Wrocław, 21.12.2021r.

Lista zadań nr 5

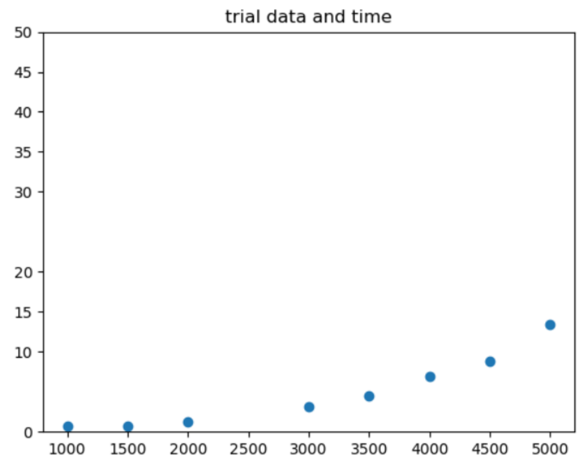
Julia Mazur, 262296

Zadanie 1.

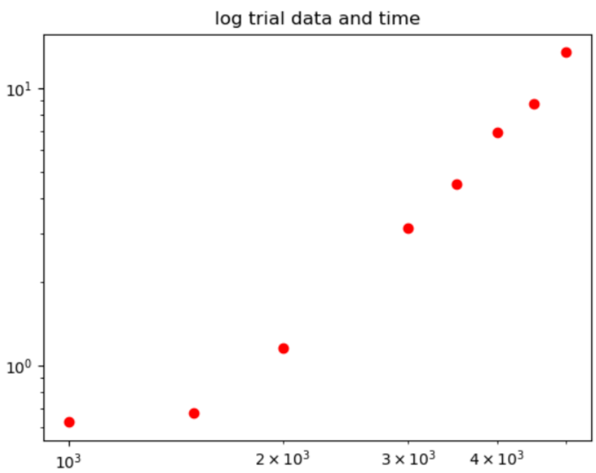
Zmierzyłam czas wykonywania funkcji *solve* dla wybranych przeze mnie danych danych.

```
1000 :    0.625
1500 :    0.671875
2000 :    1.15625
3000 :    3.125
3500 :    4.515625
4000 :    6.90625
4500 :    8.796875
5000 :    13.453125
```

Sporządziłam wykres.



Na wykresie logarytmicznym widzimy, że oprócz pierwszego wszystkie punkty układają się w prostą:



Dlatego spodziewamy się zależności potęgowej:

$y = ax^{b}$

$log(y) = log(a) + b\,log(x)$

(jest to prosta dla zlogarytmowanego wzoru).

Hipoteza podwojenia:

Zmierzyłam czas wykonania funkcji dla dwukrotnie zwiększających się danych wejściowych (czyli ilości niewiadomych w układzie równań). Następnie policzyłam stosunki $\frac{T(2N)}{T(N)}$ oraz $log_2(\frac{T(2N)}{T(N)})$. Otrzymałam takie wyniki:

a) pierwsza próba:

N	T	Ratio	Log
512	0.3125	None	None
1024	0.875	2.8	1.4854268271702415
2048	1.171875	1.3392857142857142	0.4214637684382767
4096	10.34375	8.826666666666666	3.141868716311337
8192	47.9375	4.634441087613293	2.212395360695716

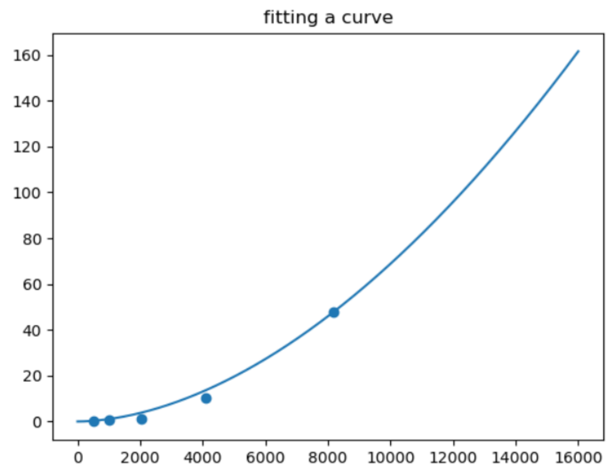
Zatem średnie $b \approx 1.815$.

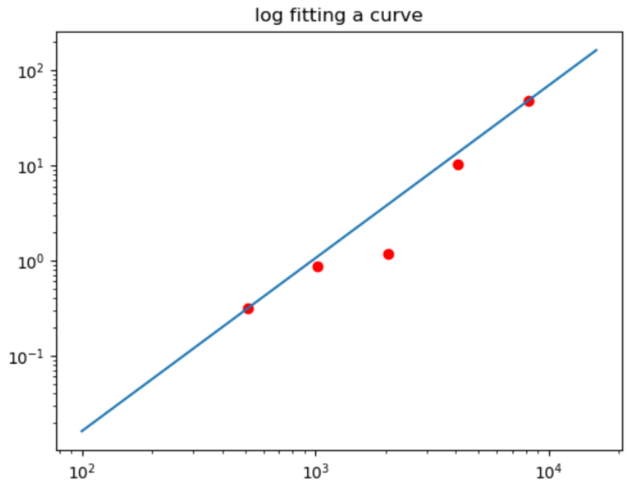
Aby obliczyć a, podstawiamy dowolne dane do równania.

Zrobimy to dla $N = 8192$, czyli $47.9375 = a * 8192^{1.815}$.

Zatem nasze $a \approx 3.773 * 10^{-6}$.

Na wykresie prezentuje się to następująco:





b) druga próba:

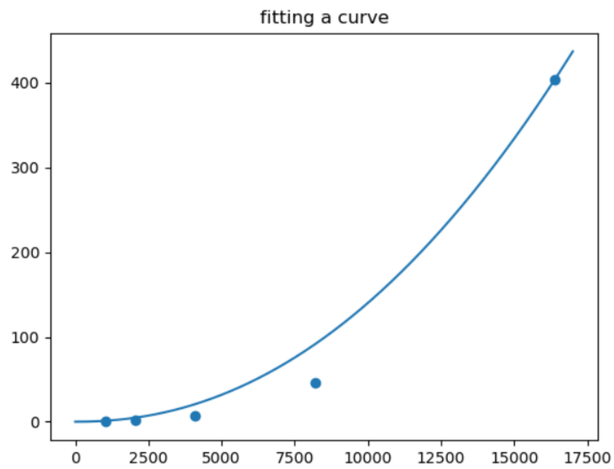
N	T	Ratio	Log
1024	1.0625	None	None
2048	1.453125	1.3676470588235294	0.451695969857692
4096	7.296875	5.021505376344086	2.3281199286016308
8192	46.09375	6.316916488222698	2.659220499426904
16384	403.9375	8.763389830508475	3.1314890371413733

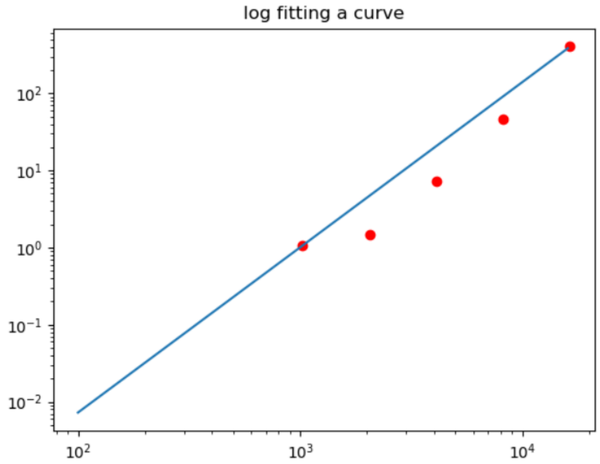
Zatem średnie $b \approx 2.143$.

Obliczamy a dla $N = 16384$, czyli $403.9375 = a * 16384^{2.143}$.

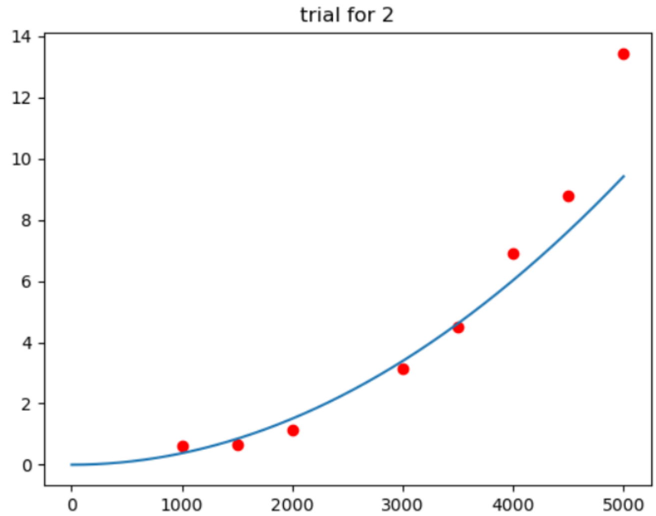
Zatem nasze $a \approx 3.770 * 10^{-7}$.

Na wykresie prezentuje się to następująco:





Możemy zauważyć, że b jest liczbą z otoczenia 2, dlatego sprawdzimy czy możemy założyć, że $b = 2$:

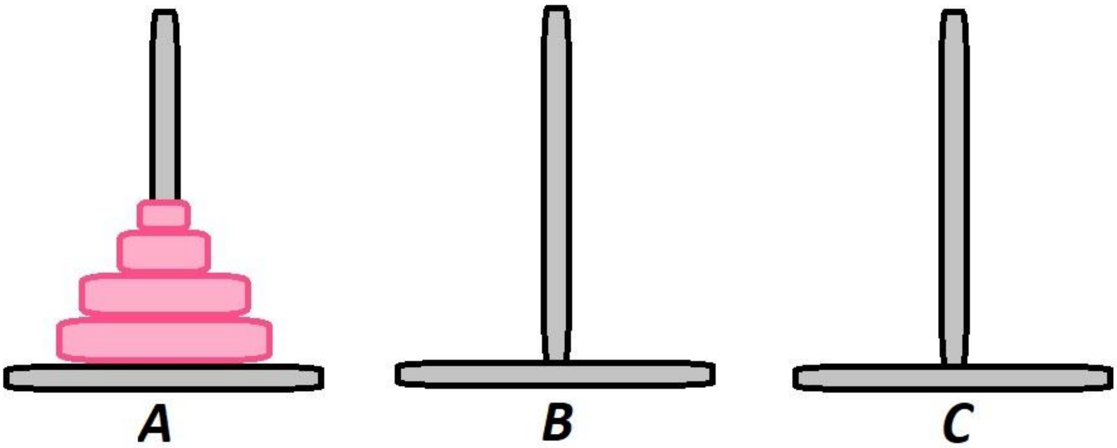


Zatem nasza hipoteza jest prawdziwa. Czyli złożoność obliczeniowa funkcji to $O(n^2)$

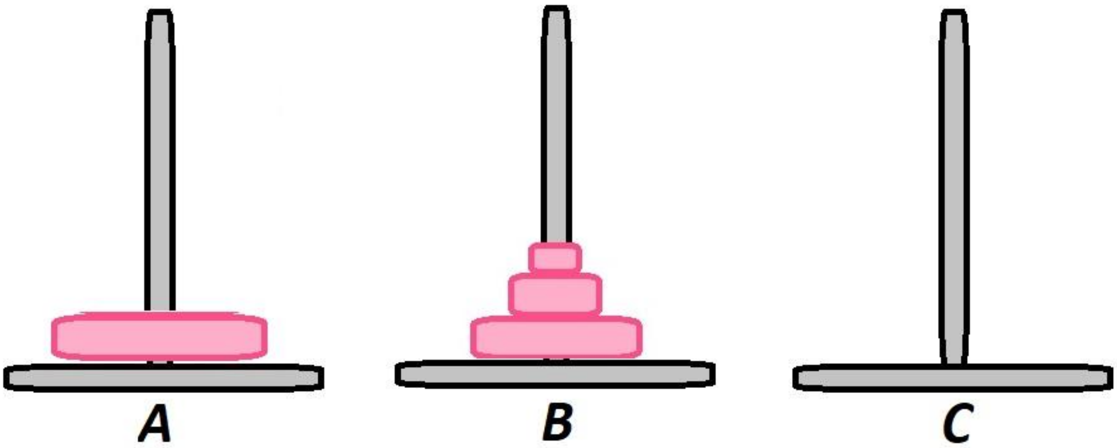
Linki do kodu i danych:

1. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/7a3d1b9448a07ca8ae41fefd5e43e405a0ac0887/lista5/1.py
2. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6e99cfcc5d834bba8a075fcc6c1fa87ea83b56b2/lista5/data1.txt
3. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6e99cfcc5d834bba8a075fcc6c1fa87ea83b56b2/lista5/data2.txt

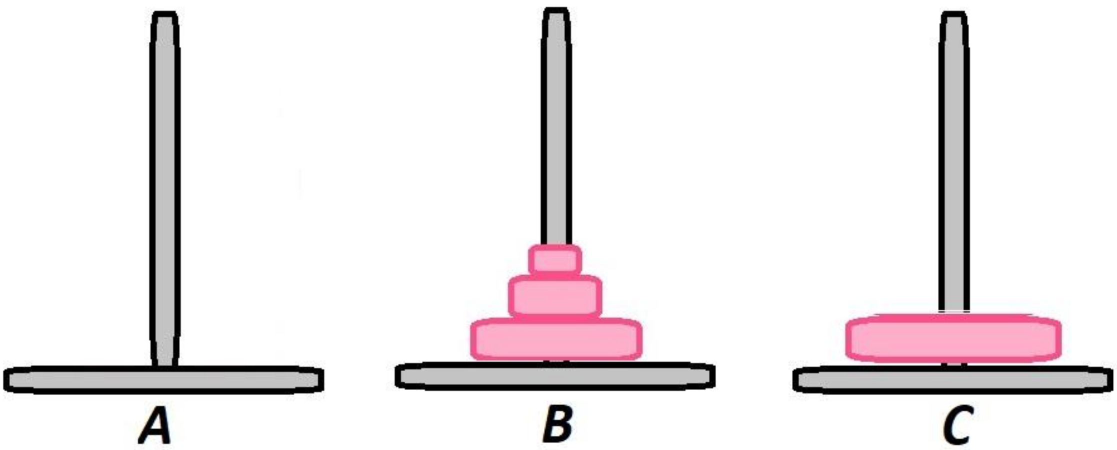
Zadanie 2.
Mamy napisać program pokazujący rozwiązanie zagadnienia wieży z Hanoi.



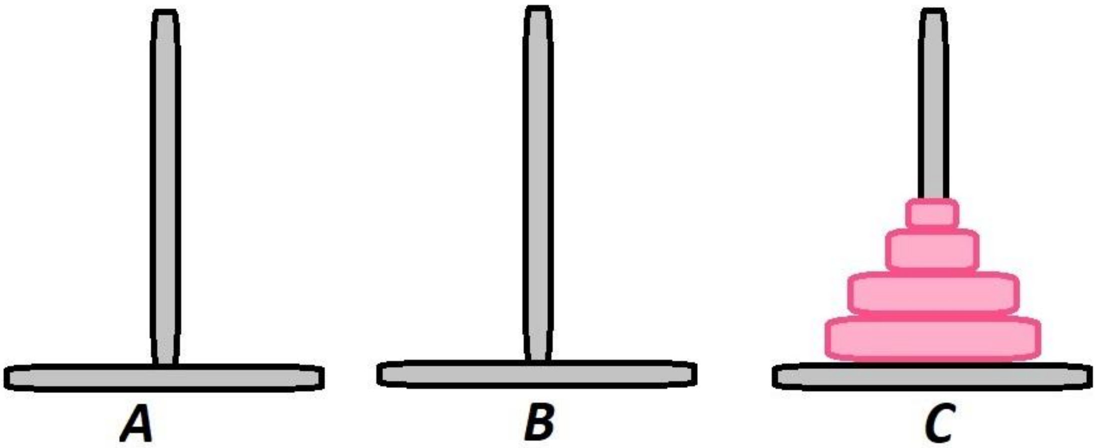
Aby przenieść n krążków ze słupka A na słupek C przy użyciu B musimy najpierw przenieść $n - 1$ na słupek B przy użyciu C.



Kiedy to nam się uda przenosimy ostatni (n – ty) krążek na słupek C.



Na koniec musimy przenieść krążki $(n - 1)$ ze słupka B na C przy użyciu A.



Przykładowe wywołania programu:

1. dla $n = 1$:

```
move disc from A to C
```

2. dla $n = 2$:

```
move disc from A to B
move disc from A to C
move disc from B to C
```

3. dla $n = 3$:

```
move disc from A to C
move disc from A to B
move disc from C to B
move disc from A to C
move disc from B to A
move disc from B to C
move disc from A to C
```

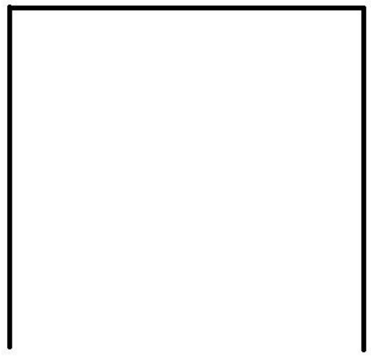
4. dla $n = 4$:

```
move disc from A to B
move disc from A to C
move disc from B to C
move disc from A to B
move disc from C to A
move disc from C to B
move disc from A to B
move disc from A to C
move disc from B to C
move disc from B to A
move disc from C to A
move disc from B to C
move disc from A to B
move disc from A to C
move disc from B to C
```

Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6e99cfcc5d834bba8a075fcc6c1fa87ea83b56b2/lista5/2.py

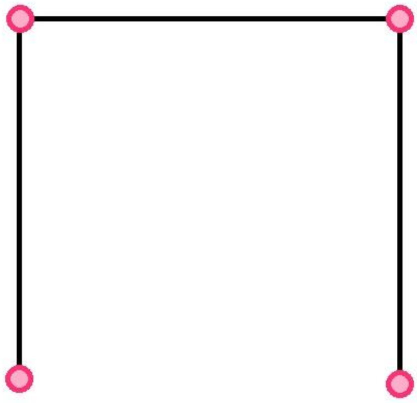
Zadanie 3.

Analizę krzywej hilberta zaczynamy od przypadku dla $n = 1$ – jest to nasz bazowy kształt



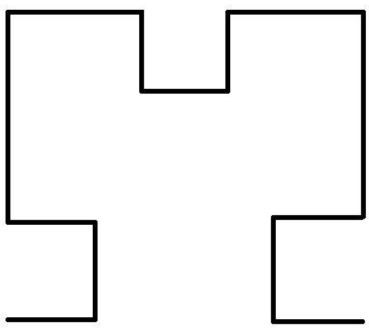
(1)

Punkty, w których występuje zmiana to:



(2)

i w tych punktach musimy wykonać ten kształt (w pierwszym i czwartym kształt będzie odwrócony**(x)**).



(3)

itd. dla kolejnych n.

Aby narysować krzywą o złożoności n:

1. Zaczynamy od obrócenia naszego żółwia o 90 stopni w lewo i powtórzeniu całego algorytmu dla $n - 1$ oraz kąta równego $- 90$ stopni (z **(x)**)"odhaczając" w ten sposób pierwszą krawędź.
2. Następnie idziemy prosto, by kontynuować bazowy kształt.
3. Obracamy żółwia o 90 stopni w prawo (również zgodnie z **(1)**) i także powtarzamy algorytm dla $n - 1$.
4. Znów kontynuujemy prosto. Jesteśmy w drugiej kropce z **(2)**.
5. Powtarzamy algorytm dla $n - 1$, ponieważ nasze ustawienie jest poprawne.
6. Jesteśmy w trzeciej kropce z **(2)**, dlatego obracamy się o 90 stopni w prawo i idziemy prosto.
7. Po raz ostatni powtarzamy cały algorytm dla $n - 1$oraz kąta równego $- 90$ stopni (z **(x)**) i "odhaczamy" ostatnią krawędź kształtu bazowego.
8. Na koniec obracamy się w lewo, żeby umożliwić zachowanie bazy i poprawne połączenie kroków (po wykonaniu kształtu bazowego, żółw kontynuuje ruch odbijając w lewo .

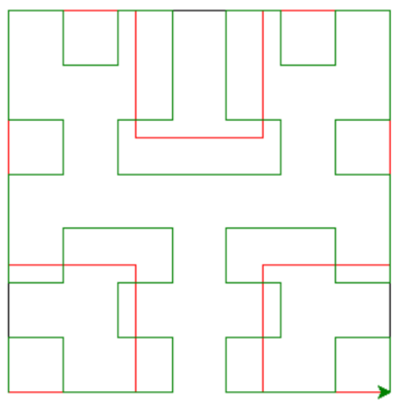
Obliczenie długości jednego odcinka krzywej:

W każdym kolejnym kroku z jednego odcinka tworzymy trzy, czyli ilość wszystkich odcinków zwiększa się o 2^n (w jedną stronę). Zatem wzór na ilość odcinków to $k_n = k_1 + \sum_{i=1}^{n-1} 2^i$, gdzie $k_1 = 1$, bo zakładamy, że dla $n = 1$ ruch w jedną stronę wynosi 1. Wtedy, korzystając z sumy ciągu geometrycznego o $a_1 = 2$ i $q = 2$, liczymy $k_n = 1 + 2 * \frac{1-2^{n-1}}{1-2} = 1 + 2^n - 2 = 2^n - 1$.

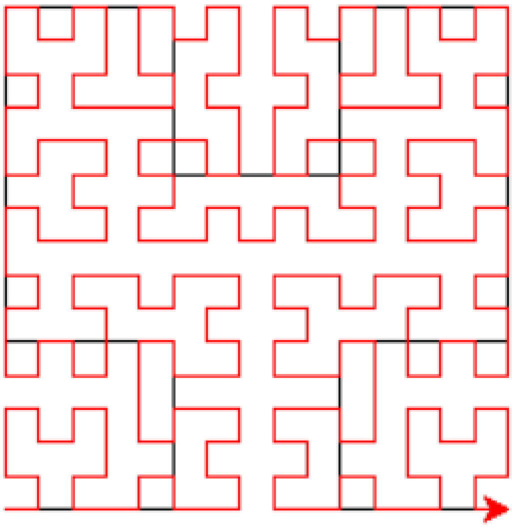
Dlatego długość odcinka krzywej wyznaczyłam jako $rozmiar/(2^n - 1)$.

Przykładowe wywołania programu:

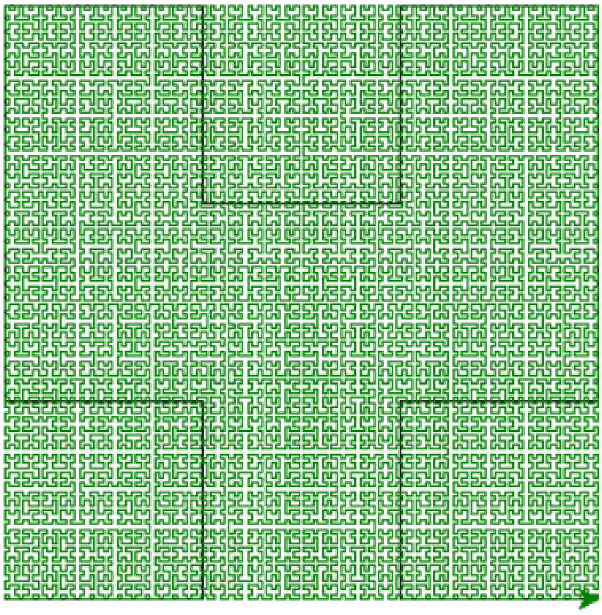
1. dla $n = 1$ (czarny), $n = 2$ (czerwony), $n = 3$ (zielony):



2. dla $n = 4$ (baza to $n = 2$ – czarny):



3. dla $n = 7$ (baza to $n = 2$ – czarny):



Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6e99cfcc5d834bba8a075fcc6c1fa87ea83b56b2/lista5/3.py

Zadanie 4.

Zacznijmy od narysowania krzywej Kocha o złożoności n:

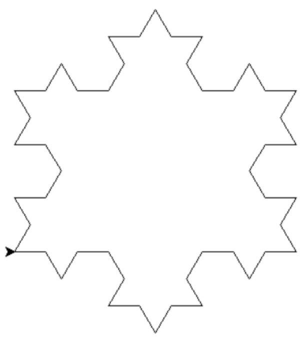
1. Dla $n = 0$ jest to odcinek, więc żółw idzie prosto.
2. W pierwszym kroku powtarzamy cały algorytm dla $n - 1$, bo dzielimy nasz odcinek na trzy części o poprzednim poziomie złożoności.
3. Teraz przechodzimy do zarysowania kształtu krzywej, czyli obracamy żółwia o 60 stopni w lewo i powtarzamy algorytm dla $n - 1$ (przemieszczanie się żółwia jest w warunku bazowym **- 1.**)
4. Następnie obracamy żółwia o 120 stopni w prawo i znów powtarzamy algorytm dla $n - 1$. W taki sposób mamy już $\frac{2}{3}$ początkowego odcinka.
5. Pozostaje już tylko obrócić żółwia o 60 stopni w lewo i powtórzyć algorytm dla $n - 1$.

Skoro dzielimy odcinek na trzy równe części to długość, jaką przebywa żółw w jednym kroku, to $\frac{l}{3^n}$, gdzie l to bazowa długość odcinka.

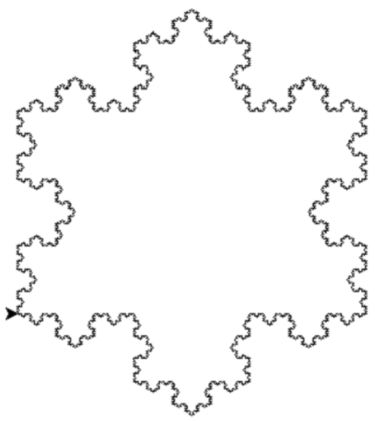
Aby stworzyć płatek kocha wystarczy połączyć trzy krzywe kocha tak, aby kąty między nimi wynosiły 60 stopni.

Przykładowe wywołania programu:

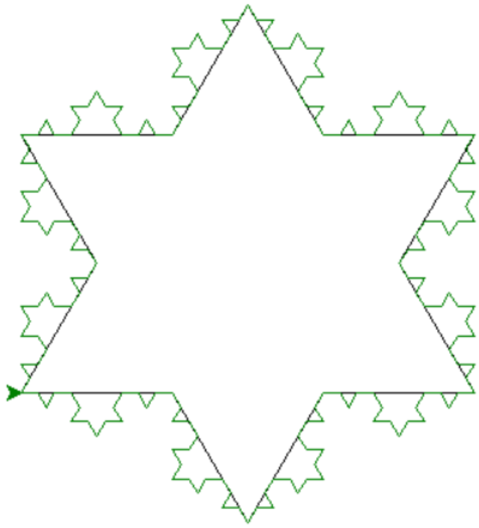
1. dla $n = 2$:



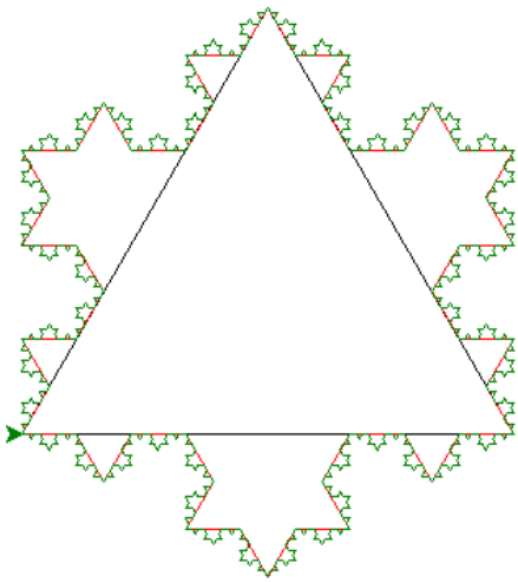
2. dla $n = 5$:



3. dla $n = 1$ (czarny), dla $n = 3$ (zielony):



4. dla $n = 0$ (czarny), dla $n = 2$ (czerwony), dla $n = 4$ (zielony):



Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6e99cfcc5d834bba8a075fcc6c1fa87ea83b56b2/lista5/4.py