Julia Grzegorzewska, Wiktoria Fimińska

RAPORT LISTA 6

ZADANIE 1

W tym zadaniu stworzyłyśmy dwie klasy, jedna określająca dany węzeł, druga całe drzewo binarne. W klasie *TreeNode* skupiamy się na węźle: czy ma ‘dzieci’ i z której strony, czy jest wierzchołkiem, możemy oddzielić węzły oraz zastosowałyśmy metodę, w której możemy zmienić właściwości węzła. W klasie *BinaryTree* określiłyśmy w metodzie __*str*__ jak ma prezentować się nasze wirtualne drzewo. Zastosowałyśmy odpowiednią skalę, aby każda wartość mieściła się i nie nachodziła na inną. Tutaj korzystamy z klasy *TreeNode*.

Obok widzimy jak prezentuje się ‘output’. A poniżej, co wpisałyśmy, aby uzyskać taki wygląd.

```
Added 3 to tree with payload 4
the level of tree
0
And tree itself
 3
```

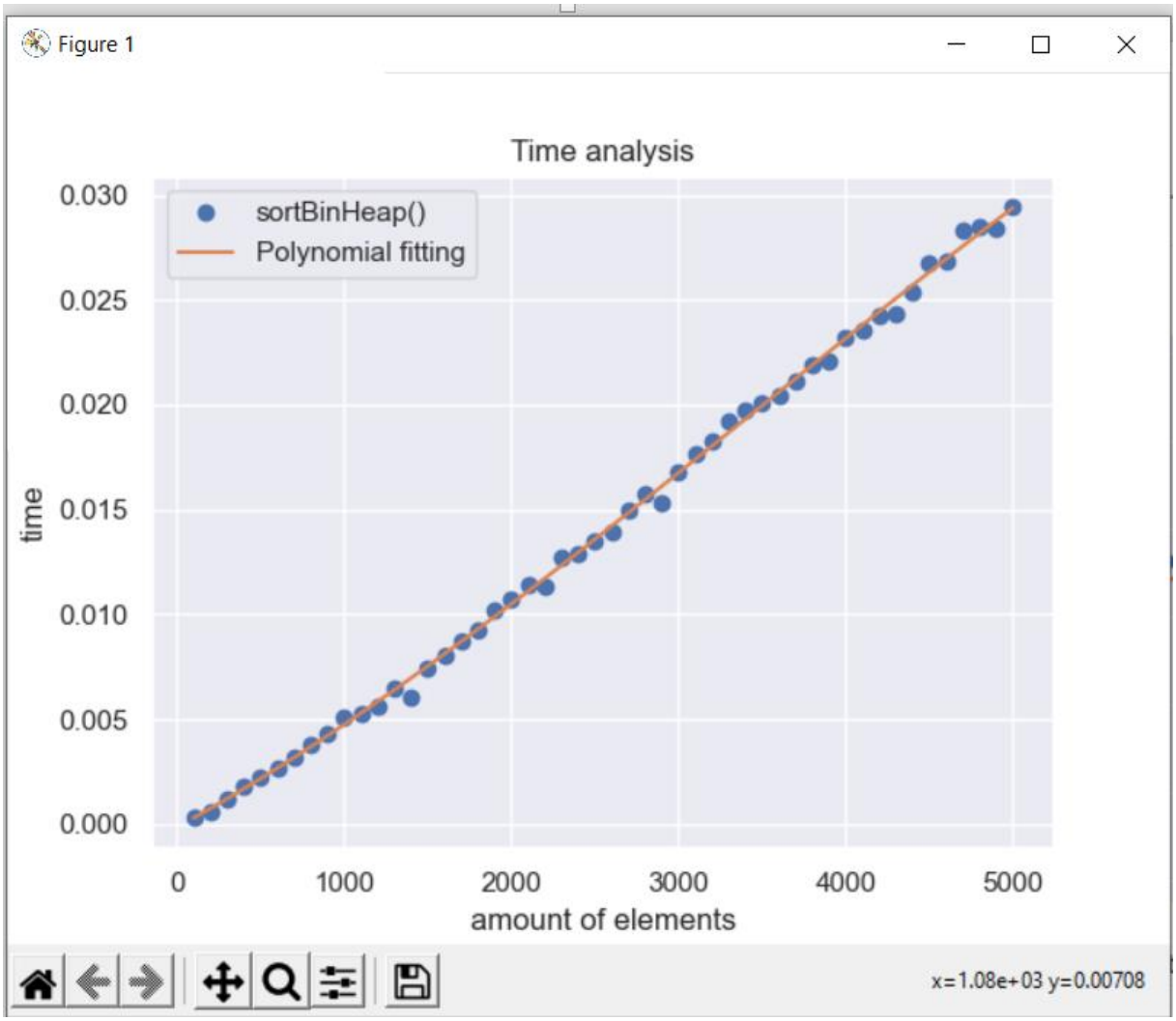
```
adding 4 to the tree
print payload of 4
4
print tree
                  3


         2                   4


                                  7


                                5    9
```

```python
mytree = BinarySearchTree()
print("Added 3 to tree with payload 4")
mytree[3] = 4
print("the level of tree")
print(mytree.max_level)
print("And tree itself")
print(mytree)
print("Added 4 to tree woth payload 1")
mytree[4] = 1
print("the level of tree")
print(mytree.max_level)
print("tree itself")
print(mytree)
print("Added 6 to tree with payload 2")
mytree[6] = 2
print("print tree")
print(mytree)
print("Added 2 to tree with payload 5")
mytree[2] = 5
print("print tree")
print(mytree)
```

```python
print("Added 7 to tree with payload 1")
mytree[7] = 1
print("print tree")
print(mytree)
print("Added 9 to tree with payload 3")
mytree[9] = 3
print("print tree")
print(mytree)
print("Added 10 to tree with payload 5")
mytree[5] = 5
print("print tree")
print(mytree)
print("Added 4 to tree with payload 3 so it's 4")
mytree[4] = 3
print("print payload of 4")
print(mytree[4])
print("deleting 4")
mytree.delete(4)
print(mytree[4])
print("delete 6 two times")
mytree.delete(6)
print("payload of 6:")
print(mytree[6])
mytree.delete(6)
print(mytree)
print("adding 4 to the tree")
mytree.put(4)
print("print payload of 4")
print(mytree[4])
print("print tree")
print(mytree)
```

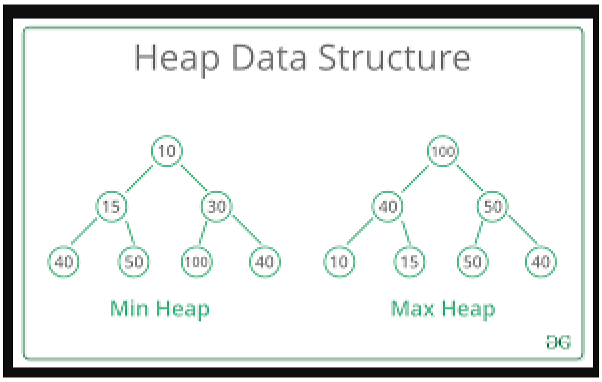
ZADANIE 2

Tutaj tworzymy klasę opisującą kopiec binarny. Definicja kopca: *„Kopiec binarny – tablicowa struktura danych reprezentująca drzewo binarne, którego wszystkie poziomy z wyjątkiem ostatniego muszą być pełne. W przypadku, gdy ostatni poziom drzewa nie jest pełny, liście ułożone są od lewej do prawej strony drzewa.”*. Aby móc posortować elementy w kopcu określiłyśmy w klasie metodę, która usuwa minimalną wartość, ale jednocześnie też zwraca ją. Z tej metody korzystamy w funkcji *sortBinHeap*, aby posortować wartości w kopcu. Zwraca nam ona listę posortowanych wartości. W funkcji *time_checker* randomowo określamy wartości w liście, z której chcemy stworzyć kopiec przy pomocy biblioteki *numpy*, a dokładnie *np.random.permutation* w zależności od długości listy i tworzymy symulację. Funkcja *simulation* zwraca plot w zależności od ilości elementów do posortowania oraz czasu, w jakim zostało wykonane sortowanie. Poniżej wynik symulacji do 5000 tysięcy elementów, wartości co 100.



ZADANIE 3

Tutaj modyfikujemy klasę *BinHeap* w klasę *LimitedBinHeap*, w której możemy tylko określoną liczbę elementów przechowywać, jednocześnie mają one być największe. Możemy dodawać zarówno pojedyncze elementy poprzez metodę *insert*, ale także całą listę w metodzie *insertList*. Trzeba pamiętać o specyfice kopca.




```
print("Create a LimitedBinHeap with given limit.")
bh = LimitedBinHeap(4)
print("Insert 5")
bh.insert(5)
print(bh)
print("Insert 2")
bh.insert(2)
print(bh)
print("Insert 7")
bh.insert(7)
print(bh)
print("Insert 5")
bh.insert(5)
print(bh)
print("Insert 6")
bh.insert(6)
print(bh)
print("Insert 3")
bh.insert(3)
print(bh)
print("Insert a list [4, 1, 6, 9]")
bh.insertList([9, 4, 1, 6])
print(bh)
```

```
C:\Users\Uzytkownik\AppData\Local\Programs
Create a LimitedBinHeap with given limit.
Insert 5
[5]
Insert 2
[5, 2]
Insert 7
[7, 2, 5]
Insert 5
[7, 2, 5]
Insert 6
[7, 6, 5, 2]
Insert 3
[7, 6, 5, 3]
Insert a list [4, 1, 6, 9]
[9, 7, 5, 6]
```

```
Given function to differentiate:
(-cos(2*x)+(x^3))
Counted derivative:
(2*sin(2*x))+(3*(x^2))
                                        +

                    *                                       *

          2                   sin                 3                   ^

                                  *                                  x          2

                              2     x
```

ZADANIE 4

W zadaniu 4 mieliśmy przy wykorzystaniu drzewa binarnego, w którym przechowujemy wyrażenie matematyczne stworzyć drzewo pochodnej tego wyrażenia. W naszym przypadku ma działać dla *sin*, *cos, exp, ln* oraz dla różnych wyrażeń z podaną niewiadomą, np. *x* lub *a*.

W metodzie *buildParseTree* tworzymy drzewko z podanego wyrażenia matematycznego. To z niego odczytuje nasza funkcja *derivative_Tree* odczytuje wartości przy pomocy *unpacking_Tree* i zwraca nam listę cząstek w drzewie. Następnie przy pomocy *derivative_from_list* tworzymy listę wartości pochodnej wyrażenia. Funkcja *clearing_derivative* jest funkcją pomocną, aby pozbyć się dwóch minusów obok siebie itp. Tworzymy listę *stringów* i budujemy nasze nowe drzewko. Przy pomocy klasy *TreeVisualisation* możemy wyprintować nasze drzewko. Poniżej przykładowe wyniki:

```
Given function to differentiate:
(sin(3*x)+(x^5))
Counted derivative:
(3*cos(3*x))+(5*(x^4))
                                  +

                 *                                   *

        3                cos                5                 ^

                             *                                 x          4

                          3     x
```

```
Given function to differentiate:
ln(x+3)
Counted derivative:
1/(x+3)
          /

     1            +

              x      3
```

```
Given function to differentiate:
exp(a^2)
Given function to differentiate:
(a^2)
Counted derivative:
(2*a)*exp(a^2)
                  *

          *                   exp

     2           a           ^

                          a      2
```

```
Given function to differentiate:
ln(t+2)
Counted derivative:
1/(t+2)
          /

     1            +

              t      2
```