CSCE 221 Cover Page

Programming Assignment#4

Due by April 1st midnight to eCampus

First name: Aldo Last Name: Leon Marquez UIN: 326004699

User Name: allema98 E-mail address: allema98@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: Aggie Honor System Office

Types of Sources			
People	TA sec 501		
Web Pages	https://piazza.com	Stack Overflow	Geeks for Geeks
Printed material	-		
Other Sources	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."

Name: Aldo Leon Marquez Date: 01/04/2019

Program Description:

The program coded for this assignment is a Binary tree implementation (not AVL). The supported operations for this class are: insert, search, print level by level, print inorder, Rule of 5, and a function that updates The search cost of each individual node. In order to implement the binary tree we need a class for the tree and a struct for the nodes that make the tree. Each node will contain a pointer to its left and its right node (<,>) and the actual value of the node. The Binary tree class will then store the root node, the characteristic values of the given tree and the supported functions.

Data Structure Description:

The data structure being program is a Binary tree, that Inserts values to the right or to the left depending if its less or greater than the current value, after several insertions the tree will visually reassemble a "tree". The implementation is done by using a node for each value and the Nodes are stored in the Binary tree Class.

Function Description:

(a) individual search cost: In my implementation two functions can update the individual search cost; update_search_cost and Print_Level_By_Level. The iteration through the tree for both is the same. The way the works is by using a vector to temporary string Nodes inside a Vector. The vector first pushes back a node representing the root then inside a while loop a new temporary node will be assigned with the current node being used, the node left and right pointers will be checked if they are null pointers, and in the case at least one of them is not the loop will continue to run. After passing these two new nodes a check if the level is empty is done, if its empty the value is dated to a variable "level" that will increase by one every time the if statement is true. A final if is done to check if a bool "done" is true(only true if all of the nodes of the level point to NULL) if true the program will exit, else the bool will be reseted and the while will start again

(b) average search cost and explain which tree operation (e.g. find, insert) was helpful.

For the average Search cost I used the recursive function of inorder. I updated a variable called "Total Cost" with the individual search cost of the node before the cout of the node. Then the Value of TotalCost is divided by the size of the tree to give the average cost

Analyze the time complexity of

(a) calculating individual search cost. After analyzing the obtained data the time complexity of a perfect and random tree is O(logn) and for the Average for a linear tree its O(n), with the normal average around n(n+1)/2, this can be explained by the fact that sum of the search values is $\sum_{n=}^{n} i$ and since the average is just dividing by n we get the n(n+1)/2 or O(n).

(b) **summing up** the search costs over all the nodes. The time complexity of a traversal over the full tree is just O(n), so the Time complexity of the sum is the same O(n)

4. Give individual search cost in terms of *n* using big-O notation. Analyze and give the average search costs of a perfect binary tree and a linear binary tree using big-O notation, assuming that the following formulas are true (*n* denotes the total number of nodes)

$$\sum_{d=0}^{\log_2(n+1)-1} 2^d(d+1) \simeq (n+1)*\log_2(n+1) - n \quad and \quad \sum_{d=1}^{n} d \simeq \frac{n(n+1)}{2}$$

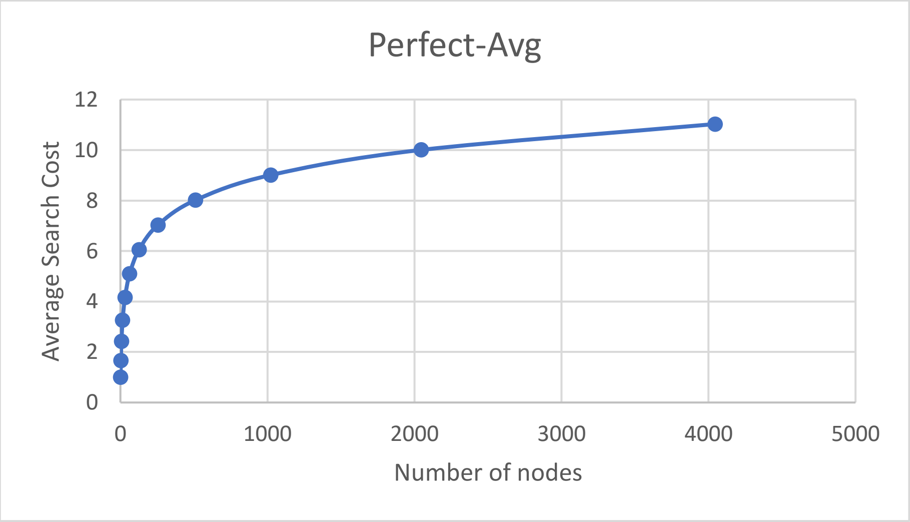
For a perfect binary tree we know that the Height of the tree is bounded by log(n), we also know that the total cost is the cost for each level multiplied by the number of nodes of each tree. That's is exactly what the left side of the first equality says, assuming the right as true we that the average cost for a perfect tree is either side of the divided by the total number of nodes: $\frac{(n+1)*\log_2(n+1)-n}{n}$ that gives us a Complexity of O(n) which is exactly what we expect and get.

For the Linear tree is even easier, knowing that the individual search cost of each inserted node will increase linearly he equality provided on the left is just the formula for a linear sum, we also know that that sum is exactly equal to the formula on the right. Then to calculate the average we just divide by n and that gives us $\frac{n(n+1)}{2n}$ with Time complexity equal to O(n).

5. Include a table and a plot of average search cost you obtain. In your discussions of the experimental results, compare the curves of search cost with the theoretical analysis results presented in item 4.

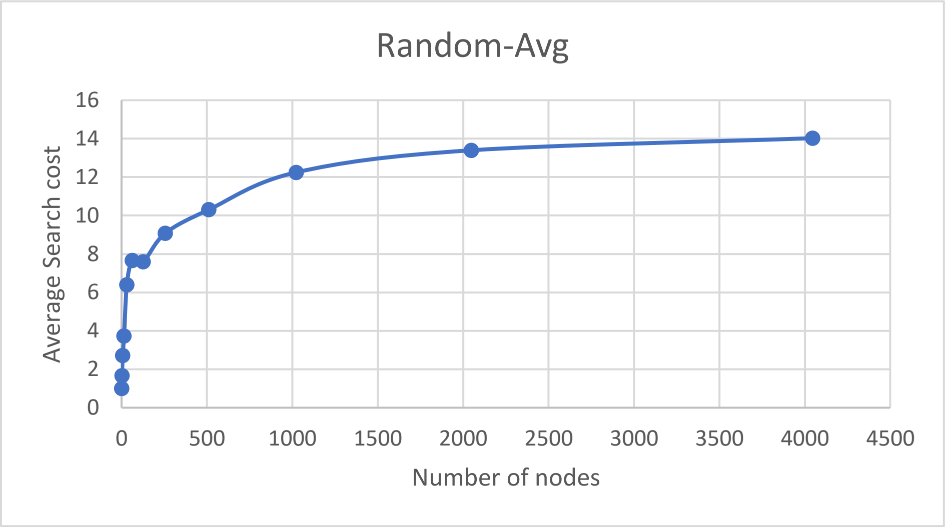
Perfect Trees

Perfect	Avg
1	1
3	1.667
7	2.42857
15	3.2667
31	4.16129
63	5.09524
127	6.05512
255	7.03137
511	8.01761
1023	9.00978
2047	10.0054
4046	11.029



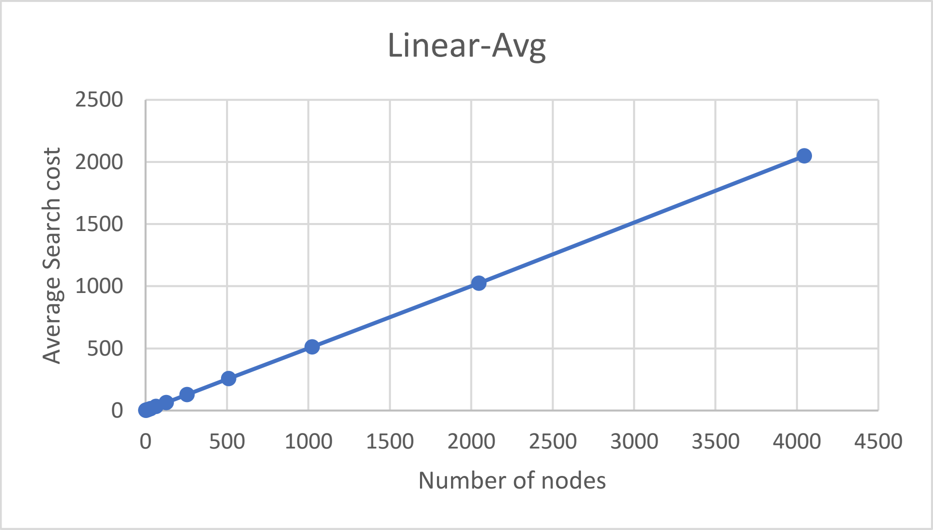
Random Trees:

Random	Avg
1	1
3	1.6667
7	2.71429
15	3.7333
31	6.3871
63	7.6667
127	7.59055
255	9.0667
511	10.3033
1023	12.2463
2047	13.3937
4046	14.0237

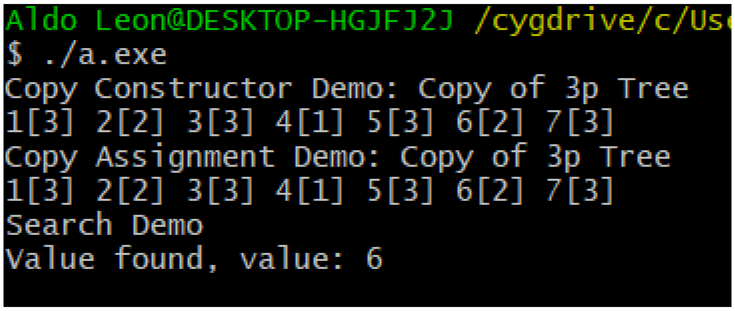


Linear Trees:

Linear	Avg
1	1
3	2
7	4
15	8
31	16
63	32
127	64
255	128
511	256
1023	512
2047	1024
4046	2048



Data and code Proof:



```
Aldo Leon@DESKTOP-HGJFJ2J /cygdrive/c/Us
$ ./a.exe
Copy Constructor Demo: Copy of 3p Tree
1[3] 2[2] 3[3] 4[1] 5[3] 6[2] 7[3]
Copy Assignment Demo: Copy of 3p Tree
1[3] 2[2] 3[3] 4[1] 5[3] 6[2] 7[3]
Search Demo
Value found, value: 6
```

Printed Trees: The printed trees proof is located in the “Text file results” folder, and additionally when the code is run the program generates all the text files with the name “results” plus a random character. This was done so that all the files could be created at once