Professional Integrity Report (CPS101)

Student Name: Alex McColm **Assignment #: 6**

It took me 10 hours to complete the assignment.

These parts of the program work well:

The Binary Search Tree is successfully converted to generic. The tree traversal methods to print in pre-order, post-order, and level order recursively are functional. The test driver successfully adds 10 random integers, prints the tree in diagram form using methods from the textbook, and traverses it with each of the tree traversal methods I wrote. The Trie has almost all* its basic functionality – Trie nodes are inserted, searched, and deleted properly. The Trie has constructors which take files, strings, or string arrays. The Trie test successfully loads the entire Dictionary file into a Trie and performs searches to make sure they are all present.**

These parts of the program don't work well (please identify the specific problem):

***** The Trie display method does not work properly. It outputs all existing nodes, but not laid out in order. The Trie constructor which takes a file does not add words which contain characters not in the alphabet (such as "ground-breaking"), it skips over them.

I learnt the following in doing the assignment:

Pushed my skills in understanding and applying recursion much further by struggling to work out and implement recursive algorithms for operating on BSTs and Tries. Saw first-hand the efficiency of both Binary Search Trees and Tries, and the thought that goes into them.

The difficulties I encountered were:

I had some difficulties with generics, specifically trying to make a GenericQueue parametrized with Node<T> objects.

Here are some other comments or suggestions:

My analysis of the time and space complexity of the methods in the Trie class follows. Let N represent the number of elements in the Trie, let M represent the number of characters in a string being operated on, and let L represent the length of the longest string in the tree.

Method	Big O Time Comp.	Big O Space Comp.	Justification
Insert()	$O(M)$	$O(M)$	At most M constant time operations are performed, adding M nodes.
Search()	$O(M)$	$O(1)$	Checks M nodes at most. Only creates 1 reference, used for traversal.
Contains()	See Search()	See Search()	Calls Search()
Delete()	$O(M)$	$O(1)$	At most, visits M nodes. Similar to Search() under the hood. Creates 2 references, character and child, during the process.