CSCE 221 Assignment 4 Cover Page

First Name: Adrian Last Name: Gamez-Rodriguez UIN:126009409

User Name: adriangamez E-mail address: adriangamez@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	TA in Lab		
Web Pages	Stackoverflow.com	Geeksforgeeks.org	
Printed Material	Textbook	Lecture Slides	

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: Adrian Gamez-Rodriguez Date:4/1/19

1. The purpose of this assignment was to create a Binary Search Tree using programming concepts such as pointer, OOP characteristics, and recursion.

The program structure is as follows: the struct Node is the fundamental building block of the Binary tree, and pointers serve as links for these nodes. The struct Nodes holds 4 parameters. It holds the data of type int, a pointer to a node for the left child, a pointer to a node for the right child, and a search cost value of type int. Using this struct we create a class/ data structure called BTree that has private data members, size and a pointer to the root node, and a data member that keeps track of the total search cost (sum of every node search cost). We implement the copy constructor, copy assignment, move constructor, move assignment, insert(x), update_search_cost(), search(x), inorder(), print_level_by_level(). The functions are self-explanatory. These functions allow for data manipulation of the binary tree. How to run the program:

Go to “BTreeCode” directory

- Once there simply type the command : ***make all***
- To run simply type the command : ***./main***

Note: all the input files will be there, and all output files will generate when program runs. Also, the program will output all 36 tree’s info, such as size and average search cost. Might be a little overwhelming.

2. The data structure we created was a binary search tree that is made up of nodes that contain a value and a search cost. A node is a simple struct that contains the data, two pointers that point to other nodes that are the right and left child, and another value that signifies the search cost associated with that node. In essence, the Binary Search Tree data structure is an encapsulation of the data struct Node, with the structure defined by a Binary tree's property. For instance, any value greater than the root node can be found in the right subtree, and any value less than the root node can be found in the left subtree. With this criterion we can construct the tree and perform operations on it.

3. I implemented the individual search cost both while inserting nodes and its own function update_search_cost(). The insert function kept track of the traversal operations and then I just assigned the search cost equal to those traversal operations. In the update_search_cost(), I created a helper function that recursive calls itself to traverse through the tree and updates the search cost for each individual node. I calculated the search cost by keeping track of the total search cost (sum all search cost of each node) in the insert function that was traversing through the list. Then I simply just took the total search cost of the tree and divided it by the size of tree.

The time complexity of the calculating the individual search cost is O(n) or O(nlogn). If the tree is unbalanced then we must process every node, in which

every node is a new level. If the tree is balanced, then it is O(logn) because we just have to go to the level in which the node is found, and the height of a balanced tree is log n.

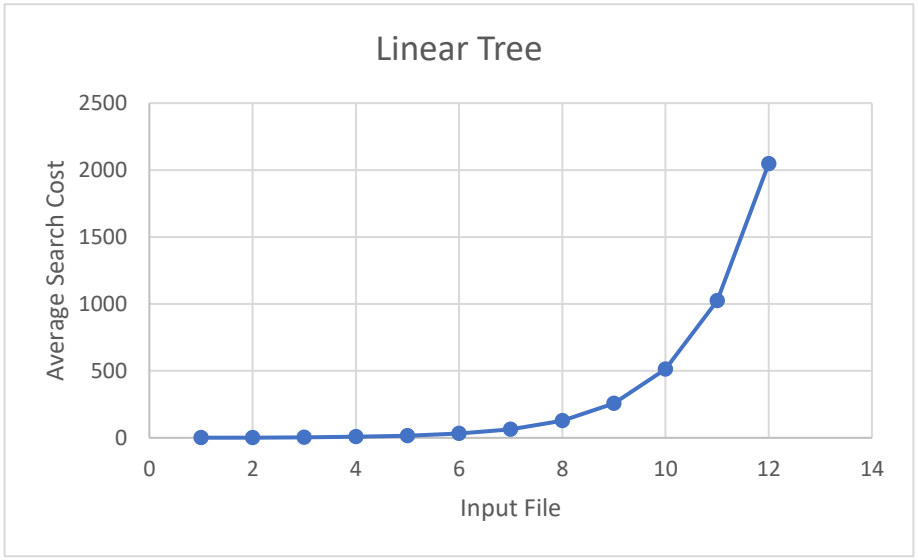
The time complexity for summing up all the search costs for all the nodes is O(n) since we must traverse to every node and add its search cost to the overall total cost of the tree.

4. For a perfect binary tree the tree is balanced, therefore the average search cost is minimized since the height is minimized because a perfect binary tree has all its level filled. So, the time complexity of the average search O(logn).

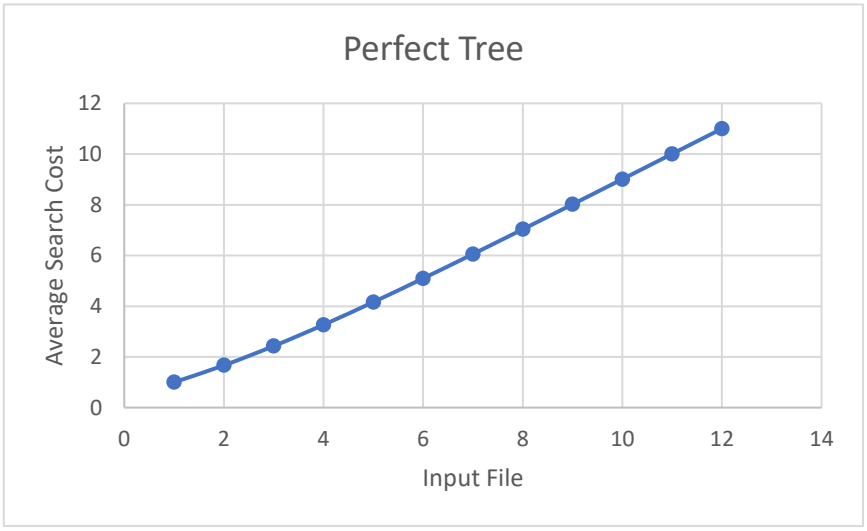
For a linear tree, this is the worst case for a binary tree since it is one sided, it would have a height of n for an input of n elements. Thus, the time complexity of the average search cost is O(n). It basically serves as an array.

5.

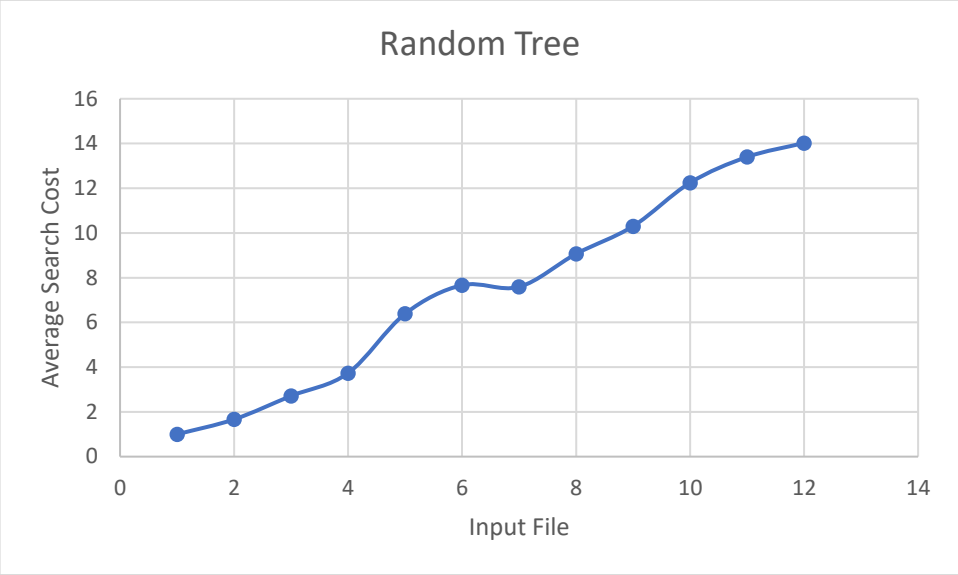
Linear	Average Search Cost
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
10	512
11	1024
12	2048



Perfect	Average Search Cost
1	1
2	1.667
3	2.428
4	3.266
5	4.161
6	5.095
7	6.055
8	7.031
9	8.017
10	9.009
11	10.005
12	11.002



Random	Average Search Cost
1	1
2	1.667
3	2.714
4	3.733
5	6.387
6	7.666
7	7.59
8	9.066
9	10.303
10	12.246
11	13.3972
12	14.023



This experiment follows the expected results from part 4. If we look at the linear trees, their average search times are far worse than the perfect or random trees. The perfect trees are the ideal input case, however, in practicality, a real-world input would look like the random tree input. So, one can say that Binary search trees are more effective for searching for an element in an input of n elements.