CSCE 221 Cover Page

Programming Assignment #4

Due **November 5** by midnight to CSNet

First Name: Quinn Last Name: Nguyen UIN: 524002419

User Name:quinnminh_nguyen

E-mail address: quinnminh_nguyen@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

Type of sources			
People	Peer TA		
Web pages (provide URL)	http://www.cplusplus.com/		
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: Quinn Nguyen Date: 11/05/2017

1. Assignment objective: Reading a list of integers from a file, put them in the binary tree form and calculate search cost for each node and average search cost for each tree. We also need to output the tree using in-order transversal or BFS algorithm to get level by level tree.

   The user can change the file input name in “inFS.open()” for different inputs.

   fileName is for the output file name of print tree in-order. If total nodes are more than 2^4, print in a file called in fileName. If not, print on the screen.

   fileName2 is for the output file name of print tree level to level. If total nodes are less than 2^4, print a level tree in the file. If not, do nothing because it is not required to do so.

   Relationship between classes:

   There are 2 classes: binaryNode and BinarySearchTree. BinarySearchTree contains a structure of binary nodes in a binary tree form.

   In addition to lecture nodes:

   BinaryNode:

   +contained searchCost as a private member to keep track search cost for each node
   +have setter and getters for each of the private variable
   +help the tree to transverse elements.

   BinarySearchTree:

   +have printQ queue to make it easier to print elements in-order to a file
   +updateCost updates cost whenever delete or insert.
   +sumCost sums all of node cost
   +queueInOrder has printQ ready to print elements in-order to a file
   +printToFile prints elements in-order to a file.
   +outputTreelevelByLevel prints tree level to level to a file
2. Theorectical definition of Binary Search Tree:

   +A single node is a full binary tree.

   +If T1 and T2 are full binary trees, there is a full binary tree T consisting of a root r together with edges connecting the root r to each of the roots of left subtree T1 and right subtree T2.

   Actual implementation:

   +The node has an extra private member SearchCost to keep track of the cost of each node.

   +The tree has an extra private member printQ to print element in order in a file.

   +To print level to level, the tree has to be balanced. In order to balance the tree, I created fake nodes (has key of -1, and left and right point to null). This way, I can print out a tree level to level with “x” to fill missing nodes.

3. **Individual Search Cost**: a search cost is “assigned” to a node after the node is inserted. This is achieved by increasing search cost by 1 whenever the node is compared to an element until it finds its place in the tree. Because the implementation of search Cost is included in insert function, individual search cost time complexity is the same as insert function. It is *O(logn).*

   Average Search Cost: use sum node function divide by the size function to find the average cost. Sum cost is achieved by going from the root and going to element on the left and on the right. Because the average search cost depends mostly on **sum cost** implementation, it is *O(N).*

   UpdateCost: is implemented by post-transverse the involved part of the tree and decreasing the searchCost of the node following after by 1. This implementation of updateCost is associated with remove. The time complexity for updateCost is *O(N)* because it iterates from deleted node to every children of every generation.

4.

Perfect Binary Tree:

+Individual search cost: O(logn)

+Total operations of inserting n elements

$$\sum_{d=0}^{\log_2(n+1)-1} 2^d(d+1) \simeq (n+1)\cdot\log_2(n+1) - n$$

+Average search cost = $\frac{(n+1)log_2(n+1)-n}{n} = \frac{nlog_2(n+1)}{n} + \frac{log_2(n+1)}{n} - 1$

=>Average search cost: O($log_2(n)$)

Linear Binary Tree:

+ Individual search cost: O(n)

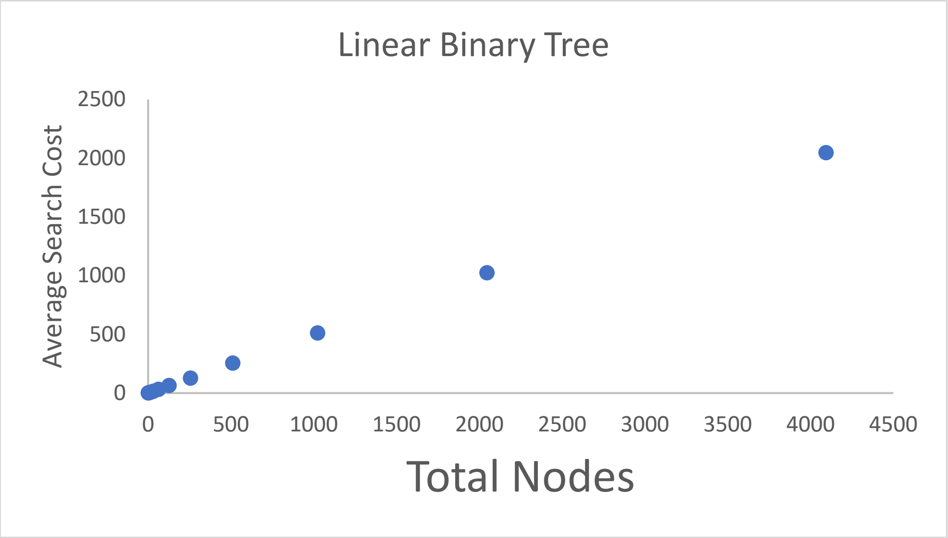
+Total operations of inserting n elements

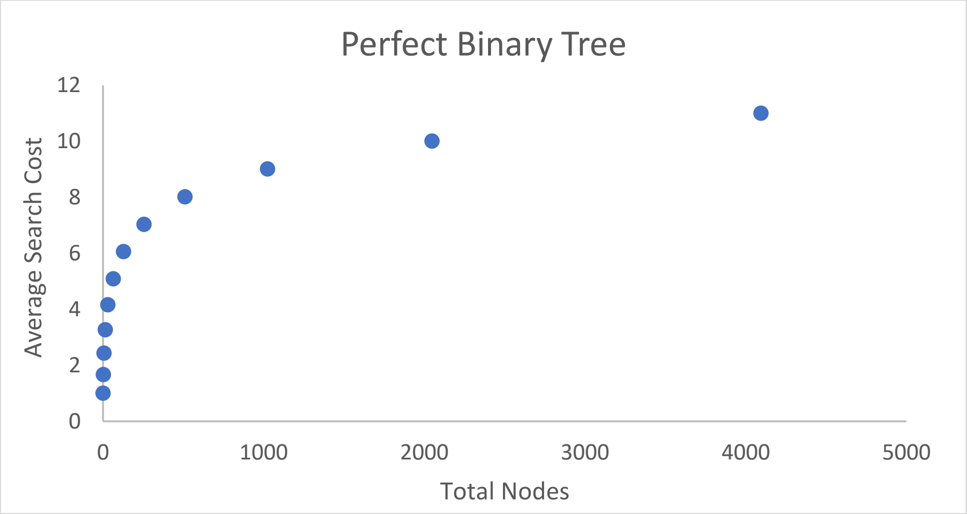
$$\sum_{d=1}^{n} d \simeq n(n+1)/2$$

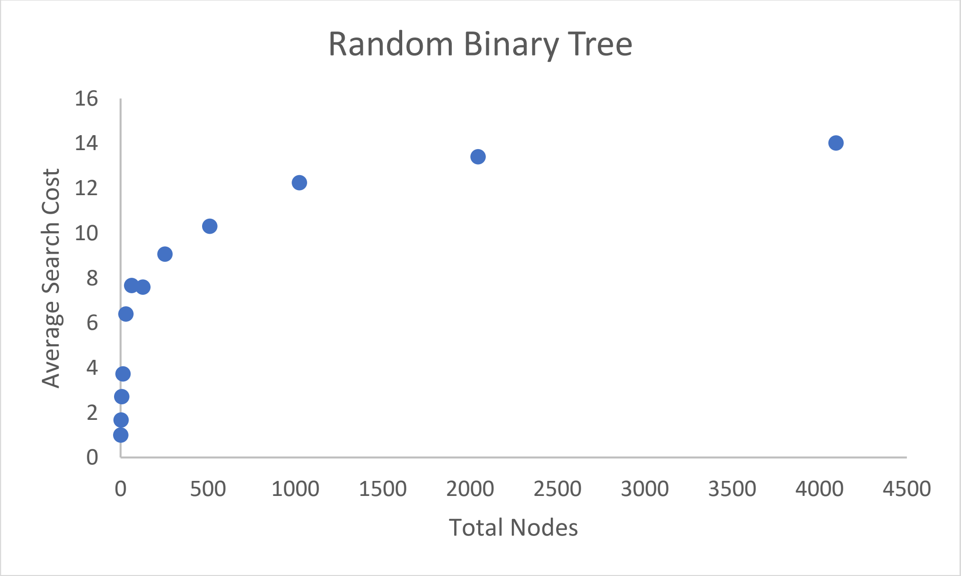
+Average search cost = $\frac{n(n+1)/2}{n} = \frac{(n+1)}{2}$

=>Average search cost: O(n)

5.


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


Perfect Binary Tree
Average Search Cost
12
10
8
6
4
2
0
0
1000
2000
3000
4000
5000
Total Nodes


Random Binary Tree
Average Search Cost
16
14
12
10
8
6
4
2
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Total Nodes

case	total nodes	Average Search Cost
1p	1	1
2p	3	1.67
3p	7	2.43
4p	15	3.27
5p	31	4.16
6p	63	5.09
7p	127	6.06
8p	255	7.03
9p	511	8.02
10p	1023	9.01
11p	2047	10
12p	4095	11
1r	1	1
2r	3	1.67
3r	7	2.71
4r	15	3.73
5r	31	6.39
6r	63	7.67
7r	127	7.59
8r	255	9.07
9r	511	10.3
10r	1023	12.25
11r	2047	13.4
12r	4095	14.02
1l	1	1
2l	3	2
3l	7	4
4l	15	8
5l	31	16
6l	63	32
7l	127	64
8l	255	128
9l	511	256
10l	1023	512
11l	2047	1024
12l	4095	2048

+According to the graph, the experimental result for **perfect binary tree** is that the average search cost grows **logarithmically** depending on n size. Therefore, it matches with theoretical results that perfect binary tree average search cost complexity is **O($\boldsymbol{log_2(n)}$).**

+According to the graph, the experimental result for **linear binary tree** is that the average search cost grows **linearly** depending on n size. Therefore, it matches with theoretical results that linear binary tree average search cost complexity is **O($\boldsymbol{n}$).**