Deciphering the Collatz Conjecture Through Recursion

Glenn Patrick King Ang

Abstract

More than 80 years has passed since the Collatz conjecture has been proposed, but since then there has been no concrete proof as to why it is stuck in the 4 to 2 to 1 loop endlessly. There had been various attempts to find the reason as to why this infinite loop occurs, but there hasn't been any widely recognized proof for the Collatz conjecture.

This paper details a different approach to explain why this infinite loop occurs and at the same time explain why the Collatz conjecture is similar to a computer source code. Section 2, 3, 4, and 5 details the process used to determine the purpose of $2n$ and $n + 1$ functions inside an iterative loop, while section 6 and 7 reveals the logical reasoning behind how the $2n + 2$ algebraic expression was obtained based on $3n + 1$ or $2n + n + 1$.

Section 8 provides proof for $2n + 2$ as the hidden algebraic expression of the Collatz conjecture, while both sections 9 and 10 provides proof that it is also possible to get stuck in the -4 to -2 to -1 loop endlessly when n is any negative odd integer by simply replacing addition with subtraction. Sections 11 provides the key reason why the Collatz conjecture is getting stuck in an endless loop of 4 to 2 to 1.

1. Introduction

Collatz conjecture states:

a. Let n be any positive odd integer greater than 0
b. If n is odd, use $3n + 1$
c. If n is even, use $\frac{n}{2}$
d. Repeat the loop until 1 is reached

2. Recursion and Iterations

In computer science, recursion and iteration are two related methods for solving problems that depends on the solution of a previous computation from the same lines of code.

> ***Recursion is a recursive function that calls itself repeatedly if a specific condition is met and will only stop once the predefined condition for it to stop is met.***
>
> ***Iteration simply means executing the same lines of code over and over until the predefined condition for it to stop is met.***

The very definition of recursion and iteration perfectly describes the Collatz conjecture in its entirety. Therefore, it is now possible to infer that the Collatz conjecture is just like any other computer source code, which can be further examined to determine what the purposes are of each function inside the iterative loop. Representing the Collatz conjecture as a computer algorithm using the python language. See the link for the python code in Appendix A.

Iterative function
filename: 3n _plus_ 1_whileloop.py

```
# Using a while loop for the Collatz conjecture
# created by Glenn Patrick King Ang 10/18/2021

# n = 1 or any positive odd integer > 0

n = 1
previous_n = 0

while n > 0 and previous_n != 2:

    previous_n = n

    if previous_n == 2:
        print("1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = (3 * n) + 1
    elif n % 2 == 0:
        n = n / 2

    print(int(n))
```

Recursive function
filename: 3n _plus_ 1_recursion.py

```
# Creating a function called Collatz conjecture
# created by Glenn Patrick King Ang 10/18/2021

def Collatz_Conjecture(n, previous_n=0):
    if previous_n == 2 or n == 2:
        print("1")
        print("end recursive programming")

    else:
        previous_n = n

        if n % 2 == 1:
            n = (3 * n) + 1
        elif n % 2 == 0:
            n = n / 2

        print(int(n))

        Collatz_Conjecture(n, previous_n)
```

	`# n = 1 or any positive odd integer > 0`
	`n = 1`
	`Collatz_Conjecture(n)`

3. Rewriting the $3n + 1$ algebraic expression

In mathematics, an algebraic expression is an expression composed of variables, constants, and algebraic operations. Furthermore, algebraic expressions can be rewritten more than one way without affecting the original algebraic expression. Rewriting $3n + 1$:

$$\begin{gathered}\boldsymbol{3n + 1} \\ \boldsymbol{= 2n + n + 1}\end{gathered}$$

Rewriting the original algebraic expression is the key to not only better understand what the function of each part of the algebraic expression is for, but also in deciphering the hidden form of the $3n + 1$ algebraic expression inside an iterative loop.

4. Revealing the functions of $2n$ and $n + 1$

In computer programming, when there are multiple functions in the source code of a program, the coder or programmer must look at and study each function to fully comprehend what it is for or what it exactly does. The same logic applies in this instance to fully prove that the $3n + 1$ algebraic expression was designed from the very beginning to create an infinite loop that goes from 4 to 2 to 1 then back to 4 loops.

By using the rewritten algebraic expression, the possibility of evaluating each part of the algebraic expression for its true purpose has finally been opened up. For this proof, the rewritten algebraic expression will be separated into $2n$ and $n + 1$.

Starting with $2n$:

The purpose of $2n$ is to always output an even integer that is always evenly divisible by 2.

Proof: Properties of even and odd numbers.

The sum of two odd numbers is always an even number.

- Rewriting $2n$ as:

$$\boldsymbol{n + n}$$

For $n + 1$:

Its purpose is to always output an even integer that is always evenly divisible by 2.

Proof: Properties of even and odd numbers

The sum of two odd numbers is always an even number.

- Let n be any positive odd integer

$$\boldsymbol{n + 1}$$

This proves that both $2n$ and $n + 1$ is designed to not only always output an even integer but ensure that the integer is always evenly divisible by 2. Since it has already been proven what $2n$ and $n + 1$ does, it is now time to decipher the code hidden inside $3n + 1$.

5. Deciphering the Collatz Conjecture's code

Although using the recursive function is far more ideal, for the purpose of keeping things simple, the while loop will be used throughout the proof instead. Both $2n$ and $n + 1$ will be put inside an iterative loop separately in order to shed light as to the true purpose of each function.

a. Starting with 2n:

This experiment will be using the exact same conditions as the Collatz conjecture, but with a slight variation where $2n$ will be used instead of $3n + 1$. Representing the Collatz conjecture as a computer algorithm using the python language. See the link for the python code in Appendix A. filename: 2n_whileloop.py

<table>
<tr>
<td>

```
# n = 1 or any positive odd
integer > 0
# created by Glenn Patrick King
Ang 10/18/2021

n = 19
previous_n = 0

while n > 0 and previous_n != 2:

    previous_n = n

    if previous_n == 2:
```

</td>
<td>

Example:

n = 19

$if\ odd: 2n$
$n = 2(19)$
$n = 38$

$if\ even: \frac{n}{2}$
$n = \frac{38}{2}$

</td>
</tr>
</table>

<pre>   print("1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = (2 * n)
    elif n % 2 == 0:
        n = n / 2
 	
    print(int(n))</pre>
	$n = 19$
	
	The loop is forever stuck in the 19 to 38 then back to 19 loops.

By looping the code with $2n$ as the algebraic expression, it is now possible to infer that for any positive odd integer, the output will always get stuck in a loop. The loop consists mainly of multiplying and dividing n by 2, which will then revert n back to its original value. Therefore, the only purpose of $2n$ is to convert the positive odd integer into a positive even integer that will always be evenly divisible by 2.

Logically speaking, the true purpose of $\mathbf{2n}$ when combined with $\mathbf{n+1}$ is to ensure that the iterative loop will never stop as long as a positive odd integer exists inside the iterative loop.

b. Starting with n+1:

This experiment will be using the exact same conditions as the Collatz conjecture, but with a slight variation where $n + 1$ will be used instead of $3n + 1$. Representing the Collatz conjecture as a computer algorithm using the python language. See the link for the python code in Appendix A. filename: n_plus_1_whileloop.py

<pre># n = 1 or any positive odd 	
integer > 0
# created by Glenn Patrick King 	
Ang 10/18/2021
 	
n = 1
previous_n = 0
while n > 0 and previous_n != 2:
    previous_n = n
    if previous_n == 2:
        print("1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = n + 1</pre>
	Example:
	
	n = 1
	
	$if\ odd{:}\, n + 1$
	$n = 1 + 1$
	$n = 2$
	
	$if\ even{:}\, \frac{n}{2}$
	$n = \frac{2}{2}$
	$n = 1$

`    elif n % 2 == 0:`
`        n = n / 2`
 	
`    print(int(n))`
	The loop is forever stuck in the 2 to 1 then back to 2 loops.

Executing the iterative loop with more examples:

Example: n = 7 $if\ odd: n = \ n+1$ $n = 7+1$ $n = 8$ $if\ even: n = \frac{n}{2}$ $n = \frac{8}{2}$ $n = 4$ Repeat the loop. Output: $n = \frac{4}{2} = 2$ $n = \frac{2}{2} = 1$ $n = 1+1 = 2$ The loop is forever stuck in the 2 to 1 then back to 2 loops	 n = 23 $if\ odd: n = \ n+1$ $n = 23+1$ $n = 24$ $if\ even: n = \frac{n}{2}$ $n = \frac{24}{2}$ $n = 12$ Repeat the loop. Output: $n = \frac{12}{2} = 6$ $n = \frac{6}{2} = 3$ $n = 3+1 = 4$ $n = \frac{4}{2} = 2$ $n = \frac{2}{2} = 1$ $n = 1+1 = 2$ The loop is forever stuck in the 2 to 1 then back to 2 loops	 n = 85 $if\ odd: n = n+1$ $n = 85+1$ $n = 86$ $if\ even: n = \frac{n}{2}$ $n = \frac{86}{2}$ $n = 43$ Repeat the loop. Output: $n = 43+1 = 44$ $n = \frac{44}{2} = 22$ $n = \frac{22}{2} = 11$ $n = 11+1 = 12$ $n = \frac{12}{2} = 6$ $n = \frac{6}{2} = 3$ $n = 3+1 = 4$ $n = \frac{4}{2} = 2$ $n = \frac{2}{2} = 1$ $n = 1+1 = 2$ The loop is forever stuck in the 2 to 1 then back to 2 loops

By looping the code with $n + 1$ as the algebraic expression, it is now safe to infer that for any positive odd integer, the output will always be forever stuck in the 2 to 1 then back to 2 loops. Therefore, the only purpose of $n + 1$ is to keep adding 1 to any positive odd integer in order to eventually convert the positive odd integer into a positive even integer that will always be evenly divisible by 2.

> ***Logically speaking, the true purpose of $\boldsymbol{n+1}$ is to ensure that the iterative loop will never stop until $\boldsymbol{n}$ becomes 2. By simply using 1 as $\boldsymbol{n}$ inside the iterative loop for $\boldsymbol{n+1}$, the output is always going to be 2 because $\boldsymbol{n+1}$ is designed to eventually convert any positive odd integer inside the iterative loop into 2.***

Therefore, both $2n$ and $n + 1$ was designed from the very beginning not only to ensure that the iterative loop will never stop until all positive odd integers are exhausted, but also for $n + 1$ to eventually convert any positive odd integer into 2 inside the iterative loop.

6. The hidden form of the $\boldsymbol{3n + 1}$ algebraic expression

After proving the functions of the $2n$ and $n + 1$ inside the iterative loop, it is now possible to make a confident inference that will reveal the algebraic expression's hidden form.

Based on observations:

- The main purpose of $2n$ is to turn any positive odd integer into an even integer that is always evenly divisible by 2. This ensures that the iterative loop will never break until all positive odd integers are exhausted inside the iterative loop.

- Moreover, $n + 1$ ensures that any positive odd integer will eventually be reduced to 2 at some point inside the iterative loop. In essence, $n + 1$ is nothing but a function with the end goal of converting any positive odd integer inside the iterative loop into 2.

Therefore, it is safe to confidently infer that $\boldsymbol{n+1}$ is basically just 2.

Now, the hidden form of the $3n + 1$ algebraic expression is finally revealed. By substituting $n + 1$ with 2, the hidden form of the $3n + 1 \; or \; 2n + n + 1$ algebraic expression is:

$$\boldsymbol{2n + 2}$$

7. Proving the logic of inference using both iterative loop and recursive function for $2n + n + 1$

The $2n + 2$ algebraic expression provides a clearer picture as to how the Collatz conjecture truly works. In essence, $2n$ mainly functions as a multiplier to ensure that the positive odd integer will always be evenly divisible by 2, whereas n+1's sole objective is to eventually convert any positive odd integer into 2. In order to prove the logic of the inference, a python code was constructed using both iterative loop and recursive function for 2n+n+1 or more commonly known as 3n+1. Representing the Collatz conjecture as a computer algorithm using the python language.

See the link for the python code in Appendix A. filename: 2n_plus_n_plus_1_recursion.py

```
# n = 1 or any positive odd integer > 0
# created by Glenn Patrick King Ang 10/18/2021

n = 1
previous_n = 0


def n_plus_1(n_recursion, previous_n_recursion=0):
    if previous_n_recursion == 2 or n_recursion ==
2:
        return n_recursion
    else:
        previous_n_recursion = n_recursion
        if n_recursion % 2 == 1:
            n_recursion = n_recursion + 1
        elif n_recursion % 2 == 0:
            n_recursion = n_recursion / 2

        return n_plus_1(n_recursion,
previous_n_recursion)

while n > 0 and previous_n != 2:

    previous_n = n

    if previous_n == 2:
        print("end while loop for 2n + recursion at
1")
        break
    elif n % 2 == 1:
        n = (2 * n) + n_plus_1(n)
    elif n % 2 == 0:
        n = n / 2
    print(f"While loop for 2n + recursion (n+1):
{int(n)}")
```

$3n + 1 = 2n + n + 1$

Using a while loop for $2n + n + 1$

Replacing $n + 1$ with a recursive function called n_plus_1

This python code for the Collatz conjecture provides a conclusive proof that $n + 1$ is basically just 2.

Therefore, this also proves that the $2n + 2$ algebraic expression for the Collatz conjecture is its hidden form.

After obtaining the hidden form of the Collatz conjecture's $3n + 1$ algebraic expression, it is now possible to prove that all positive odd integer will always end up in the 4 to 2 to 1 loop using $2n + 2$.

8. All positive odd integers will loop 4 to 2 to 1 endlessly using $2n + 2$

Using the same conditions as the Collatz conjecture:

a. Let n be any positive odd integer greater than 0
b. If n is odd, use $2n + 2$
c. If n is even, use $\frac{n}{2}$
d. Repeat the loop until 1 is reached

Representing the Collatz conjecture as a computer algorithm using python language. See the link for the python code in Appendix A. filename: 2n_plus_2_whileloop.py

```
# n = 1 or any positive odd
integer > 0
# created by Glenn Patrick King
Ang 10/18/2021

n = 7
previous_n = 0

while n > 0 and previous_n != 2:

    previous_n = n

    if previous_n == 2:
        print("1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = (2 * n) + 2
    elif n % 2 == 0:
        n = n / 2

    print(int(n))
```

n = 7

$if\ odd: n = 2n + 2$
$n = 2(7) + 2$
$n = 16$

$if\ even: n = \frac{n}{2}$
$n = \frac{16}{2}$
$n = 8$

Repeat the loop.

Output:
$n = \frac{8}{2} = 4$
$n = \frac{4}{2} = 2$
$n = \frac{2}{2} = 1$
$n = 2(1) + 2 = 4$

The loop is forever stuck in the 4 to 2 to 1 then back to 4 loops

9. Further Proof: All negative odd integers will loop -4 to -2 to -1 endlessly using $3n - 1$

It is now possible to prove that the Collatz conjecture also works with all negative odd integers using $3n + 1$ as long as some changes are made. Since it has already been proven that the only function of $n + 1$ is to output 2, it is now safe to infer that to get the -4 to -2 to -1 then back to -4 loops, a simple switch from addition to subtraction will suffice to make the $3n + 1$ work for all negative odd integers. Therefore, the new algebraic expression is:

$$\mathbf{3n - 1}$$

Using the same conditions as the Collatz conjecture:

a. Let n be any negative odd integer less than 0
b. If n is odd, use $3n - 1$
c. If n is even, use $\frac{n}{2}$
d. Repeat the loop until -1 is reached

Representing the Collatz conjecture as a computer algorithm using python language. See the link for the python code in Appendix A. filename: 3n_minus_1_whileloop.py

<table>
<tr>
<td>

```
# n = -1 or any negative odd
integer < 0
# created by Glenn Patrick King
Ang 10/18/2021

n = -1
previous_n = 0

while n < 0 and previous_n != -2:

    previous_n = n

    if previous_n == -2:
        print("-1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = (3 * n) - 1
    elif n % 2 == 0:
        n = n / 2

    print(int(n))
```

</td>
<td>

Example:

n = -1

$if\ odd: 3n - 1$
$n = 3(-1) - 1 = -3 - 1$
$n = -4$

$if\ even: \frac{n}{2}$
$n = \frac{-4}{2} = -2$

$n = \frac{-2}{2} = -1$
$n = 3(-1) - 1 = -4$

The loop is forever stuck in the -4 to -2 to -1 then back to -4 loops. More examples are provided in Appendix B

</td>
</tr>
</table>

10. Further proof: $2n + 2$ will also adhere to the same changes made on the $3n + 1$ for all negative odd integers

Since it has already been proven that $3n + 1$ works on all negative odd integer by simply replacing addition with subtraction, it is now possible to infer that its hidden form will also adhere to the same changes as well. To get the -4 to -2 to -1 then back to -4 loops with $2n + 2$, replacing addition with subtraction will give a new algebraic expression:

$$\mathbf{2n - 2}$$

Using the same conditions as the Collatz conjecture:

a. Let n be any negative odd integer less than 0
b. If n is odd, use $2n - 2$
c. If n is even, use $\frac{n}{2}$
d. Repeat the loop until you reach -1

Representing the Collatz conjecture into a computer algorithm using python language. See the link for the python code in Appendix A. filename: 2n_minus_2_whileloop.py

<table>
<tr>
<td>

```
# n = -1 or any negative odd
integer < 0
# created by Glenn Patrick King
Ang 10/18/2021

n = -1
previous_n = 0

while n < 0 and previous_n != -2:

    previous_n = n

    if previous_n == -2:
        print("-1")
        print("end loop")
        break
    elif n % 2 == 1:
        n = (2 * n) - 2
    elif n % 2 == 0:
        n = n / 2

    print(int(n))
```

</td>
<td>
Example:

n = -1

$if\ odd: 2n - 2$
$n = 2(-1) - 2 = -2 - 2$
$n = -4$

$if\ even: \frac{n}{2}$
$n = \frac{-4}{2} = -2$

$n = \frac{-2}{2} = -1$
$n = 2(-1) - 2 = -4$

The loop is forever stuck in the -4 to -2 to -1 then back to -4 loops.
</td>
</tr>
</table>

11. Key reason why the Collatz conjecture is always getting stuck in an endless loop of 4 to 2 to 1

The only reason why all positive odd integers are always getting stuck in the 4 to 2 to 1 loop endlessly is because of $2n$ in $2n + n + 1$. The $2n$ ensures that any positive odd integer will always be converted to an even number that is always greater than or equal to 2 in order for it to be evenly divisible by 2. This can be proven by simply removing $2n$ from $2n + n + 1$, the iterative loop will always get stuck in the 2 to 1 then back to 2 loops for any positive odd integer.

Moreover, the only reason why all positive odd integers are always getting stuck in the 4 to 2 to 1 loop endlessly is because of 3 in $3n + 1$. The $3n + 1$ ensures any positive odd integer will always be turned to an even number that is always greater than 2 in order for it to be evenly divisible by 2. This can be proven by simply removing 3 from $3n + 1$, the iterative loop will always get stuck in the 2 to 1 then back to 2 loops for any positive odd integer.

Therefore, the Collatz conjecture is going to loop endlessly for any positive integer because the $3n + 1$ algebraic expression used in the iterative loop was designed not only to eventually reduce any positive odd integer to 2, but to also make sure that once the even integer 2 is further reduced to 1, the next positive even number created is 4.

12. Conclusion

Perceiving the Collatz conjecture as a computer code revealed the hidden function deep inside the seemingly simple $3n + 1$ algebraic expression. The beauty and complexity of this problem lies on the need to discern the patterns that emerges from evaluating the intent or purpose of the programmer and or designer for each function inside the iterative loop.

From all the proof that has been given, it is now possible to confirm that:

a. All positive odd integer will loop 4 to 2 to 1 then back to 4 endlessly given that the equation is $3n + 1$ or its hidden form $2n + 2$.

b. All negative odd integer will loop -4 to -2 to -1 then back to -4 endlessly given that the equation is $3n - 1$ or its hidden form $2n - 2$.

c. The Collatz conjecture is looping endlessly because the $3n + 1$ algebraic expression used in the iterative loop was designed not only to eventually reduce any positive odd integer to 2, but also to make sure that once the even integer 2 is further reduced to 1, the next positive even number created is 4.

Acknowledgements

I would like to express my gratitude toward my parents for their continuous love, support, and belief in me as I find my way in life. This research paper would have not been possible without their love and support.

I would also like to express my gratitude toward my wife Natassia for her continuous love, understanding, and constant support that she has given me ever since. Thank you for always being a source of inspiration and a partner that I can always depend on.

Glenn Patrick King Ang

https://github.com/07231985

glenn.patrick.king.ang@outlook.com (preferred)

glenn.p.ang@alumni.uts.edu.au (not affiliated, only an alumni)

Appendix A:

Link to the GitHub repository containing the python codes used in this research paper:

https://github.com/07231985/collatzconjecture

Appendix B:

More examples:

a. n = -7
b. n = -23
c. c = -85

a. n = -7 $if\ odd: n = 3n - 1$ $n = 3(-7) - 1$ $n = -21 - 1$ $n = -22$ $if\ even: n = \frac{n}{2}$ $n = \frac{-22}{2}$ $n = -11$ Repeat the loop. Output: $n = 3(-11) - 1 = -34$ $n = \frac{-34}{2} = -17$ $n = 3(-17) - 1 = -52$ $n = \frac{-52}{2} = -26$ $n = \frac{-26}{2} = -13$ $n = 3(-13) - 1 = -40$ $n = \frac{-40}{2} = -20$ $n = \frac{-20}{2} = -10$ $n = \frac{-10}{2} = -5$ $n = 3(-5) - 1 = -16$ $n = \frac{-16}{2} = -8$ $n = \frac{-8}{2} = -4$ $n = \frac{-4}{2} = -2$ $n = \frac{-2}{2} = -1$ $n = 3(-1) - 1 = -4$ The loop is forever stuck in the -4 to -2 to -1 then back to -4 loops	b. n = -23 $if\ odd: n = 3n - 1$ $n = 3(-23) - 1$ $n = -70$ $if\ even: n = \frac{n}{2}$ $n = \frac{-70}{2}$ $n = -35$ Repeat the loop. Output: $n = 3(-35) - 1 = -106$ $n = \frac{-106}{2} = -53$ $n = 3(-53) - 1 = -160$ $n = \frac{-160}{2} = -80$ $n = \frac{-80}{2} = -40$ $n = \frac{-40}{2} = -20$ $n = \frac{-20}{2} = -10$ $n = \frac{-10}{2} = -5$ $n = 3(-5) - 1 = -16$ $n = \frac{-16}{2} = -8$ $n = \frac{-8}{2} = -4$ $n = \frac{-4}{2} = -2$ $n = \frac{-2}{2} = -1$ $n = 3(-1) - 1 = -4$ The loop is forever stuck in the -4 to -2 to -1 then back to -4 loops	c. n = -85 $if\ odd: n = 3n - 1$ $n = 3(-85) - 1$ $n = -256$ $if\ even: n = \frac{n}{2}$ $n = \frac{-256}{2}$ $n = -128$ Repeat the loop. Output: $n = \frac{-128}{2} = -64$ $n = \frac{-64}{2} = -32$ $n = \frac{-32}{2} = -16$ $n = \frac{-16}{2} = -8$ $n = \frac{-8}{2} = -4$ $n = \frac{-4}{2} = -2$ $n = \frac{-2}{2} = -1$ $n = 3(-1) - 1 = -4$ The loop is forever stuck in the -4 to -2 to -1 then back to -4 loops