DATA STRUCTURES AND ALGORITHMS

HOMEWORK – 3

Q1 Develop an implementation of the basic symbol-table API that uses 2-3 trees that are not necessarily balanced as the underlying data structure. Allow 3-nodes to lean either way. Hook the new node onto the bottom with a black link when inserting into a 3-node at the bottom.

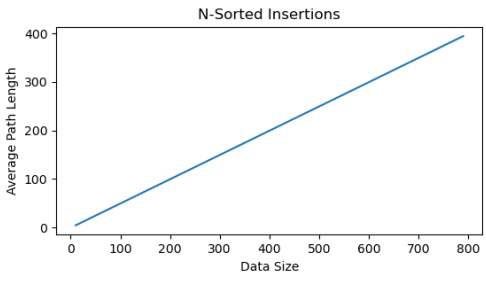
Ans: The output that I obtained for running the program symboltable.py is shown below:

```
C:\Users\prana\PycharmProjects\q1\venv\Scripts\python.exe C:/Users/prana/PycharmProjects/q1/symbol_table.py
Empty symbol table created!
is_empty() returns: True
Adding keys and corresponding values to the table...
is_empty() = False
keys() = {'H', 'R', 'S', 'G', 'I', 'E', 'T', 'M', 'N', 'A'}
size() = 10
contains(T) = True
delete(A) = None
keys() = {'H', 'R', 'S', 'G', 'I', 'E', 'T', 'M', 'N'}
contains(A) = False

Process finished with exit code 0
```

Q2. Run experiments to develop a hypothesis estimating the average path length in a tree built from (i) N-random insertions. (ii) N-sorted insertions?

Ans: The output I obtained by running avgpathlength.py is shown below:



	N-Random Insertions	N-Sorted Insertions
10	3.50	4.50
110	5.80	54.50
210	8.46	104.50
310	8.71	154.50
410	9.44	204.50
510	9.08	254.50
610	9.89	304.50
710	10.02	354.50

Since the type of tree to be implemented was not specified in the question, I implemented a binary search tree for this question.

For N-random insertions, the values are being inserted into the tree and are shuffled randomly. The corresponding graph for this scenario resembles a logarithmic function. Thus, the hypothesis can be a logarithmic function in this case.

In contrast, for N-sorted insertions, where the values inserted into the tree are sorted while inserting, the average path length of the tree for the different number of nodes remains linear throughout. Thus, the hypothesis can be a linear function in this case.

Q3. Write a program that computes the percentage of red nodes in a given red-black tree. Test program by running at least 100 trials of the experiment of increasing N random keys into an initially empty tree for N=10^4, 10^5 and 10^6 and formulate a hypothesis.

Ans: The percentage of red nodes in the red-black tree for number of nodes N = 10,000, 100,000 and 1,000,000 are:

N = 10,000 -> Percentage of red nodes = 27.15%

N = 100,000 -> Percentage of red nodes = 25.42%

N = 1,000,000 -> Percentage of red nodes = 24.93%

The basic idea behind this problem is we recursively traverse the red-black tree; first, we recursively traverse the left subtree of the red-black tree and count the number of red nodes. Then, we recursively traverse the right subtree of the red-black tree and add the number of red nodes to the previous value obtained. We divide this by the total number of nodes, or size of the tree, and multiply by 100 to obtain the percentage.

As we can see from the figures obtained, the percentage of red nodes remains in the neighborhood of 25%.

Q4. Run empirical studies to compute the average and std deviation of the average length of a path to a random node (internal path length divided by tree size) in a red-black BST built by insertion of N random keys into an initially empty tree, for N from 1 to 10,000. Do at least 1,000 trials for each size.

Ans: The values obtained for the mean and standard deviation of the average internal path length to a random node is shown below:

```
For 10 trials and 1 insertions, the mean path length is 0.00 with a standard deviation of 0.00
For 10 trials and 500 insertions, the mean path length is 7.31 with a standard deviation of 0.07
For 10 trials and 1000 insertions, the mean path length is 8.38 with a standard deviation of 0.08
For 10 trials and 1500 insertions, the mean path length is 8.97 with a standard deviation of 0.08
For 10 trials and 2000 insertions, the mean path length is 9.39 with a standard deviation of 0.06
For 10 trials and 2500 insertions, the mean path length is 9.73 with a standard deviation of 0.10
For 10 trials and 3000 insertions, the mean path length is 9.99 with a standard deviation of 0.09
For 10 trials and 3500 insertions, the mean path length is 10.23 with a standard deviation of 0.09
For 10 trials and 4000 insertions, the mean path length is 10.42 with a standard deviation of 0.08
For 10 trials and 4500 insertions, the mean path length is 10.59 with a standard deviation of 0.07
For 10 trials and 5000 insertions, the mean path length is 10.75 with a standard deviation of 0.06
For 10 trials and 5500 insertions, the mean path length is 10.89 with a standard deviation of 0.12
For 10 trials and 6000 insertions, the mean path length is 11.10 with a standard deviation of 0.14
For 10 trials and 6500 insertions, the mean path length is 11.10 with a standard deviation of 0.04
For 10 trials and 7000 insertions, the mean path length is 11.21 with a standard deviation of 0.06
For 10 trials and 7500 insertions, the mean path length is 11.37 with a standard deviation of 0.13
For 10 trials and 8000 insertions, the mean path length is 11.42 with a standard deviation of 0.06
For 10 trials and 8500 insertions, the mean path length is 11.52 with a standard deviation of 0.05
For 10 trials and 9000 insertions, the mean path length is 11.61 with a standard deviation of 0.05
For 10 trials and 9500 insertions, the mean path length is 11.70 with a standard deviation of 0.05
For 10 trials and 10000 insertions, the mean path length is 11.79 with a standard deviation of 0.04
```

As we can see from the values obtained, the mean of the average path length to a random node ranges from 7 to 11. Comparing with the number of node insertions (N), we can say that the mean has an approximate logarithmic variation with the number of node insertions. In addition, the standard deviation of the path length is also less.

Q5. Implement the rank() and select() ordered operations for a BST. Use data set linked below. (i) What is the value of select(7) for the data set? (ii) What is the value of rank(7) for the data set?

Ans: The values for rank(7) and select(7) are shown below:

Rank(7) = 6

Select(7) = 8

The rank of a given node in a binary search tree basically tells the number of keys in the tree that have a value less than the key of that node. This shows that in the binary search tree implemented, there are 6 keys which have a value less than 7.

The select operation in a binary search tree finds the key of a given node such that precisely k other keys in the binary search tree are smaller. This shows that in the binary search tree implemented, the 8 is the key of the 7^{th} node such that there are 7 smaller keys.