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. Roughly 8 hours spent developing the data structures, 4 hours developing the word count and correlator algorithms, 1 hour determining and selecting the relevant text files for study, 6 hours developing automated tests, 3 hours writing the report. Total time contributed ~20-25 hours.

1. Before you started, which data structure did you expect would be the fastest?

The expectation is that Hash Table would be the fastest. Insertions are O(1) versus O(n log n) for AVL tree.

1. Which data structure is the fastest? Why were you right or wrong?

AVL tree is the fastest for the majority of the test cases run with WordCount.java. We were both right and wrong; it depends on the use case; for super large text files (i.e. The King James Bible) we were right as the Hash Table wins, but for “regular” size text files (like the plays, classic novels, scientific papers), we were wrong as the AVL Tree wins. The regular size files are the typical use case.

1. 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)

??????????????????

1. Are there cases in which a particular data structure performs really well or badly in the correlator? Enumerate the cases for each data structure.

In general Correlator runtimes are analogous to the WordCount runtimes. The test was run for 12 comparison cases for each of the ADT (BST, AVL, HT). AVL performs the best, followed by BST, followed by HashTable. The runtimes however are very close to each other. A summary is listed below:

AVL: 7.73 seconds

BST: 7.86 seconds

HT: 8.76 seconds

See **Appendix A: Figure 1** for the summary of the test run

1. 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.

For the test methodology we used multiple works of Shakespeare and Bacon, as well as a couple of other texts for comparison. In order the texts are:

* Shakespeare: hamlet, othello, the-tempest
* Sir Francis Bacon: essays, novum (scientific), the-new-atlantis
* King James Bible
* alice (Alice in Wonderland)

For the Test design:

0) Use HashTable backend for all tests (assuming it's the most efficient)

1) Compare all permutations of Shakespeare's works to find errorSums. Note the median errorSum for these 3 Shakespeare works (expected errorSum when comparing works within single author) which is (3 choose 2). Use the text that produces the median errorSum to compare against others.

2) Do the same for Sir Francis Bacon

3) Run the different permutations (4 choose 2)

s b, s k, s a, b k, b a, k a

Based on the test results, **we believe that they are different authors**. Shakespeare – Shakespeare errorSum is about: 1E-4. Bacon – Bacon errorSum is about: 4E-5. Shakespeare – Bacon errorSum is significantly higher, about: 5.6E-4. For detailed analysis, see **Appendix B**.

1. 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?

Measuring runtime of the count scripts (shell, perl), and WordCount with the 3 data structures (BST, AVL, HT) by doing direct comparisons on 8 total texts.

* 1. What is the definition of “better” given your measurement?

Better means giving the same results in less time (lower runtime)

* 1. Why is the measurement interesting in determining which is the superior algorithm for the project?

For real life applications, response and calculation time are incredibly important. Searching, insertion, deletions are regular used. Examples include simple sorting algorithms in determining order.

* 1. What was your method of benchmarking?

Method of benchmarking was running the programs through a PowerShell script. Each of the 5 count frequency runs (shell, perl, Java BST, Java AVL, Java HT) for each of the 8 texts was timed individually. Frequency is noted for results < 1% or > 0.01 %. Short times are noted. See **Appendix A: Figure 2** for detailed results.

* 1. What were the sources of errors?

Sources of error mostly came from the data structure implementation. As the data structure needed to implement the incCount() method. Also, the initial Hash Table size wasn’t efficient for these set of tests. We needed a much larger prime so that resizing (overhead) did not occur as much.

* 1. What were your results?

Results were that for HashTable is the fastest for the large text case (King James Bible). However, for all other cases, AVL Tree won. This is true for both the WordCount and Correlator timing tests. See **Appendix A: Figure 3** for detailed results.

* 1. How did you interpret your results?

Results were mostly interpreted by the runtimes. For the errorSum analysis, basic statistical techniques such as range, mean, median were used to determine “uniqueness” of a work. For Shakespeare and Bacon, the “average” text was used for different test permutations.

* 1. What were your conclusions?

From these texts, for a single author the average errorSum is about 8E-5 and largest difference of 1.75E-4. For different authors, there is about a 4E-4 overall average errorSum and largest range of 8E-4. There seems to be about a **4 factor average difference between different authors**. The errorSum range for a single both single and different authors are about **2 X factor larger than the average**.

* 1. Are there any interesting directions for future study?

More detailed analysis would need to be used with more precise word stemming techniques. In this case most of the texts used were at least a few hundred years old; more detailed testing would have more texts from the modern era. In addition, text size was not varied; future testing will have short stories, different genres of text, modern scientific papers and larger texts (e.g. the Epic).

1. What did you enjoy about this assignment? What did you hate? Could we have done anything better?

We enjoyed the having the flexibility to perform our own analysis. Picking the texts for our cases was fun. Coming up with the automated testing is a good skill to have.

One thing we did not like is the assumed reliance on Unix/Linux systems for running the timings. To improve, the instructors can have additional instructions for Windows users (e.g. basics of powershell timing).

**Appendix** A: Raw results

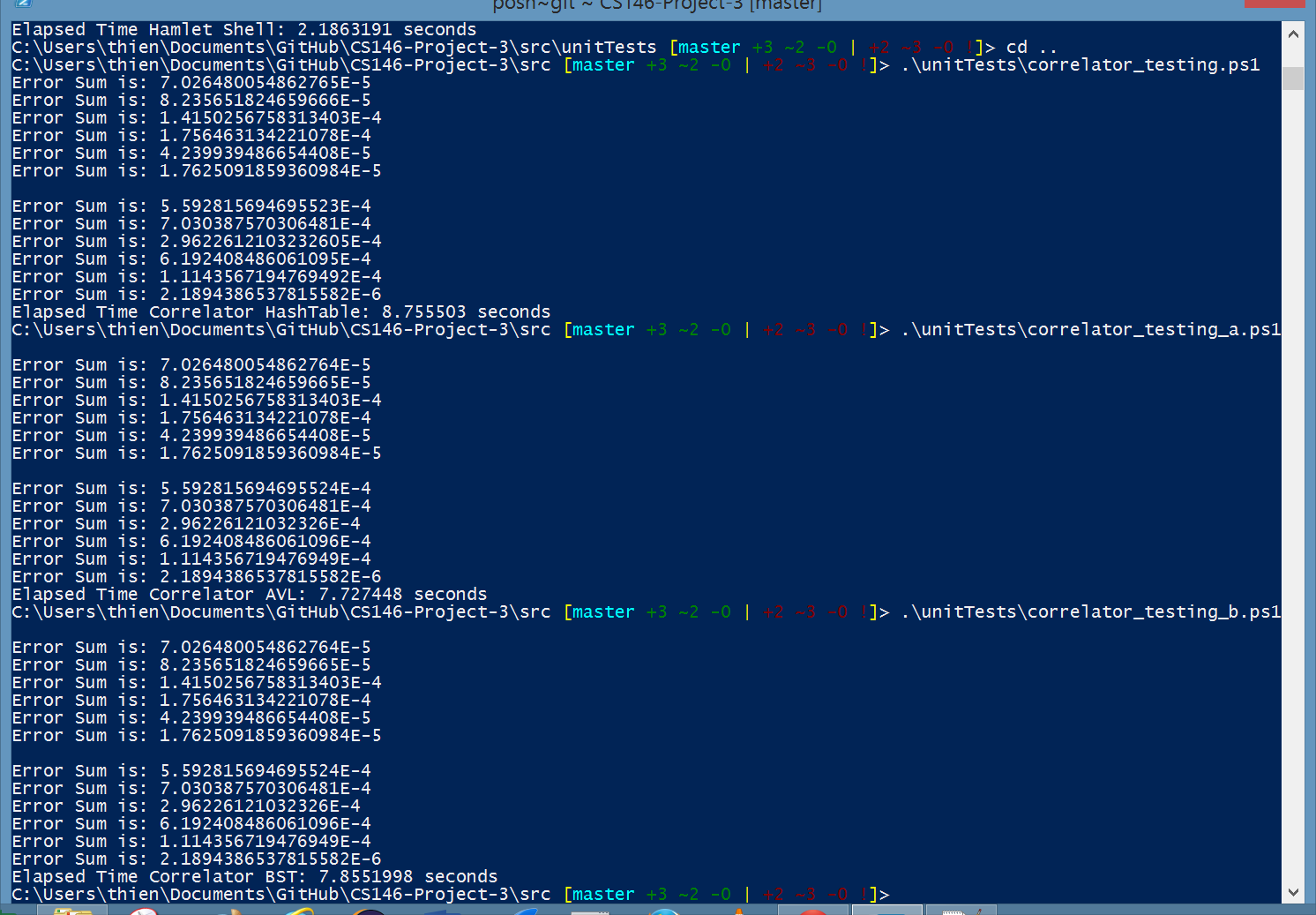
Figure 1: Correlator time comparisons for 12 comparisons using HashTable, AVL, BST respectively

Figure 2: Error sums of the various permutations

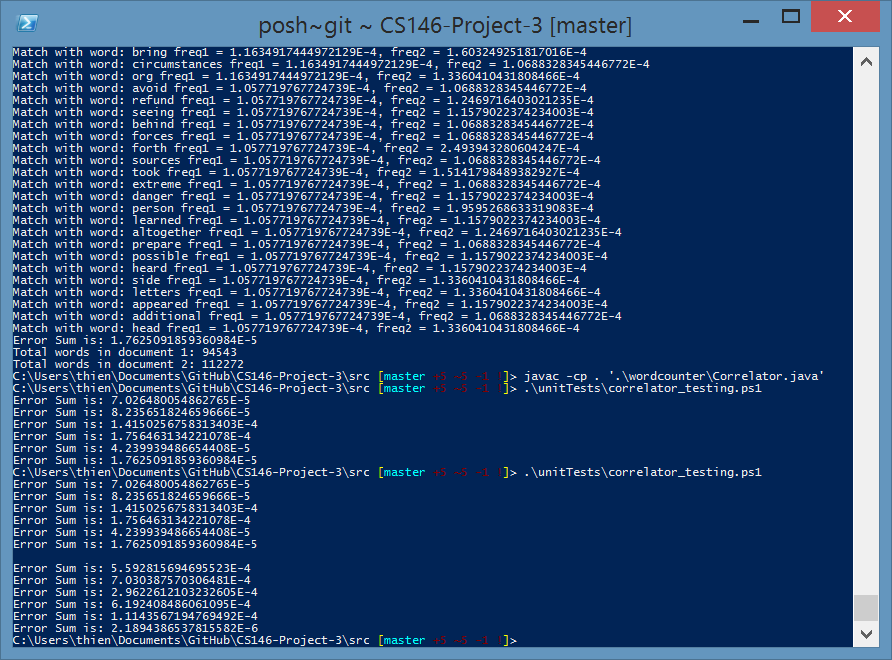
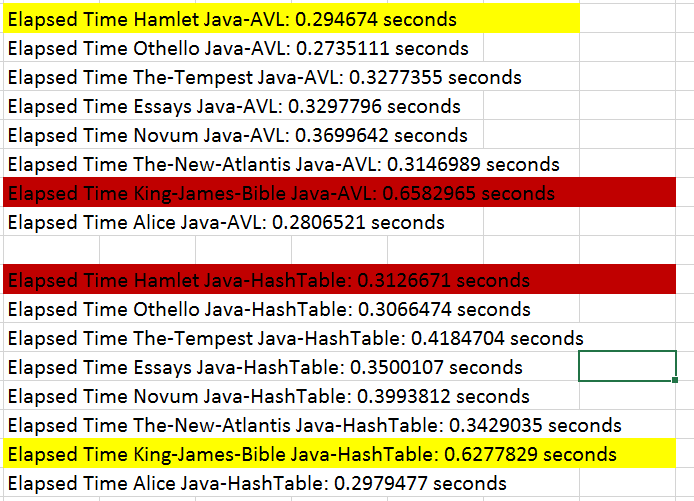


Figure 3: Runtimes of WordCount.java for AVL and HashTable



**Appendix** B: Raw analysis

