 31 = 3935 MOD8: 7"
[1] "3943 MOD Power: 16 * 246 + 7 = 3943 MOD8: 7"
[1] "3951 MOD Power: 32 * 123 + 15 = 3951 MOD8: 7"
[1] "3959 MOD Power: 16 * 247 + 7 = 3959 MOD8: 7"
```

```
[1] "3967 MOD Power: 256 * 15 + 127 = 3967 MOD8: 7"
[1] "3975 MOD Power: 16 * 248 + 7 = 3975 MOD8: 7"
[1] "3983 MOD Power: 32 * 124 + 15 = 3983 MOD8: 7"
[1] "3991 MOD Power: 16 * 249 + 7 = 3991 MOD8: 7"
[1] "3999 MOD Power: 64 * 62 + 31 = 3999 MOD8: 7"
[1] "4007 MOD Power: 16 * 250 + 7 = 4007 MOD8: 7"
[1] "4015 MOD Power: 32 * 125 + 15 = 4015 MOD8: 7"
[1] "4023 MOD Power: 16 * 251 + 7 = 4023 MOD8: 7"
[1] "4031 MOD Power: 128 * 31 + 63 = 4031 MOD8: 7"
[1] "4039 MOD Power: 16 * 252 + 7 = 4039 MOD8: 7"
[1] "4047 MOD Power: 32 * 126 + 15 = 4047 MOD8: 7"
[1] "4055 MOD Power: 16 * 253 + 7 = 4055 MOD8: 7"
[1] "4063 MOD Power: 64 * 63 + 31 = 4063 MOD8: 7"
[1] "4071 MOD Power: 16 * 254 + 7 = 4071 MOD8: 7"
[1] "4079 MOD Power: 32 * 127 + 15 = 4079 MOD8: 7"
[1] "4087 MOD Power: 16 * 255 + 7 = 4087 MOD8: 7"
[1] "4095 MOD Power: 8192 * 0 + 4095 = 4095 MOD8: 7"
[1] "4103 MOD Power: 16 * 256 + 7 = 4103 MOD8: 7"
[1] "4111 MOD Power: 32 * 128 + 15 = 4111 MOD8: 7"
[1] "4119 MOD Power: 16 * 257 + 7 = 4119 MOD8: 7"
[1] "4127 MOD Power: 64 * 64 + 31 = 4127 MOD8: 7"
[1] "4135 MOD Power: 16 * 258 + 7 = 4135 MOD8: 7"
[1] "4143 MOD Power: 32 * 129 + 15 = 4143 MOD8: 7"
[1] "4151 MOD Power: 16 * 259 + 7 = 4151 MOD8: 7"
[1] "4159 MOD Power: 128 * 32 + 63 = 4159 MOD8: 7"
[1] "4167 MOD Power: 16 * 260 + 7 = 4167 MOD8: 7"
[1] "4175 MOD Power: 32 * 130 + 15 = 4175 MOD8: 7"
[1] "4183 MOD Power: 16 * 261 + 7 = 4183 MOD8: 7"
[1] "4191 MOD Power: 64 * 65 + 31 = 4191 MOD8: 7"
[1] "4199 MOD Power: 16 * 262 + 7 = 4199 MOD8: 7"
[1] "4207 MOD Power: 32 * 131 + 15 = 4207 MOD8: 7"
[1] "4215 MOD Power: 16 * 263 + 7 = 4215 MOD8: 7"
[1] "4223 MOD Power: 256 * 16 + 127 = 4223 MOD8: 7"
[1] "4231 MOD Power: 16 * 264 + 7 = 4231 MOD8: 7"
[1] "4239 MOD Power: 32 * 132 + 15 = 4239 MOD8: 7"
[1] "4247 MOD Power: 16 * 265 + 7 = 4247 MOD8: 7"
[1] "4255 MOD Power: 64 * 66 + 31 = 4255 MOD8: 7"
[1] "4263 MOD Power: 16 * 266 + 7 = 4263 MOD8: 7"
[1] "4271 MOD Power: 32 * 133 + 15 = 4271 MOD8: 7"
[1] "4279 MOD Power: 16 * 267 + 7 = 4279 MOD8: 7"
[1] "4287 MOD Power: 128 * 33 + 63 = 4287 MOD8: 7"
[1] "4295 MOD Power: 16 * 268 + 7 = 4295 MOD8: 7"
[1] "4303 MOD Power: 32 * 134 + 15 = 4303 MOD8: 7"
[1] "4311 MOD Power: 16 * 269 + 7 = 4311 MOD8: 7"
[1] "4319 MOD Power: 64 * 67 + 31 = 4319 MOD8: 7"
[1] "4327 MOD Power: 16 * 270 + 7 = 4327 MOD8: 7"
[1] "4335 MOD Power: 32 * 135 + 15 = 4335 MOD8: 7"
[1] "4343 MOD Power: 16 * 271 + 7 = 4343 MOD8: 7"
[1] "4351 MOD Power: 512 * 8 + 255 = 4351 MOD8: 7"
[1] "4359 MOD Power: 16 * 272 + 7 = 4359 MOD8: 7"
[1] "4367 MOD Power: 32 * 136 + 15 = 4367 MOD8: 7"
[1] "4375 MOD Power: 16 * 273 + 7 = 4375 MOD8: 7"
[1] "4383 MOD Power: 64 * 68 + 31 = 4383 MOD8: 7"
[1] "4391 MOD Power: 16 * 274 + 7 = 4391 MOD8: 7"
[1] "4399 MOD Power: 32 * 137 + 15 = 4399 MOD8: 7"
[1] "4407 MOD Power: 16 * 275 + 7 = 4407 MOD8: 7"
[1] "4415 MOD Power: 128 * 34 + 63 = 4415 MOD8: 7"
[1] "4423 MOD Power: 16 * 276 + 7 = 4423 MOD8: 7"
[1] "4431 MOD Power: 32 * 138 + 15 = 4431 MOD8: 7"
[1] "4439 MOD Power: 16 * 277 + 7 = 4439 MOD8: 7"
[1] "4447 MOD Power: 64 * 69 + 31 = 4447 MOD8: 7"
[1] "4455 MOD Power: 16 * 278 + 7 = 4455 MOD8: 7"
[1] "4463 MOD Power: 32 * 139 + 15 = 4463 MOD8: 7"
[1] "4471 MOD Power: 16 * 279 + 7 = 4471 MOD8: 7"
```

```
[1] "4479 MOD Power: 256 * 17 + 127 = 4479 MOD8: 7"
[1] "4487 MOD Power: 16 * 280 + 7 = 4487 MOD8: 7"
[1] "4495 MOD Power: 32 * 140 + 15 = 4495 MOD8: 7"
[1] "4503 MOD Power: 16 * 281 + 7 = 4503 MOD8: 7"
[1] "4511 MOD Power: 64 * 70 + 31 = 4511 MOD8: 7"
[1] "4519 MOD Power: 16 * 282 + 7 = 4519 MOD8: 7"
[1] "4527 MOD Power: 32 * 141 + 15 = 4527 MOD8: 7"
[1] "4535 MOD Power: 16 * 283 + 7 = 4535 MOD8: 7"
[1] "4543 MOD Power: 128 * 35 + 63 = 4543 MOD8: 7"
[1] "4551 MOD Power: 16 * 284 + 7 = 4551 MOD8: 7"
[1] "4559 MOD Power: 32 * 142 + 15 = 4559 MOD8: 7"
[1] "4567 MOD Power: 16 * 285 + 7 = 4567 MOD8: 7"
[1] "4575 MOD Power: 64 * 71 + 31 = 4575 MOD8: 7"
[1] "4583 MOD Power: 16 * 286 + 7 = 4583 MOD8: 7"
[1] "4591 MOD Power: 32 * 143 + 15 = 4591 MOD8: 7"
[1] "4599 MOD Power: 16 * 287 + 7 = 4599 MOD8: 7"
[1] "4607 MOD Power: 1024 * 4 + 511 = 4607 MOD8: 7"
[1] "4615 MOD Power: 16 * 288 + 7 = 4615 MOD8: 7"
[1] "4623 MOD Power: 32 * 144 + 15 = 4623 MOD8: 7"
[1] "4631 MOD Power: 16 * 289 + 7 = 4631 MOD8: 7"
[1] "4639 MOD Power: 64 * 72 + 31 = 4639 MOD8: 7"
[1] "4647 MOD Power: 16 * 290 + 7 = 4647 MOD8: 7"
[1] "4655 MOD Power: 32 * 145 + 15 = 4655 MOD8: 7"
[1] "4663 MOD Power: 16 * 291 + 7 = 4663 MOD8: 7"
[1] "4671 MOD Power: 128 * 36 + 63 = 4671 MOD8: 7"
[1] "4679 MOD Power: 16 * 292 + 7 = 4679 MOD8: 7"
[1] "4687 MOD Power: 32 * 146 + 15 = 4687 MOD8: 7"
[1] "4695 MOD Power: 16 * 293 + 7 = 4695 MOD8: 7"
[1] "4703 MOD Power: 64 * 73 + 31 = 4703 MOD8: 7"
[1] "4711 MOD Power: 16 * 294 + 7 = 4711 MOD8: 7"
[1] "4719 MOD Power: 32 * 147 + 15 = 4719 MOD8: 7"
[1] "4727 MOD Power: 16 * 295 + 7 = 4727 MOD8: 7"
[1] "4735 MOD Power: 256 * 18 + 127 = 4735 MOD8: 7"
[1] "4743 MOD Power: 16 * 296 + 7 = 4743 MOD8: 7"
[1] "4751 MOD Power: 32 * 148 + 15 = 4751 MOD8: 7"
[1] "4759 MOD Power: 16 * 297 + 7 = 4759 MOD8: 7"
[1] "4767 MOD Power: 64 * 74 + 31 = 4767 MOD8: 7"
[1] "4775 MOD Power: 16 * 298 + 7 = 4775 MOD8: 7"
[1] "4783 MOD Power: 32 * 149 + 15 = 4783 MOD8: 7"
[1] "4791 MOD Power: 16 * 299 + 7 = 4791 MOD8: 7"
[1] "4799 MOD Power: 128 * 37 + 63 = 4799 MOD8: 7"
[1] "4807 MOD Power: 16 * 300 + 7 = 4807 MOD8: 7"
[1] "4815 MOD Power: 32 * 150 + 15 = 4815 MOD8: 7"
[1] "4823 MOD Power: 16 * 301 + 7 = 4823 MOD8: 7"
[1] "4831 MOD Power: 64 * 75 + 31 = 4831 MOD8: 7"
[1] "4839 MOD Power: 16 * 302 + 7 = 4839 MOD8: 7"
[1] "4847 MOD Power: 32 * 151 + 15 = 4847 MOD8: 7"
[1] "4855 MOD Power: 16 * 303 + 7 = 4855 MOD8: 7"
[1] "4863 MOD Power: 512 * 9 + 255 = 4863 MOD8: 7"
[1] "4871 MOD Power: 16 * 304 + 7 = 4871 MOD8: 7"
[1] "4879 MOD Power: 32 * 152 + 15 = 4879 MOD8: 7"
[1] "4887 MOD Power: 16 * 305 + 7 = 4887 MOD8: 7"
[1] "4895 MOD Power: 64 * 76 + 31 = 4895 MOD8: 7"
[1] "4903 MOD Power: 16 * 306 + 7 = 4903 MOD8: 7"
[1] "4911 MOD Power: 32 * 153 + 15 = 4911 MOD8: 7"
[1] "4919 MOD Power: 16 * 307 + 7 = 4919 MOD8: 7"
[1] "4927 MOD Power: 128 * 38 + 63 = 4927 MOD8: 7"
[1] "4935 MOD Power: 16 * 308 + 7 = 4935 MOD8: 7"
[1] "4943 MOD Power: 32 * 154 + 15 = 4943 MOD8: 7"
[1] "4951 MOD Power: 16 * 309 + 7 = 4951 MOD8: 7"
[1] "4959 MOD Power: 64 * 77 + 31 = 4959 MOD8: 7"
[1] "4967 MOD Power: 16 * 310 + 7 = 4967 MOD8: 7"
[1] "4975 MOD Power: 32 * 155 + 15 = 4975 MOD8: 7"
[1] "4983 MOD Power: 16 * 311 + 7 = 4983 MOD8: 7"
```

```
[1] "4991 MOD Power: 256 * 19 + 127 = 4991 MOD8: 7"
[1] "4999 MOD Power: 16 * 312 + 7 = 4999 MOD8: 7"
[1] "5007 MOD Power: 32 * 156 + 15 = 5007 MOD8: 7"
[1] "5015 MOD Power: 16 * 313 + 7 = 5015 MOD8: 7"
[1] "5023 MOD Power: 64 * 78 + 31 = 5023 MOD8: 7"
[1] "5031 MOD Power: 16 * 314 + 7 = 5031 MOD8: 7"
[1] "5039 MOD Power: 32 * 157 + 15 = 5039 MOD8: 7"
[1] "5047 MOD Power: 16 * 315 + 7 = 5047 MOD8: 7"
[1] "5055 MOD Power: 128 * 39 + 63 = 5055 MOD8: 7"
[1] "5063 MOD Power: 16 * 316 + 7 = 5063 MOD8: 7"
[1] "5071 MOD Power: 32 * 158 + 15 = 5071 MOD8: 7"
[1] "5079 MOD Power: 16 * 317 + 7 = 5079 MOD8: 7"
[1] "5087 MOD Power: 64 * 79 + 31 = 5087 MOD8: 7"
[1] "5095 MOD Power: 16 * 318 + 7 = 5095 MOD8: 7"
[1] "5103 MOD Power: 32 * 159 + 15 = 5103 MOD8: 7"
[1] "5111 MOD Power: 16 * 319 + 7 = 5111 MOD8: 7"
[1] "5119 MOD Power: 2048 * 2 + 1023 = 5119 MOD8: 7"
[1] "5127 MOD Power: 16 * 320 + 7 = 5127 MOD8: 7"
[1] "5135 MOD Power: 32 * 160 + 15 = 5135 MOD8: 7"
[1] "5143 MOD Power: 16 * 321 + 7 = 5143 MOD8: 7"
[1] "5151 MOD Power: 64 * 80 + 31 = 5151 MOD8: 7"
[1] "5159 MOD Power: 16 * 322 + 7 = 5159 MOD8: 7"
[1] "5167 MOD Power: 32 * 161 + 15 = 5167 MOD8: 7"
[1] "5175 MOD Power: 16 * 323 + 7 = 5175 MOD8: 7"
[1] "5183 MOD Power: 128 * 40 + 63 = 5183 MOD8: 7"
[1] "5191 MOD Power: 16 * 324 + 7 = 5191 MOD8: 7"
[1] "5199 MOD Power: 32 * 162 + 15 = 5199 MOD8: 7"
[1] "5207 MOD Power: 16 * 325 + 7 = 5207 MOD8: 7"
[1] "5215 MOD Power: 64 * 81 + 31 = 5215 MOD8: 7"
[1] "5223 MOD Power: 16 * 326 + 7 = 5223 MOD8: 7"
[1] "5231 MOD Power: 32 * 163 + 15 = 5231 MOD8: 7"
[1] "5239 MOD Power: 16 * 327 + 7 = 5239 MOD8: 7"
[1] "5247 MOD Power: 256 * 20 + 127 = 5247 MOD8: 7"
[1] "5255 MOD Power: 16 * 328 + 7 = 5255 MOD8: 7"
[1] "5263 MOD Power: 32 * 164 + 15 = 5263 MOD8: 7"
[1] "5271 MOD Power: 16 * 329 + 7 = 5271 MOD8: 7"
[1] "5279 MOD Power: 64 * 82 + 31 = 5279 MOD8: 7"
[1] "5287 MOD Power: 16 * 330 + 7 = 5287 MOD8: 7"
[1] "5295 MOD Power: 32 * 165 + 15 = 5295 MOD8: 7"
[1] "5303 MOD Power: 16 * 331 + 7 = 5303 MOD8: 7"
[1] "5311 MOD Power: 128 * 41 + 63 = 5311 MOD8: 7"
[1] "5319 MOD Power: 16 * 332 + 7 = 5319 MOD8: 7"
[1] "5327 MOD Power: 32 * 166 + 15 = 5327 MOD8: 7"
[1] "5335 MOD Power: 16 * 333 + 7 = 5335 MOD8: 7"
[1] "5343 MOD Power: 64 * 83 + 31 = 5343 MOD8: 7"
[1] "5351 MOD Power: 16 * 334 + 7 = 5351 MOD8: 7"
[1] "5359 MOD Power: 32 * 167 + 15 = 5359 MOD8: 7"
[1] "5367 MOD Power: 16 * 335 + 7 = 5367 MOD8: 7"
[1] "5375 MOD Power: 512 * 10 + 255 = 5375 MOD8: 7"
[1] "5383 MOD Power: 16 * 336 + 7 = 5383 MOD8: 7"
[1] "5391 MOD Power: 32 * 168 + 15 = 5391 MOD8: 7"
[1] "5399 MOD Power: 16 * 337 + 7 = 5399 MOD8: 7"
[1] "5407 MOD Power: 64 * 84 + 31 = 5407 MOD8: 7"
[1] "5415 MOD Power: 16 * 338 + 7 = 5415 MOD8: 7"
[1] "5423 MOD Power: 32 * 169 + 15 = 5423 MOD8: 7"
[1] "5431 MOD Power: 16 * 339 + 7 = 5431 MOD8: 7"
[1] "5439 MOD Power: 128 * 42 + 63 = 5439 MOD8: 7"
[1] "5447 MOD Power: 16 * 340 + 7 = 5447 MOD8: 7"
[1] "5455 MOD Power: 32 * 170 + 15 = 5455 MOD8: 7"
[1] "5463 MOD Power: 16 * 341 + 7 = 5463 MOD8: 7"
[1] "5471 MOD Power: 64 * 85 + 31 = 5471 MOD8: 7"
[1] "5479 MOD Power: 16 * 342 + 7 = 5479 MOD8: 7"
[1] "5487 MOD Power: 32 * 171 + 15 = 5487 MOD8: 7"
[1] "5495 MOD Power: 16 * 343 + 7 = 5495 MOD8: 7"
```

```
[1] "5503 MOD Power: 256 * 21 + 127 = 5503 MOD8: 7"
[1] "5511 MOD Power: 16 * 344 + 7 = 5511 MOD8: 7"
[1] "5519 MOD Power: 32 * 172 + 15 = 5519 MOD8: 7"
[1] "5527 MOD Power: 16 * 345 + 7 = 5527 MOD8: 7"
[1] "5535 MOD Power: 64 * 86 + 31 = 5535 MOD8: 7"
[1] "5543 MOD Power: 16 * 346 + 7 = 5543 MOD8: 7"
[1] "5551 MOD Power: 32 * 173 + 15 = 5551 MOD8: 7"
[1] "5559 MOD Power: 16 * 347 + 7 = 5559 MOD8: 7"
[1] "5567 MOD Power: 128 * 43 + 63 = 5567 MOD8: 7"
[1] "5575 MOD Power: 16 * 348 + 7 = 5575 MOD8: 7"
[1] "5583 MOD Power: 32 * 174 + 15 = 5583 MOD8: 7"
[1] "5591 MOD Power: 16 * 349 + 7 = 5591 MOD8: 7"
[1] "5599 MOD Power: 64 * 87 + 31 = 5599 MOD8: 7"
[1] "5607 MOD Power: 16 * 350 + 7 = 5607 MOD8: 7"
[1] "5615 MOD Power: 32 * 175 + 15 = 5615 MOD8: 7"
[1] "5623 MOD Power: 16 * 351 + 7 = 5623 MOD8: 7"
[1] "5631 MOD Power: 1024 * 5 + 511 = 5631 MOD8: 7"
[1] "5639 MOD Power: 16 * 352 + 7 = 5639 MOD8: 7"
[1] "5647 MOD Power: 32 * 176 + 15 = 5647 MOD8: 7"
[1] "5655 MOD Power: 16 * 353 + 7 = 5655 MOD8: 7"
[1] "5663 MOD Power: 64 * 88 + 31 = 5663 MOD8: 7"
[1] "5671 MOD Power: 16 * 354 + 7 = 5671 MOD8: 7"
[1] "5679 MOD Power: 32 * 177 + 15 = 5679 MOD8: 7"
[1] "5687 MOD Power: 16 * 355 + 7 = 5687 MOD8: 7"
[1] "5695 MOD Power: 128 * 44 + 63 = 5695 MOD8: 7"
[1] "5703 MOD Power: 16 * 356 + 7 = 5703 MOD8: 7"
[1] "5711 MOD Power: 32 * 178 + 15 = 5711 MOD8: 7"
[1] "5719 MOD Power: 16 * 357 + 7 = 5719 MOD8: 7"
[1] "5727 MOD Power: 64 * 89 + 31 = 5727 MOD8: 7"
[1] "5735 MOD Power: 16 * 358 + 7 = 5735 MOD8: 7"
[1] "5743 MOD Power: 32 * 179 + 15 = 5743 MOD8: 7"
[1] "5751 MOD Power: 16 * 359 + 7 = 5751 MOD8: 7"
[1] "5759 MOD Power: 256 * 22 + 127 = 5759 MOD8: 7"
[1] "5767 MOD Power: 16 * 360 + 7 = 5767 MOD8: 7"
[1] "5775 MOD Power: 32 * 180 + 15 = 5775 MOD8: 7"
[1] "5783 MOD Power: 16 * 361 + 7 = 5783 MOD8: 7"
[1] "5791 MOD Power: 64 * 90 + 31 = 5791 MOD8: 7"
[1] "5799 MOD Power: 16 * 362 + 7 = 5799 MOD8: 7"
[1] "5807 MOD Power: 32 * 181 + 15 = 5807 MOD8: 7"
[1] "5815 MOD Power: 16 * 363 + 7 = 5815 MOD8: 7"
[1] "5823 MOD Power: 128 * 45 + 63 = 5823 MOD8: 7"
[1] "5831 MOD Power: 16 * 364 + 7 = 5831 MOD8: 7"
[1] "5839 MOD Power: 32 * 182 + 15 = 5839 MOD8: 7"
[1] "5847 MOD Power: 16 * 365 + 7 = 5847 MOD8: 7"
[1] "5855 MOD Power: 64 * 91 + 31 = 5855 MOD8: 7"
[1] "5863 MOD Power: 16 * 366 + 7 = 5863 MOD8: 7"
[1] "5871 MOD Power: 32 * 183 + 15 = 5871 MOD8: 7"
[1] "5879 MOD Power: 16 * 367 + 7 = 5879 MOD8: 7"
[1] "5887 MOD Power: 512 * 11 + 255 = 5887 MOD8: 7"
[1] "5895 MOD Power: 16 * 368 + 7 = 5895 MOD8: 7"
[1] "5903 MOD Power: 32 * 184 + 15 = 5903 MOD8: 7"
[1] "5911 MOD Power: 16 * 369 + 7 = 5911 MOD8: 7"
[1] "5919 MOD Power: 64 * 92 + 31 = 5919 MOD8: 7"
[1] "5927 MOD Power: 16 * 370 + 7 = 5927 MOD8: 7"
[1] "5935 MOD Power: 32 * 185 + 15 = 5935 MOD8: 7"
[1] "5943 MOD Power: 16 * 371 + 7 = 5943 MOD8: 7"
[1] "5951 MOD Power: 128 * 46 + 63 = 5951 MOD8: 7"
[1] "5959 MOD Power: 16 * 372 + 7 = 5959 MOD8: 7"
[1] "5967 MOD Power: 32 * 186 + 15 = 5967 MOD8: 7"
[1] "5975 MOD Power: 16 * 373 + 7 = 5975 MOD8: 7"
[1] "5983 MOD Power: 64 * 93 + 31 = 5983 MOD8: 7"
[1] "5991 MOD Power: 16 * 374 + 7 = 5991 MOD8: 7"
[1] "5999 MOD Power: 32 * 187 + 15 = 5999 MOD8: 7"
[1] "6007 MOD Power: 16 * 375 + 7 = 6007 MOD8: 7"
```

```
[1] "6015 MOD Power: 256 * 23 + 127 = 6015 MOD8: 7"
[1] "6023 MOD Power: 16 * 376 + 7 = 6023 MOD8: 7"
[1] "6031 MOD Power: 32 * 188 + 15 = 6031 MOD8: 7"
[1] "6039 MOD Power: 16 * 377 + 7 = 6039 MOD8: 7"
[1] "6047 MOD Power: 64 * 94 + 31 = 6047 MOD8: 7"
[1] "6055 MOD Power: 16 * 378 + 7 = 6055 MOD8: 7"
[1] "6063 MOD Power: 32 * 189 + 15 = 6063 MOD8: 7"
[1] "6071 MOD Power: 16 * 379 + 7 = 6071 MOD8: 7"
[1] "6079 MOD Power: 128 * 47 + 63 = 6079 MOD8: 7"
[1] "6087 MOD Power: 16 * 380 + 7 = 6087 MOD8: 7"
[1] "6095 MOD Power: 32 * 190 + 15 = 6095 MOD8: 7"
[1] "6103 MOD Power: 16 * 381 + 7 = 6103 MOD8: 7"
[1] "6111 MOD Power: 64 * 95 + 31 = 6111 MOD8: 7"
[1] "6119 MOD Power: 16 * 382 + 7 = 6119 MOD8: 7"
[1] "6127 MOD Power: 32 * 191 + 15 = 6127 MOD8: 7"
[1] "6135 MOD Power: 16 * 383 + 7 = 6135 MOD8: 7"
[1] "6143 MOD Power: 4096 * 1 + 2047 = 6143 MOD8: 7"
[1] "6151 MOD Power: 16 * 384 + 7 = 6151 MOD8: 7"
[1] "6159 MOD Power: 32 * 192 + 15 = 6159 MOD8: 7"
[1] "6167 MOD Power: 16 * 385 + 7 = 6167 MOD8: 7"
[1] "6175 MOD Power: 64 * 96 + 31 = 6175 MOD8: 7"
[1] "6183 MOD Power: 16 * 386 + 7 = 6183 MOD8: 7"
[1] "6191 MOD Power: 32 * 193 + 15 = 6191 MOD8: 7"
[1] "6199 MOD Power: 16 * 387 + 7 = 6199 MOD8: 7"
[1] "6207 MOD Power: 128 * 48 + 63 = 6207 MOD8: 7"
[1] "6215 MOD Power: 16 * 388 + 7 = 6215 MOD8: 7"
[1] "6223 MOD Power: 32 * 194 + 15 = 6223 MOD8: 7"
[1] "6231 MOD Power: 16 * 389 + 7 = 6231 MOD8: 7"
[1] "6239 MOD Power: 64 * 97 + 31 = 6239 MOD8: 7"
[1] "6247 MOD Power: 16 * 390 + 7 = 6247 MOD8: 7"
[1] "6255 MOD Power: 32 * 195 + 15 = 6255 MOD8: 7"
[1] "6263 MOD Power: 16 * 391 + 7 = 6263 MOD8: 7"
[1] "6271 MOD Power: 256 * 24 + 127 = 6271 MOD8: 7"
[1] "6279 MOD Power: 16 * 392 + 7 = 6279 MOD8: 7"
[1] "6287 MOD Power: 32 * 196 + 15 = 6287 MOD8: 7"
[1] "6295 MOD Power: 16 * 393 + 7 = 6295 MOD8: 7"
[1] "6303 MOD Power: 64 * 98 + 31 = 6303 MOD8: 7"
[1] "6311 MOD Power: 16 * 394 + 7 = 6311 MOD8: 7"
[1] "6319 MOD Power: 32 * 197 + 15 = 6319 MOD8: 7"
[1] "6327 MOD Power: 16 * 395 + 7 = 6327 MOD8: 7"
[1] "6335 MOD Power: 128 * 49 + 63 = 6335 MOD8: 7"
[1] "6343 MOD Power: 16 * 396 + 7 = 6343 MOD8: 7"
[1] "6351 MOD Power: 32 * 198 + 15 = 6351 MOD8: 7"
[1] "6359 MOD Power: 16 * 397 + 7 = 6359 MOD8: 7"
[1] "6367 MOD Power: 64 * 99 + 31 = 6367 MOD8: 7"
[1] "6375 MOD Power: 16 * 398 + 7 = 6375 MOD8: 7"
[1] "6383 MOD Power: 32 * 199 + 15 = 6383 MOD8: 7"
[1] "6391 MOD Power: 16 * 399 + 7 = 6391 MOD8: 7"
[1] "6399 MOD Power: 512 * 12 + 255 = 6399 MOD8: 7"
[1] "6407 MOD Power: 16 * 400 + 7 = 6407 MOD8: 7"
[1] "6415 MOD Power: 32 * 200 + 15 = 6415 MOD8: 7"
[1] "6423 MOD Power: 16 * 401 + 7 = 6423 MOD8: 7"
[1] "6431 MOD Power: 64 * 100 + 31 = 6431 MOD8: 7"
[1] "6439 MOD Power: 16 * 402 + 7 = 6439 MOD8: 7"
[1] "6447 MOD Power: 32 * 201 + 15 = 6447 MOD8: 7"
[1] "6455 MOD Power: 16 * 403 + 7 = 6455 MOD8: 7"
[1] "6463 MOD Power: 128 * 50 + 63 = 6463 MOD8: 7"
[1] "6471 MOD Power: 16 * 404 + 7 = 6471 MOD8: 7"
[1] "6479 MOD Power: 32 * 202 + 15 = 6479 MOD8: 7"
[1] "6487 MOD Power: 16 * 405 + 7 = 6487 MOD8: 7"
[1] "6495 MOD Power: 64 * 101 + 31 = 6495 MOD8: 7"
[1] "6503 MOD Power: 16 * 406 + 7 = 6503 MOD8: 7"
[1] "6511 MOD Power: 32 * 203 + 15 = 6511 MOD8: 7"
[1] "6519 MOD Power: 16 * 407 + 7 = 6519 MOD8: 7"
```

```
[1] "6527 MOD Power: 256 * 25 + 127 = 6527 MOD8: 7"
[1] "6535 MOD Power: 16 * 408 + 7 = 6535 MOD8: 7"
[1] "6543 MOD Power: 32 * 204 + 15 = 6543 MOD8: 7"
[1] "6551 MOD Power: 16 * 409 + 7 = 6551 MOD8: 7"
[1] "6559 MOD Power: 64 * 102 + 31 = 6559 MOD8: 7"
[1] "6567 MOD Power: 16 * 410 + 7 = 6567 MOD8: 7"
[1] "6575 MOD Power: 32 * 205 + 15 = 6575 MOD8: 7"
[1] "6583 MOD Power: 16 * 411 + 7 = 6583 MOD8: 7"
[1] "6591 MOD Power: 128 * 51 + 63 = 6591 MOD8: 7"
[1] "6599 MOD Power: 16 * 412 + 7 = 6599 MOD8: 7"
[1] "6607 MOD Power: 32 * 206 + 15 = 6607 MOD8: 7"
[1] "6615 MOD Power: 16 * 413 + 7 = 6615 MOD8: 7"
[1] "6623 MOD Power: 64 * 103 + 31 = 6623 MOD8: 7"
[1] "6631 MOD Power: 16 * 414 + 7 = 6631 MOD8: 7"
[1] "6639 MOD Power: 32 * 207 + 15 = 6639 MOD8: 7"
[1] "6647 MOD Power: 16 * 415 + 7 = 6647 MOD8: 7"
[1] "6655 MOD Power: 1024 * 6 + 511 = 6655 MOD8: 7"
[1] "6663 MOD Power: 16 * 416 + 7 = 6663 MOD8: 7"
[1] "6671 MOD Power: 32 * 208 + 15 = 6671 MOD8: 7"
[1] "6679 MOD Power: 16 * 417 + 7 = 6679 MOD8: 7"
[1] "6687 MOD Power: 64 * 104 + 31 = 6687 MOD8: 7"
[1] "6695 MOD Power: 16 * 418 + 7 = 6695 MOD8: 7"
[1] "6703 MOD Power: 32 * 209 + 15 = 6703 MOD8: 7"
[1] "6711 MOD Power: 16 * 419 + 7 = 6711 MOD8: 7"
[1] "6719 MOD Power: 128 * 52 + 63 = 6719 MOD8: 7"
[1] "6727 MOD Power: 16 * 420 + 7 = 6727 MOD8: 7"
[1] "6735 MOD Power: 32 * 210 + 15 = 6735 MOD8: 7"
[1] "6743 MOD Power: 16 * 421 + 7 = 6743 MOD8: 7"
[1] "6751 MOD Power: 64 * 105 + 31 = 6751 MOD8: 7"
[1] "6759 MOD Power: 16 * 422 + 7 = 6759 MOD8: 7"
[1] "6767 MOD Power: 32 * 211 + 15 = 6767 MOD8: 7"
[1] "6775 MOD Power: 16 * 423 + 7 = 6775 MOD8: 7"
[1] "6783 MOD Power: 256 * 26 + 127 = 6783 MOD8: 7"
[1] "6791 MOD Power: 16 * 424 + 7 = 6791 MOD8: 7"
[1] "6799 MOD Power: 32 * 212 + 15 = 6799 MOD8: 7"
[1] "6807 MOD Power: 16 * 425 + 7 = 6807 MOD8: 7"
[1] "6815 MOD Power: 64 * 106 + 31 = 6815 MOD8: 7"
[1] "6823 MOD Power: 16 * 426 + 7 = 6823 MOD8: 7"
[1] "6831 MOD Power: 32 * 213 + 15 = 6831 MOD8: 7"
[1] "6839 MOD Power: 16 * 427 + 7 = 6839 MOD8: 7"
[1] "6847 MOD Power: 128 * 53 + 63 = 6847 MOD8: 7"
[1] "6855 MOD Power: 16 * 428 + 7 = 6855 MOD8: 7"
[1] "6863 MOD Power: 32 * 214 + 15 = 6863 MOD8: 7"
[1] "6871 MOD Power: 16 * 429 + 7 = 6871 MOD8: 7"
[1] "6879 MOD Power: 64 * 107 + 31 = 6879 MOD8: 7"
[1] "6887 MOD Power: 16 * 430 + 7 = 6887 MOD8: 7"
[1] "6895 MOD Power: 32 * 215 + 15 = 6895 MOD8: 7"
[1] "6903 MOD Power: 16 * 431 + 7 = 6903 MOD8: 7"
[1] "6911 MOD Power: 512 * 13 + 255 = 6911 MOD8: 7"
[1] "6919 MOD Power: 16 * 432 + 7 = 6919 MOD8: 7"
[1] "6927 MOD Power: 32 * 216 + 15 = 6927 MOD8: 7"
[1] "6935 MOD Power: 16 * 433 + 7 = 6935 MOD8: 7"
[1] "6943 MOD Power: 64 * 108 + 31 = 6943 MOD8: 7"
[1] "6951 MOD Power: 16 * 434 + 7 = 6951 MOD8: 7"
[1] "6959 MOD Power: 32 * 217 + 15 = 6959 MOD8: 7"
[1] "6967 MOD Power: 16 * 435 + 7 = 6967 MOD8: 7"
[1] "6975 MOD Power: 128 * 54 + 63 = 6975 MOD8: 7"
[1] "6983 MOD Power: 16 * 436 + 7 = 6983 MOD8: 7"
[1] "6991 MOD Power: 32 * 218 + 15 = 6991 MOD8: 7"
[1] "6999 MOD Power: 16 * 437 + 7 = 6999 MOD8: 7"
[1] "7007 MOD Power: 64 * 109 + 31 = 7007 MOD8: 7"
[1] "7015 MOD Power: 16 * 438 + 7 = 7015 MOD8: 7"
[1] "7023 MOD Power: 32 * 219 + 15 = 7023 MOD8: 7"
[1] "7031 MOD Power: 16 * 439 + 7 = 7031 MOD8: 7"
```

```
[1] "7039 MOD Power: 256 * 27 + 127 = 7039 MOD8: 7"
[1] "7047 MOD Power: 16 * 440 + 7 = 7047 MOD8: 7"
[1] "7055 MOD Power: 32 * 220 + 15 = 7055 MOD8: 7"
[1] "7063 MOD Power: 16 * 441 + 7 = 7063 MOD8: 7"
[1] "7071 MOD Power: 64 * 110 + 31 = 7071 MOD8: 7"
[1] "7079 MOD Power: 16 * 442 + 7 = 7079 MOD8: 7"
[1] "7087 MOD Power: 32 * 221 + 15 = 7087 MOD8: 7"
[1] "7095 MOD Power: 16 * 443 + 7 = 7095 MOD8: 7"
[1] "7103 MOD Power: 128 * 55 + 63 = 7103 MOD8: 7"
[1] "7111 MOD Power: 16 * 444 + 7 = 7111 MOD8: 7"
[1] "7119 MOD Power: 32 * 222 + 15 = 7119 MOD8: 7"
[1] "7127 MOD Power: 16 * 445 + 7 = 7127 MOD8: 7"
[1] "7135 MOD Power: 64 * 111 + 31 = 7135 MOD8: 7"
[1] "7143 MOD Power: 16 * 446 + 7 = 7143 MOD8: 7"
[1] "7151 MOD Power: 32 * 223 + 15 = 7151 MOD8: 7"
[1] "7159 MOD Power: 16 * 447 + 7 = 7159 MOD8: 7"
[1] "7167 MOD Power: 2048 * 3 + 1023 = 7167 MOD8: 7"
[1] "7175 MOD Power: 16 * 448 + 7 = 7175 MOD8: 7"
[1] "7183 MOD Power: 32 * 224 + 15 = 7183 MOD8: 7"
[1] "7191 MOD Power: 16 * 449 + 7 = 7191 MOD8: 7"
[1] "7199 MOD Power: 64 * 112 + 31 = 7199 MOD8: 7"
[1] "7207 MOD Power: 16 * 450 + 7 = 7207 MOD8: 7"
[1] "7215 MOD Power: 32 * 225 + 15 = 7215 MOD8: 7"
[1] "7223 MOD Power: 16 * 451 + 7 = 7223 MOD8: 7"
[1] "7231 MOD Power: 128 * 56 + 63 = 7231 MOD8: 7"
[1] "7239 MOD Power: 16 * 452 + 7 = 7239 MOD8: 7"
[1] "7247 MOD Power: 32 * 226 + 15 = 7247 MOD8: 7"
[1] "7255 MOD Power: 16 * 453 + 7 = 7255 MOD8: 7"
[1] "7263 MOD Power: 64 * 113 + 31 = 7263 MOD8: 7"
[1] "7271 MOD Power: 16 * 454 + 7 = 7271 MOD8: 7"
[1] "7279 MOD Power: 32 * 227 + 15 = 7279 MOD8: 7"
[1] "7287 MOD Power: 16 * 455 + 7 = 7287 MOD8: 7"
[1] "7295 MOD Power: 256 * 28 + 127 = 7295 MOD8: 7"
[1] "7303 MOD Power: 16 * 456 + 7 = 7303 MOD8: 7"
[1] "7311 MOD Power: 32 * 228 + 15 = 7311 MOD8: 7"
[1] "7319 MOD Power: 16 * 457 + 7 = 7319 MOD8: 7"
[1] "7327 MOD Power: 64 * 114 + 31 = 7327 MOD8: 7"
[1] "7335 MOD Power: 16 * 458 + 7 = 7335 MOD8: 7"
[1] "7343 MOD Power: 32 * 229 + 15 = 7343 MOD8: 7"
[1] "7351 MOD Power: 16 * 459 + 7 = 7351 MOD8: 7"
[1] "7359 MOD Power: 128 * 57 + 63 = 7359 MOD8: 7"
[1] "7367 MOD Power: 16 * 460 + 7 = 7367 MOD8: 7"
[1] "7375 MOD Power: 32 * 230 + 15 = 7375 MOD8: 7"
[1] "7383 MOD Power: 16 * 461 + 7 = 7383 MOD8: 7"
[1] "7391 MOD Power: 64 * 115 + 31 = 7391 MOD8: 7"
[1] "7399 MOD Power: 16 * 462 + 7 = 7399 MOD8: 7"
[1] "7407 MOD Power: 32 * 231 + 15 = 7407 MOD8: 7"
[1] "7415 MOD Power: 16 * 463 + 7 = 7415 MOD8: 7"
[1] "7423 MOD Power: 512 * 14 + 255 = 7423 MOD8: 7"
[1] "7431 MOD Power: 16 * 464 + 7 = 7431 MOD8: 7"
[1] "7439 MOD Power: 32 * 232 + 15 = 7439 MOD8: 7"
[1] "7447 MOD Power: 16 * 465 + 7 = 7447 MOD8: 7"
[1] "7455 MOD Power: 64 * 116 + 31 = 7455 MOD8: 7"
[1] "7463 MOD Power: 16 * 466 + 7 = 7463 MOD8: 7"
[1] "7471 MOD Power: 32 * 233 + 15 = 7471 MOD8: 7"
[1] "7479 MOD Power: 16 * 467 + 7 = 7479 MOD8: 7"
[1] "7487 MOD Power: 128 * 58 + 63 = 7487 MOD8: 7"
[1] "7495 MOD Power: 16 * 468 + 7 = 7495 MOD8: 7"
[1] "7503 MOD Power: 32 * 234 + 15 = 7503 MOD8: 7"
[1] "7511 MOD Power: 16 * 469 + 7 = 7511 MOD8: 7"
[1] "7519 MOD Power: 64 * 117 + 31 = 7519 MOD8: 7"
[1] "7527 MOD Power: 16 * 470 + 7 = 7527 MOD8: 7"
[1] "7535 MOD Power: 32 * 235 + 15 = 7535 MOD8: 7"
[1] "7543 MOD Power: 16 * 471 + 7 = 7543 MOD8: 7"
```

```
[1] "7551 MOD Power: 256 * 29 + 127 = 7551 MOD8: 7"
[1] "7559 MOD Power: 16 * 472 + 7 = 7559 MOD8: 7"
[1] "7567 MOD Power: 32 * 236 + 15 = 7567 MOD8: 7"
[1] "7575 MOD Power: 16 * 473 + 7 = 7575 MOD8: 7"
[1] "7583 MOD Power: 64 * 118 + 31 = 7583 MOD8: 7"
[1] "7591 MOD Power: 16 * 474 + 7 = 7591 MOD8: 7"
[1] "7599 MOD Power: 32 * 237 + 15 = 7599 MOD8: 7"
[1] "7607 MOD Power: 16 * 475 + 7 = 7607 MOD8: 7"
[1] "7615 MOD Power: 128 * 59 + 63 = 7615 MOD8: 7"
[1] "7623 MOD Power: 16 * 476 + 7 = 7623 MOD8: 7"
[1] "7631 MOD Power: 32 * 238 + 15 = 7631 MOD8: 7"
[1] "7639 MOD Power: 16 * 477 + 7 = 7639 MOD8: 7"
[1] "7647 MOD Power: 64 * 119 + 31 = 7647 MOD8: 7"
[1] "7655 MOD Power: 16 * 478 + 7 = 7655 MOD8: 7"
[1] "7663 MOD Power: 32 * 239 + 15 = 7663 MOD8: 7"
[1] "7671 MOD Power: 16 * 479 + 7 = 7671 MOD8: 7"
[1] "7679 MOD Power: 1024 * 7 + 511 = 7679 MOD8: 7"
[1] "7687 MOD Power: 16 * 480 + 7 = 7687 MOD8: 7"
[1] "7695 MOD Power: 32 * 240 + 15 = 7695 MOD8: 7"
[1] "7703 MOD Power: 16 * 481 + 7 = 7703 MOD8: 7"
[1] "7711 MOD Power: 64 * 120 + 31 = 7711 MOD8: 7"
[1] "7719 MOD Power: 16 * 482 + 7 = 7719 MOD8: 7"
[1] "7727 MOD Power: 32 * 241 + 15 = 7727 MOD8: 7"
[1] "7735 MOD Power: 16 * 483 + 7 = 7735 MOD8: 7"
[1] "7743 MOD Power: 128 * 60 + 63 = 7743 MOD8: 7"
[1] "7751 MOD Power: 16 * 484 + 7 = 7751 MOD8: 7"
[1] "7759 MOD Power: 32 * 242 + 15 = 7759 MOD8: 7"
[1] "7767 MOD Power: 16 * 485 + 7 = 7767 MOD8: 7"
[1] "7775 MOD Power: 64 * 121 + 31 = 7775 MOD8: 7"
[1] "7783 MOD Power: 16 * 486 + 7 = 7783 MOD8: 7"
[1] "7791 MOD Power: 32 * 243 + 15 = 7791 MOD8: 7"
[1] "7799 MOD Power: 16 * 487 + 7 = 7799 MOD8: 7"
[1] "7807 MOD Power: 256 * 30 + 127 = 7807 MOD8: 7"
[1] "7815 MOD Power: 16 * 488 + 7 = 7815 MOD8: 7"
[1] "7823 MOD Power: 32 * 244 + 15 = 7823 MOD8: 7"
[1] "7831 MOD Power: 16 * 489 + 7 = 7831 MOD8: 7"
[1] "7839 MOD Power: 64 * 122 + 31 = 7839 MOD8: 7"
[1] "7847 MOD Power: 16 * 490 + 7 = 7847 MOD8: 7"
[1] "7855 MOD Power: 32 * 245 + 15 = 7855 MOD8: 7"
[1] "7863 MOD Power: 16 * 491 + 7 = 7863 MOD8: 7"
[1] "7871 MOD Power: 128 * 61 + 63 = 7871 MOD8: 7"
[1] "7879 MOD Power: 16 * 492 + 7 = 7879 MOD8: 7"
[1] "7887 MOD Power: 32 * 246 + 15 = 7887 MOD8: 7"
[1] "7895 MOD Power: 16 * 493 + 7 = 7895 MOD8: 7"
[1] "7903 MOD Power: 64 * 123 + 31 = 7903 MOD8: 7"
[1] "7911 MOD Power: 16 * 494 + 7 = 7911 MOD8: 7"
[1] "7919 MOD Power: 32 * 247 + 15 = 7919 MOD8: 7"
[1] "7927 MOD Power: 16 * 495 + 7 = 7927 MOD8: 7"
[1] "7935 MOD Power: 512 * 15 + 255 = 7935 MOD8: 7"
[1] "7943 MOD Power: 16 * 496 + 7 = 7943 MOD8: 7"
[1] "7951 MOD Power: 32 * 248 + 15 = 7951 MOD8: 7"
[1] "7959 MOD Power: 16 * 497 + 7 = 7959 MOD8: 7"
[1] "7967 MOD Power: 64 * 124 + 31 = 7967 MOD8: 7"
[1] "7975 MOD Power: 16 * 498 + 7 = 7975 MOD8: 7"
[1] "7983 MOD Power: 32 * 249 + 15 = 7983 MOD8: 7"
[1] "7991 MOD Power: 16 * 499 + 7 = 7991 MOD8: 7"
[1] "7999 MOD Power: 128 * 62 + 63 = 7999 MOD8: 7"
```

Log of numbers added to the 27 Run

- I have added the log results for the number and for the remainder
  - N LOG shows the log of the number
- From all R LOGS that are 0 you will see that
  - N LOG + 3 steps will be a Lower N LOG
  - All R LOG 0 are 1MOD4 numbers
    - This shows the loop resets that are happening with the next step

Steps	Number	MOD 8	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	Correct Power	*	Powers	+	Remainder	N Log	R Log
	27	3	0	0	0	0	0	0	0	0	0	1	1	0	1	1	8	*	3	+	3	4.7549	1.5850
1	82	2	0	0	0	0	0	0	0	1	0	1	0	0	1	0	8	*	10	+	2	6.3576	1.0000
2	41	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	4	*	10	+	1	5.3576	0.0000
3	124	4	0	0	0	0	0	0	0	1	1	1	1	1	0	0	256	*	0	+	124	6.9542	6.9542
4	62	6	0	0	0	0	0	0	0	0	1	1	1	1	1	0	128	*	0	+	62	5.9542	5.9542
5	31	7	0	0	0	0	0	0	0	0	0	1	1	1	1	1	64	*	0	+	31	4.9542	4.9542
6	94	6	0	0	0	0	0	0	0	1	0	1	1	1	1	0	64	*	1	+	30	6.5546	4.9069
7	47	7	0	0	0	0	0	0	0	0	1	0	1	1	1	1	32	*	1	+	15	5.5546	3.9069
8	142	6	0	0	0	0	0	0	1	0	0	0	1	1	1	0	32	*	4	+	14	7.1497	3.8074
9	71	7	0	0	0	0	0	0	0	1	0	0	0	1	1	1	16	*	4	+	7	6.1497	2.8074
10	214	6	0	0	0	0	0	0	1	1	0	1	0	1	1	0	16	*	13	+	6	7.7415	2.5850
11	107	3	0	0	0	0	0	0	0	1	1	0	1	0	1	1	8	*	13	+	3	6.7415	1.5850
12	322	2	0	0	0	0	0	1	0	1	0	0	0	0	1	0	8	*	40	+	2	8.3309	1.0000
13	161	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	4	*	40	+	1	7.3309	0.0000
14	484	4	0	0	0	0	0	1	1	1	1	0	0	1	0	0	16	*	30	+	4	8.9189	2.0000
15	242	2	0	0	0	0	0	0	1	1	1	1	0	0	1	0	8	*	30	+	2	7.9189	1.0000
16	121	1	0	0	0	0	0	0	0	1	1	1	1	0	0	1	4	*	30	+	1	6.9189	0.0000
17	364	4	0	0	0	0	0	1	0	1	1	0	1	1	0	0	32	*	11	+	12	8.5078	3.5850
18	182	6	0	0	0	0	0	0	1	0	1	1	0	1	1	0	16	*	11	+	6	7.5078	2.5850
19	91	3	0	0	0	0	0	0	0	1	0	1	1	0	1	1	8	*	11	+	3	6.5078	1.5850
20	274	2	0	0	0	0	0	1	0	0	0	1	0	0	1	0	8	*	34	+	2	8.0980	1.0000
21	137	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	4	*	34	+	1	7.0980	0.0000
22	412	4	0	0	0	0	0	1	1	0	0	1	1	1	0	0	64	*	6	+	28	8.6865	4.8074
23	206	6	0	0	0	0	0	0	1	1	0	0	1	1	1	0	32	*	6	+	14	7.6865	3.8074
24	103	7	0	0	0	0	0	0	0	1	1	0	0	1	1	1	16	*	6	+	7	6.6865	2.8074
25	310	6	0	0	0	0	0	1	0	0	1	1	0	1	1	0	16	*	19	+	6	8.2761	2.5850
26	155	3	0	0	0	0	0	0	1	0	0	1	1	0	1	1	8	*	19	+	3	7.2761	1.5850
27	466	2	0	0	0	0	0	1	1	1	0	1	0	0	1	0	8	*	58	+	2	8.8642	1.0000
28	233	1	0	0	0	0	0	0	1	1	1	0	1	0	0	1	4	*	58	+	1	7.8642	0.0000
29	700	4	0	0	0	0	1	0	1	0	1	1	1	1	0	0	128	*	5	+	60	9.4512	5.9069
30	350	6	0	0	0	0	0	1	0	1	0	1	1	1	1	0	64	*	5	+	30	8.4512	4.9069
31	175	7	0	0	0	0	0	0	1	0	1	0	1	1	1	1	32	*	5	+	15	7.4512	3.9069
32	526	6	0	0	0	0	1	0	0	0	0	0	1	1	1	0	32	*	16	+	14	9.0389	3.8074
33	263	7	0	0	0	0	0	1	0	0	0	0	0	1	1	1	16	*	16	+	7	8.0389	2.8074
34	790	6	0	0	0	0	1	1	0	0	0	1	0	1	1	0	16	*	49	+	6	9.6257	2.5850
35	395	3	0	0	0	0	0	1	1	0	0	0	1	0	1	1	8	*	49	+	3	8.6257	1.5850
36	1,186	2	0	0	0	1	0	0	1	0	1	0	0	0	1	0	8	*	148	+	2	10.2119	1.0000
37	593	1	0	0	0	0	1	0	0	1	0	1	0	0	0	1	4	*	148	+	1	9.2119	0.0000
38	1,780	4	0	0	0	1	1	0	1	1	1	1	0	1	0	0	16	*	111	+	4	10.7977	2.0000
39	890	2	0	0	0	0	1	1	0	1	1	1	1	0	1	0	8	*	111	+	2	9.7977	1.0000
40	445	5	0	0	0	0	0	1	1	0	1	1	1	1	0	1	4	*	111	+	1	8.7977	0.0000
41	1,336	0	0	0	0	1	0	1	0	0	1	1	1	0	0	0	128	*	10	+	56	10.3837	5.8074
42	668	4	0	0	0	0	1	0	1	0	0	1	1	1	0	0	64	*	10	+	28	9.3837	4.8074
43	334	6	0	0	0	0	0	1	0	1	0	0	1	1	1	0	32	*	10	+	14	8.3837	3.8074
44	167	7	0	0	0	0	0	0	1	0	1	0	0	1	1	1	16	*	10	+	7	7.3837	2.8074
45	502	6	0	0	0	0	0	1	1	1	1	1	0	1	1	0	16	*	31	+	6	8.9715	2.5850
46	251	3	0	0	0	0	0	0	1	1	1	1	1	0	1	1	8	*	31	+	3	7.9715	1.5850
47	754	2	0	0	0	0	1	0	1	1	1	1	0	0	1	0	8	*	94	+	2	9.5584	1.0000
48	377	1	0	0	0	0	0	1	0	1	1	1	1	0	0	1	4	*	94	+	1	8.5584	0.0000
49	1,132	4	0	0	0	1	0	0	0	1	1	0	1	1	0	0	32	*	35	+	12	10.1447	3.5850
50	566	6	0	0	0	0	1	0	0	0	1	1	0	1	1	0	16	*	35	+	6	9.1447	2.5850
51	283	3	0	0	0	0	0	1	0	0	0	1	1	0	1	1	8	*	35	+	3	8.1447	1.5850
52	850	2	0	0	0	0	1	1	0	1	0	1	0	0	1	0	8	*	106	+	2	9.7313	1.0000
53	425	1	0	0	0	0	0	1	1	0	1	0	1	0	0	1	4	*	106	+	1	8.7313	0.0000
54	1,276	4	0	0	0	1	0	0	1	1	1	1	1	1	0	0	512	*	2	+	252	10.3174	7.9773
55	638	6	0	0	0	0	1	0	0	1	1	1	1	1	1	0	256	*	2	+	126	9.3174	6.9773
56	319	7	0	0	0	0	0	1	0	0	1	1	1	1	1	1	128	*	2	+	63	8.3174	5.9773

Steps	Number	MOD 8	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	Correct Power	*	Powers	+	Remainder	N Log	R Log
57	958	6	0	0	0	0	1	1	1	0	1	1	1	1	1	0	128	*	7	+	62	9.9039	5.9542
58	479	7	0	0	0	0	0	1	1	1	0	1	1	1	1	1	64	*	7	+	31	8.9039	4.9542
59	1,438	6	0	0	0	1	0	1	1	0	0	1	1	1	1	0	64	*	22	+	30	10.4898	4.9069
60	719	7	0	0	0	0	1	0	1	1	0	0	1	1	1	1	32	*	22	+	15	9.4898	3.9069
61	2,158	6	0	0	1	0	0	0	0	1	1	0	1	1	1	0	32	*	67	+	14	11.0755	3.8074
62	1,079	7	0	0	0	1	0	0	0	0	1	1	0	1	1	1	16	*	67	+	7	10.0755	2.8074
63	3,238	6	0	0	1	1	0	0	1	0	1	0	0	1	1	0	16	*	202	+	6	11.6609	2.5850
64	1,619	3	0	0	0	1	1	0	0	1	0	1	0	0	1	1	8	*	202	+	3	10.6609	1.5850
65	4,858	2	0	1	0	0	1	0	1	1	1	1	1	0	1	0	8	*	607	+	2	12.2461	1.0000
66	2,429	5	0	0	1	0	0	1	0	1	1	1	1	1	0	1	4	*	607	+	1	11.2461	0.0000
67	7,288	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	256	*	28	+	120	12.8313	6.9069
68	3,644	4	0	0	1	1	1	0	0	0	1	1	1	1	0	0	128	*	28	+	60	11.8313	5.9069
69	1,822	6	0	0	0	1	1	1	0	0	0	1	1	1	1	0	64	*	28	+	30	10.8313	4.9069
70	911	7	0	0	0	0	1	1	1	0	0	0	1	1	1	1	32	*	28	+	15	9.8313	3.9069
71	2,734	6	0	0	1	0	1	0	1	0	1	0	1	1	1	0	32	*	85	+	14	11.4168	3.8074
72	1,367	7	0	0	0	1	0	1	0	1	0	1	0	1	1	1	16	*	85	+	7	10.4168	2.8074
73	4,102	6	0	1	0	0	0	0	0	0	0	0	0	1	1	0	16	*	256	+	6	12.0021	2.5850
74	2,051	3	0	0	1	0	0	0	0	0	0	0	0	0	1	1	8	*	256	+	3	11.0021	1.5850
75	6,154	2	0	1	1	0	0	0	0	0	0	0	1	0	1	0	8	*	769	+	2	12.5873	1.0000
76	3,077	5	0	0	1	1	0	0	0	0	0	0	0	1	0	1	4	*	769	+	1	11.5873	0.0000
77	9,232	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	64	*	144	+	16	13.1724	4.0000
78	4,616	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	32	*	144	+	8	12.1724	3.0000
79	2,308	4	0	0	1	0	0	1	0	0	0	0	0	1	0	0	16	*	144	+	4	11.1724	2.0000
80	1,154	2	0	0	0	1	0	0	1	0	0	0	0	0	1	0	8	*	144	+	2	10.1724	1.0000
81	577	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	4	*	144	+	1	9.1724	0.0000
82	1,732	4	0	0	0	1	1	0	1	1	0	0	0	1	0	0	16	*	108	+	4	10.7582	2.0000
83	866	2	0	0	0	0	1	1	0	1	1	0	0	0	1	0	8	*	108	+	2	9.7582	1.0000
84	433	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	4	*	108	+	1	8.7582	0.0000
85	1,300	4	0	0	0	1	0	1	0	0	0	1	0	1	0	0	16	*	81	+	4	10.3443	2.0000
86	650	2	0	0	0	0	1	0	1	0	0	0	1	0	1	0	8	*	81	+	2	9.3443	1.0000
87	325	5	0	0	0	0	0	1	0	1	0	0	0	1	0	1	4	*	81	+	1	8.3443	0.0000
88	976	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	64	*	15	+	16	9.9307	4.0000
89	488	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	32	*	15	+	8	8.9307	3.0000
90	244	4	0	0	0	0	0	0	1	1	1	1	0	1	0	0	16	*	15	+	4	7.9307	2.0000
91	122	2	0	0	0	0	0	0	0	1	1	1	1	0	1	0	8	*	15	+	2	6.9307	1.0000
92	61	5	0	0	0	0	0	0	0	0	1	1	1	1	0	1	4	*	15	+	1	5.9307	0.0000
93	184	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	128	*	1	+	56	7.5236	5.8074
94	92	4	0	0	0	0	0	0	0	1	0	1	1	1	0	0	64	*	1	+	28	6.5236	4.8074
95	46	6	0	0	0	0	0	0	0	0	1	0	1	1	1	0	32	*	1	+	14	5.5236	3.8074
96	23	7	0	0	0	0	0	0	0	0	0	1	0	1	1	1	16	*	1	+	7	4.5236	2.8074
97	70	6	0	0	0	0	0	0	0	1	0	0	0	1	1	0	16	*	4	+	6	6.1293	2.5850
98	35	3	0	0	0	0	0	0	0	0	1	0	0	0	1	1	8	*	4	+	3	5.1293	1.5850
99	106	2	0	0	0	0	0	0	0	1	1	0	1	0	1	0	8	*	13	+	2	6.7279	1.0000
100	53	5	0	0	0	0	0	0	0	0	1	1	0	1	0	1	4	*	13	+	1	5.7279	0.0000
101	160	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	128	*	1	+	32	7.3219	5.0000
102	80	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	64	*	1	+	16	6.3219	4.0000
103	40	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	32	*	1	+	8	5.3219	3.0000
104	20	4	0	0	0	0	0	0	0	0	0	1	0	1	0	0	16	*	1	+	4	4.3219	2.0000
105	10	2	0	0	0	0	0	0	0	0	0	0	1	0	1	0	8	*	1	+	2	3.3219	1.0000
106	5	5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	4	*	1	+	1	2.3219	0.0000
107	16	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	64	*	0	+	16	4.0000	4.0000
108	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	32	*	0	+	8	3.0000	3.0000
109	4	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	2.0000	2.0000
110	2	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	1.0000	1.0000
111	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	0.0000	0.0000
	4	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	16	*	0	+	4	2.0000	2.0000
	2	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	*	0	+	2	1.0000	1.0000
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	*	0	+	1	0.0000	0.0000

The Collatz Map or Transformation

Overview of the Collatz Map

The Collatz Map involves examining the transformation of odd numbers through a sequence of mod forms, each progressively covering a larger set of odd numbers. This method demonstrates that every odd number eventually reduces to a smaller form, supporting the conjecture's assertion that all numbers reach 1.

Methodology

Initial Step: 1mod4

1. **Starting Point**: Consider an odd number of the form 4k+1, which represents 50% of all odd numbers.
2. **Transformation**:
   - Applying the Collatz function: 3(4k+1) +1 = 12k+4 = 4(3k+1).
   - Since 4(3k+1) is divisible by 4, it will reduce through division steps to 3k+1.
3. **Conclusion**: In 3 steps, the number is reduced below its original value. This step covers 50% of all odd numbers.

Next Step: 3mod8

1. **Starting Point**: Consider an odd number of the form 8k+3, representing 50% of the remaining odd numbers.
2. **Transformation**:
   - Applying the Collatz function: (8k+3) = (12k+5) = (9k+4).
   - Reducing: 12k+5, is a 1MOD (4) number and we know is below itself in 3 steps.
3. **Conclusion**: This step covers an additional 50% of the remaining odd numbers.

Progressive Steps

1. **General Form**: For an odd number of the form $(2^{(n+1)})-1 \bmod 2^{(n+2)}$, the transformation involves progressively doubling the coefficients of k and adding appropriate constants to ensure the number remains odd.
2. **Transformation Rule**:
   - For the S value: Multiply k by 2 and the constant by 2x+1.
   - For the M value: Multiply k by 3 and the constant by 3x+2.
   - For the L value: Multiply k by 3 and the constant by 3x+1.
   - We can use these formulas to find all S,M,L values for the next 2 step to as high as necessary to solve for any number.
3. **Illustration**:
   - 3mod8: Starting from 8k+3:
     - S=8k+3, M=12k+5, L=9k+4
   - 7mod16: Starting from 16k+7:

        - S=16k+7, M=36k+17, L=27k+13
    - 15mod32: Starting from 32k+15:
        - S=32k+15, M=108k+53, L=81k+40
4. **Progressive Coverage**: Each step further divides the set of odd numbers, demonstrating that every number eventually maps to a smaller form. By continually applying these rules, the process covers an increasingly larger percentage of odd numbers.

Detailed Example

For a detailed example, consider the following steps for the first few iterations:

- MOD is the module of the number; ALL odd numbers fall into one of these mods
- S is the starting k value, M is the k value that is 1MOD4 in this sequence, L is the k value that is below the M in 3 steps.
- Steps is the total amount of steps required to go from S to L.
- At 40 levels we covered 99.99999999995453% of all Odd numbers.
- We can keep applying `(2^(n+1))−1 mod 2^(n+2)` to find the next MOD level to infinity.

Mod	S	M	L	Steps
1mod4	4k+1	4k+1	3k+1	3
3mod8	8k+3	12k+5	9k+4	5
7mod16	16k+7	36k+17	27k+13	7
15mod32	32k+15	108k+53	81k+40	9
31mod64	64k+31	324k+161	243k+121	11
63mod128	128k+63	972k+485	729k+364	13
127mod256	256k+127	2916k+1457	2187k+1093	15
255mod512	512k+255	8748k+4373	6561k+3280	17
511mod1024	1024k+511	26244k+13121	19683k+9841	19
1023mod2048	2048k+1023	78732k+39365	59049k+29524	21
2047mod4096	4096k+2047	236196k+118097	177147k+88573	23
4095mod8192	8192k+4095	708588k+354293	531441k+265720	25
8191mod16384	16384k+8191	2125764k+1062881	1594323k+797161	27
16383mod32768	32768k+16383	6377292k+3188645	4782969k+2391484	29
32767mod65536	65536k+32767	19131876k+9565937	14348907k+7174453	31
65535mod131072	131072k+65535	57395628k+28697813	43046721k+21523360	33
131071mod262144	262144k+131071	172186884k+86093441	129140163k+64570081	35
262143mod524288	524288k+262143	516560652k+258280325	387420489k+193710244	37
524287mod1048576	1048576k+524287	1549681956k+774840977	1162261467k+581130733	39
1048575mod2097152	2097152k+1048575	4649045868k+2324522933	3486784401k+1743392200	41

2097151mod4194304	4194304k+2097151	13947137604k+6973568801	10460353203k+5230176601	43
4194303mod8388608	8388608k+4194303	41841412812k+20920706405	31381059609k+15690529804	45
8388607mod16777216	16777216k+8388607	125524238436k+62762119217	94143178827k+47071589413	47
16777215mod33554432	33554432k+16777215	376572715308k+188286357653	282429536481k+141214768240	49
33554431mod67108864	67108864k+33554431	1129718145924k+564859072961	847288609443k+423644304721	51
67108863mod134217728	134217728k+67108863	3389154437772k+1694577218885	2541865828329k+1270932914164	53
134217727mod268435456	268435456k+134217727	10167463313316k+5083731656657	7625597484987k+3812798742493	55
268435455mod536870912	536870912k+268435455	30502389939948k+15251194969973	22876792454961k+11438396227480	57
536870911mod1073741824	1073741824k+536870911	91507169819844k+45753584909921	68630377364883k+34315188682441	59
1073741823mod2147483648	2147483648k+1073741823	274521509459532k+137260754729765	205891132094649k+102945566047324	61
2147483647mod4294967296	4294967296k+2147483647	823564528378596k+411782264189297	617673396283947k+308836698141973	63
4294967295mod8589934592	8589934592k+4294967295	2470693585135788k+1235346792567893	1853020188851841k+926510094425920	65
8589934591mod17179869184	17179869184k+8589934591	7412080755407364k+3706040377703681	5559060566555523k+2779530283277761	67
17179869183mod34359738368	34359738368k+17179869183	22236242266222092k+11118121133111046	16677181699666568k+8338590849833284	69
34359738367mod68719476736	68719476736k+34359738367	66708726798666272k+33354363399333136	50031545098999704k+25015772549499852	71
68719476735mod137438953472	137438953472k+68719476735	200126180395998816k+100063090197999408	150094635296999104k+75047317648499552	73
137438953471mod274877906944	274877906944k+137438953471	600378541187996416k+300189270593998208	450283905890997312k+225141952945498656	75
274877906943mod549755813888	549755813888k+274877906943	1801135623563989248k+900567811781994624	1350851717672992000k+675425858836496000	77
549755813887mod1099511627776	1099511627776k+549755813887	5403406870691968000k+2701703435345984000	4052555153018976256k+2026277576509488128	79
1099511627775mod2199023255552	2199023255552k+1099511627775	16210220612075905024k+8105110306037952512	12157665459056928768k+6078832729528464384	81

Conclusion

The Collatz Map or Transformation provides a systematic approach to demonstrating that all odd numbers, through a sequence of mod forms, eventually reduce to a smaller form under the Collatz function. By continuously applying the transformation rules, it becomes evident that every odd number will eventually reach 1, supporting the conjecture's validity.

(2^(n+1))−1 mod 2^(n+2)

There is a very simple formula to use for calculating all the mods and steps to solve for each n. Everything is calculated from 4k+1 (0 Steps) -> 4k+1 (3 Steps) -> 3k+1. All future n's are found using these.

Every time you increase n to n+1 you split the remaining number into 2 new groups of numbers.

Every time you increase n to n+1 the rules change. At n=0 there are 6 patterns numbers follow. At n=1 there are 12 patterns all numbers will follow, and at n=2 there are 24 patterns. Not only do the patterns double by size but they change in their path. Every even number has 2 paths they take and all odd number have 1 path so at MOD(4) you have 4 even paths and 2 odd paths. MOD (8) you have 8 even and 4 odd. And so on.

Visual Aid

MOD2	Number	Binary
0	0	0000 0000
1	1	0000 0001
0	2	0000 0000
1	3	0000 0001
0	4	0000 0000
1	5	0000 0001
0	6	0000 0000
1	7	0000 0001

0	0	0000 0000
0	0	0000 0000
0	0	0000 0000
1	1	0000 0001
1	1	0000 0001
0	0	0000 0000

Here you can see that at MOD(2) we have three paths number will follow:

- Number 0:
  - When you divide 0 by 2
    - It can remain 0
    - It can change to a 1
  - When you apply 3n+1 to 1
    - It will always be 0
- All even numbers always have 2 paths
- All odd numbers have 1 path

MOD4	Number	Binary
0	0	0000 0000
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
0	4	0000 0000
1	5	0000 0001
2	6	0000 0010
3	7	0000 0011

0	0	0000 0000
0	0	0000 0000
0	0	0000 0000
2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
2	2	0000 0010
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
3	3	0000 0011
2	2	0000 0010

Here you can see that at MOD(4) we have six paths number will follow:

- Number 0:
  - When you divide 0 by 2
    - It can remain 0
    - It can change to a 2
  - When you apply 3n+1 to 1
    - It will always be 0
  - When you divide 2
    - It will change to 1
    - It will change to 3
  - When you apply 3n+1 to 3
    - It will always be 2

I will post MOD(8) and MOD(16) to view

- For MOD(8) and MOD(16) as well as the above I run double the numbers so you can see they seem to follow a path that at higher MOD levels it changes
- At each level we see more details into the patterns the numbers follow.
- After MOD(4) nothing new happens in regard to numbers.
  - All 1MOD(4) numbers do a loop reset
  - All 3MOD(4) numbers divide their remainder in half and perform a 3x+1 on the p ower side of the number until they switch to 1MOD(4) number and switch to the other pattern.

MOD8	Number	Binary
0	0	0000 0000
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
4	4	0000 0100
5	5	0000 0101
6	6	0000 0110
7	7	0000 0111
0	8	0000 0000
1	9	0000 0001
2	10	0000 0010
3	11	0000 0011
4	12	0000 0100
5	13	0000 0101
6	14	0000 0110
7	15	0000 0111

0	0	0000 0000
0	0	0000 0000
0	0	0000 0000
4	4	0000 0100
1	1	0000 0001
4	4	0000 0100
2	2	0000 0010
1	1	0000 0001
2	2	0000 0010
5	5	0000 0101
3	3	0000 0011
2	2	0000 0010
4	4	0000 0100
2	2	0000 0010
4	4	0000 0100
6	6	0000 0110
5	5	0000 0101
0	0	0000 0000
6	6	0000 0110
3	3	0000 0011
6	6	0000 0110
7	7	0000 0111
7	7	0000 0111
6	6	0000 0110

MOD16	Number	Binary
0	0	0000 0000
1	1	0000 0001
2	2	0000 0010
3	3	0000 0011
4	4	0000 0100
5	5	0000 0101
6	6	0000 0110
7	7	0000 0111
8	8	0000 1000
9	9	0000 1001
10	10	0000 1010
11	11	0000 1011
12	12	0000 1100
13	13	0000 1101
14	14	0000 1110
15	15	0000 1111
0	16	0000 0000
1	17	0000 0001
2	18	0000 0010
3	19	0000 0011
4	20	0000 0100
5	21	0000 0101
6	22	0000 0110
7	23	0000 0111
8	24	0000 1000
9	25	0000 1001
10	26	0000 1010
11	27	0000 1011
12	28	0000 1100
13	29	0000 1101
14	30	0000 1110
15	31	0000 1111

0	0	0000 0000
0	0	0000 0000
0	0	0000 0000
8	8	0000 1000
1	1	0000 0001
4	4	0000 0100
2	2	0000 0010
1	1	0000 0001
2	2	0000 0010
9	9	0000 1001
3	3	0000 0011
10	10	0000 1010
4	4	0000 0100
2	2	0000 0010
4	4	0000 0100
10	10	0000 1010
5	5	0000 0101
0	0	0000 0000
6	6	0000 0110
3	3	0000 0011
6	6	0000 0110
11	11	0000 1011
7	7	0000 0111
6	6	0000 0110

8	8	0000 1000
4	4	0000 0100
8	8	0000 1000
12	12	0000 1100
9	9	0000 1001
12	12	0000 1100
10	10	0000 1010
5	5	0000 0101
10	10	0000 1010
13	13	0000 1101
11	11	0000 1011
2	2	0000 0010
12	12	0000 1100
6	6	0000 0110
12	12	0000 1100
14	14	0000 1110
13	13	0000 1101
8	8	0000 1000
14	14	0000 1110
7	7	0000 0111
14	14	0000 1110
15	15	0000 1111
15	15	0000 1111
14	14	0000 1110

Summary

1. **All numbers have a MOD power slot**: that divides the powers and the remainder of the number from each other. This is specifically outlined for numbers interacting with 3 as the multiplier.

2. **All odd numbers fall into 2 groups**

    a. 3MOD(4) these number divide the remainder of an odd number by one and move it to the power side
        i. Power multiplied by 3 does a 3x+1 to the power side
        ii. This will continue until the number is 1MOD(4)
    b. 1MOD(4) cannot move the 2^0 bit from the remainder to the power so a loop is created and the number resets to a new power slot
        i. This repeats until we reach the number 1

3. **No 2^0 can be divided either 0 or 1**

    a. Dividing by 0 becomes undefined
    b. Dividing 1 in half creates a loop
        i. No other number has only 1 bit
        ii. All other 1MOD(4) number will reset after dividing the 0 out to:
            1. Another 1MOD(4) and loop reset
            2. A 3MOD(4) and work its way back to another 1MOD(4)

4. **All number fall below themselves**

    a. Even number do this in 1 step
    b. We can use the Transformation Map to show all Odd number fall below themselves
        i. $(2^{(n+1)})-1 \bmod 2^{(n+2)}$ is used for all odd numbers
        ii. n=0 starts us with 4k+1 (0 Steps) -> 4k+1 (3 Steps) -> 3k+1
            1. This solves for 50% of all odd numbers
        iii. n=1 adds 2 steps and solves for another 50% of all remaining odd numbers
        iv. n=2 adds 2 more steps and another 50% of all odd numbers
        v. This continues for all odd numbers of which they are infinite

5. **There can be no other loops**

    a. All 1MOD(4) numbers create the same loop as the number 1 with the exception being they have multiple 1 bits in their binary number

6. **We can build a right sided tree and plot all numbers to a specific location on the tree**

    a. There is only 1 path from 1 to any number in the tree
        i. This couldn't be possible if there if there were any other unknowns
    b. $4 * 0 + 1$ builds are power tree
        i. From there we branch out at all 1MOD(3) even numbers
        ii. $(N*0)+((N-1) / 3)$ is used to branch
            1. The first 1MOD(3) on a branch is a new base which we build a new tree
        iii. Every odd number has a matching even 1MOD(3) number
            1. 4 and 1, 10 and 3, 16 and 5, 22 and 7, this covers all
                a. Even 1MOD(3) numbers
                b. All odd numbers

7. **After the number 7 nothing new happens with any numbers that have any impact on the Collatz Conjecture**

    a. The number 3 has no power to effect the binary bits past 2^2
    b. All the magic is happening in the first 3 bits of all numbers:
        i. All 7MOD(8) will change to 6MOD(8)
        ii. All 5MOD(8) will change to 0MOD(8)
        iii. All 3MOD(8) will change to 2MOD(8)
        iv. All 1MOD(8) will become 4MOD(8) causing a 4,2,1 loop
        v. 6MOD(8) change to 3MOD(8) or back to 7MOD(8)
        vi. 4MOD(8) change to 2MOD(8) or 6MOD(8)
        vii. 2MOD(8) change to 1MOD(8) or 5MOD(8)
        viii. 0MOD(8) change to 4MOD(8) or stay 0MOD(8)
    c. The loop pattern highlighted remains constant no matter the MOD level
    d. If we MOD(16) details will become clearer, but the 4, 2, 1 loop will remain constant through all MOD increases
    e. Every number only has 1 pattern it follows, but until we are at the correct MOD for that number we cannot see it true pattern
    f. Each MOD increase splits out 50% of the numbers into their true pattern