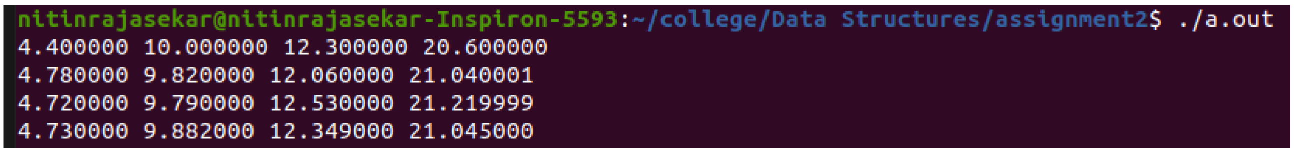
1)
Bstsort is implemented using inorder traversal .
First, the leftmost node of the bst is printed, next the value of its parent node is printed, and then the value of the right node of the parent .
In this way, the entire tree is traversed in ascending order .
Each node is visited once, and the time complexity is O(n) .

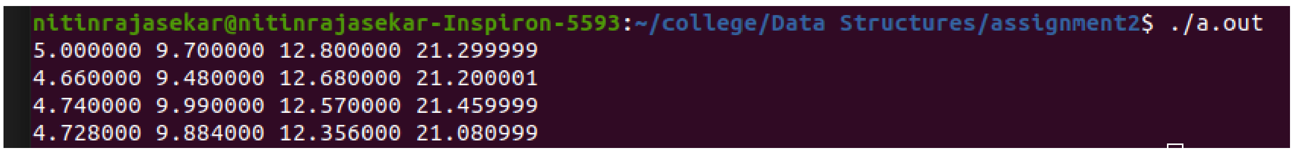
2)
Here finddepth uses helper function findepth_get(for datatype conversion) and increase .
Increase is used to traverse down the tree in search for the node in search of the particular node in the tree, if not found it recursively checks the left and right subtrees, and increments a depth variable by 1 everytime it does so .
Every node is checked once, so the time complexity is linear and O(n) .

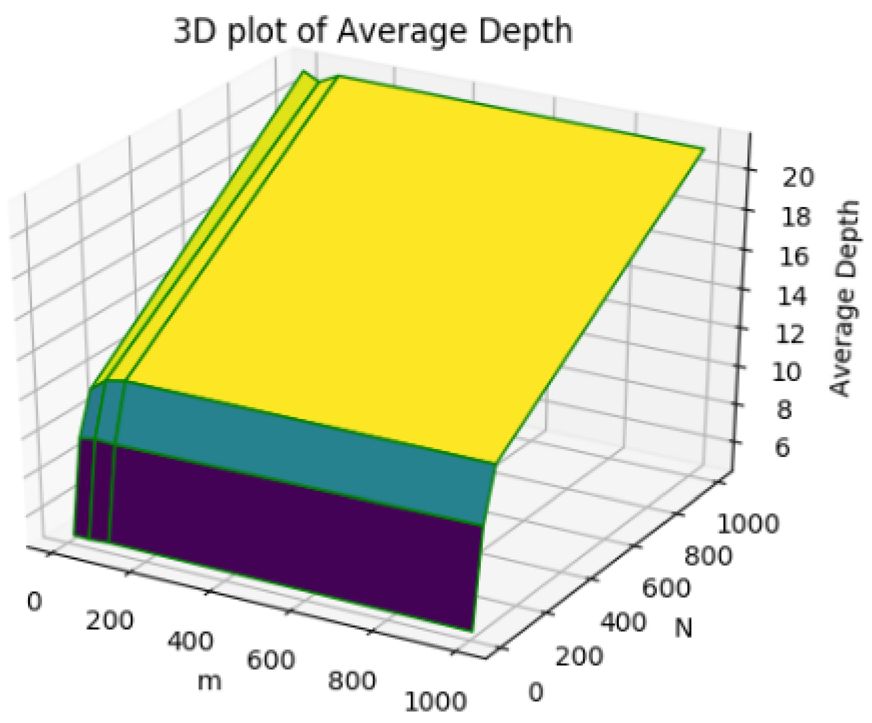
3)
Height_get recursively checks the height of the the left and right subtrees of the parent node, beginning from the root .
It then checks which cubtree has the biggest height and returns it, adding one in the end as it is the parent node .Again the traversal has time complexity O(n) .

4)
Isbst_trav checks whether for every node the maximum value in the left subtree is smaller than the parent node and that the minimum value on the right subtree is bigger than the parent node, keeping track of the current max and min values, and looks at each node once, the time complexity is O(n) .

5)
The function avgdepth() calls srand() and then calls the function randomgenerator ()multiple times . RandomBST calls srand() and calls randomgenerator() once .Randomgenerator() does not contain srand() and genereates a random tree within the input range, by this arrangement random trees can be generated without calling srand multiple times within the same time span . Avgdepth implements the required output by calling the functions within nested loops .
Looking at how the slope of the height to the number of nodes decreases drastically with time while the net height increases, it appears that the relation between the height and the number of nodes is a function of the order **O(log(n))** .


```
nitinrajasekar@nitinrajasekar-Inspiron-5593:~/college/Data Structures/assignment2$ ./a.out
4.400000 10.000000 12.300000 20.600000
4.780000 9.820000 12.060000 21.040001
4.720000 9.790000 12.530000 21.219999
4.730000 9.882000 12.349000 21.045000
```


```
nitinrajasekar@nitinrajasekar-Inspiron-5593:~/college/Data Structures/assignment2$ ./a.out
5.000000 9.700000 12.800000 21.299999
4.660000 9.480000 12.680000 21.200001
4.740000 9.990000 12.570000 21.459999
4.728000 9.884000 12.356000 21.080999
```



trinket_plot.png

6)
Similar to bstsort, inRange uses inorder traversal, if the value of the current node is bigger than the range, the left subtree is checked, else if it is smaller, the right subtree is checked, else the value is printed. Each node is traversed once, and the time complexity is O(n) .