

## CSCE 221 Assignment 4 Cover Page

First Name Yingjian Last Name Wang UIN 726004401

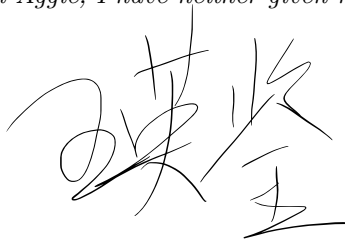
User Name Yingjian Wang E-mail address wang19485@tamu.edu

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)	<a href="https://piazza.com/class/jqwtg8502r96um">https://piazza.com/class/jqwtg8502r96um</a>
Printed material	Data Structures and Algorithms in C++ Second Edition ISBN-13 978-0-470-38327-8, page 11
Other Sources	Slides on ecampus

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



Date 03/31/2019

# Assignment 4 (100 pts)

Program: Due March 24 at 11:59 pm

## Objectives: Programming (70 points)

Write a C++ program to create binary search tree. Your program should empirically calculate the average search cost for each node in a tree and output a tree, level by level and in-order traversal, in a text format.

### 1. Programming (70 points).

(a) Build a binary search tree according to the instructions provided below:

- Read integer data from the provided files. Every line of the file contains one unique integer number (no duplicates). The size of any input file is not given and the termination of input data is done by detecting EOF.
- Print the input data on the screen only for small input size (do not do this for larger sizes).
- Create a binary search tree using the input data. Use the definition of a node and binary search tree provided in the **header file**.
- Calculate the search cost for each node **when it is inserted** in a binary search tree.
  - A node element has two data fields: value and search\_cost.
  - For each node, count and store the number of comparisons required for searching a node (i.e., the number of comparisons = the search cost for the node = 1+ the depth of the node).
- Print the tree data by performing in-order traversal operation. If the number of nodes is less than  $2^4$ , print on the screen the resulting tree along with the information about the nodes (for each node, print its value and search cost). Otherwise, print this tree and information about its nodes into a file. See the example below.
- Print the binary search tree on the screen level-by-level in the case when the number of nodes is less than or equal to  $2^4$  along with the information about the nodes (for each node, print its value and search cost). For trees larger than  $2^4$ , you do not print to a file as the file may be large, see the example at 2.
- Calculate the average search cost for all nodes in the tree and print it.
  - Sum up the search cost over all the nodes in your tree as you traverse the tree by in-order traversal.
  - Divide the sum by the total number of nodes in the tree, and the result is the average search cost for all the tree nodes
- Print total number of nodes

### 2. Example:

Input data:

3  
5  
7  
9  
10  
11

Create a binary search tree and provide information about each node when you display the tree.

Value	Search Cost
5	1
3	2
9	2
7	3
10	3
11	4

The total number of nodes is 6.

Use the in-order traversal to output a tree on the screen or to a file. Each node is represented in the following format Value[SearchCost]:

```
3[2] 5[1] 7[3] 9[2] 10[3] 11[4]
```

The search cost over all the nodes in the tree is:  $2 + 1 + 3 + 2 + 3 + 4 = 15$ . Average search cost is:  $15/6 = 2.5$ .

Output the tree level-by-level to a file (x stands for an empty node):

```
5[1]
3[2] 9[2]
X X 7[3] 10[3]
X X X X X X X 11[4]
```

### 3. Download files containing integer data from the course website.

- The files *1p*, *2p*, ..., *12p* contain  $2^1 - 1$ ,  $2^2 - 1$ , ..., and  $2^{12} - 1$  integers respectively. The integers make 12 **perfect binary trees** where all leaves are at the same depth. Calculate and record the average search cost for each perfect binary tree.
- The files *1r*, *2r*, ..., *12r* contain the same number of integers as above. The integers are randomly ordered and make 12 **random binary trees**. Calculate and record the average search cost for each tree.
- The files *1l*, *2l*, ..., *12l* contain same number of integers as above. The integers are in increasing order and make 12 **linear binary trees**. Calculate and record the average search cost for each tree.
- Make a table and a plot showing the average search cost (y-axis) versus the number of tree nodes (x-axis) of all perfect binary trees, random binary trees and linear binary trees.
- Provide the output in the text format for the files *1p-4p*, *1r-4r*, and *1l-4l*. Another option for tree drawing is to output elements under text mode using a queue. The nodes will be printed according to their depth level. Empty nodes will be represented by the symbol x. A possible solution is to fill the empty nodes with fake nodes in the data structure when printing the tree.

EXAMPLE:

```
5[1]
3[2] 9[2]
X X 7[3] 10[3]
X X X X X X X 11[4]
```

### 4. Hints

- Besides using links/pointers to represent a binary search tree, you may store the binary tree in a vector. This implementation might be useful, especially for the printing of a tree level-by-level. Please refer to section 7.3.5 at p. 295 of the textbook.
- To output a binary search tree level-by-level, you may use breadth-first-search (BFS) algorithm based on queue.
  - You may use the doubly linked list class from the assignment 3, or use the STL queue: [Link](#)

- ii. The following is a sample of pseudo-code of the BFS algorithm

```

Algorithm OutputTreeLevelByLevel(BinarySearchTree T)
Queue q
q.enqueue(T.root)
while (not q.empty()) do
    TreeNode node = q.dequeue()
    /* output a node (you need to determine whether to output a newline) */
    print node.value
    if (there are still un-enqueued non-null nodes in the tree) then
        /* enqueue children here for later output */
    endif
done

```

### Report (30 points)

Write a brief report that includes the following:

1. A description of the assignment objective, how to compile and run your programs, and an explanation of your program structure (i.e. a description of the classes you use, the relationship between the classes, and the functions or classes in addition to those in the lecture notes).
  - Use gcc to compile and run. `read_file` is for input and output. In the main, the function will be call multiple times. The first output operator will output both `print_level_by_level()` and `inorder()`. The input operator will create a tree by calling `insert()` function which will insert nodes through comparison. The copy constructor/assignment operator call the helper function `copyTree()` to copy. `inorder` just print out tree in infix order. `print_level_by_level` will output the tree level by level and do other outputs like size, search cost and table of cost.
2. A brief description of the data structure you created (i.e. a theoretical definition of the data structure and the actual data arrangement in the classes).
  - There are binary search tree, stack, vector. The tree basically shows up in whole project; stack is used in the implementation of `inorder()` function. Vector is used in `print_level_by_level` function.
3. A description of how you implement the calculation of (a) individual search cost and (b) average search cost and explain which tree operation (e.g. find, insert) was helpful. Analyze the time complexity of (a) calculating individual search cost and (b) **summing up** the search costs over all the nodes.
  - I calculate the cost while printing the tree level by level. (a) For individual cost, it is equal to the layer or depth of levels plus one. (b) For the average, sum the individual's cost and divide them by the number of nodes. Insert function is only be used in creating trees. (a) complexity of calculating individual search cost is  $O(\log n)$ . (b) summing up is  $O(n)$ .
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).
 
$$\sum_{d=0}^{\log_2(n+1)-1} 2^d(d+1) \simeq (n+1) \cdot \log_2(n+1) - n \quad \text{and} \quad \sum_{d=1}^n d \simeq n(n+1)/2$$
  - Individual: (a) perfect bst: cost =  $\log_2(n+1) = O(\log n)$  (b) linear bst: cost =  $(n+1)/2 = O(n)$
  - Individual search cost for perfect bst and linear bst respectlly:  $O(\log n)$ ,  $O(n)$
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.
  - The experimental results exactly match the theoretical analysis.

Balanced BST	
Size	average cost
1	1
3	1.66667
7	2.42857
15	3.26667
31	4.16129
63	5.09524
127	6.05512
255	7.03137
511	8.01761
1023	9.00978
2047	10.0054
4095	11.0029

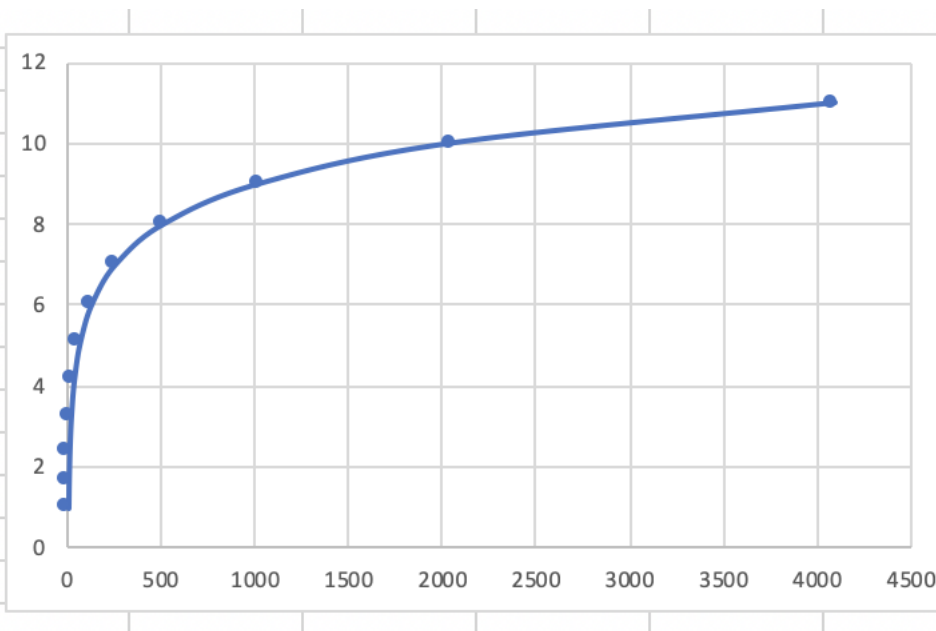


Figure 1:

Linear	
size	avg cost
1	1
3	2
7	4
15	8

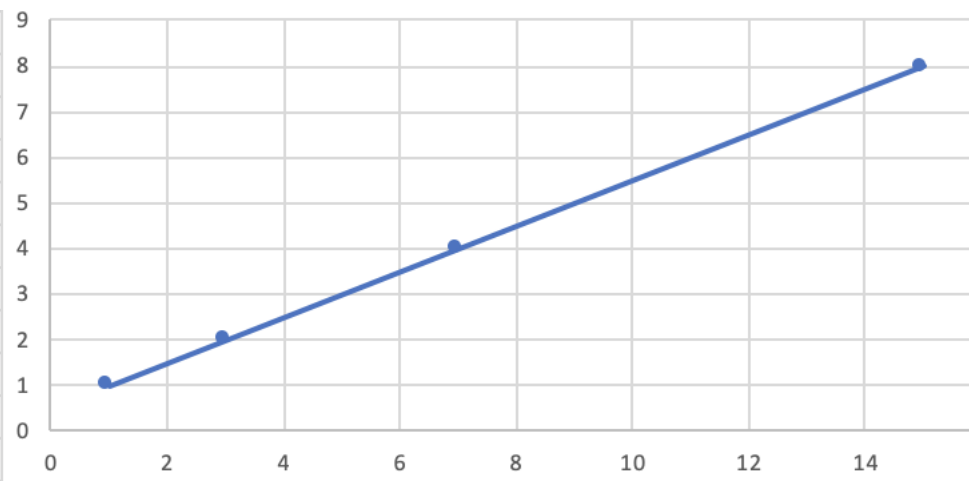


Figure 2: