SUMMARY

~~Project 2 is an exercise in the implementation and run-time analysis of BST (Binary Search Tree) and AVL Trees. As a short recap, BSTs are special trees with 2 conditions: nodes are ordered, and each node has at most 2 children. AVL trees are an extension of BSTs with the additional requirement that they are self-balancing; every insertion or removal requires a balance check.~~

~~Each~~ **~~part of the program is run from a Driver\_part1 and Driver\_part2 respectively~~**~~. Much of the code for this project for BST, AVL construction and presentation has been adopted and modified from:~~

* ~~my previous data structures and algorithms from Michael Loceff at Foothill College~~
* ~~CS 146 material from Professor Ron Mak at San Jose State University~~

~~The corresponding algorithms from the implementation match those shown in the CS 146 Fall 2015 lectures. In addition, the key algorithms of balancing by rotation, insertion and removal have been verified to work by working the problems by hand with trees on the whiteboard.~~

~~For part 1 of the assignment, the BST and AVL tree are verified to work correctly. They are each instantiated, and nodes are inserted until about height 5. The results can be seen in~~ **~~Appendix~~** ~~A.~~

~~For part 2 of the assignment, performance of each tree is analyzed for different but very large n size values. Part 2.1 tests different numbers of insertions, 2.2 tests searching for k random integers, 2.3 analyzes the effects of both insertion and searching. The table below shows the results of the timings and the analysis. Red shows the raw times gathered from each run. Pink shows the calculations for Part 2.1, Light Green for 2.2, and Turquoise for 2.3 respectively.~~

QUESTIONS ANSWERED:

1. Who is in your group?

Elton Vinh, Thien Van

1. How long did the project take?

Project initiated on September 24th, 2015 and ended on October 4th, 2015. Spanned about 11 days. Total time contributed ~20-25 hours.

1. Before you started, which data structure did you expect would be the fastest?
2. Which data structure is the fastest? Why were you right or wrong?
3. In general, which DataCounter dictionary implementation was “better”: trees or hash tables? Note that you will need to define “better” (ease of coding, ease of debugging, memory usage, disk access patterns, runtime for average input, runtime for all input, etc)
4. Are there cases in which a particular data structure performs really well or badly in the correlator? Enumerate the cases for each data structure.
5. Give a one to two paragraph explanation of whether or not you think Bacon wrote Shakespeare’s plays based on the data you collected. No fancy statistical analysis here (formal analysis comes later); keep it fun and simple.
6. Writeup your benchmarks (this is long). Your mission is to convince us that your benchmark makes sense and that we should be interested in it if we are trying to ascertain which data structure is better suited for your input. You will need to answer at least the following (all formal analysis should answer something similar):
   1. What are you measuring?

**Appendix** A:

**Appendix** B: