Collatz - Conjecture

Rudolph, Leon

August 2, 2022

1 Erhebung von Daten

Das Collatzproblem lässt sich mittels folgender Funktion $f(n)$ beschreiben

$$f(n) = \begin{cases} 3n+1 & \text{wenn } n \mod 2 = 1 \\ \frac{n}{2} & \text{sonst} \end{cases}$$

Um diese Funktion näher zu betrachen habe ich mir hierfür zusätzlich ein Pythonskript erstellt um die erhobenen Daten schneller zu berechnen. Das vollständiges verstehen des Skript ist hierbei für den Leser nicht erfordlich, da die einzelnen Schritte stets erklärt werden. Es dient hierbei lediglich als Anhang um die einzelnen Berechnungsschritte schlüssig nachvollziehen zu können.

```
1  import matplotlib.pyplot as plt
2
3  pair = []
4
5  def collatz(x: int, index: int):
6      if x == 1:
7          return
8      elif x % 2 == 0:
9          pair.__getitem__(index).__getitem__(1).append(int(x / 2))
10         collatz(x / 2, index)
11     else:
12         pair.__getitem__(index).__getitem__(1).append(int(3 * x + 1))
13         collatz(3 * x + 1, index)
14
15
16 def printPair():
17     for a, b in pair:
18         print(a, b)
19
20
21 def groupPair(x: int):
22     for a, b in pair:
23         if b.__contains__(x):
24             print(a, b)
25
26 def plotPairByGroup(x: int):
27     fig1 = plt.figure()
28     ax1 = fig1.add_subplot(111)
29     for a, b in pair:
30         if b.__contains__(x):
31             ax1.plot(b)
```

```
32
33      plt.show()
34
35
36  def plotPair():
37      fig1 = plt.figure()
38      ax1 = fig1.add_subplot(111)
39
40      for a, b in pair:
41          ax1.plot(b)
42
43      plt.show()
44
45
46  def main(startIndex: int, endIndex: int):
47      for i in range(startIndex, endIndex):
48          pair.append((i, []))
49          collatz(i, i - startIndex)
50
51      printPair()
52      plotPair()
```

Als Ausgabe nach dem ausführen des Skripts erhalten wir folgendes

```
3 [10, 5, 16, 8, 4, 2, 1]
4 [2, 1]
5 [16, 8, 4, 2, 1]
6 [3, 10, 5, 16, 8, 4, 2, 1]
7 [22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
8 [4, 2, 1]
9 [28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
10 [5, 16, 8, 4, 2, 1]
11 [34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
12 [6, 3, 10, 5, 16, 8, 4, 2, 1]
13 [40, 20, 10, 5, 16, 8, 4, 2, 1]
14 [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
...
```

Hierbei zeigt sich auf den ersten Blick eine Gruppenbildung zwischen den verschiedenen Werten. Alle Werte die Elemente von 2^x sind, können stets ohne Anwendung von $3x+1$ auf den Zyklus $4 \to 2 \to 1 \to 4 \to ....$ zurückgeführt werden. Sind die Werte nicht Element von 2^x, so zeigt sich hierbei eine gewisse Abweichung bezüglich der Funktion 2^x. Um diese Abweichungen zu filtern definiere ich mir zwei Zusatzfunktionen die bereits im obigen Skript angegeben sind. Jeweils mit "plotPairByGroup(x: int)" und "groupPair(x: int)". Im folgenden wurden die Werte nach einer Gruppe gefiltert, die eine 5 enthielten.

```
3 [10, 5, 16, 8, 4, 2, 1]
6 [3, 10, 5, 16, 8, 4, 2, 1]
7 [22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
9 [28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
10 [5, 16, 8, 4, 2, 1]
11 [34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
12 [6, 3, 10, 5, 16, 8, 4, 2, 1]
13 [40, 20, 10, 5, 16, 8, 4, 2, 1]
14 [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
15 [46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1]
17 [52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
18 [9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
19 [58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
20 [10, 5, 16, 8, 4, 2, 1]
22 [11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

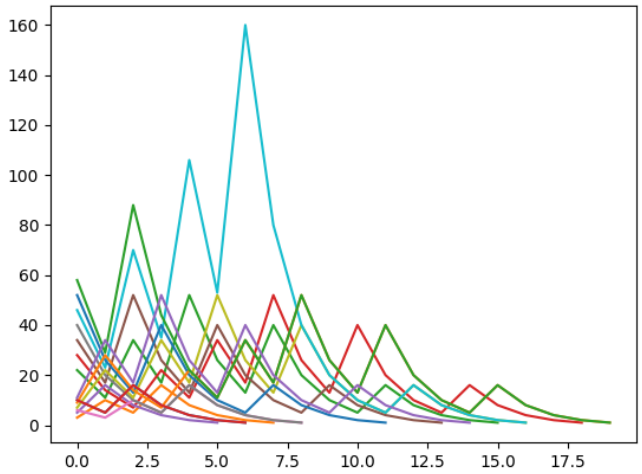


Figure 1: Gruppenbildung mit dem Wert 5

Die X-Achse aus dem Graphen beschreibt dabei die Anzahl der Elemente in der Liste, die Y-Achse die Wertigkeit der jeweiligen Stelle. Hierbei ist interessant zu beobachten, dass sich die Listen stets fortführen lassen. Auch wenn 3 mittels folgenden Werten terminiert,

```
3 [10, 5, 16, 8, 4, 2, 1]
```

so tauchen diese Werte in späteren Listen wieder auf.

```
6 [3, 10, 5, 16, 8, 4, 2, 1]
```

```
12 [6, 3, 10, 5, 16, 8, 4, 2, 1]
...
```

Um dahingehend eine allgemeine Funktion aufzustellen, die mir die Liste berechnet, verwende ich die Umkehrfunktion von $f(n)$ mit

$$f^{-1}(n) = \begin{cases} f^{-1}(\frac{n-1}{3}) & \text{wenn } \frac{n-1}{3} \mod 2 = 1 \\ f^{-1}(2n) & \text{sonst} \end{cases}$$

Im weiteren Teil des Papers werde ich ausschließlich die Umkehrfunktion näher betrachten und ihr Verhalten bei beiden Fällen untersuchen.

2 Verhalten der Umkehrfunktion

Für die Umkehrfunktion $f^{-1}(n)$ habe ich ebenfalls wieder ein Pythonskript erstellt um mir die Berechnungen der einzelnen Listen zu erleichtern. Hierbei ist es ebenfalls nicht erforderlich das Skript vollständig zu verstehen

```
import matplotlib.pyplot as plt

rebuildedCollatz = [(0, [])]


def printCollatz():
    for a, b in rebuildedCollatz:
        if a == 2:
            print(b)


def plotPairByRebuild():
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)
    for a, b in rebuildedCollatz:
        if a == 2:
            ax1.plot(b, 'ro')

    plt.show()


def listContainsArray(array):
    for a, b in rebuildedCollatz:
        if b == array:
            return True
        else:
            return False


def addLinkArray(array):
    temp = []
    for a in array:
        temp.append(a)
    return temp
```

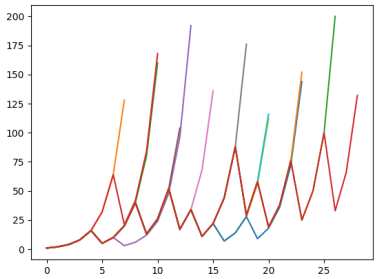
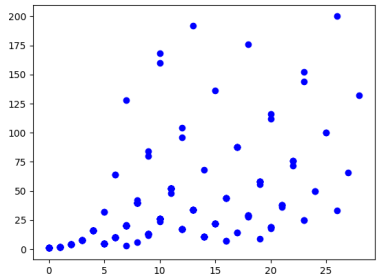
```
def rebuildCollatzByNumber(x: int, array, index: int, withLink: bool):
    if listContainsArray(array):
        return

    if x > 100:
        rebuildedCollatz.append((index, array))
        return

    if ((x - 1) / 3) % 2 == 1:
        if not array.__contains__(int((x - 1) / 3)):
            temp = []
            if withLink:
                temp = addLinkArray(array)
            temp.append(int((x - 1) / 3))
            rebuildCollatzByNumber(int((x - 1) / 3), temp, index+1, withLink)

    array.append(x * 2)
    rebuildCollatzByNumber(x * 2, array, index, withLink)
```

Mit dem Aufrufen folgender Funktion lässt sich das Geschehen näher untersuchen.

```
import RebuildCollatz as rbCollatz

rbCollatz.rebuildCollatzByNumber(1, [1], 0, True)
rbCollatz.rebuildedCollatz.sort()

rbCollatz.plotPairByRebuild()
rbCollatz.printCollatz()
```

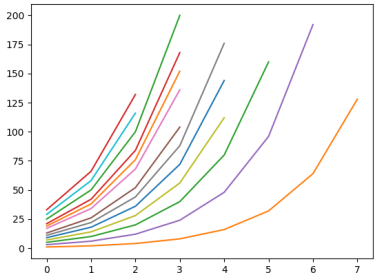
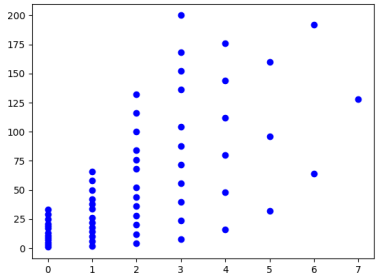
Als Ausgabe erhalten wir mit $x > 100$ als Abbruchbedingung folgende Werte

```
(0, [])
(0, [1, 2, 4, 8, 16, 32, 64, 128])
(1, [1, 2, 4, 8, 16, 5, 10, 20, 40, 80, 160])
(1, [1, 2, 4, 8, 16, 32, 64, 21, 42, 84, 168])
(2, [1, 2, 4, 8, 16, 5, 10, 3, 6, 12, 24, 48, 96, 192])
(2, [1, 2, 4, 8, 16, 5, 10, 20, 40, 13, 26, 52, 104])
(3, [1, 2, 4, 8, 16, 5, 10, 20, 40, 13, 26, 52, 17, 34, 68, 136])
(4, [1, 2, 4, 8, 16, 5, 10, 20, 40, 13, 26, 52, 17, 34, 11, 22, 44, 88, 176])
```

Hierbei gilt es zu beachten, dass der linke Graph, eine falsche Darstellung unserer Umkehrfunktion ist, da wir uns ausschließlich in $\mathbb{N}$ bewegen ist der rechte Graph die korrekte Darstellung.
Entfernt man die Verlinkungen und betrachten die Listen einzeln bei ihren Bruchstellen so erhalten wir dafür folgende Ergebnisse

```
(0, [])
(0, [1, 2, 4, 8, 16, 32, 64, 128])
(1, [5, 10, 20, 40, 80, 160])
(1, [21, 42, 84, 168])
(2, [3, 6, 12, 24, 48, 96, 192])
(2, [13, 26, 52, 104])
(3, [17, 34, 68, 136])
(4, [11, 22, 44, 88, 176])
(5, [7, 14, 28, 56, 112])
(5, [29, 58, 116])
(6, [9, 18, 36, 72, 144])
(6, [19, 38, 76, 152])
(7, [25, 50, 100, 200])
(8, [33, 66, 132])
```

So lässt sich unsere Umkehrfunktion wie folt umformulieren.

$$f^{-1}(n) = \begin{cases} f^{-1}(\frac{n*2^x-1}{3}) & \text{wenn } \frac{n*2^x-1}{3} \mod 2 = 1 \\ n * 2^x \end{cases}$$

Da für jedes $n \in \mathbb{N}$ nach einem Bruch stets 2^x berechnet wird erhalten wir folgendes

$$f_k^{-1}(n) = f_k^{-1}(2 * f_{k-1}^{-1}(...2 * f_1^{-1}(2 * n)...))$$

Hierbei ist es völlig unabhängig ob unser $\frac{n*2^x-1}{3} \mod 2 = 1$ gilt. So existiert beispielsweise im Collatz Problem der Pfad $5 \to 10 \to 20 \to 40 \to ...$ gleichzeitig existiert für unseren Wert 5 ebenfalls der Pfad $5 \to 10 \to 3 \to 6 \to ....$ Folglich wird unser n stets weiter berechnet für jeden weiteren Wert mit $n * 2^x$.

Aus dieser Vereinfachung folgen ein paar interessante Eigenschaften für $f^{-1}(n)$. Um das Problem weiter mathematisch zu beschreiben, definiere ich mir ein paar zusätzliche Symbole.

So ist P_{Coll} abgekürzt das Collatz-Problem.
$L_{P_{Coll}}$ ist die Sprache von Collatz oder anders geschrieben
$L_{P_{Coll}} \subseteq \{0,1,2,3,4,5,6,7,8,9\}^*$
$A_{n \mod 2=0} = \{n \in \mathbb{N} | n \mod 2 = 0\}$
$B_{(n+1) \mod 2=0} = \{n \in \mathbb{N} | (n+1) \mod 2 = 0\}$

Folglich gelten für die oben genannten Mengen auch folgende Eigenschaften.

(1) $A_{n \mod 2=0} \cup B_{(n+1) \mod 2=0} = \mathbb{N}$
(2) $A_{n \mod 2=0} \cap B_{(n+1) \mod 2=0} = \emptyset$

$A_{n \mod 2=0} \cup B_{(n+1) \mod 2=0} = L_{P_{Coll}}$ gilt hierbei nicht, dass wäre ein trugschluss. Denn, wenn $A_{n \mod 2=0} \cup B_{(n+1) \mod 2=0} = L_{P_{Coll}}$, so wäre aus (1), $\mathbb{N} = L_{P_{Coll}}$, da bereits $A_{n \mod 2=0} \cup B_{(n+1) \mod 2=0} = \mathbb{N}$.
Nach dem Collatz-Problem gilt es allerdings zu zeigen, dass entweder (3) oder (4) gilt.

(3) $L_{P_{Coll}} = \mathbb{N}$
(4) $L_{P_{Coll}} \subset \mathbb{N}$

Da wir aus der Umformulierung der Umkehrfunktion wissen, dass für jede ungerade Zahl n, $n \in L_{P_{Coll}}$, auch $n * 2^x \in L_{P_{Coll}}$. Lässt sich daraus folgern, dass jedes 2^x, mit $x \geq 1$ Vielache von n auch stets gerade ist, da

$$2^x n \mod 2 = 0$$
$$2^x n = 2m$$
$$2^{x-1} n = m$$

Es wäre jetzt äußert trügerisch wenn man sagen könnte $\forall b \in B_{(n+1) \mod 2=0}$, ist $b \in L_{P_{Coll}}$. Doch wäre diese Annahme gegeben so ließe sich mit Hilfe der Primfaktorzerlegung zeigen, dass folglich auch $\forall a \in A_{n \mod 2=0}$ mit $a \in L_{P_{Coll}}$ gilt. Um diese trügerische Aussage zu beweisen verwende ich hierfür als Hilfestellung folgende Sätze.

(5)

$$y = 2m + 1, x = 2k + 1$$
$$y * x = (2m + 1) * (2k + 1)$$
$$y * x = 4mk + 2m - 2k - 1$$
$$y * x = 2(2mk + m - k) - 1$$

(6)

$$y = 2m - 1, x = 2k$$
$$y * x = (2m - 1) * 2k$$
$$y * x = 4mk - 2k$$

Unsere Aussage (5) lässt sich hierbei noch erweitern.

$$x_i = 2m_i + 1, \text{ mit } i, k \in \mathbb{N}, i \leq k$$
$$\exists y \in B_{(n+1) \mod 2=1}, \text{ mit } y = x_0 * x_1 * ... * x_k$$

$$(x_0 * x_1) * ... * (x_{k-1} * x_k) \text{ wenn } k \mod 2 = 0$$
$$(x_0 * x_1) * ... * (x_{k-2} * x_{k-1}) * x_k \text{ wenn } k \mod 2 = 1$$

$\forall p \in B_{(n+1) \mod 2=1}, p \leq k : (x_{p-1} * x_p) = l_o$ nach (5) ist $l_o \in B_{(n+1) \mod 2=1}$ mit

$$o \in \mathbb{N}, o \leq h(k) = \begin{cases} \frac{k}{2} & \text{wenn } k \mod 2 = 0 \\ \frac{k-1}{2} & \text{sonst} \end{cases}$$

Dieser Schritt mit Klammersetzen und anwenden von (5) lässt sich
für die komplette Formel y fortsetzen bis nur noch eine Variable z übrig bleibt
da $(l_0 * ... * l_o) = z, \text{ mit } z \in B_{(n+1) \mod 2=1}$
folgt hieraus, dass für $k \mod 2 = 1 : z * x_k \in B_{(n+1) \mod 2=1}$

Aus dem Satz der Primfaktorzerlegung geht hervor.
$\forall n \in \mathbb{N}, \exists p_0, \exists p_1, ... \exists p_k \in \mathbb{P}$, sodass $n = p_0^{x_0} * p_1^{x_1} * ... * p_k^{x_k}$, mit $k, x \in \mathbb{N}$

Daraus lässt sich folgendes Schlussfolgern, die einzige gerade Zahl in $\mathbb{P}$ ist 2. Wenn das nicht der Fall wäre, so wäre unser beliebiges $p \in \mathbb{P}$, keine Primzahl, da das der Definition einer Primzahl widersprechen würde, dass p nur durch p und 1 teilbar ist. Wenn p aber nicht durch 2 teilbar ist, so zeigt sich, dass p stets ungerade ist. Heißt für irgendein beliebiges $i \leq k$ in $n = p_0^{x_0} * p_1^{x_1} * ... * p_k^{x_k}$, muss ein $p_i^{x_i} \mod 2 = 0$ sein, wenn $n \mod 2 = 0$. So ist aus der Erweiterung von (5) unser Zähler in $n = \frac{p_0^{x_0} * p_1^{x_1} * ... * p_k^{x_k}}{p_i^{x_i}}$ stets ungerade, da wir im Bruch unsere einzige gerade Zahl entfernt haben. Wenden wir als Hilfestellung die Funktion $g(n)$ an.

$$g(n) = \begin{cases} g(\frac{n}{2}) & \text{wenn } n \mod 2 = 0 \\ n & \text{sonst} \end{cases}$$

So bereinigt unsere Hilfsfunktion $g(n)$ unsere beliebig gewähltes n und führt es stets auf eine ungerade Zahl zurück. Da wir oben trügerisch angenommen haben $\forall b \in B_{(n+1) \mod 2=0}$, ist $b \in L_{P_{Coll}}$, wissen wir mittels $g(n)$ gilt auch $\forall a \in A_{n \mod 2=0}$ ist $a \in L_{P_{Coll}}$, da jede gerade Zahl unter Anwendung von $g(n)$ stets auf eine ungerade Zahl zurückgeführt werden kann, und dieses $b_a \in L_{P_{Coll}}$. Dieses b_a wird unter Anwendung der Umkehrfunktion stets wieder jedes gerade 2^x Vielfache von b_a herleitet. Daraus lässt sich Schlussfolgern $L_{P_{Coll}} = A_{n \mod 2=0} \cup B_{(n+1) \mod 2=0}$ und demnach nach (1) gilt (3) $L_{P_{Coll}} = \mathbb{N}$.

Im weiteren Teil des Papers werde ich mich noch mit der folgenden Eigenschaft der Funktion $f^{-1}(n)$ auseinandersetzen.

$$f^{-1}(\frac{n-1}{3}) \text{ wenn } \left(\frac{n-1}{3}\right) \mod 2 = 1$$

Bis jetzt haben wir uns ausschließlich mit $n * 2^x$ auseinandergesetzt. Können aber mit Sicherheit schon sagen, dass

(7) $L_{P_{Coll}} = \mathbb{N}$, wenn $(L_{P_{Coll}} \setminus A_{n \mod 2=0}) = B_{(n+1) \mod 2=0}$
(8) $L_{P_{Coll}} = \mathbb{N}$, wenn $(L_{P_{Coll}} \setminus B_{(n+1) \mod 2=0}) = A_{n \mod 2=0}$
(9)$L_{P_{Coll}} \subset \mathbb{N}$, wenn $(L_{P_{Coll}} \setminus A_{n \mod 2=0}) \subset B_{(n+1) \mod 2=0}$
(10) $L_{P_{Coll}} \subset \mathbb{N}$, wenn $(L_{P_{Coll}} \setminus B_{(n+1) \mod 2=0}) \subset A_{n \mod 2=0}$

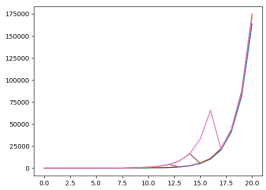
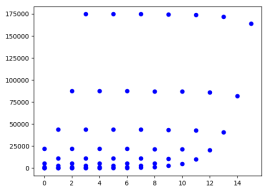
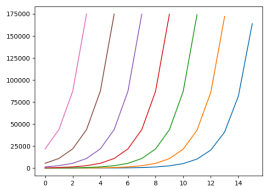
Um weiter zu verstehen wie sich ungerade Zahlen in $f^{-1}(n)$ verhalten, wissen wir bereits, dass bei jedem weiteren rekursiven Funktionsaufruf n mod 2 = 1 gilt. Im Folgenden werde ich den obigen Python-Code modellieren und weitere Eigenschaft der Umkehrfunktion $f^{-1}(n)$ untersuchen. Unsere Funktion printCollatz() und plotPairByRebuild() ändere ich wie folgt, dass jeweils nur noch die erste Bruchstelle ausgegeben wird.

```
def printCollatz():
    for a, b in rebuildedCollatz:
        if a == 1:
            print(b)


def plotPairByRebuild():
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)
    for a, b in rebuildedCollatz:
        if a == 1:
            ax1.plot(b)

    plt.show()
```

Ausgabe nach dem Aufruf

```
[1, 2, 4, 8, 16, 5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, ...]
[1, 2, 4, 8, 16, 32, 64, 21, 42, 84, 168, 336, 672, 1344, 2688, 5376, ...]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 85, 170, 340, 680, 1360, 2720, 5440, ...]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 341, 682, 1364, 2728, 5456, ...]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 1365, 2730, 5460, ...]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 5461, ...]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, ...]
```

```
[5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960, 81920, 163840]
[21, 42, 84, 168, 336, 672, 1344, 2688, 5376, 10752, 21504, 43008, 86016, 172032]
[85, 170, 340, 680, 1360, 2720, 5440, 10880, 21760, 43520, 87040, 174080]
[341, 682, 1364, 2728, 5456, 10912, 21824, 43648, 87296, 174592]
[1365, 2730, 5460, 10920, 21840, 43680, 87360, 174720]
[5461, 10922, 21844, 43688, 87376, 174752]
[21845, 43690, 87380, 174760]
```

Betrachtet man die ersten Bruchstellen genauer, so lässt sich hierbei folgende Vermutung aufstellen. Auch wenn das folgende noch keine bewiesene Aussage ist, so ist für unsere n_0 mit $n_0 = 1$, jeder weitere Wert im Abstand $4n_{i-1} + 1$. Schaut man sich die Exponenten von 2^x an, so gilt anscheinend das $x \in A_{n \mod 2=0}$. Der Bruch bei 2^2, wäre hier bei 4 und endet wie bereits bekannt im Zyklus $4 \to 2 \to 1 \to 4 \to ...$ Folglich könnte man folgende Vermutung aufstellen, mit $x > 1$.

$$\forall x \in A_{n \mod 2=0} : \frac{2^x - 1}{3} \mod 2 = 1$$
$$\frac{2^x - 1}{3} = 2m + 1$$
$$2^x - 1 = 6m + 3$$
$$2^x - 2^2 = 2 * 3m$$
$$\text{Da } x \in A_{n \mod 2=0} \text{ ist folglich } x = 2k, \text{ mit } k \in \mathbb{N}$$
$$2^{2k} - 2^2 = 2 * 3m$$
$$2^{2k-1} - 2 = 3m$$

Um die Gleichheit in unserer Gleichung besser zu verstehen, müssen wir unser m näher betrachten. Wir konnten unser x bereits abändern zu $x = 2k$, da unser $x \in A_{n \mod 2=0}$. Unser m ist hierbei immer noch beliebig gewählt und es muss noch durch eine passende Funktion beschrieben werden. Schaut man sich hierbei den linken Teil der Formel an so erhält man als Ergebnisse von k jeweils, 2, 8, 32, ... minus zwei und anschließend durch drei geteilt ergibt sich hierbei die Reihe, 0, 2, 10, 42, 170, ... Um die Reihe zu beschreiben, ergibt sich hierbei wieder ein ähnliches Muster, wie in der Collatzfunktion beim ersten Bruch. Zur Erinnerung waren unsere Werte beim ersten Bruch Elemente der Reihe 0, 1, 5, 21, 85, 341, ... Multipliziert man diese besagte Reihe mit zwei so ergibt sich unser gewünschtes Ergebnis. Um unsere Reihe zu definieren, verwende ich hierfür r_n mit folgenden Werten.

$$r_0 = 4^0$$
$$r_1 = 4^1 + 4^0$$
$$r_k = 4^k + 4^{k-1} + ... + 4^0$$

So lässt sich unser r_k zu der Summer abändern

$$\sum_{i=0}^{k} r_k$$

Unsere Reihe lässt sich dann mit Hilfe der Geometrische Reihe beschreiben

$$\sum_{i=0}^{k} q^k = \frac{q^{k+1}-1}{q-1}, \text{ für } q > 1$$

$$\sum_{i=0}^{k} 4^k = \frac{4^{k+1}-1}{3}$$

So lässt sich unsere Vereinfachung der Gleichung für $\frac{2^x-1}{3} \mod 2 = 1$ wie folgt fortsetzen.

$$2^{2k-1} - 2 = 3m$$

$$\frac{2^{2k-1}-2}{3} = m$$

unser m folgt hier aus der Geometrischen Reihe, wichtig ist dabei der Operator $* 2$

$$\frac{2^{2k-1}-2}{3} = 2 * (\frac{4^{n+1}-1}{3})$$

$$\frac{2^{2k-1}-2}{6} = \frac{4^{n+1}-1}{3}$$

$$\frac{2^{2k-2}-1}{3} = \frac{4^{n+1}-1}{3}$$

$$k = n + 2 \Leftrightarrow n = k - 2$$

Folglich gilt unser obige Eigenschaft mit dem Beweis, dass es natürlich Schnittstellen für unser m gibt. Daraus folgt, dass es sich bestimmen lässt wann die Operation $\frac{n-1}{3}$ ausgeführt wird mit folgender Eigenschaft:

$$\forall x \in A_n \mod 2=0 : \frac{2^x-1}{3} \mod 2 = 1$$

Man darf hier aber nicht vergessen, dass wir es nur für den ersten Bruch gezeigt haben. Schaut man sich die nächsten Brüche so lassen sich unsere nächsten Vorhersagen nicht mehr so leicht treffen. So ist unsere Zahlenreihen für den Index gleich zwei folgende:

```
[1, 2, 4, 8, 16, 5, 10, 3, 6, 12, 24, 48, 96, 192, 384, 768, 1536]
[1, 2, 4, 8, 16, 5, 10, 20, 40, 13, 26, 52, 104, 208, 416, 832, 1664]
[1, 2, 4, 8, 16, 5, 10, 20, 40, 80, 160, 53, 106, 212, 424, 848, 1696]
[1, 2, 4, 8, 16, 5, 10, 20, 40, 80, 160, 320, 640, 213, 426, 852, 1704]
[1, 2, 4, 8, 16, 5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560, 853, 1706]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 85, 170, 340, 113, 226, 452, 904, 1808]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 85, 170, 340, 680, 1360, 453, 906, 1812]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 85, 170, 340, 680, 1360, 2720, 5440, 1813]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 341, 682, 227, 454, 908, 1816]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 341, 682, 1364, 2728, 909, 1818]
```

Die 21 ist in unserer Reihen für den nächsten Bruch nicht vertreten. Geht man hierbei noch eins höher, indem wir unseren Index auf 3 setzen so ist die 3 ebenfalls nicht vertreten. Und dieses Verhalten lässt sich dahingehend auch schnell beweisen. Denn wenn unser $n \mod 3 = 0$, so ist $n \in C_{3*x}$ mit $x \in \mathbb{N}$ und $n = 3x$. Da unser $n \in C_{3*x}$ so ist auch $n * 2^k \in C_{3*x}$ mit $k \in \mathbb{N}$. Da sich hierfür unser x umformulieren lässt zu $x' = 2^k * x$ und daraus folgt $n * 2^k = 3 * x * 2^k \in C_{3*x}$. Folglich wissen wir, dass unser $n * 2^k \mod 3 = 0$ und so ist $(n * 2^k) - 1 \mod 3 = 2$. Demnach ist auch $((n * 2^k) - 1)/3 \notin \mathbb{N}$, da $n * 2^k - 1$ stets den Rest 2 besitzt. Daraus folgt, dass wenn unsere Umkehrfunktion ein Wert erreicht, wo $n \mod 3 = 0$, so wird unsere Umkehrfunktion für jeden weiteren Wert $n * 2^k$ nie wieder brechen.

Man könnte aus der gewonnen Erkenntnis trügerisch behaupten unsere Umkehrfunktion würde dann terminieren, aber dass ist diesbezüglich nicht möglich, da immer noch für $n \mod 3 = 0$, $n * 2^k$ berechnet werden muss mit $k > 0$.
Würde hierbei unsere Umkehrfunktion mit einem Wert n starten, wobei n die Eigenschaft besitzt $n \mod 3 = 0$ so können wir aus der obigen Erkenntnis behaupten, dass unsere berechnete Menge der Umkehrfunktion E die Eigenschaft besitzt $E \subset \mathbb{N}$, weil E nur die Elemente $n * 2^k$ beinhaltet.

Betrachtet man die oben angegebenen Reihen weiter für den Index 2 und 3 so sieht man, dass 4^x nicht fortgeführt wird. So hat beispielsweise 5 den nächsten Bruch bei 10, 40 .... Folglich muss hierfür für jede weitere Zahl ein anderer Zusammenhang gelten. Betrachtet man dafür unsere bereits bewiesene Aussage $\frac{2^{2x}-1}{3} \mod 2 = 1$ so ergibt sich hieraus $4^x \mod 3 = 1$. Folglich ließe es sich hierfür fortsetzten, dass man behaupten könnte $((3k+1) * 4^x) \mod 3 = 1$. Betrachtet man hierfür die Brüche bei 5, 53, ... so ließe sich hierfür folgendes herleiten $((3k+2) * 2^{2x+1})$, da $5 \mod 3 = 2$, ... und dafür die noch unbewiesene Definition aufstellen.

(11)

$$\frac{(3k+1) * 4^n - 1}{3} = 2m + 1$$

(12)

$$\frac{(3k+2) * 2^{2n+1} - 1}{3} = 2m + 1$$

Beweis zu (12)

$$\frac{(3k+2)*2^{2n+1}-1}{3} = 2m+1$$
$$3k*2^{2n+1}+2*2^{2n+1}-1 = 6m+3$$
$$3k*2^{2n+1}+4^{n+1}-4 = 6m$$
$$k*2^{2n+1}+\frac{4^{n+1}-4}{3} = 2m$$
$$\frac{4^{n+1}-4}{3} = \frac{4^{n+1}-1}{3} - \frac{3}{3} = \Big(\sum_{i=0}^{n} 4^n\Big) - 1$$

Da wir bereits bewiesen haben, dass

$$\sum_{i=0}^{n} 4^n \mod 2 = 1 \text{ folgt daraus, dass}$$

$$\Big(\sum_{i=0}^{n} 4^n\Big) - 1 \mod 2 = 0$$

und da unabhängig von unserem $k, k*2^{2n+1}$ ebenfalls gerade ist (6), folgt

$$k*2^{2n+1} = 2l, \frac{4^{n+1}-4}{3} = 2o$$
$$2l+2o = 2m$$
$$2(l+o) = 2m$$
$$l+o = m$$

Beweis zu (11)

$$\frac{(3k+1)*4^n-1}{3} = 2m+1$$
$$3k*4^n+4^n-1 = 6m+3$$
$$3k*4^n+4^n-4 = 6m$$
$$k*4^n+\frac{4^n-4}{3} = 2m$$

Analog zu obigem Beweis ist $\Big(\sum_{i=0}^{n} 4^n\Big) - 1 \mod 2 = 0$

und unabhängig vom k ist $k*2^{2n} \in A_2 \mod n=0$

Analoger Beweis zu oben ist unsere Gleichung stets gerade, da

$2l+2o = 2(l+o)$, folglich

$$2(l+o) = 2m$$
$$m = l+o$$

Aus der oben gewonnen Erkenntnis lässt sich eine zweite Funktion f'^{-1} aufstellen die uns unsere Wörter in einer Untermenge $L' \subseteq L_{P_{Coll}}$ berechnet.

$$f'^{-1}(n) = \begin{cases} n * 2^x & \forall x \in \mathbb{N} \\ f'^{-1}(\frac{n*4^x-1}{3}) & \forall x \in \mathbb{N}, \text{ wenn } n \mod 3 = 1 \\ f'^{-1}(\frac{n*2^{2x+1}-1}{3}) & \forall x \in \mathbb{N}, \text{ wenn } n \mod 3 = 2 \end{cases}$$

Hieraus lässt sich folgendes schließen

$$f'^{-1}(n) : F_{2^x} \to D$$
$$f^{-1}(n) : F_{2^x} \to E$$
$$F_{2^x} = \{1, 2, 4, 8, 16, ...\}$$

Wobei wir hier folgendes behaupten können, da $f^{-1}(n)$ die Umkehrfunktion zu unserem Collatzproblem ist, so lässt sich hierbei unser Collatzproblem wie folgt beschreiben, $f(n) : E \to F_{2^x}$. Da jedes Element in E unter Anwendung von $3e + 1$ und $\frac{e}{2}$ auf ein Element f aus F_{2^x} zurückgeführt werden kann.

So ist hierbei unklar wie aus (3) und (4) folgt, ob $E \subset \mathbb{N}$ oder $E = \mathbb{N}$. Da wir nicht eindeutig behaupten können, dass unserer Hilfsfunktion $f'^{-1} : F_{2^x} \to D$ unsere Umkehrfunktion genauestens beschreibt, lässt sich hierbei nur daraus folgern, dass $D \subseteq E$. Unsere Umkehrfunktion f^{-1} berechnet hierbei für jedes Element x der Reihe $k * 2^x$ den Bruch. Unsere Hilfsfunktion f'^{-1} ist diesbezüglich beschränkt auf die Restklassen von 3. So kann unsere Umkehrfunktion $f^{-1}(n)$ immer noch für irgendwelche beliebigen Werte brechen die wir in f'^{-1} nicht erfasst haben.

Hierbei lässt sich aus den Eigeschaften für D folgender Beweis aufstellen.

$$1 \in D$$

$$\forall x\Big(1 \in D \to \big(1 * 2^x \in D \wedge \frac{1 * 4^x - 1}{3} \in D\big)\Big)$$

$$\forall x\Bigg(\frac{1 * 4^x - 1}{3} \in D \to \exists y \exists k \exists a\Big(y = \frac{1 * 4^a - 1}{3} \wedge \big((y = 3k + 1) \vee (y = 3k + 2)\big) \wedge y * 2^x \in D\Big)\Bigg)$$

$$\exists k \forall x\Bigg(((y = 3k + 1) * 2^x) \in D \to \Big((y * 2^x \in D) \wedge (\frac{y * 4^x - 1}{3} \in D)$$

$$\to \exists z \exists m \exists a\Big(z = \frac{y * 4^a - 1}{3} \wedge \big((z = 3m + 1) \vee (z = 3m + 2)\big) \wedge z \in D\Big)\Big)\Bigg)$$

$$\exists k \forall x\Bigg(((y = 3k + 2) * 2^x) \in D \to \big((y * 2^x \in D) \wedge (\frac{y * 2^{2x+1} - 1}{3} \in D)$$

$$\to \exists z \exists m \exists a\Big(z = \frac{y * 2^{2a+1} - 1}{3} \wedge \big((z = 3m + 1) \vee (z = 3m + 2)\big) \wedge z \in D\Big)\Big)\Bigg)$$

$$\forall D\Bigg(1 \in D \to \forall x\Bigg(1 * 2^x \in D \wedge \frac{1 * 4^x - 1}{3} \in D$$

$$\to \exists y \exists k \exists a\Big(y = \frac{1 * 4^x - 1}{3} \wedge \big((y = 3k + 1) \vee (y = 3k + 2)\big)\Big)$$

$$\to \exists k\Big(\big((y = 3k + 1) * 2^x\big) \in D \to \exists z \exists m \exists a\big(z = \frac{y * 4^a - 1}{3} \wedge z \in D \wedge ((z = 3m + 1) \vee (z = 3m + 2))\big) \vee$$

$$\big(((y = 3k + 2) * 2^x) \in D \to \exists z \exists m \exists a(z = \frac{y * 2^{2a+1} - 1}{3} \wedge z \in D \wedge ((z = 3m + 1) \vee (z = 3m + 2)))\big)\Big)$$

$$\Bigg) \to \mathbb{N} \subseteq D\Bigg)$$

$$\mathbb{N} \subseteq D \subseteq E$$

$$\mathbb{N} \subseteq E$$

3 Skriptanhang

Bei der Ausarbeitung des Paper habe ich zusätzlich noch drei Skripts geschrieben die alle den aktuellen Stand der Forschung widerspiegeln und die Sachverhalte veranschaulichen sollen.

Die Datei SieveCollatz.py, filtert hierbei ähnlich wie das Sieb des Eratosthenes.

Die Datei TraceCollatz.py, findet hierbei für eine beliebige Zahl ihren Weg auf den derzeitig bekanntesten kürzesten Weg.

Die Datei NewCollatz.py, arbeitet auf dem letzten bewiesenen Erkenntnisstand uns jede Reihe aus die sich mit der Funktion $f'^{-1}(n)$ berechnen lässt. Hierbei bricht unsere Skript nach einer gewissen Wertigkeit ab, aktuell im Paper festgelegt sind 50.

SieveCollatz.py

```python
import NewCollatz as nwCollatz

matrix = []

# works for 1-100
def initMatrix():
    for i in range(1, 102):
        matrix.append(i)

def printMatrix():
    for i in range(len(matrix)):
        if (i+1) % 10 == 0:
            if matrix[i] == -1:
                print(f'    \n', end='')
            else:
                print(f' {matrix[i]} \n', end='')
        else:
            if matrix[i] == -1:
                print(f'    ', end='')
            elif i < 10:
                print(f'  {matrix[i]} ', end='')
            elif matrix[i] == 101:
                print()
            else:
                print(f' {matrix[i]} ', end='')

def removeCollatz(array):
    for a, b in array:
        print(b)
        for i in range(len(matrix)):
            for temp in range(len(b)):
                if b[temp] > 100:
                    break
                elif matrix[i] == b[temp]:
                    matrix[i] = -1
        printMatrix()

initMatrix()

nwCollatz.rebuildCollatzByMod(1, 0)
nwCollatz.l.sort()

removeCollatz(nwCollatz.l)
```

Ausgabe des Siebes für unser Collatz-Problem

```
[1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0]
          3       5   6   7       9  10
 11  12  13  14  15      17  18  19  20
 21  22  23  24  25  26  27  28  29  30
 31      33  34  35  36  37  38  39  40
 41  42  43  44  45  46  47  48  49  50
 51  52  53  54  55  56  57  58  59  60
 61  62  63      65  66  67  68  69  70
 71  72  73  74  75  76  77  78  79  80
 81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99  100

[5.0, 10.0, 20.0, 40.0, 80.0, 160.0, 320.0, 640.0]
          3           6   7       9
 11  12  13  14  15      17  18  19
 21  22  23  24  25  26  27  28  29  30
 31      33  34  35  36  37  38  39
 41  42  43  44  45  46  47  48  49  50
 51  52  53  54  55  56  57  58  59  60
 61  62  63      65  66  67  68  69  70
 71  72  73  74  75  76  77  78  79
 81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99  100

[...] #Cutout for good reasons


[261.0, 522.0, 1044.0, 2088.0, 4176.0, 8352.0, 16704.0, 33408.0]

                15
          23            27          30
 31               35              39
 41       43          46  47
 51           54  55      57      59  60
 61  62  63               67          70
 71       73      75          78  79
 81  82  83          86  87      89
 91  92  93  94  95      97      99
```

TraceCollatz.py

```python
import math

pair = []


def traceNumber(x):
    n = math.log(x, 2)
    logs = []
    pair.append(x)

    for i in range(math.floor(n)):
        logs.append(math.pow(2, i))

    logs.reverse()

    while not n.is_integer():
        for i in logs:
            if (x / i).is_integer():
                traceNumber((x / i)*3+1)
                return


traceNumber(45679)
print(pair)
```

Ausgaben für beliebige Zahlen:

37

```
[37, 37.0, 112.0, 7.0, 22.0, 11.0, 34.0, 17.0, 52.0, 13.0, 40.0, 5.0, 16.0]
```

3

```
[3, 3.0, 10.0, 5.0, 16.0]
```

7

```
[7, 7.0, 22.0, 11.0, 34.0, 17.0, 52.0, 13.0, 40.0, 5.0, 16.0]
```

20

```
[20, 5.0, 16.0]
```

30

```
[30, 15.0, 46.0, 23.0, 70.0, 35.0, 106.0, 53.0, 160.0, 5.0, 16.0]
```

NewCollatz.py

```python
import math

l = [(0, [])]


def printArray():
    for a, b in l:
        print(f'{a} {b}')


def getNumberArray(m):
    temp = []
    for i in range(0, 8):
        temp.append(m * math.pow(2, i))
    return temp


def checkIfArrayExists(array):
    exists = False
    for a, b in l:
        if b.__eq__(array):
            exists = True

    return exists


def isNumberInArray(n):
    exists = False
    for a, b in l:
        if b.__contains__(n):
            exists = True

    return exists
```

```
def rebuildCollatzByMod(n, index):
    array = []
    for i in range(0, 8):
        array.append(n * math.pow(2, i))

    if not checkIfArrayExists(array):
        l.append((index, array))

    for m in array:
        if m < 50:
            for k in range(0, 8):
                if m % 3 == 1:
                    futureBreak = ((m * math.pow(4, k)) - 1) / 3
                    if not isNumberInArray(futureBreak):
                        rebuildCollatzByMod(futureBreak, index + 1)
                elif m % 3 == 2:
                    futureBreak = ((m * math.pow(2, (2 * k + 1))) - 1) / 3
                    if not isNumberInArray(futureBreak):
                        rebuildCollatzByMod(futureBreak, index + 1)
                elif m % 3 == 0:
                    if not checkIfArrayExists(array):
                        l.append((index, getNumberArray(m)))
                else:
                    print(end="")
        else:
            return


rebuildCollatzByMod(1, 0)
l.sort()
printArray()
```

Ausgabe:

```
0 []
0 [1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0]
1 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
1 [5.0, 10.0, 20.0, 40.0, 80.0, 160.0, 320.0, 640.0]
1 [21.0, 42.0, 84.0, 168.0, 336.0, 672.0, 1344.0, 2688.0]
1 [85.0, 170.0, 340.0, 680.0, 1360.0, 2720.0, 5440.0, 10880.0]
1 [341.0, 682.0, 1364.0, 2728.0, 5456.0, 10912.0, 21824.0, 43648.0]
2 [3.0, 6.0, 12.0, 24.0, 48.0, 96.0, 192.0, 384.0]
2 [13.0, 26.0, 52.0, 104.0, 208.0, 416.0, 832.0, 1664.0]
2 [53.0, 106.0, 212.0, 424.0, 848.0, 1696.0, 3392.0, 6784.0]
2 [213.0, 426.0, 852.0, 1704.0, 3408.0, 6816.0, 13632.0, 27264.0]
3 [17.0, 34.0, 68.0, 136.0, 272.0, 544.0, 1088.0, 2176.0]
3 [69.0, 138.0, 276.0, 552.0, 1104.0, 2208.0, 4416.0, 8832.0]
3 [277.0, 554.0, 1108.0, 2216.0, 4432.0, 8864.0, 17728.0, 35456.0]
4 [11.0, 22.0, 44.0, 88.0, 176.0, 352.0, 704.0, 1408.0]
4 [45.0, 90.0, 180.0, 360.0, 720.0, 1440.0, 2880.0, 5760.0]
4 [181.0, 362.0, 724.0, 1448.0, 2896.0, 5792.0, 11584.0, 23168.0]
5 [7.0, 14.0, 28.0, 56.0, 112.0, 224.0, 448.0, 896.0]
5 [29.0, 58.0, 116.0, 232.0, 464.0, 928.0, 1856.0, 3712.0]
5 [117.0, 234.0, 468.0, 936.0, 1872.0, 3744.0, 7488.0, 14976.0]
5 [469.0, 938.0, 1876.0, 3752.0, 7504.0, 15008.0, 30016.0, 60032.0]
6 [9.0, 18.0, 36.0, 72.0, 144.0, 288.0, 576.0, 1152.0]
6 [19.0, 38.0, 76.0, 152.0, 304.0, 608.0, 1216.0, 2432.0]
6 [37.0, 74.0, 148.0, 296.0, 592.0, 1184.0, 2368.0, 4736.0]
6 [77.0, 154.0, 308.0, 616.0, 1232.0, 2464.0, 4928.0, 9856.0]
6 [149.0, 298.0, 596.0, 1192.0, 2384.0, 4768.0, 9536.0, 19072.0]
6 [309.0, 618.0, 1236.0, 2472.0, 4944.0, 9888.0, 19776.0, 39552.0]
7 [25.0, 50.0, 100.0, 200.0, 400.0, 800.0, 1600.0, 3200.0]
7 [49.0, 98.0, 196.0, 392.0, 784.0, 1568.0, 3136.0, 6272.0]
7 [101.0, 202.0, 404.0, 808.0, 1616.0, 3232.0, 6464.0, 12928.0]
7 [197.0, 394.0, 788.0, 1576.0, 3152.0, 6304.0, 12608.0, 25216.0]
7 [405.0, 810.0, 1620.0, 3240.0, 6480.0, 12960.0, 25920.0, 51840.0]
8 [33.0, 66.0, 132.0, 264.0, 528.0, 1056.0, 2112.0, 4224.0]
8 [65.0, 130.0, 260.0, 520.0, 1040.0, 2080.0, 4160.0, 8320.0]
8 [133.0, 266.0, 532.0, 1064.0, 2128.0, 4256.0, 8512.0, 17024.0]
8 [261.0, 522.0, 1044.0, 2088.0, 4176.0, 8352.0, 16704.0, 33408.0]
```