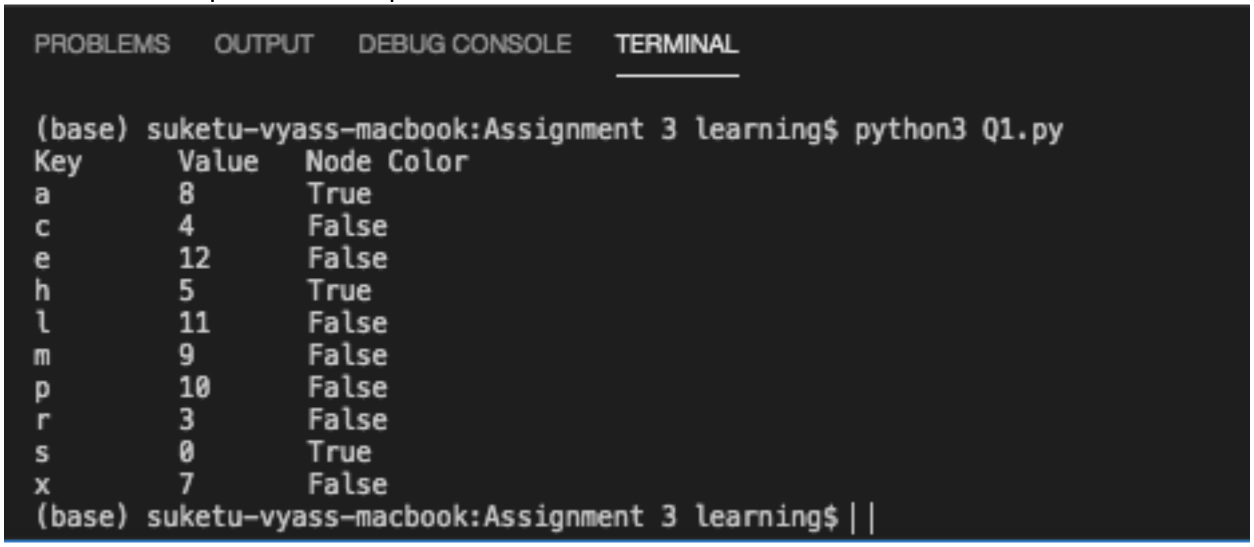
Homework 3: Pratik Mistry – DSA Spring 2020 (pdm79)

Solution 1:

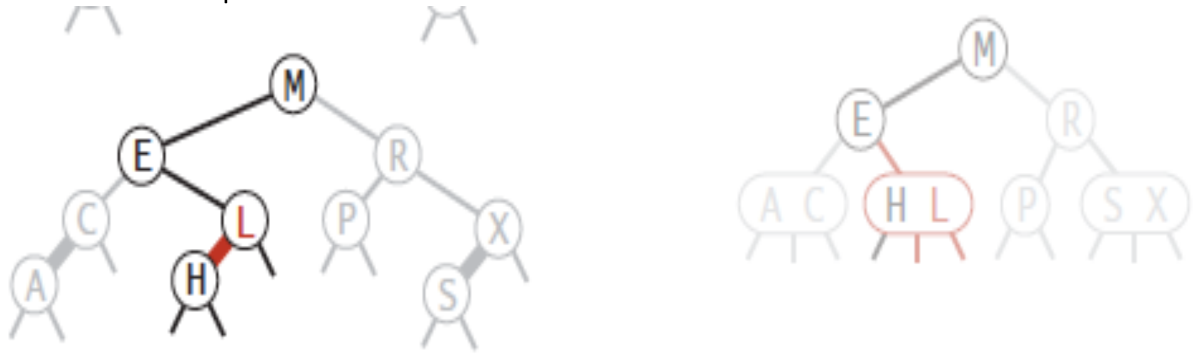
Explanation:

- The Symbol Table API implemented is 2-3 Red Black BST and some of the important operations implemented are: put(), inorder(), rotateLeft(), rotateRight(), flipColors()
- The main function has two main input functions: **1.** Hard coded array example from the slides i.e. "SEARCHEXAMPLE" and **2.** Dataset set given i.e. select-data.txt
- The output of the code is the inorder traversal of the 2-3 Red-Black tree that would be created by the program which will print: **Key, Value and Node Color.** The node color is printed will be either "True" or "False" as True denotes "RED" node and False denotes "BLACK" node.
- Below is the output for first input:



```
(base) suketu-vyass-macbook:Assignment 3 learning$ python3 Q1.py
Key      Value   Node Color
a        8       True
c        4       False
e        12      False
h        5       True
l        11      False
m        9       False
p        10      False
r        3       False
s        0       True
x        7       False
(base) suketu-vyass-macbook:Assignment 3 learning$ ||
```

- The above output for the input taken can be verified with the diagrammatic view of the tree structure present in the slides as shown below:



- Thus, we can see that the output of program is correct as Nodes A, H and S is red nodes (and part of 3 node).
- Similarly, the output of the second input will be on similar lines. I am not including/attaching output for second input because the dataset is huge and can be best viewed when executed.

- **NOTE**:
  - For input 1: uncomment **main_input1()** and for input 2: uncomment **main_input2()** respectively in the main function.
  - Some of the logic for the program is referred from lecture slides: symbol-tables.pdf

Solution 2: I have calculated Average Path Lengths for ordered insertion and random insertion of keys in both – Binary Search Tree and 2-3 Red-Black BST. Implementation for 2-3 Red-Black BST is just for the sake of understanding concepts better.

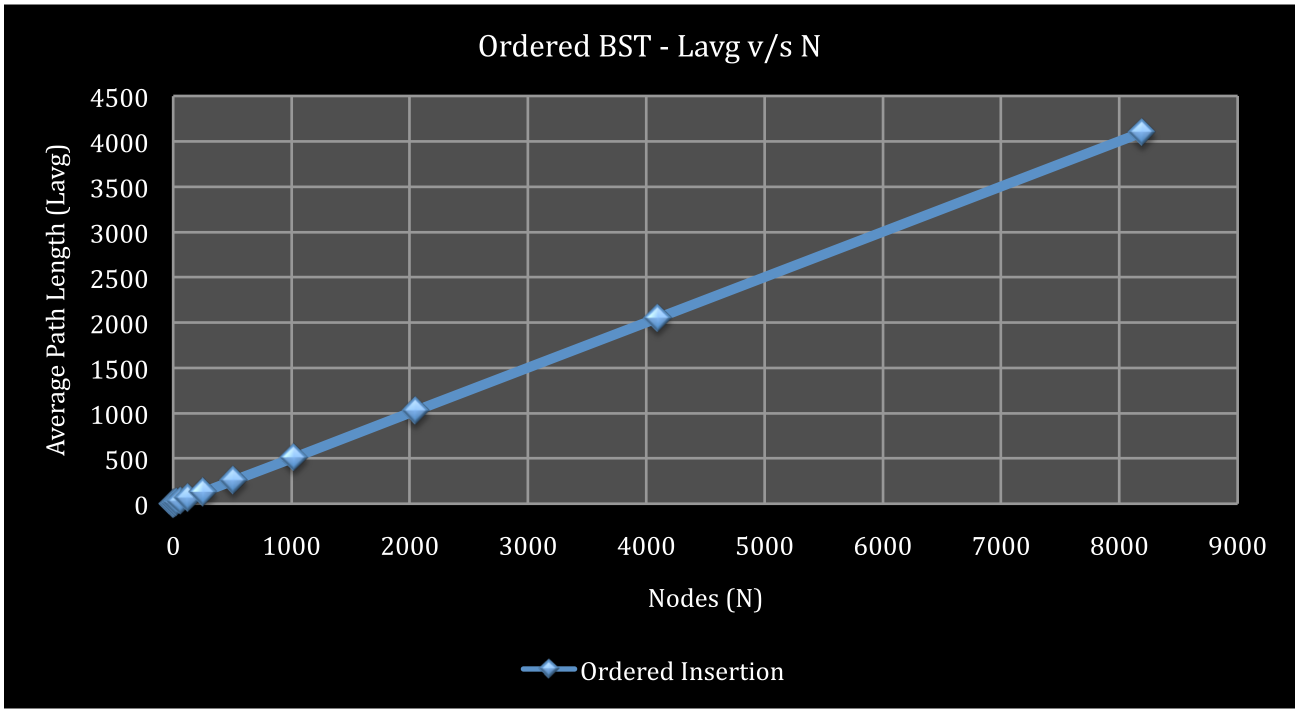
A. **Average Path Length For Binary Search Tree**: Use Q2_BST.py for this part

Output Table:

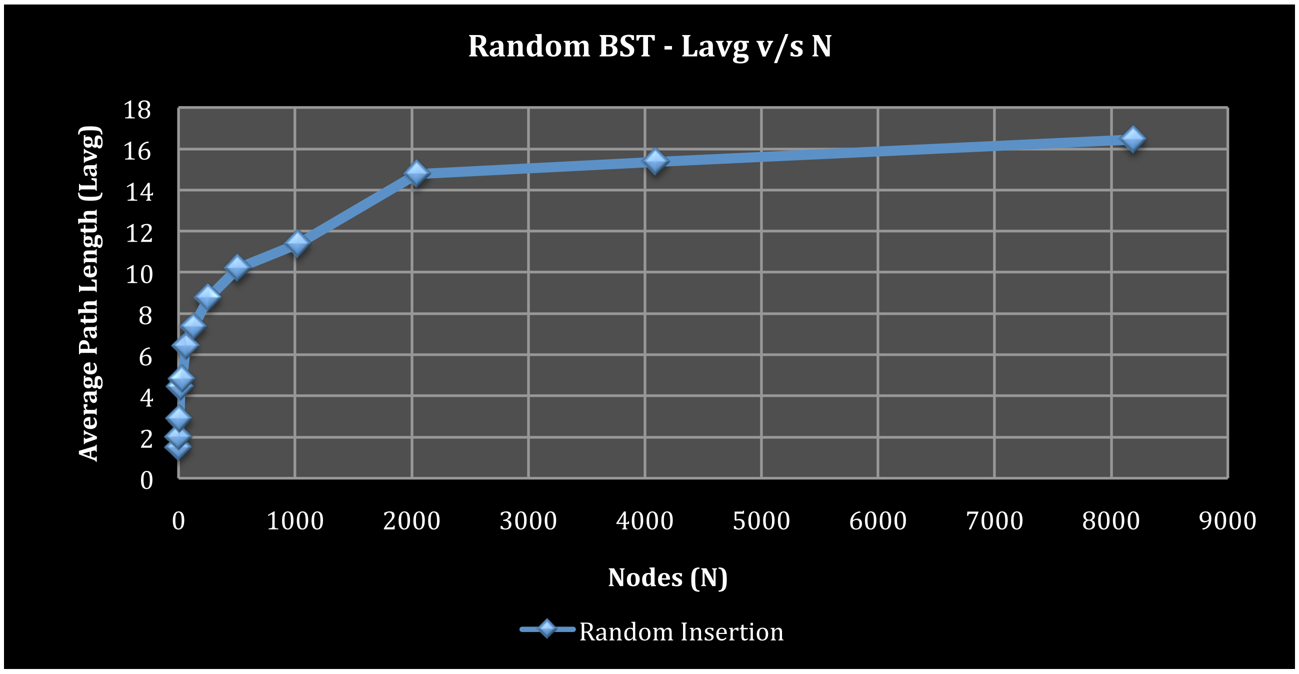
Binary Search Tree		
N	**Average Path Length (Ordered Inserted)**	**Average Path Length (Random Inserted)**
2	1.5	1.5
4	2.5	2
8	4.5	2.875
16	8.5	4.438
32	16.5	4.812
64	32.5	6.422
128	64.5	7.359
256	128.5	8.77
512	256.5	10.21
1024	512.5	11.377
2048	1024.5	14.786
4096	2048.5	15.38
8192	4096.5	16.455

Graphs:

a. Average Path Length v/s Nodes – Ordered BST



b. Average Path Length v/s Nodes – Random BST



Explanation:

- The tree structure implemented is Binary Search Tree and average path length is calculated for trees with: Ordered Insertion of elements and Random Insertion of elements
- The path length for a Node N is given as:
  Length (N) = Length (Left Tree) + Length (Right Tree) + Size of Tree at node N
- Thus, average path length of tree is given as:
  (Length at Root Node)/ (Size of Tree at Root Node)
- The reason why Average Path Length of ordered inserted tree is higher than random inserted tree is because while inserting keys in order, the height of the tree goes on increasing i.e. the tree is not balanced leading to asymmetric in nature
- While, when the keys are randomly inserted in the tree, the trees are is balanced in nature compared to ordered inserted tree (but not perfectly balanced) due to which the height of tree remains less compared to ordered insertion thus more symmetric in nature
- By looking at the table and graphs above, we can see that Average Path Length
  - For Ordered Inserted Keys: (N/2) + 0.5 ~ **N**
  - For Random Inserted Keys: **~log (N)**

B. **<u>Average Path Length For 2-3 Red-Black BST:</u>** Use Q2_RB_BST.py for this part

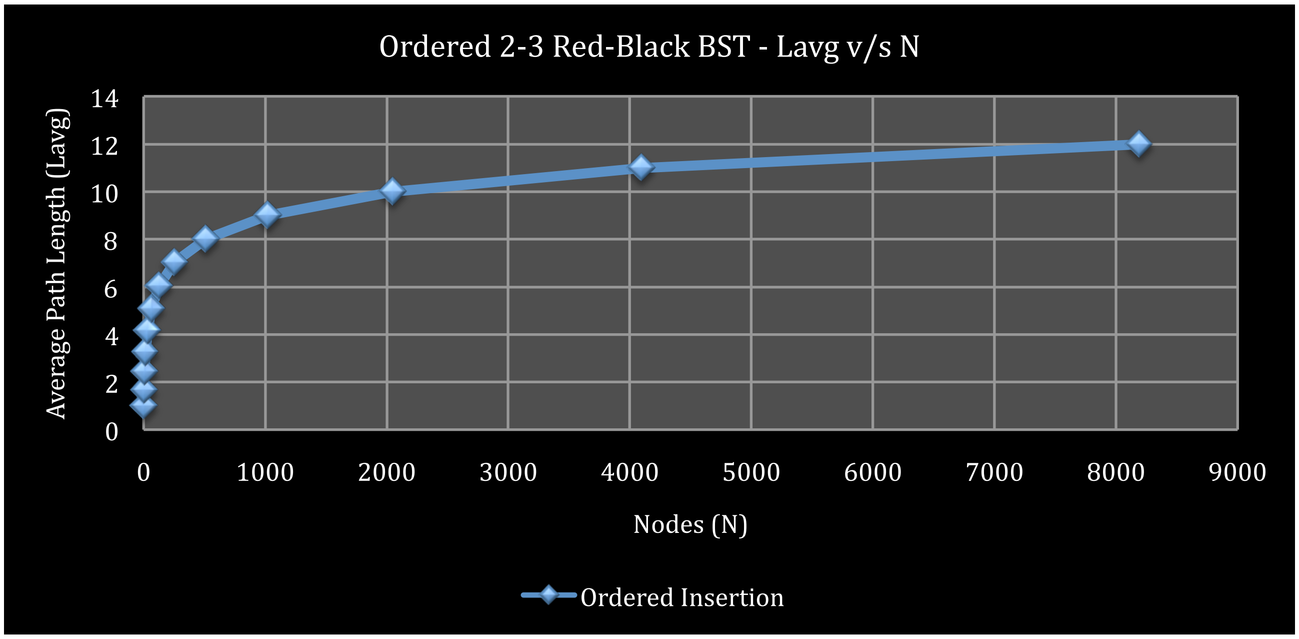
Output Table:

2-3 Red-Black BST		
N	**Average Path Length (Ordered Inserted)**	**Average Path Length (Random Inserted)**
2	1	1
4	1.667	1.667
8	2.429	2.429
16	3.267	2.545
32	4.161	3.435
64	5.095	4.353
128	6.055	5.253
256	7.031	5.335
512	8.018	7.234
1024	9.01	7.244

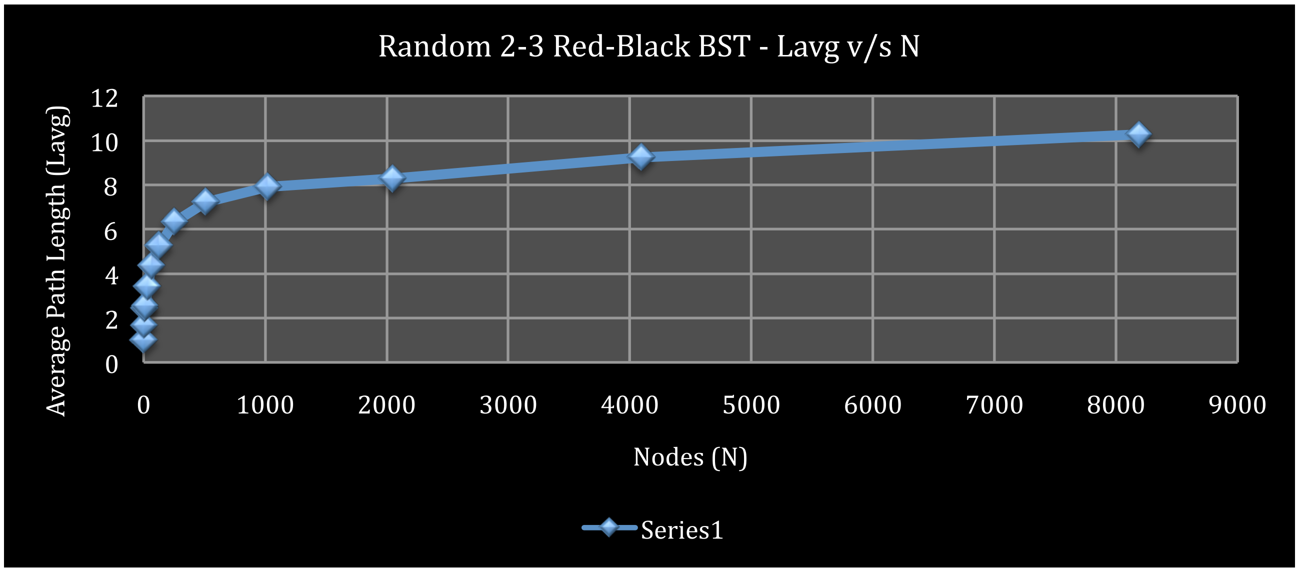
2048	10.005	8.28
4096	11.003	9.252
8192	12.002	10.267

Graphs:

a. Average Path Length v/s Nodes – Ordered 2-3 Red-Black BST



b. Average Path Length v/s Nodes – Random 2-3 Red-Black BST



Explanation:

- The path length of tree is given as sum of all the paths to each node (except the red nodes) from root.
- The average path length of tree is given as:
  (Length at Root Node)/ ((Size of Tree at Root Node) – (Total Red Nodes))
- The reason why Average Path Length of ordered inserted tree is higher than random inserted tree is because the number of red nodes in random insertion is way more than ordered insertion and I have excluded red nodes while calculating average path length.
- By looking at the table and graphs above, we can see that Average Path Length:
  - For Ordered inserted tree: log(N)-1 ~ **log(N)**
  - For Random Inserted tree: **less than log(N)-1**

Conclusion:

- When the average path length of the tree created using Binary Search Tree is much higher than that of 2-3 Red-Black BST because tree in BST is not balanced and symmetric compared to that of 2-3 Red Black BST. Also, height of tree is less incase of Red Black BST (when red nodes are not taken into account for calculating path length and height)
- The average path length of tree created using ordered insertion of keys is very less than random insertion of keys again due to balancing criteria between the two.
- Average Path Length (Lavg) for all the implementations can be summarized as:
  Lavg(2-3_RB_BST_random_insertion) < Lavg(2-3_RB_BST_ordered_insertion) < Lavg(BST_random_insertion) < Lavg(BST_ordered_insertion)

NOTE:

1. Use Q2_BST.py for Average Path Length in Binary Search Tree
2. Use Q2_RB_BST.py for Average Path Length in 2-3 Red-Black BST
3. Concept for path length is referred from:
   https://www.coursera.org/lecture/analysis-of-algorithms/path-length-Bzppb

Solution 3:

Explanation**:**

- The tree structure implemented is balanced 2-3 Red-Black BST and tree sizes are 10^4,10^5 and 10^6 as per mentioned in question
- The keys to be inserted in the tree is randomly shuffled and then inserted
- Percentage of red nodes for tree size 10000 is: 25.408499999999993
- Percentage of red nodes for tree size 100000 is: 25.391710000000003
- Percentage of red nodes for tree size 1000000 is: 25.38891999999995
- Thus, the average percentage of red nodes for all the tree sizes is: 25.39636733333333
- It took approximately 5 hours for the program to execute successfully
- Machine specifications are:
    - Processor: 2 GHz Intel Core 2 Duo
    - Memory: 8 GB 1067 MHz DDR3
    - Graphics: NVIDIA GeForce 9400M 256 MB
    - OS: Mac OS Yosemite 10.10.5

Solution 4:

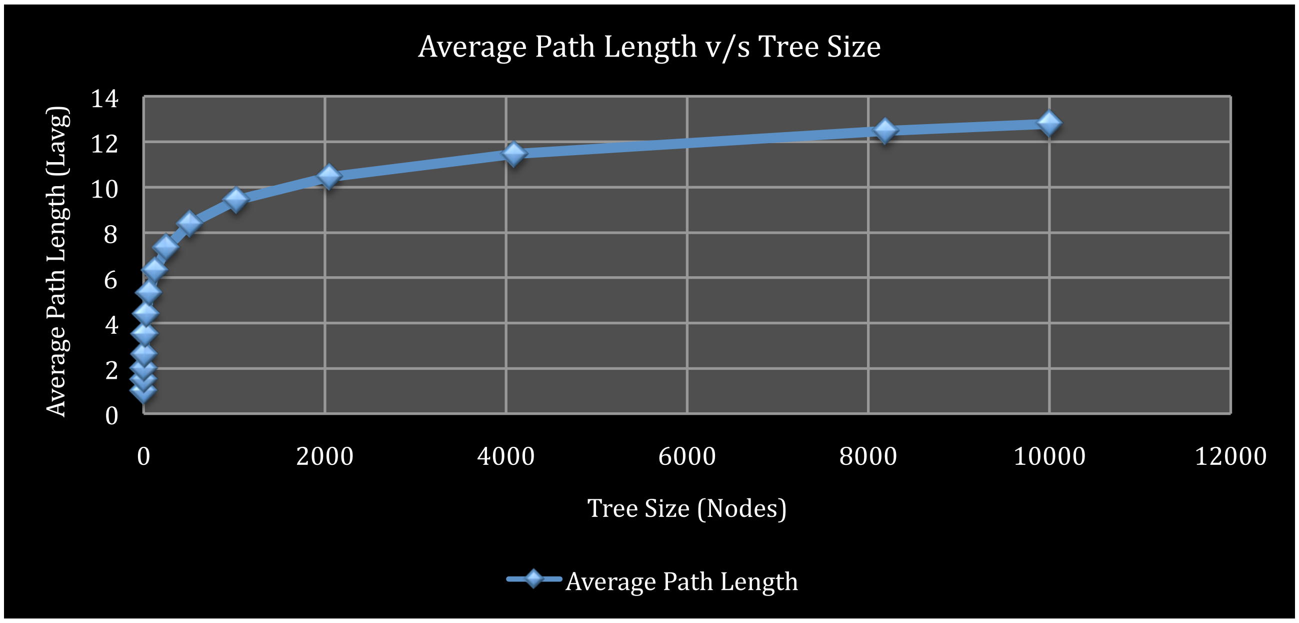
Output Table:

Nodes	**Average Path Length**	**Standard Deviation**
1	1	0
2	1.5	0
4	2	0
8	2.642625	0.043502694
16	3.5175625	0.099687774
32	4.4009375	0.067017226
64	5.336484375	0.058951564
128	6.309273438	0.056999366
256	7.331199219	0.088562159
512	8.378990234	0.102316111
1024	9.417942383	0.089987404
2048	10.43488232	0.068985668

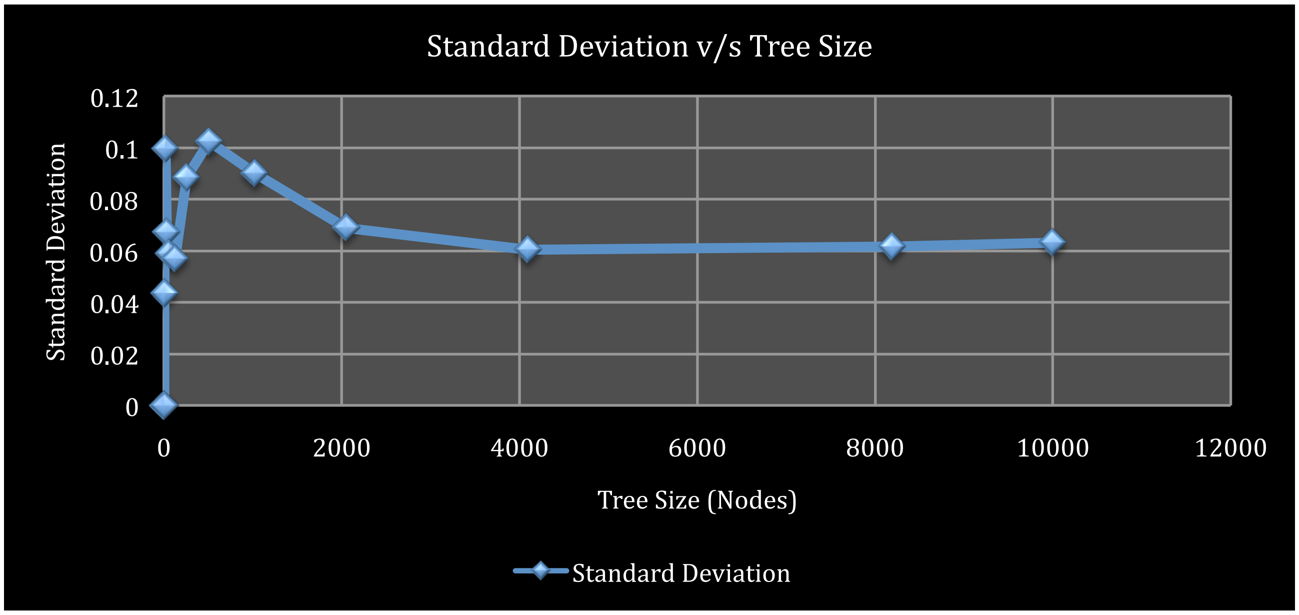
4096	11.45197974	0.060488062
8192	12.47056458	0.061625681
10000	12.7930549	0.063247453

Graphs:

a. Average Path Length v/s Tree Size



b. Standard Deviation v/s Tree Size



Explanation**:**

- The tree structure implemented is 2-3 Red-Black BST
- Firstly, I tried to run program for tree sizes 1 to 10000 but it took more than 40 hours to complete tree with sizes from 1 to 2105 and had to stop the program. Please find the readings of the output average path length and standard deviation of nodes 1 to 2105 in the **q4_results_allNodes.csv** file.
- Thus, I decided to execute trees for selected sizes only for faster results. Sizes are: 1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192 and 10000 i.e. powers of 2 till 10000 whose output is in **q4_results_selectedNodes.csv** file.
- The internal path length for a Node N is given as:
  Length (N) = Length (Left Tree) + Length (Right Tree) + Size of Tree at node N
- Thus the average length is given as is: Length (N)/(Size of tree at Root Node)
- Red nodes are also taken into consideration while calculating the length because we need to calculate internal path lengths
- As seen from graph 1,the average path length for 2-3 Red-Black Tree created by random insertion is approximately log (N)-1 ~ **log (N)**
- The average standard deviation is approximately 0.06 for above input sizes as seen in graph and table and will surely increase by 0.01 to 0.02 if all the nodes are taken into considered instead of selected nodes, which is quite small.
- It took around 45-60 mins for the execution of program.
- The logic to write results in CSV file is referred from: https://www.programiz.com/python-programming/writing-csv-files
- Machine specifications are:
  - Processor: 2 GHz Intel Core 2 Duo
  - Memory: 8 GB 1067 MHz DDR3
  - Graphics: NVIDIA GeForce 9400M 256 MB
  - OS: Mac OS Yosemite 10.10.5
- **NOTE**:
  - Since, the program took very long time i.e. 40 hours to find results for nodes 1 to 2105, I had to stop the execution and then find results for only selected nodes.
  - For considering all nodes, uncomment **main_allNodes()** function in the main function (but only recommended if computer configuration is very high) and execute the code.
  - For considering only selected nodes, uncomment **main_selectedNodes()** function in the main function and execute the code.
  - Some of the logic of the program of 2-3 Red-Black tree is referred from lecture slides – symbol-tables.pdf

Solution 5:

Explanation**:**

- The tree structure implemented is Binary Search Tree and dataset taken is select-data.txt which was uploaded on Sakai.
- The select () and rank() operations are implemented in the code which will return the “Key” for the entered rank and return the “Rank” for the entered key
- The output for select (7) is: 8
- The output for rank (7) is: 6
- The concept of rank() and select() is referred from the lectures slides – symbol-tables.pdf