Program Assignment 3

CSCE 221

For program Assignment three we were to build a binary search tree through the provided code that was given. The structure of our program contained three files, the main.cpp, a header file, and second cpp file that defines the member functions from the header file. Within the main file there is a read function and an output function that allows the taking in of data files and outputting of them. Also, within the main is code for collecting data average search cost for the given binary tress. The header file contains two struct. The first is Node, this struct is built to define the object within a node, such as value, search cost and what this node points to. The second struct is BSTree, this struct contains the root and the process to build a binary search tree and functions to access data about it. Struct Node is a subclass of BSTree and BSTree contains copy and move constructors and assignments as well. The header file is included in both the main.cpp and the second cpp file the defines all the header files member functions. To compile the code, you must develop a make file to run the program on a Unix terminal.

A binary search tree a tree that stores the value of different entries and sorts them in an order. Because of how a binary search tree is made by going left or right at every node to find or insert a node it allows for a best case big-O of O(log n) while its worst case being O(n). In my program you can see the runs times played out by the search cost value that each node gets assign. A node is given a search cost based on 1+ depth of the node. So, for example the root has a depth of zero so 1+0 = 1 giving the root a search cost of 1. While a node is inserted the depth of that node is tracked and then given a search cost based on that. Also, there is function update\_search\_times which is a recursive function that’s only job it to make sure all search costs are correct at each node. To get the average search time code was given and implemented for us. First a function get\_average\_search\_time was called which then passes the root of the tree to a different function get\_total\_search\_total. This function was recursive and returned the total of all search times by calling the root->left, root, and root->right as the parameters when calling itself. The total was returned to get\_average\_search\_time were is was then divided by the size and returned.The time complexity for an individual search based on the search function I implemented has T(n)=logn +1 and its big-O is O (log n). While to sum up the search cost the T(n) = 1 + T(n/4) + c.

Part of the lab focus was to calculate average search cost and see how different shaped trees can have massive effect on time complexity. We can find the perfect and linear binary tress average cost from these two formulas. The left being perfect and right being linear.

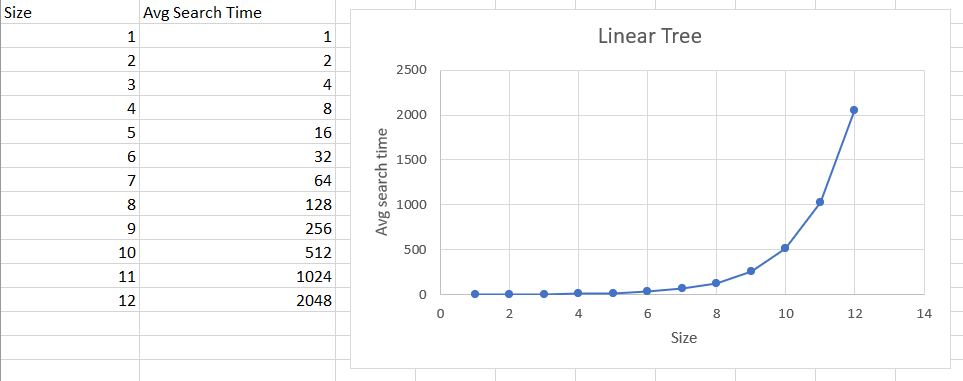


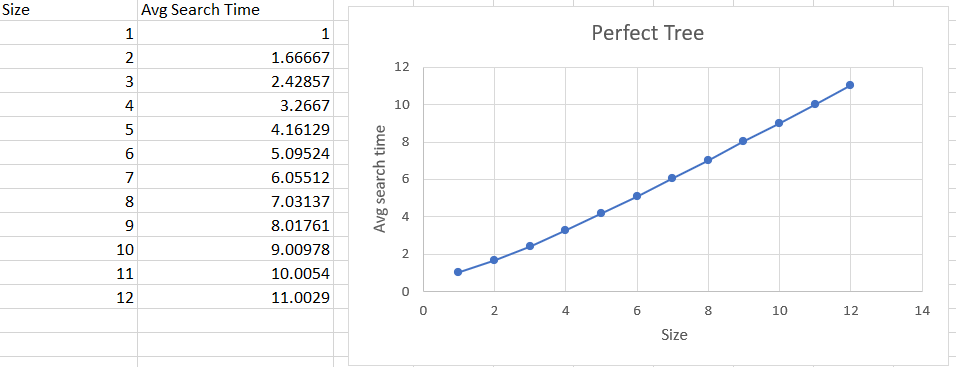
Where n is the size of the input of the data.

All of this leads to a conclusion that the big-O search cost for perfect tree is O (log n) while a linear big-O is O(n).

Data from test files:

Linear:



Perfect:

Random:

