CSCE 221 Cover Page

First Name: Maria Last Name: Dmitrievskaia UIN: 927009911

User Name: E-mail address: dmimar382@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: http://aggiehonor.tamu.edu/

Type of sources				
People				
Web pages (provide URL)	https://www.geeksforgeeks.org/binary-search-tree-data-structure/			
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work. On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name: Maria Dmitrievskaia Date: 03/29/20

1. A description of the assignment objective, how to compile and run your programs, and an explanation of your program structure (i.e. a description of the classes you use, the relationship between the classes, and the functions or classes in addition to those in the lecture notes).

Assignment objective:

The goal of this assignment was to create a Binary Search Tree class. Through creating different kinds of trees- linear, perfect, random – I can calculate the search cost for each node and the average search cost for each tree and illustrate the difference in search times between the three types of trees. With this calculated data, I illustrate the differences in average search times for linear, perfect, and random trees through graphs that can be seen in question 5.

How to compile and run my program/ program structure:

My program consists of 3 main files: BSTree.h , BSTree.cpp and BSTree_main.cpp. The file BSTree.h contains the Binary Search Tree class declaration, the node declaration and definition, as well as the declaration of the input and output operators.

The file BSTree.cpp contains all the class function definitions, as well as the definitions for the input and output operators. The class contains functions like insert, search, update search times, get average search time, in ordered traversal through the binary search tree. The class also defines a function to print the tree level by level. Helper function are created in order to implement the listed functions.

The file BSTree_main.cpp defines function to read from a file and to input data into a file and contains the main function in which tests are written to ensure the correct operation of the binary search tree class.

A Makefile was made in order to assist in compiling the program. All the user needs to do is type “make” in putty to compile the program.

2. A brief description of the data structure you create (i.e. a theoretical definition of the data structure and the actual data arrangement in the classes).

The theoretical definition of a Binary Search Tree according to GeeksforGeeks is as follows:

> A node-based binary tree data structure which has the following properties:
>
> - The left subtree of a node contains only nodes with keys lesser than the node’s key.
> - The right subtree of a node contains only nodes with keys greater than the node’s key.
> - The left and right subtree each must also be a binary search tree.

The Binary Search Tree class has public and private members. Public members include the default constructor, the move constructor/assignment operator, the copy constructor/assignment operator, the destructor. Several functions, such as insert, search, and in order and public, however, have private helper functions in order to prevent the user from accessing restricted information, such as the root of the tree of the tree’s size.

3. A description of how you implemented the calculation of (a) individual search cost and (b) average search cost and explain which tree operation (e.g. find, insert) was helpful. Analyze the time complexity of (a) calculating individual search cost and (b) summing up the search costs over all the nodes.

Individual node search cost implementation:

In order to calculate the individual node search cost, I created an integer to keep track of how many times the update search times helper function was called. The helper function took in that integer and a node as arguments and went through the tree in order, thus updating all the search times. In the insert function, after inserting a node, I update the size of the tree and call the update search times function.

Average search cost implementation:

In order to calculate the average search cost of for a tree, I first calculate the total search time for the given tree. I do so by recursively calling the "get_total_search_time" function to add up all the individual node search times. Then I divide the total search time by the size of the tree in order to get the average search cost.

Helpful functions:

The functions I found helpful were insert, insert helper, update search times, update search times helper, get average search time and get total search time.

Time complexity of Individual Node search cost:

Best Case:

In the best case, the Binary Search Tree is perfect, meaning the height of the tree is $\log_2(n+1)$, where n is the number of nodes. So, in order to find a value or to validate that a value is not in the tree, $\log_2(n)$ number of comparisons would have to be made. Thus, the time complexity of individual node search cost is the best case is $O(\log_2(n))$.

Average Case:

On average, the tree will be less balanced than a perfect tree, however, will be balanced (not a linear tree). So, on average, to find a value or to validate that a value is not in the tree $\log_2(n)$ number of comparisons would have to be made. Thus, the time complexity of individual node search cost is the average case is $O(\log_2(n))$.

Worst Case:

In the worst case, the Binary Search Tree is linear, meaning in order to find a value or to validate that a value is not in the tree, n number of comparisons would have to be made, where n is the total number of nodes in the tree. Thus, the time complexity of individual node search cost in the worst case is O(n).

Time complexity of summing up individual search costs:

Best Case:

The time complexity of individual node search cost is the best case is $O(\log_2(n))$. To sum up all the individual search costs, the tree would have to visit each of the n nodes. So, the time complexity of summing up all the individual nodes in the best case is $O(n\log_2(n))$.

Average Case:

The time complexity of individual node search cost is the average case is $O(\log_2(n))$. To sum up all the individual search costs, the tree would have to visit each of the n nodes. So, the time complexity of summing up all the individual nodes in the best case is $O(n\log_2(n))$.

Worst Case:

The time complexity of individual node search cost in the worst case is O(n). To sum up all the individual search costs, the tree would have to visit each of the n nodes. So, the time complexity of summing up all the individual nodes in the best case is $O(n^2)$.

4. Give an individual search cost in terms of n using big-O notation. Analyze and give the average search costs of a perfect binary tree and a linear binary tree using big-O notation, assuming that the following formulas are true (n denotes the total number of input data).

$$\sum_{d=0}^{\log_2(n+1)-1} 2^d(d+1) \simeq (n+1)\cdot\log_2(n+1) - n \quad \text{and} \quad \sum_{d=1}^{n} d \simeq n(n+1)/2$$

Assuming the equations are true, the first equation represents the total search cost of a perfect tree and the second equation represents the total search cost for a linear tree, where n is the number of nodes. In order to get the average search cost, the equations need to be divided by the number of nodes.

So, the equation that represents the average search cost for a perfect tree is:

$\frac{n+1}{n} log_2(n+1) - \frac{n}{n}$ =

$= (1 + \frac{1}{n}) log_2(n+1) - 1$

$\frac{1}{n} < 1$ for all n, so, the time complexity for the average search cost for a perfect binary tree is $O(\log_2 n)$.

Thus, the time complexity for the average search cost for a perfect tree is $O(\log_2 n)$.

So, the equation that represents the average search cost for a linear tree is:

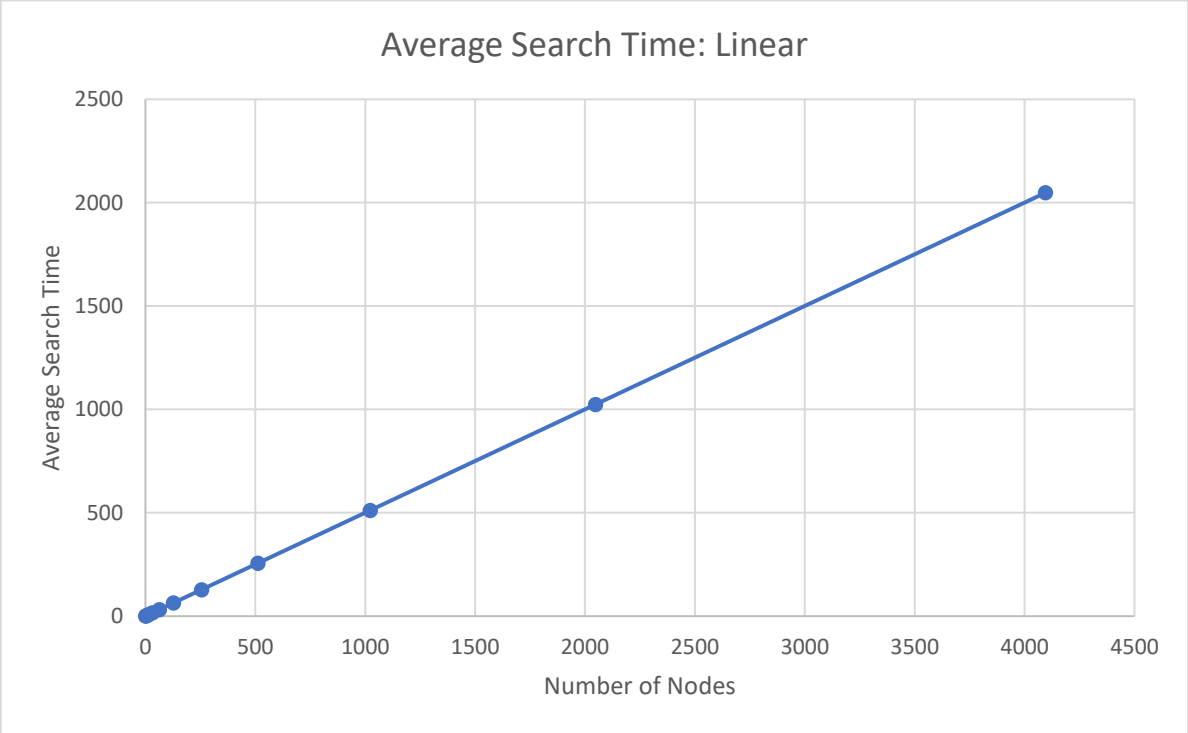
$\frac{n(n+1)}{2n}$ =

$= (\frac{(n+1)}{2})$

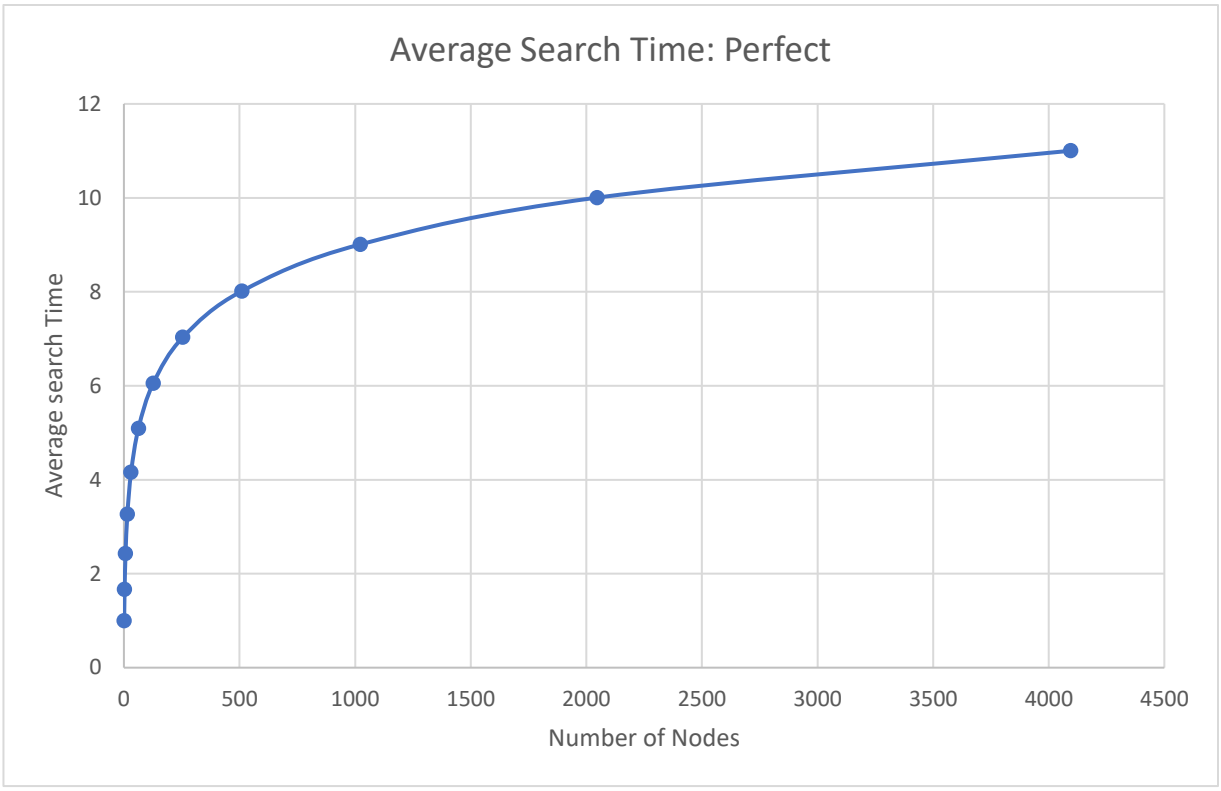
Thus, the time complexity for the average search cost for a linear tree is O(n).

5. Include a table and a plot of an average search costs you obtain.

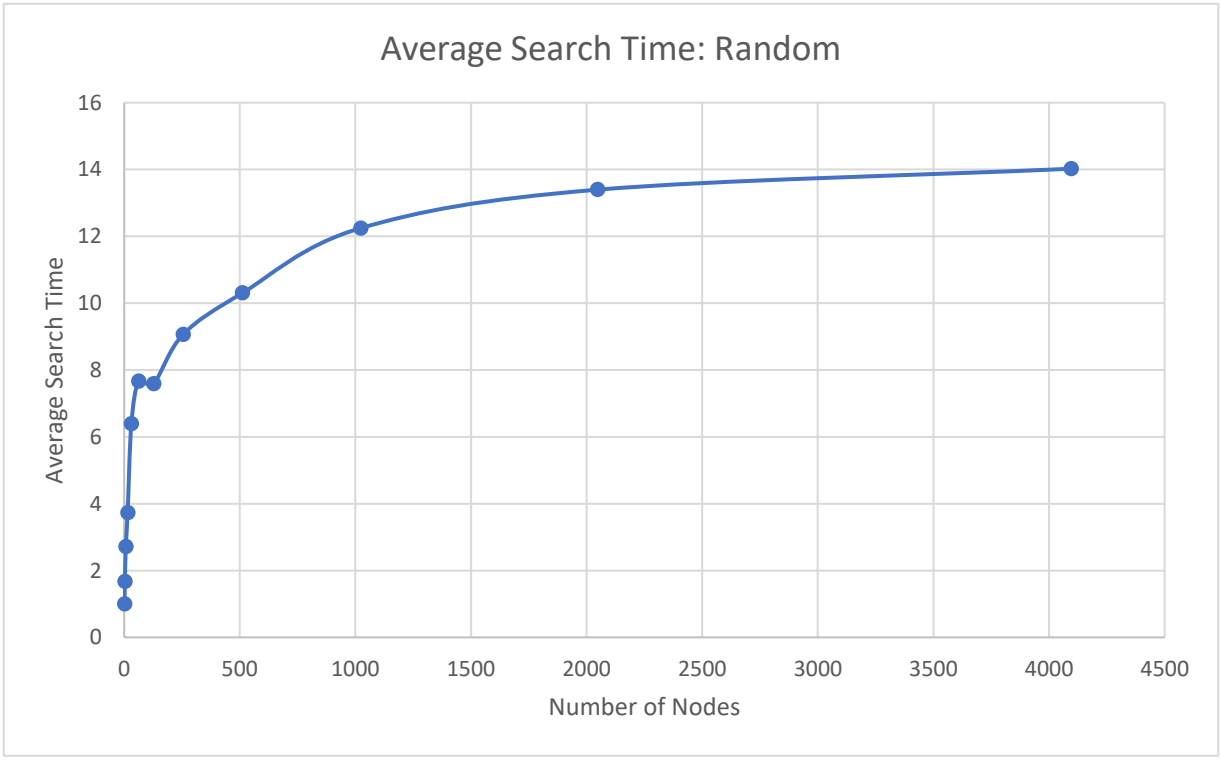
Number of nodes	Average search linear	Average search perfect	Average search random
1	1	1	1
3	2	1.6666	1.6666
7	4	2.4286	2.7143
15	8	3.2666	3.7333
31	16	4.1613	6.3871
63	32	5.0952	7.6666
127	64	6.0551	7.5905
255	128	7.0318	9.0666
511	256	8.0176	10.3033
1023	512	9.0098	12.2463
2047	1024	10.0054	13.3972
4095	2048	11.0029	14.0237



Graph 1: Average Search Time of a Linear Binary Search Tree



Graph 2: Average Search Time of a Perfect Binary Search Tree



Graph 3: Average Search Time of a Random Binary Search Tree

In your discussions of experimental results, compare the curves of search costs with your theoretical analysis results in item 3.

In Question 3 the following results were obtained for the time complexity of summing up individual search costs:

Best Case: $O(n\log_2(n))$

Average Case: $O(n\log_2(n))$

Worst Case: $O(n^2)$

In order to get the average search costs for each of the cases, we need to divide by the number of nodes. So, the following results are obtained:

Best Case (Perfect Tree): $O(\log_2(n))$

Average Case (Random Tree): $O(\log_2(n))$

Worst Case (Linear Tree): $O(n)$

For the prefect and random trees, the graphs are represented by $y = \log_2(n)$, thus confirming the time complexities for these cases. For the linear tree, the graph is represented by $y = n$, thus confirming the time complexity for the linear tree.