CSCE 221 Cover Page
Homework Assignment

First Name: Dilanka Last Name: Weerasinghe UIN: 126007816

User Name: dweerasinghe E-mail address: dweerasinghe@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 in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources					
People	Used Lab Table	Used Discord for help			
Web pages (provide URL)					
Printed material					
Other Sources	ecampus PPT				

I certify that I have listed all the sources that I used to develop the solutions/code 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 (signature) Dilanka Weerasinghe Date 3/29/2020

1. The description of an assignment problem.
   (a) This assignment was to create a binary search tree that had the normal parameters for a binary search tree as well as the capability to calculate the search time that it requires to find the specific node.
   (b) The way to compile my program is to use the:
      i. >> g++ BSTree_main.cpp BSTree.cpp
      ii. I tried to make a makefile but I am terrible at doing that for c++ code. It does not click in my head.
   (c) The classes that I used were the Node and the Tree class to make the binary search tree. The libraries used were iostream vector and queue, but in the end queue was omitted.
      i. The Node class has pointers to the left and to the right and holds the values obj and search time. The object is the integer being stored and the search times is the cost for how long it takes to find that node.
      ii. The tree class is an assortment of Nodes connected by their left and right pointers. This will place in the nodes based on their values.
   (d)
2. The description of data structures and algorithms used to solve the problem.

(a) Provide definitions of data structures by using Abstract Data Types (ADTs)

i. The main data structure being used is a tree. More specifically a binary search tree.

(b) Write about the ADTs implementation in C++.

i. Most of the work is done with pointers. There is a Node that holds pointers to each of its child nodes and holds in itself an object. The Nodes are placed into the tree by detecting the values held in the Node and comparing until the object can be placed in the tree in accordance with the ordering of the objects. It will travel left along the branches if the object is less than and right if the object is greater than. The Tree class uses the Node class to branch across itself, adding Nodes whenever an object is inserted into the tree.

(c) Describe algorithms used to solve the problem.

i. Many of the algorityms used in this binary search tree use recursion. This is the easiest way to navigate through the search tree because any specific loop has no way of branching out in multiple directions and no way of knowing how far it needs to go. Recursivly this is solved because the node itself can check if it can go any farther.

(d) Analyze the algorithms according to assignment requirements.

i. The algorityms used in this assigment include a binary search tree. This is what was created and this is what was used.

3. A description of how you implemented the calculation of

   (a) individual search cost and

      i. Because this is a binary search tree the depth of the Node determines the search cost of the Node. The specific equation is 1+ the depth of the node. I went about this in two ways. One way was to calculate the depth of the root and calculate the depth of the Node. Here I would take the 1+ depth(root) - depth(Node). This would work for a fully filled binary tree but because there are many blank spots around a tree this was not viable. I decided to move on to making the assignment of the search cost recursive by counting the number of times it would have to go get cost.

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

      i. THe most helpful of the operations was the insert function. Doing this alone helped me understand how to navigate through the tree effectivly. Doing the search function was just like the insert function in every way. The average search cost was calculated by the total search time / the size. This was easy as the size was incrememted up as I added things to the tree and the search cost was updated as mentioned above.

   (c) Analyze the time complexity of calculating individual search cost and

      i. The time complexity of calculating individual search cost is O(n) in the worst case as it would need to travel linearly but in the best case O(logn)

   (d) (b) summing up the search costs over all the nodes.

      i. Summing up the cost of all the nodes takes O(n) because it will have to access every single node.

4. Give an 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 input data).

   (a) Perfect Binary Tree...

      i. The height of the tree is log(n), therefore the searc time would be equivilent the the height of the tree
      ii. $log(1+1)+2(log(2+1))+2^2(log(3+1))...$
      iii. $(n+1)(log(n+1)-n)$
      iv. $O(log(n))$

   (b) Lineary Tree

      i. Height of the tree is always equal to n-1
      ii. $n(n+1)/2$
      iii. $\frac{n(n+1)}{2n}$
      iv. $O(n)$

5. Include the table and plot of an average search cost that you obtain. Compare.

   (a) Code - Terminal

      i. Initiating Dilanka Test: Printing my Test: 1[1]
      ii. 1 Printing copy of Test 1[1] 1 1 1 average search time linear 1 1 1 average search time perfect 1 1 1 average search time random 1 1 1 prefect tree 1 1[1]
      iii. average search time linear 2 2 3 average search time perfect 2 1.66667 3 average search time random 2 1.66667 3 prefect tree 2 2[1] 1[2] 3[2]
      iv. average search time linear 3 4 7 average search time perfect 3 2.42857 7 average search time random 3 2.71429 7 prefect tree 3 4[1] 2[2] 6[2] 1[3] 3[3] 5[3] 7[3]
      v. average search time linear 4 8 15 average search time perfect 4 3.26667 15 average search time random 4 3.73333 15 prefect tree 4 8[1] 4[2] 12[2] 2[3] 6[3] 10[3] 14[3] 1[4] 3[4] 5[4] 7[4] 9[4] 11[4] 13[4] 15[4]
      vi. average search time linear 5 16 31 average search time perfect 5 4.16129 31 average search time random 5 6.3871 31

vii. average search time linear 6 32 63 average search time perfect 6 5.09524 63 average search time random 6 7.66667 63

viii. average search time linear 7 64 127 average search time perfect 7 6.05512 127 average search time random 7 7.59055 127

ix. average search time linear 8 128 255 average search time perfect 8 7.03137 255 average search time random 8 9.06667 255

x. average search time linear 9 256 511 average search time perfect 9 8.01761 511 average search time random 9 10.3033 511

xi. average search time linear 10 512 1023 average search time perfect 10 9.00978 1023 average search time random 10 12.2463 1023

xii. average search time linear 11 1024 2047 average search time perfect 11 10.0054 2047 average search time random 11 13.3972 2047

xiii. average search time linear 12 2048 4095 average search time perfect 12 11.0029 4095 average search time random 12 14.0237 4095

(b)

Node	Linear	Random	Perfect
1	1	1	1
3	2	1.666	1.666
7	4	2.71	2.714
15	8	3.73	3.266
31	16	6.387	4.161
63	32	7.666	5.095
127	64	7.59	6.055
255	128	9.066	7.031
511	256	10.3	8.0176
1023	512	12.24	9.009
2047	1024	13.398	10.005
4095	2048	14.023	11.0029