Wrocław, 18.01.2022r.

Lista zadań nr 6

Julia Mazur, 262296

Zadanie 1.

Udoskonaliłam klasę *BinarySearchTree* o zliczanie liczby tych samych kluczy (jeśli dany klucz otrzymuje przypisanie kilka razy to jego wartość to ostatnia wartość dodana do tego klucza).

Weźmy sobie przykładowe drzewo o elementach wrzuconych w kolejności na liście $[5, 7, -1, 3, 0, 4, 2]$ i odpowiadających im wartościach $['meow', 'doggo', 'piglet', 'duck', 'oink', 'cat', 'kwak']$.

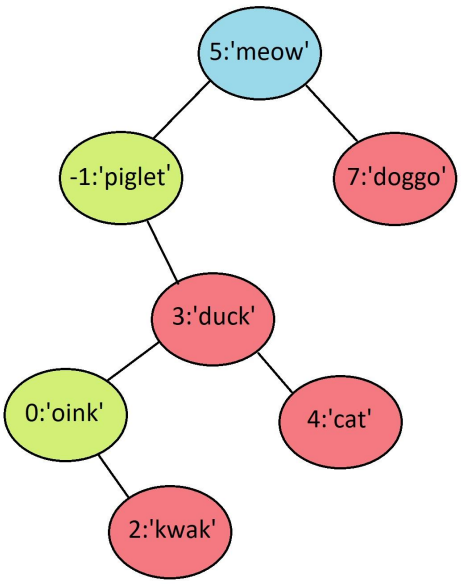
Wtedy drzewo wygląda następująco:

```
5
-1
None    3
0
None    2
None    None    4
None    None    7
None    None
```

```
meow
piglet
None    duck
oink
None    kwak
None    None    cat
None    None    doggo
None    None
```

(Dane wyświetlane są tak, że po danej wartości mamy jej wartość z lewej i tak aż do końca, czyli do dwukrotnego napotkania wartości None, kiedy cofamy się do poprzedniego węzła i wtedy podajemy jego wartość z prawej)

Co po rozrysowaniu wygląda tak:



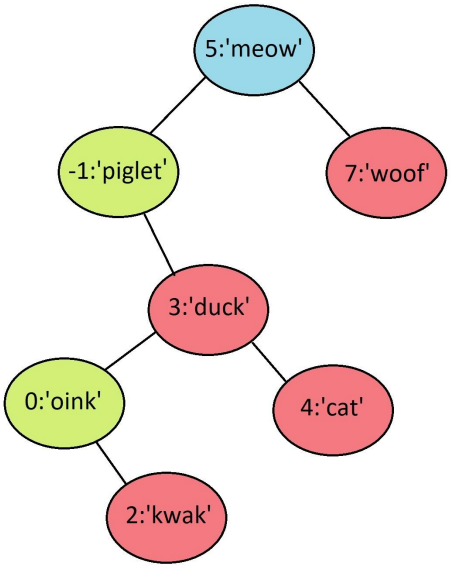
(niebieski to początek drzewa, czerwone to prawe dzieci a zielone to lewe dzieci)

```
counter for 7: 1
```

Żeby sprawdzić działanie wprowadzonych przez nas poprawek dodamy sobie nowy element o kluczu: 7 i wartości: '*woof*'.

```
5                       meow
-1                      piglet
None    3               None    duck
0                       oink
None    2               None    kwak
None    None    4       None    None    cat
None    None    7       None    None    woof
None    None            None    None
```

Jak widzimy jedynym co się zmieniło jest wartość przy kluczu wynoszącym 7.



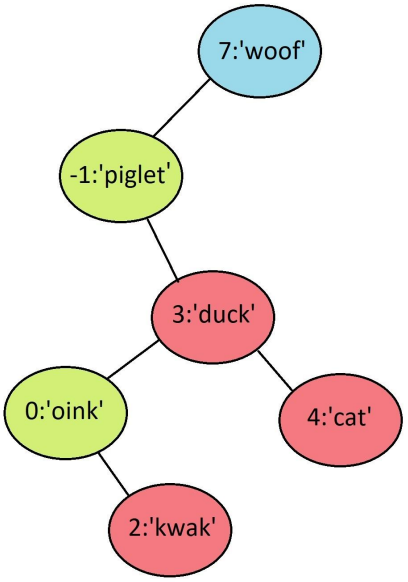

```
counter for 7: 2
```

Zatem przy dodawaniu elementu counter działa poprawnie.
Zobaczmy, co się stanie gdy usuniemy pojedynczy element.

```
delete 5

counter for 5: 0
```

```
7                       woof
-1                      piglet
None    3               None    duck
0                       oink
None    2               None    kwak
None    None    4       None    None    cat
None    None    None    None    None    None
```



Czyli dla pojedynczych elementów również wszystko się zgadza. Na koniec usuniemy element o kluczu 7.

```
delete 7

counter for 7: 1
```

```
7
-1
None    3
0
None    2
None    None    4
None    None    None
```

```
woof
piglet
None    duck
oink
None    kwak
None    None    cat
None    None    None
```

Zmieniła się jedynie wartość countera, co oznacza, że program działa poprawnie.

Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/1.py

Zadanie 2.

Napisałam funkcję korzystającą z kopca binarnego do sortowania list w czasie $O(n * logn)$. Złożoność wywodzi się z tego, że stworzenie kopca binarnego kosztuje nas $logn$ operacji i po znalezieniu najmniejszego elementu za każdym razem tworzymy nowy kopiec, więc musimy stworzyć n kopców. Zatem czas tego sortowania to $O(n * logn)$.

Przykłady działania:

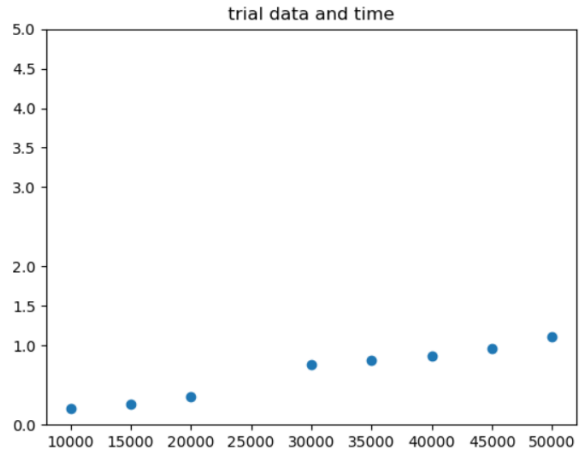
```
[2, 0, 9, 7, 0, 8, 5, 8, 2, 3, 1]
[0, 0, 1, 2, 2, 3, 5, 7, 8, 8, 9]
[20, 10, -9, -7, 0, -8, 5, 8, 2, 3, 100]
[-9, -8, -7, 0, 2, 3, 5, 8, 10, 20, 100]
```

Analiza eksperymentalna czasu wykonywania algorytmu:

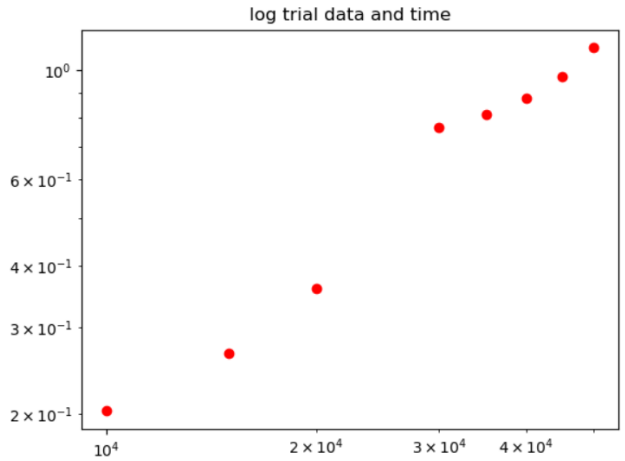
Zmierzyłam czas wykonywania sortowania dla wybranych przeze mnie danych.

```
10000 :             0.203125
15000 :             0.265625
20000 :             0.359375
30000 :             0.765625
35000 :             0.8125
40000 :             0.875
45000 :             0.96875
50000 :             1.109375
```

Wykres:



Na wykresie logarytmicznym widzimy, że można, mimo kilku odstępstw, przybliżyć wyniki prostą:



Teraz zastosowałam hipotezę podwojenia, czyli zmierzyłam czas wykonania funkcji dla dwukrotnie zwiększających się danych wejściowych (czyli ilości elementów w liście do posortowania). Następnie policzyłam stosunki $\frac{T(2N)}{T(N)}$ oraz $log_2(\frac{T(2N)}{T(N)})$.

Otrzymałam takie wyniki:

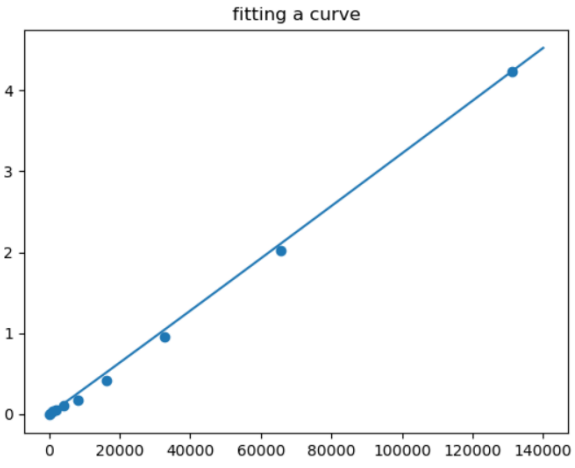
a) pierwsza próba:

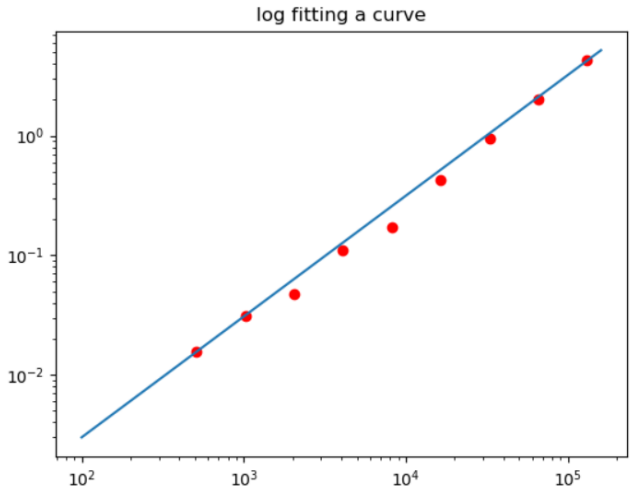
N	T	Ratio	Log
256	0.0	None	None
512	0.015625	None	None
1024	0.03125	2.0	1.0
2048	0.046875	1.5	0.5849625007211562
4096	0.109375	2.3333333333333335	1.222392421336448
8192	0.171875	1.5714285714285714	0.6520766965796931
16384	0.421875	2.4545454545454546	1.2954558835261714
32768	0.953125	2.259259259259259	1.1758498353994178
65536	2.015625	2.1147540983606556	1.0804899178603677
131072	4.234375	2.10077519379845	1.0709217859306175

Obliczyłam średnie $b \approx 1.010$.

Wyznaczyłam a, podstawiając dowolne dane do równania. Dla $N = 131072$: $4.234375 = a * 131072^{1.01}$. Zatem nasze $a \approx 2.86 * 10^{-5}$

Zobaczmy poprawność dopasowania na wykresach:





Czyli dopasowanie jest dobre.

b) druga próba (dla większych N):

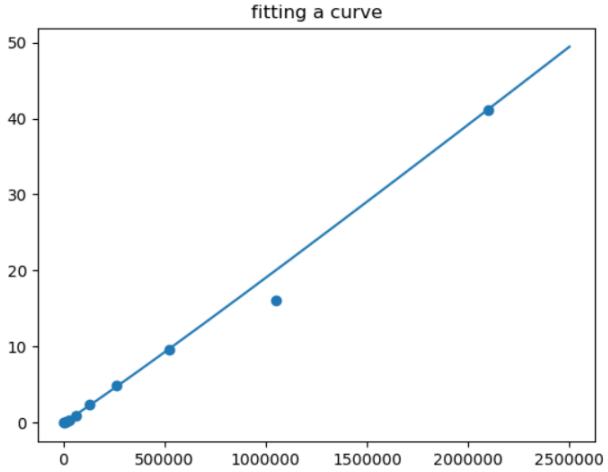
N	T	Ratio	Log
4096	0.0625	None	None
8192	0.09375	1.5	0.5849625007211562
16384	0.1875	2.0	1.0
32768	0.359375	1.9166666666666667	0.9385994553358568
65536	0.96875	2.6956521739130435	1.4306343543298623
131072	2.34375	2.419354838709675	1.2746223801090057
262144	4.859375	2.0733333333333333	1.0519520796347215
524288	9.625	1.9807073954983923	0.9860157705642988
1048576	16.109375	1.6737012987012987	0.743042076673207
2097152	41.171875	2.5557710960232782	1.3537586291564685

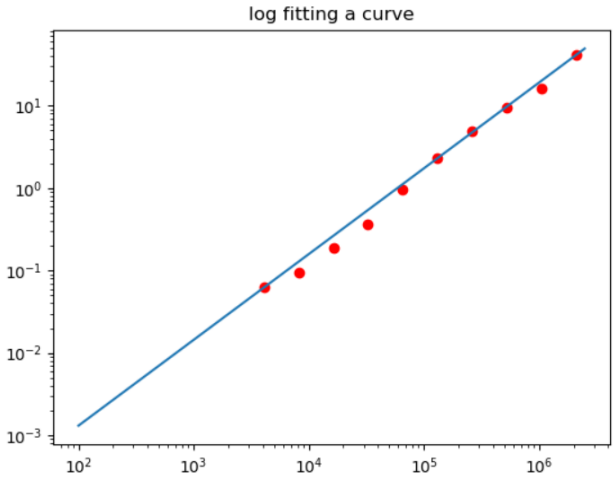
Zatem średnie $b \approx 1.040$.

Liczymy a dla $N = 2097152$. Mamy $41.171875 = a * 2097152^{1.04}$,

a z tego $a \approx 1.09 * 10^{-5}$.

Sprawdźmy dopasowanie:





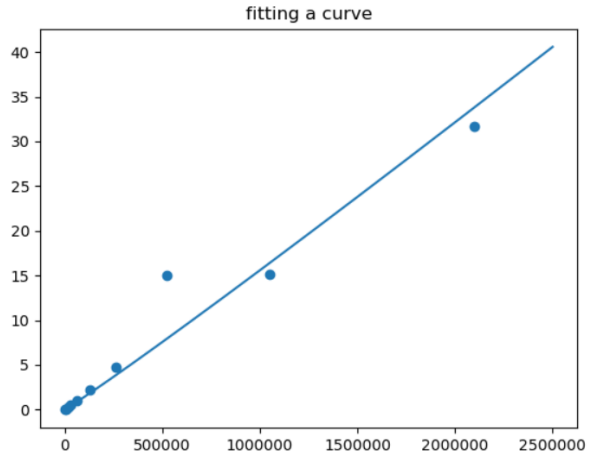
Widzimy, że również w tym przypadku dopasowanie jest dobre.

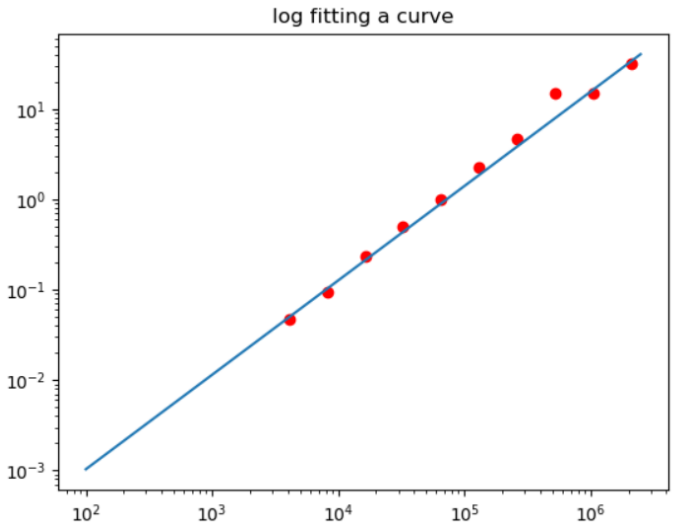
Współczynnik b przybiera wartości z okolicy 1, mniejszy czas wykonywania funkcji może również wynikać z zakresu liczb do posortowania.

c) trzecia próba:
W tym przypadku zmieniłam zakres losowania zmiennych do list z $(-\ 10, 10)$ na $(-\ 100, 100)$.

N	T	Ratio	Log
4096	0.046875	None	None
8192	0.09375	2.0	1.0
16384	0.234375	2.5	1.3219280948873624
32768	0.5	2.1333333333333333	1.0931094043914815
65536	1.0	2.0	1.0
131072	2.25	2.25	1.1699250014423124
262144	4.71875	2.0972222222222223	1.0684797378827666
524288	14.984375	3.1754966887417218	1.666982265693052
1048576	15.109375	1.008342022940563	0.011985074458710356
2097152	31.71875	2.099276111685626	1.0698919325956975

Średnie $b \approx 1.045$. Dla $N = 2097152$ mamy:
$31.71875 = a * 2097152^{1.045}$, czyli $a \approx 8.4 * 10^{-6}$.
Wykresy:





Dopasowanie jest dobre.

Linki do kodu i danych:

1. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/2.py
2. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/data1.txt
3. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/data2_1.txt
4. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/data2_2.txt
5. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/data3.txt

Zadanie 3.

Zaimplementowałam kopiec binarny o ograniczonej ilości elementów (tworzący zbiór największych podanych elementów)

Przykłady działania:

```
base: [4, 7, 2, 8, 0, 6, 3, 12, 7, 5]
n = 7:
[4, 7, 5, 8, 7, 12, 6]
n = 2:
[8, 12]
n = 1:
[12]
```

```
base: [40, 20, -8, -3, -7, 2, 8, 0, 6, 3, 12, 7, 15]
n = 7:
[6, 8, 7, 12, 20, 40, 15]
n = 2:
[20, 40]
n = 1:
[40]
```

Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/3.py

Zadanie 4.

Moja funkcja przyjmuje za bazę do obliczeń drzewo typu *parse_tree*, który przekształca *string* na drzewo binarne, i zwraca pochodną danego wyrażenia. Działa dla liczb całkowitych jednocyfrowych - dodatnich i ujemnych (ujemne powinny znajdować się w nawiasach) i dowolnego podanego operatora (jednoznakowego). (UWAGA: gdy wpisujemy funkcje: sin, cos, ln, exp nie dajemy nawiasów wokół tej funkcji oraz jeśli ‘zawartość’ powyższych funkcji jest jednoznakowa, czyli np. sinx zapisujemy to bez nawiasów wewnątrz funkcji).
Przykłady działania:

```
ln(5*n)
n
((1/(5*n))*((0*n)+(5*1)))
```

$(ln(5 * n))' = \frac{5}{5n} = \frac{1}{n}$

$((1/(5 * n)) * ((0 * n) + (5 * 1))) = \frac{1}{5n} * 5 = \frac{1}{n}$

```
(4*cos(y+5))
y
((0*cos(y+5))+(4*((-1)*(sin(y+5)*(1+0)))))
```

$(4 * cos(y + 5))' = (-4) * sin(y + 5)$

$((0 * cos(y + 5)) + (4 * ((-1) * (sin(y + 5) * (1 + 0))))) = (-4)sin(y + 5)$

```
(exp(a+5)*4)
a
(((exp(a+5)*(1+0))*4)+(exp(a+5)*0))
```

$(exp(a + 5) * 4)' = 4 * exp(a + 5)$

$(((exp(a + 5) * (1 + 0)) * 4) + (exp(a + 5) * 0)) = 4 * exp(a + 5)$

```
(((3+5)*4)*sin(x+1))
x
(((((0+0)*4)+((3+5)*0))*sin(x+1))+(((3+5)*4)*(cos(x+1)*(1+0))))
```

$$(((3 + 5) * 4) * sin(x + 1))' = ((8 * 4) * sin(x + 1))' = (32 * sin(x + 1))' = 32 * cos(x + 1)$$

$$(((((0 + 0) * 4) + ((3 + 5) * 0)) * sin(x + 1)) + (((3 + 5) * 4) * (cos(x + 1) * (1 + 0)))) = 0 + ((8 * 4) * cos(x + 1)) = 32 * cos(x + 1)$$

Link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/ba15f9b322de1b77bfe4c334f7f5e6ed0a3aa547/lista6/4.py