CSCE 221 Assignment 3 Cover Page

First Name Chris Last Name Comeaux UIN 622006681

User Name cmc236 E-mail address cmc236@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	Peer Teacher	Dr. Teresa Leyk
Web pages (provide URL)	http://www.cplusplus.com/	http://www.geeksforgeeks.org/618/
Web cont.	http://www.chegg.com/homework-help /definitions/binary-search-tree-3	http://stackoverflow.com/questions/19018294/c-to-check -if-user-input-is-a-number-not-a-character-or-a-symbol
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 Chris Comeaux Date 3/19/2016

Assignment Objective

The objective of this assignment is to create a Binary Search Tree from scratch. In doing this students learn how to create a Binary Search Tree, learn about recursive functions, and learn how to implement recursive functions to do various tasks. The students then used the Binary Search Tree class they created to create a simple program. First the program read in a list of integers from a file. Then that list was used to create a Binary Search Tree. As each node was being entered into the tree, 3 things happened. First, the node was being added to the tree, next the search cost for each node was being calculated, and third, the size of the tree was being updated. Once the tree is complete, the program outputs each node and its search cost. It then uses preorder, postorder, and inorder traversals to output the tree to either the screen or a file. Next the program calculates the average search cost of the tree. Finally, the program asks the user to enter a number to delete from the tree. It will update the tree, output the final tree, and then terminate.

Purpose of the Assignment

The purpose of the assignment was to grow student's knowledge of Binary Trees. The students had to use their knowledge and understanding of Binary Trees and recursive functions to write their own Binary Tree class and write an application for it.

Instructions to Compile and Run your Program

To compile the program, navigate to the PA4 directory using cd csce221/PA4. Then just use the 'make' command. This will compile the program and produce and executable file. To run the program, use the command ./Main. The program will ask you which you would like to to open. Enter a file name. Then the program will enter the data into the tree and begin working on it. Then the program will ask you to enter a number to remove from the tree. After the program has updated the tree, it will then terminate.

Program Organization and Description of Classes

I used two classes to create my Binary Search Tree. The first class, BTreeNode, was used to create the individual tree node. It has a member to hold key, a member to hold the search cost, and 3 members that are pointers called left_child, right_child, and parent, which pointed to the node's left child, right child, and parent respectfully. The class also has member functions to help gain access to private members and to check if the node is external or not. The next class, BSTree, was used to create the actual tree structure. It has 2 members. One member was the size of the tree and the other member was a pointer to a BTreeNode called root, which was the root of the tree. BSTree has multiple member functions. They help the user print, gain access to its members, get the height of the tree, insert a node into the tree, remove node from a tree, find the minimum node and remove it, find the average search cost, and traverse the tree. Also, BSTree is a friend class of BTreeNode, so it can access all of BTreeNode's private members. Both classes, BTreeNode and BSTree, have their declarations in BinarySearchTree.h. The implementation of BSTree is in BinarySeachTree.cpp, however, since all of BTreeNodes helper function are defined in the class, a separate .cpp file was not needed.

Data Structure Description

I created my Binary Search Tree using a Linked List type data structure. Each node is linked to its two children and its parent through pointers. The tree starts out as a single node called "root." When a node is inserted into a tree, it is only simply linked by the left_child or right_child pointers and the parent pointer. In other words, the data can be in any place in memory (i.e. does not have to be in a sequence of bites). As described in the previous section, the BTreeNode class represents the individual nodes of the tree, and the BSTree represents the nodes linked together to form a tree.

Calculations

- **Individual Search Cost**: The individual search cost of each node was calculated in the insert_node function. Insert_node is implemented by recursion. If the function does not find the right place to enter the node it calls itself with either its left or right child. Every time it is called the it keeps track of the previous nodes search cost. This allows the function to simple add one to the parents search cost when initializing the new node and adding it to the tree.

- **Average Search Cost:** The average search cost has two parts, the size and the total search cost of the tree. To calculate the size, I added one to the tree's member 'size' every time the a node was inserted and subtracted one from 'size' every time a node was removed. The total search cost is calculated in the postOrder function. Every time it outputs a node, it also accesses the nodes search cost and adds it to the tree's member 'TotalSC', which holds the total search cost of the tree. Once these two values have been calculated, the average search cost is calculated in the function aveSC. It simple returns the 'TotalSC' divided by 'size'.

- **Updated Search Cost:** This does not apply to my class because in my implementation of my binary tree, when a node is removed, the search costs of the nodes are not affected. With a perfect or a random tree, the remove function simply switches the keys of the deleted node with the left child's key (if it is an external node) or switches the key with the least node in the right sub-tree the deletes that node. The search costs of the nodes are not affected. For a linear tree, the keys of the nodes below the deleted nodes are simple shifted up one and the last node is deleted.

Time Complexity

- **Individual Search Cost**: The time complexity of calculating the individual search cost would be O(log(n)) where n is the number of nodes in the tree at that point. This is because, on average, about half of the nodes have to be visited for a node to be inserted into the tree and since the search cost is calculated during insertion, it will have the same Big-O notation as the insertion function.

- **Updated Search Cost:** The time complexity of updating the search cost would be same as deleting a node from the tree. The Big-O notation would be O(log(n)) where n is the number of nodes in the tree at that point. This is because, on average, about half of the nodes have to be visited for a node to be deleted from the tree.

- **Summing Up Search Cost:** The time complexity of summing up the search cost of the tree is O(n) where n is the number of nodes in the tree at that point. This is because every node must be visited inorder to sum up all the search costs.

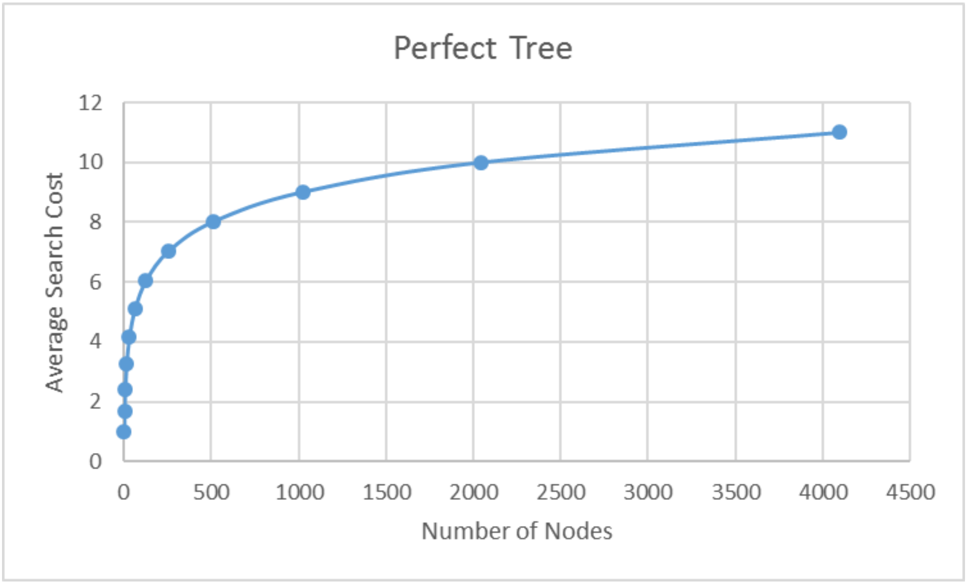
Search Costs

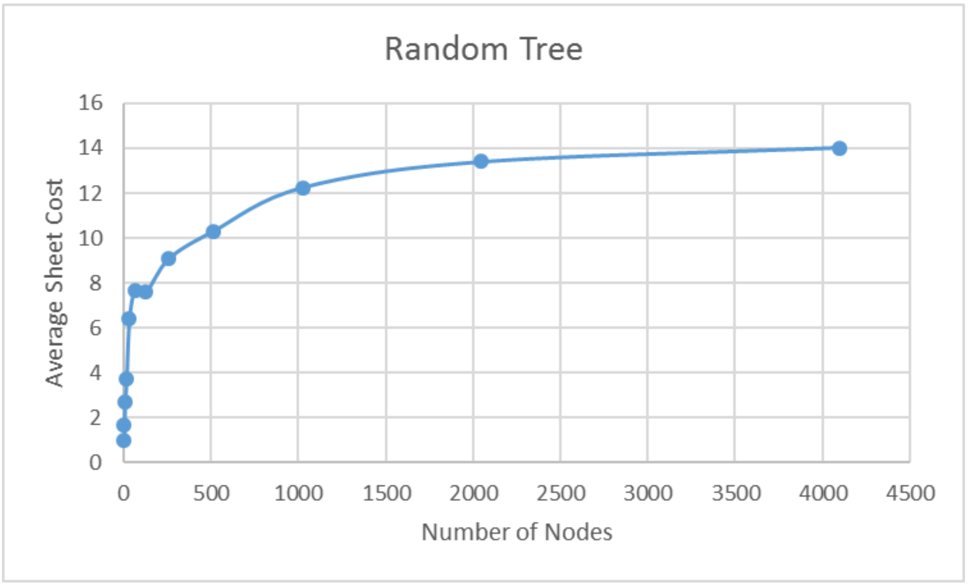
If we let n be the individual search cost of each node of a binary tree, then the best and average case for a binary tree would be O(log(n)). These cases are would result from a perfect and random binary tree. The worst case for a binary tree would be O(n), which comes with a linear tree. For the perfect tree and random tree (best and average cases), the height is log(n), on each level there are 2^k nodes, and the total search time is (log1+1) + 2(log2+1) + 2^2(log3+1) + 2^3(log4) +...+ 2^(log(n+1)) - 1(log(n)+1) = (n+1)log(n+1) - n. This means that the average time is O(log(n)). For the linear tree (worst case), the height is n-1 and the total search cost is n(n+1)/2. Dividing these you would get (n+2)/2 which is O(n).

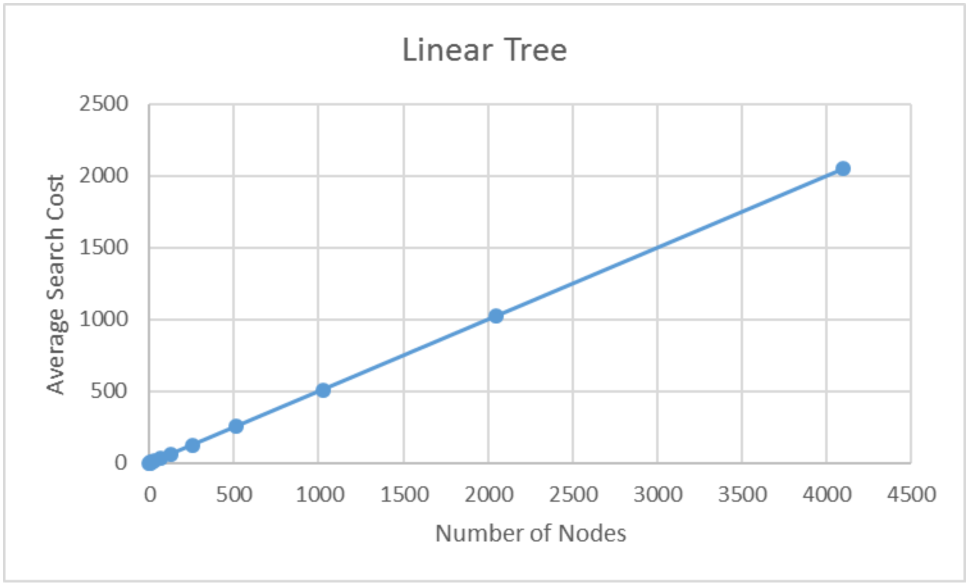
Graphs and Tables

After analyzing the tables and graphs we see that the theoretical results in part 4 match the results shown here. Analyzing both the Perfect and Random trees (Best and Average case respectfully) we see that they are both O(log(n)) which is what we found in part 4. Analyzing the Linear tree (worst case) we see that the average case for the Linear tree is 2n which is O(n) which is also what we found in part 4.

Nodes	Perfect	Random	Linear
1	1	1	1
3	1.66667	1.66667	2
7	2.42857	2.71429	4
15	3.26667	3.73333	8
31	4.16129	6.3871	16
63	5.09524	7.66667	32
127	6.05512	7.59055	64
255	7.03137	9.06667	128
511	8.01761	10.3033	256
1023	9.00978	12.2463	512
2047	10.0054	13.3972	1024
4095	11.0029	14.0237	2048






Linear Tree
Average Search Cost
Number of Nodes
0
500
1000
1500
2000
2500
3000
3500
4000
4500