CSCE 221 Assignment 4 Cover Page

First Name Anders Last Name Wallace UIN 925000221

User Name awallace2 E-mail address awallace2@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)				
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 Anders Wallace Date 11/6/17

1 Description

1.1 Assignment Objective

The purpose of this assignment was to take data as an input from a file and create a binary search tree from it. We could take user input to choose which file to take data from, as well as which nodes to delete from the tree. This assignment had 2 classes: A binary node class and a Binary Search Tree class that use the Binary Nodes to populate it and various functions to edit the tree. Some of these functions included resetSearchCost which would go through the tree and update the search cost of each node after one had been removed. In order to run the program, simply type the command:

```
g++ -std=c++14 *.cpp
```

and then type:

```
./a.out
```

2 Binary Search Tree Description

2.1 Data Structures Created and Used

This assignment called for us to create a binary search tree, where a tree consists of a root node, followed by children where the right child has a key greater than the key of the parent, and the left child has a key less than the key of the parent. While the data structure itself was a tree, a few others were used to complete this assignment, such as a queue for the level-by-level output of the tree.

3 Calculations

3.1 Implementation

a) Individual Search Cost: For this function, I update the search cost of each node every time the insertNode function is called. The function itself is recursive, and everytime the recursive call is made, I increment search cost again in order to show there is another layer of searching to be done on the node. The time complexity for this function is $O(n)$ in the worst case, and $O(logn)$ in the best case. b) Average Search Cost: To calculate the total cost, I divide the totalSearchCost of the tree by the number of nodes in the tree. These values are saved before the calculation is made. The time complexity of this function is $O(n)$

c) Updated Search Cost: In this function we call resetSearchCost after a node has been removed, where it traverses the tree and updates the nodes after one has been removed. The time complexity is $O(n)$

4 Tree Search Costs

4.1 Individual Search Costs

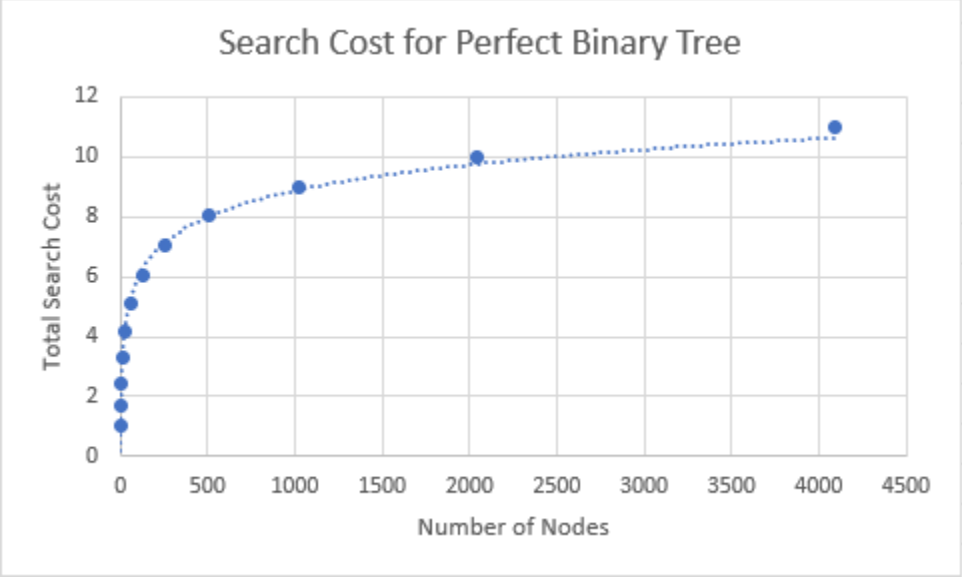
The search cost of a perfect binary tree can be found given there are 2^n nodes at each level of the tree, and after traversing all levels we find that this is equal to $2^{log(n+1)-1} \cdot log(n+1)$ which is equal to $(n+1)(log(n+1))$ which is $O(logn)$. For a linear tree, we know the total cost of searching for a key is $\frac{n(n+1)}{2}$ and after accounting for n subproblems we find that it simplifies to $(n+1)/2$ which is $O(n)$

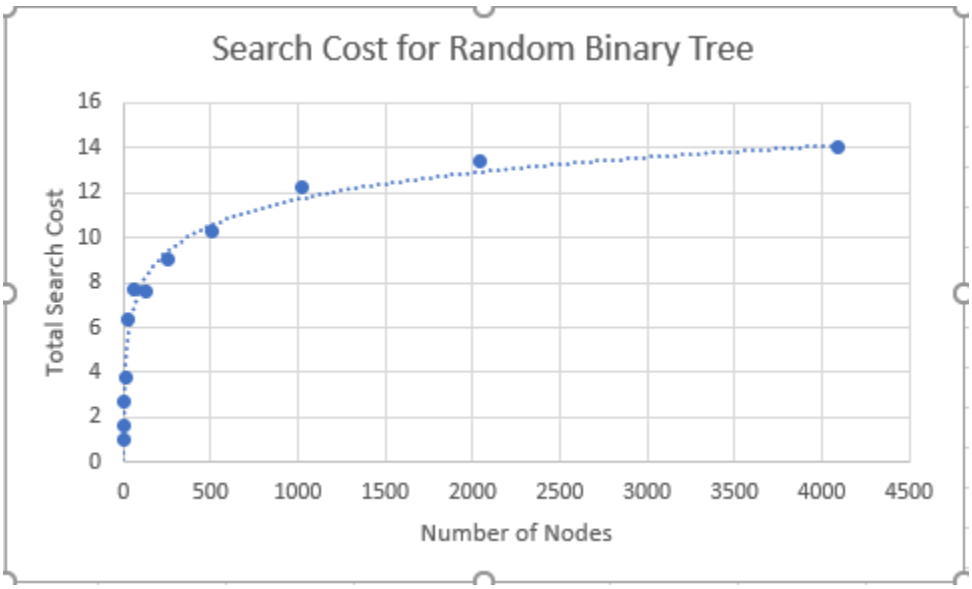
5 Graphs and Data

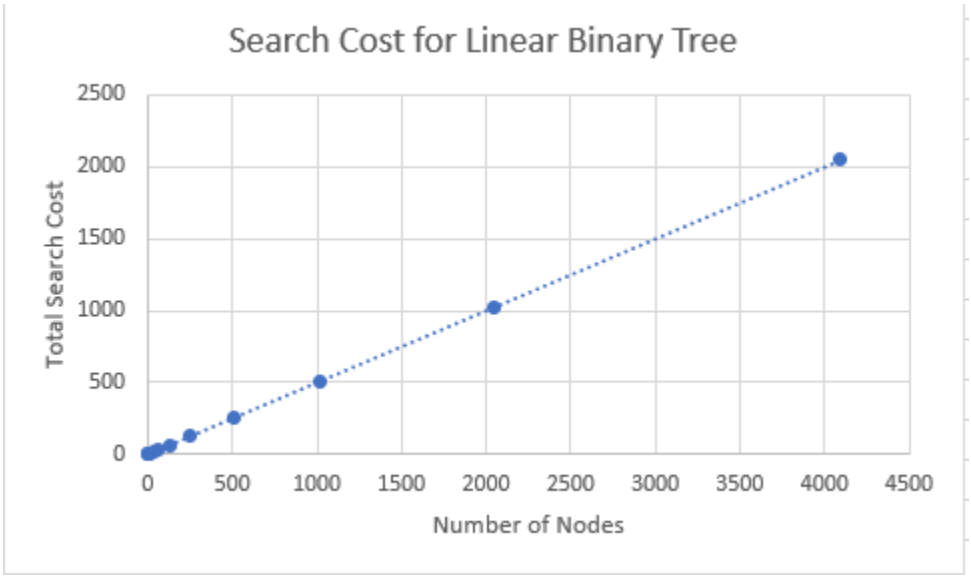
5.1 Data on Linear and Perfect Binary Search Trees

Below are tables and graphs on various data values for random, perfect, and linear trees.

# of Nodes	Linear	Perfect	Random
1	1	1	1
3	2	1.67	1.67
7	4	2.43	2.71
15	8	3.27	3.73
31	16	4.16	6.39
63	32	5.1	7.67
127	64	6.06	7.59
255	128	7.03	9.07
511	256	8.02	10.3
1023	512	9.01	12.25
2047	1024	10.01	13.4
4095	2048	11	14.02







As we can see by the graphs above, the perfect binary tree has logarithmic graph, which alligns with the theoretical search cost of $O(logn)$ and the random binary tree also has a logaritmic graph. The linear tree has a corresponding linear graph, which matches its search cost of $O(n)$.