CSE 332 Project 2 Write up

* Note: Three of the last questions require you to write code, collect data, and produce graphs of your results together with relatively long answers. Do not wait until the last minute to start this write up!

1. **Who is in your group (Give name & UW NetID of each person)?**

2. **What assistance did you receive on this project? Include anyone or anything *except* your partner, the course staff, and the printed textbook.**

3. **a) How long did the project take?**
   b) Which parts were most difficult?
   c) How could the project be better?

4. **(OPTIONAL) What "above and beyond" projects did you implement? What was interesting or difficult about them? Describe in detail how you implemented them.**

5. **a) How did you design your JUnit tests & what properties did you test?**
   b) What properties did you NOT test ("I tested everything" is NOT a valid answer since it is impossible to test every property with every possible input)?
   c) What boundary cases did you consider?

6. **a) The iterator for Binary Search Tree used a Stack as an internal data structure. Why does the BST iterator need to use an internal data structure?**
   b) If you were to write an iterator specifically for the AVL Tree, how could you guarantee that no resizing of the Stack (No size increase of the internal array) occurs after iteration has begun (which may require changing the interface of `GStack`)? Start by thinking about what would be the smallest size of an array that guarantees no resizing.

7. **If `DataCounter`'s iterator returned elements in "most-frequent words first" order, you would not need to sort before printing. For each `DataCounter` (BST, AVL, MoveToFrontList, HashTable), explain how you would write such iterator and what its big-O running time would be.**

8. **For your HashTable to be <u>CORRECT</u> (not necessarily *efficient*), what must be true about the arguments to the constructor (Think about the relationship between the two arguments)?**

9. **Conduct experiments to determine which `DataCounter` implementation (BST, AVL, MoveToFrontList, HashTable) & Sorting implementation (insertionSort, heapSort, OtherSort) is the fastest for large input texts.**

   a) Describe your experimental setup:
   1) Inputs used
   2) How you collected timing information
   3) Any details that would be needed to replicate your experiments

   b) Experimental Results (Place your graphs and tables of results here).
   You need to conduct experiments for all possible combinations, 4 DataCounters X 3 Sorting algorithms = 12 experiments if you measured the runtime of DataCounter and Sorting together, or 4 DataCounters + 3 Sorting algorithms = 7 experiments if you measured DataCounter and Sorting runtimes separately. Don't forget to give a title and label the axes for all graphs. Make sure to choose appropriate graphs to clearly show the important points of your data (i.e. How do the runtimes of the 3 Sorting algorithms compare to each other?)

   c) Interpretation of Experimental Results
   1) What did you expect about the results and why?
   2) Did your results agree with your expectations?
   3) If the results did not match with your expectations, why do you think this happened?
   4) According to your experiments, which DataCounter & Sorting Algorithm combo is the best?

   d) Are there (perhaps contrived) texts that would produce a different answer, especially considering how MoveToFrontList works?

10. **Conduct experiments to determine if changing the hash function affects the runtime of your HashTable.**

    a) Brief description of your hash functions

    b) Experimental Results (Place your graphs and tables of results here).
    Experiment with at least 2 hash functions (3 Sorting Algorithms X 2 Hashing functions = 6 OR 2 Hashing functions = 2 experiments depending on how you measured the runtime)
    Don't forget to give each graph a title and label the axes.

    c) Interpretation (Your expectations and why? Did it match your results? If not, why?)

11. **Conduct experiments to determine whether it is faster to use your *O(n log k)* approach to finding the top *k* most-frequent words or the simple *O(n log n)* approach (using the fastest sort you have available).**

    a) Produce a graph showing the time for the two approaches for various values of *k* (where *k* ranges from 1 to n).
    If you measure runtime including the time it takes to print, you should print the same number of words (i.e. print top-k words for both n log k and n log n algorithms) to account for time it takes to print. You don't have to measure runtime for every possible value of k; you can use something like increments of 10 or 20. Don't forget to give a title and label the axes for graphs. Make sure to choose *appropriate graphs* to clearly show the important point of your data (i.e. Some Bar graphs may not be ideal to show how runtime changes as k increases).

    b) Interpretation of Experimental Results
    1) What did you expect about the results and why?
    2) Did your results agree with your expectations?
    3) If the results did not match with your expectations, why do you think this happened?

    c) How could you modify your implementation to take advantage of your experimental conclusion in b)?

12. **Using `Correlator`, does your experimentation suggest that Bacon wrote Shakespeare's plays? Show at least one (you can experiment with more texts if you want) correlation value for each of:**
    a) Shakespeare's work compared to Shakespeare's work
    b) Bacon's work compared to Bacon's work
    c) Shakespeare's work compared to Bacon's work
    According to the results of your experiments, did Bacon write Shakespeare's plays?

13. **<u>If you worked with a partner</u>:**
    a) Describe the process you used for developing and testing your code. If you divided it, describe that. If you did everything together, describe the actual process used (eg. how long you talked about what, what order you wrote and tested, and how long it took).
    b) Describe each group member's contributions/responsibilities in the project.
    c) Describe at least one good thing and one bad thing about the process of working together.

Appendix

Place anything else that you want to add here.