**CS 260 Self Evaluation for Assignment 6 – Enhance Binary Search Tree**

|  |  |
| --- | --- |
| Your name:  David Oftedahl | Date:  12/02/2022 |
| Are you willing to allow your code to be used in example debugging demonstrations or documentation?  Yes  No | |

**Instructions – Part 1**  
This document is to be turned in alongside solution of this lab. You will use this document to indicate your status on the lab, as well as areas where you are struggling conceptually or in converting concept to code. Please use the space underneath each evaluation criteria to describe any errors you are receiving or challenges you are having implementing the required functionality for your code.

**Project Organization (please fill in for your language)**

|  |  |
| --- | --- |
| ***File Structure (C++)*** | |
| Is your class split into a header and a source file? |  |
|  | |
| Does your destructor delete all nodes from the tree? |  |
|  | |
| ***File Structure (C#)*** | |
| Is your class declared in a class library, separate from your console driver? |  |
|  | |
| ***File Structure (Python)*** | |
| Is your class declared in a separate, imported file? |  |
| Yes | |

**Functionality**

|  |  |  |
| --- | --- | --- |
| **Criteria** | | |
|  | |  |
| ***Basic Class Design*** | | |
| Do all parts of the program compile without errors or warnings? |  | |
| Yes, compiles without error. | | |
| Does the test program run with your class without crashing? |  | |
| Yes, runs without crashing | | |
| Does it complete all the tests properly? | | |
| ***Binary Search Tree*** | | |
| Does insertValue work correctly? |  | |
| Yes, insertValue works correctly | | |
| Does your inOrder traversal properly display values from smallest to largest? |  | |
| Yes, the inOrder displays smallest to largest | | |
| Do all of your traversals return the expected values? |  | |
| Yes, all traversals match the driver. | | |
| Does findValue work correctly? |  | |
| Yes, findValue works as expected. | | |
| Does removeValue work correctly? |  | |
| Yes, removeValue works correctly, locating the item and marking it as deleted without actually removing it. | | |
| ***FindNext Search Tree*** | | |
| Does the findLarger method return the closest value that is greater than or equal to the value requested? |  | |
| Yes, the findLarger returns the next largest value if the value cannot be found. | | |
| Does the removeLarger method work properly? |  | |
| Yes, the removeLarger uses most of the findLarger code, yet also marks the item returned as removed. | | |

**Instructions – Part 2**   
Please answer the following questions, in your own words, regarding your experiences throughout this lab.

**Experiential Review**

|  |  |
| --- | --- |
| **What aspects of this lab did you find most challenging?** | |
| I found the most challenging part of the lab to be the findLarger function. It took me a few different iterations and re-writing my pseudocode a few times before I was able to get it to function properly. |  |
| **What concept from this lab do you feel you have the best grasp on now?** | |
| I feel I have a pretty good grasp on how to build the recursive functions that are first called from the master function. Having the answer to the previous lab really helped me understand this concept. | |
| **What must be true for a binary search tree to be efficient?** | |
| For a binary search tree to be efficient, it’s important for them to be balanced. An unbalanced binary search tree could end up as a linked list if the items are all being inserted in ascending order (or descending order) without any balancing, which would make them much less efficient. | |
| **Describe in your own words how binary search trees achieve their Big O efficiency:** | |
| Binary search trees achieve their Big O efficiency by maintaining balance and allowing us to cut our remaining searches in half with each iteration of searching. Because we know that each greater item will be inserted to the right, and lesser item inserted to the left, we are basically cutting the tree in half each time we make a comparison. | |
| **Compare and contrast the efficiency of using a binary search tree vs other search methods you have used. Include Big O efficiencies.** | |
| Most of the search methods we have used up to this point have O(N) or even O(N^2) efficiency, whereas a binary search tree would have O(LogN) efficiency. As our N grows larger, searching through a binary search tree becomes significantly more efficient. | |