Programming Assignment 4

Josh Zschieshce

CSCE 221-505

UIN:523000614

March,22,2016

Part1: Assignment Objective Description

We created two classes, one that made a binary search tree and another class that was the nodes for the binary search tree. We had to complete tasks with these data structures, namely print the tree and all of its nodes to the screen, the program also prints out the nodes in inorder, preorder, and postorder. To run my program, simply have all of the test files in the same directory as my files, and type make clean then type make and run with ./main_PA4 and follow all of the instructions that come up. The class binary search tree makes the tree and the class binary nodes holds the actual data value that makes up the tree. The class binary search tree has the functions, insert, remove, remove min and height and size.

1. Insert: Inserts a new node into the binary search tree.
2. Remove: Removes a node specified by the user.
3. Remove Min: Removes the minimum number not designated by the user
4. Height: returns height of the tree (Max search class)
5. Size: returns the size of the tree (Number of nodes)

Part2: Brief Description of the Data Structure

Theoretical: Binary search trees is a container that keeps its nodes in a sorted order. This makes searching, adding, and removing much easier and quicker as all of the operations operate off of the binary search algorithm. They traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, based on the comparison, to continue searching in the left or right subtrees. On average, this means that each comparison allows the operations to skip about half of the tree.

Actual: The program is organized into 5 files, BinarySearchTree.cpp, BinarySearchTree.h, BinaryNode.h, BinaryNode.cpp, main_PA4.cpp. The header files have only declarations, and no definitions (except for the inOrder, PreOrder, and PostOrder) and the .cpp files have the definitions associated with all of the declarations in the header files. The main_PA4.cpp is the main, and it uses the classes and handles where to remove, insert, and read in from a file. The actual binary search tree has all of the properties of a theoretical binary search tree, without being able to store its search cost.

Part3: Description of Implementation of Various Functions.

Individual Search Cost: To calculate the search cost, I make a global variable called s and this can be edited from anywhere in the class, and every time that the insert function is called, I add 1 to s and that gets returned as the search cost. After the insert is called and the node is done inserting, search cost is reset to one for the next node to be inserted.

Average Search Cost: The average search cost is calculated by simply having another global called total_search_cost and adding s to that every time that the node is inserted, and this is divided by the number of nodes, which is calculated while reading in from the file. Then, the total cost is divided by the number of nodes and returned to the screen.

Updtaed Search Cost: This function is not implemented.

The implementation of search cost is implemented using the insert, as every time that the tree inserts a node, if it goes further down either the left or right subtree, then the search cost is incremented by 1 and when the insert is done, it returns search cost and then resets search cost.

Time Complexity

Individual Search Cost: O(n)

Updating Search Cost: Does Not Exist

Summing Search Cost: O(nlog(n))

Part4:

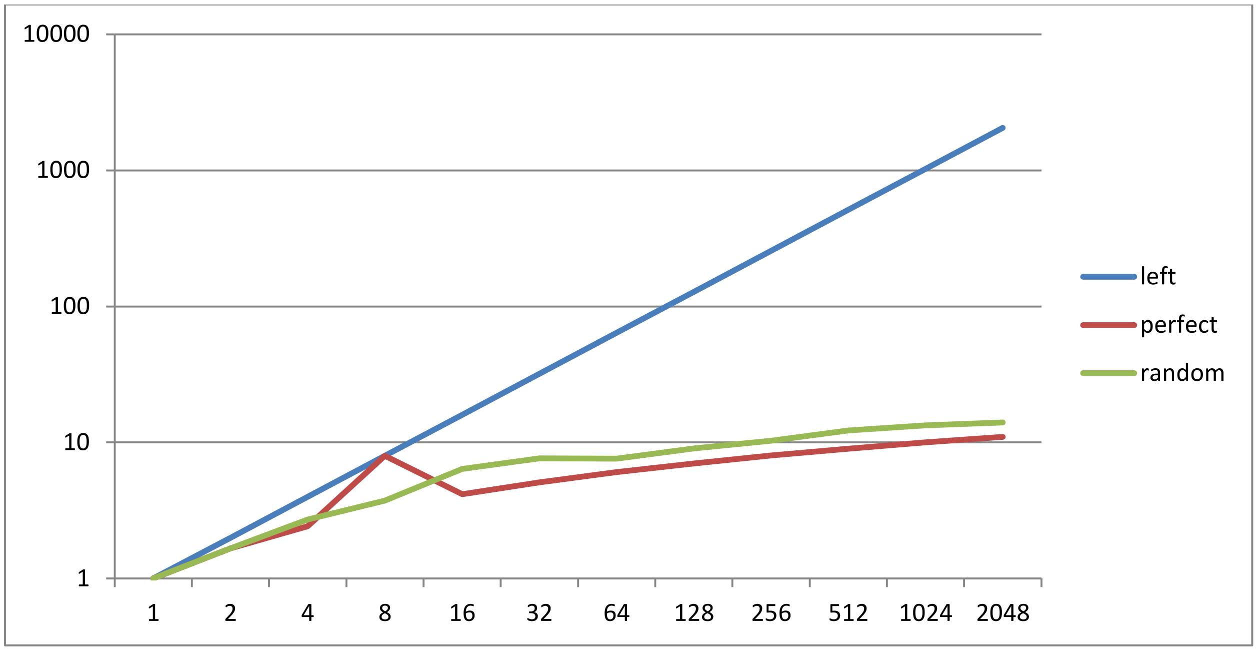
The individual search cost in terms of n for the linear file is O(n), as the linear tree makes a long tree with every node after the other all on the left-most branches. This makes a tree that is n search cost deep, meaning that as the tree grows, the search cost increases by n.

The formula for the upper bound (total search cost) of a linear search tree is n*(n+1)/2, this gets divided by n (number of nodes), to become (n+1)/2, which will be the average search cost for the linear tree. The average complexity for a linear tree is also O(n).

The formula for the upper bound (total search cost) of a perfect binary tree is (n+1)*log(n+1) – n, this gets divided by n (number of nodes), to become ((n+1)*log(n+1)-n)/n, with time complexity of O(nlog(n))

Part5:

FILE	ASC	FILE	ASC	FILE	ASC
1l	1	1p	1	1r	1
2l	2	2p	1.66667	2r	1.66667
3l	4	3p	2.42857	3r	2.71429
4l	8	4p	8	4r	3.7333
5l	16	5p	4.16129	5r	6.3871
6l	32	6p	5.09524	6r	7.6667
7l	64	7p	6.05512	7r	7.59055
8l	128	8p	7.03137	8r	9.06667
9l	256	9p	8.01761	9r	10.3033
10l	512	10p	9.00978	10r	12.2463
11l	1024	11p	10.0054	11r	13.3972
12l	2048	12p	11.0029	12r	14.0237



Note: This program has no error handling, so you must do exactly as the instructions say!