Analysis Document for Assignment 4

Philippe David (u0989696)

1. **Who is your programming partner? Which of you submitted the source code of your program?**

My programming partner was Anthony Chyr (u0627375). He is the one who submitted the source code for this assignment.

2. **What did you learn from your partner? What did your partner learn from you?**

We spoke a lot about test driven development while working on this assignment. This development technique is a approach where tests are created before any functional code is written. Many of the tests in this assignment were created before we even wrote our main program. Anthony has an excellent testing methodology and I've become better at thinking about appropriate test cases since I've worked with him.

Another thing I've learned from him was how to use Java lambda expressions that are new in Java 8. A Java lambda expression is a function that can be created without belonging to any class. The trim white space function in our file reading method in the main code is example of a lambda expression. It was very interesting to talk with him about the potential uses of these functions.

I've taken CS 1410 previously and therefore I was a little more familiar with Java and the Eclipse IDE than Anthony was. Some of the debugging features were unfamiliar to him and I was able to show him what I know. I also contributed to our coding of course. We have different ways of thinking about the problems but we work well together.

3. **Evaluate your programming partner. Do you plan to work with this person again?**

Anthony was an excellent partner for this assignment. It was great to work with him and I had a great time problem solving with him when we got together on Friday. I would definitely be happy to work with him again.

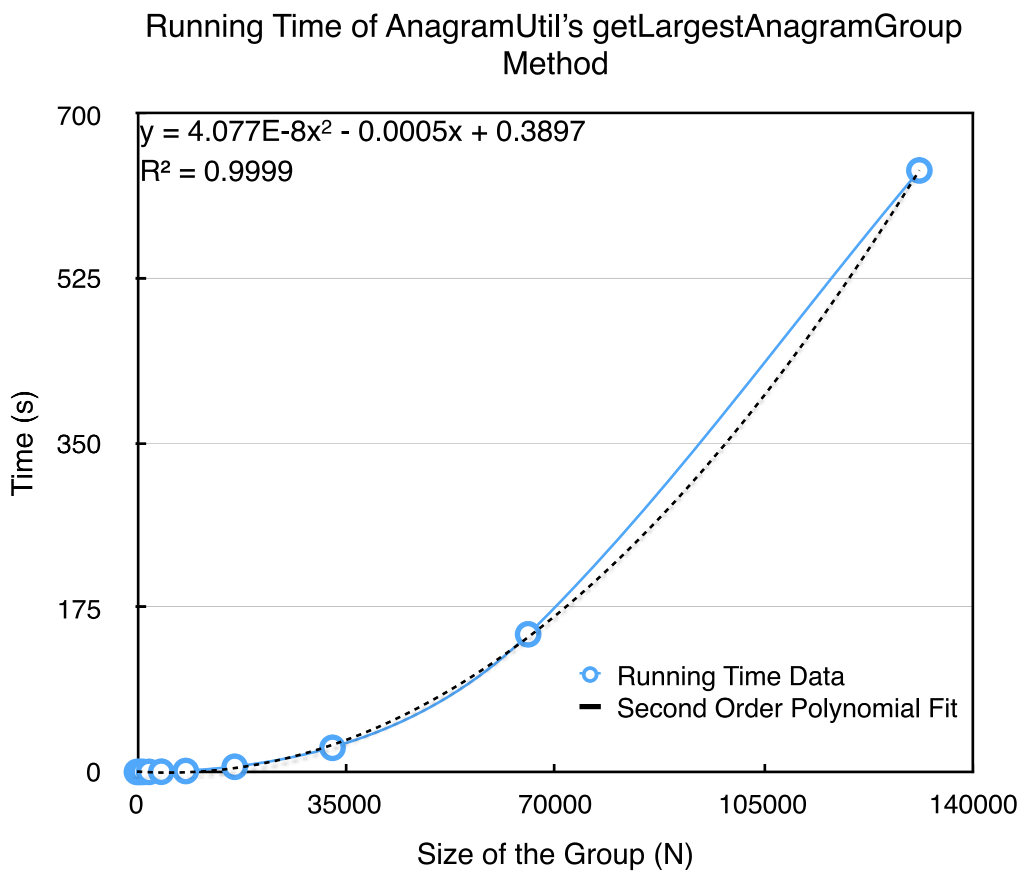
4. **Analyze the run-time performance of the areAnagrams method. What is the Big-O behavior and why? Be sure to define N.-Plot the running time for various problem sizes (up to you to choose problem sizes that sufficiently analyze the problem). Does the growth rate of the plotted running times match the Big-O behavior you predicted?**

The areAnagrams method is a quadratic (i.e., O(N^2)) algorithm where N corresponds to average length of the two String arguments required for this method to function. The areAnagrams method complexity is completely dominated by the insertion sort algorithm used to lexicographically sort each of the words passed in. The insertion sort algorithm is an O(N^2) algorithm. The pseudo code for insertion sort shown below:

```
for i ← 1 to length(A)-1
   j ← i
   while j > 0 and A[j-1] > A[j]
      swap A[j] and A[j-1]
      j ← j - 1
   end while
end for
```

shows that the O(N^2) complexity comes from the for loop containing a nested while loop. As we have discussed in class, the best case for inserting sort is when the items being sorted are already in order allowing the algorithm to never enter the nested while loop resulting in O(n) time complexity. In the average and worst cases, this algorithm has O(N^2) complexity since on average there will be (number of possible pairs) / 2 inversions. The worst case for this algorithm is when the items are sorted backwards and every possible pair is an inversion.

The growth rate of the plotted running times (shown below) matches the Big-O behavior predicted. The fit included on the graph also appears to indicated O(N^2) complexity.

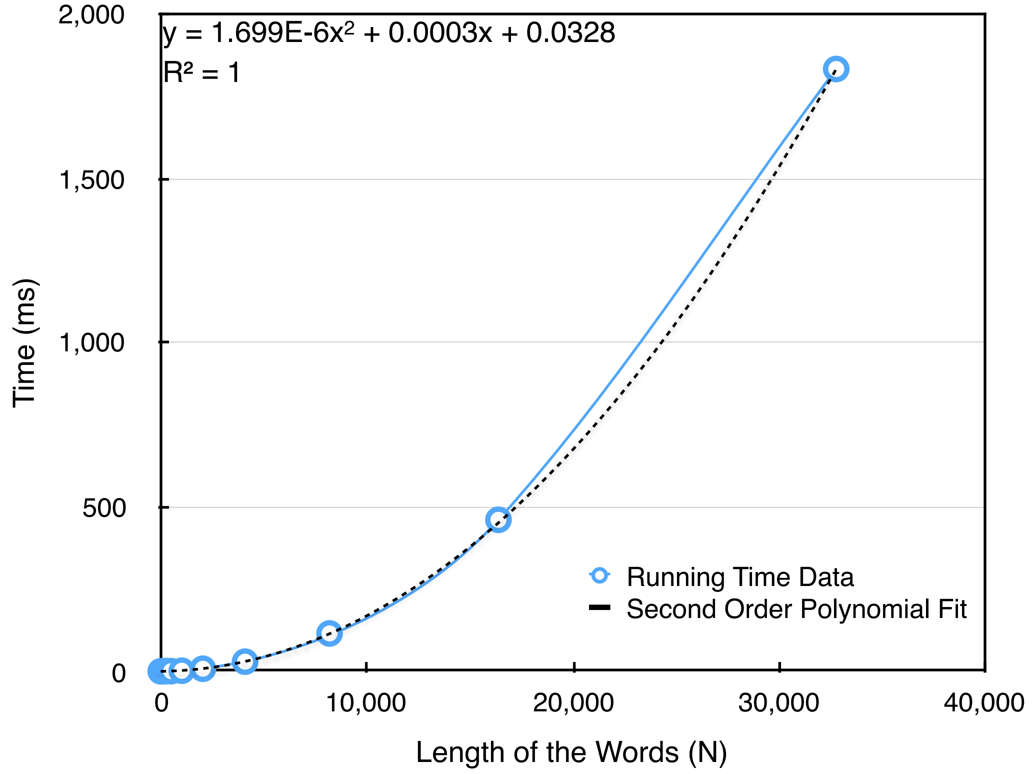


5. **Analyze the run-time performance of the getLargestAnagramGroup method using your insertion sort algorithm. (Use the same list of guiding questions as in #4.) Note that in this case, N is the number of words, not the length of words. Finding the largest group of anagrams involves sorting the entire list of words based on some criteria (not the natural ordering). To get varying input size, consider using the very large list of words linked on the assignment page, save it as a file, and take out words as necessary to get different problem sizes, or use a random word generator, provided in AnagramTester.java. If you use the random word generator, use a modest word length, such as 5-15 characters.**

The getLargestAnagramGroup method is a quadratic (i.e., O(N^2)) algorithm where N corresponds to is the number of words (Strings) contained in the list passed to the method as an argument. This algorithm has average and worst case run-time performance of O(N^2) which comes from the insertion sort performed on the list of lexicographically sorted words. This also gives a best possible case of O(N) complexity which can occur if the list is already sorted.

The growth rate of the plotted running times (shown below) matches the Big-O behavior predicted. The fit included on the graph also appears to indicated O(N^2) complexity.

Running Time of AnagramUtil's areAnagrams Method

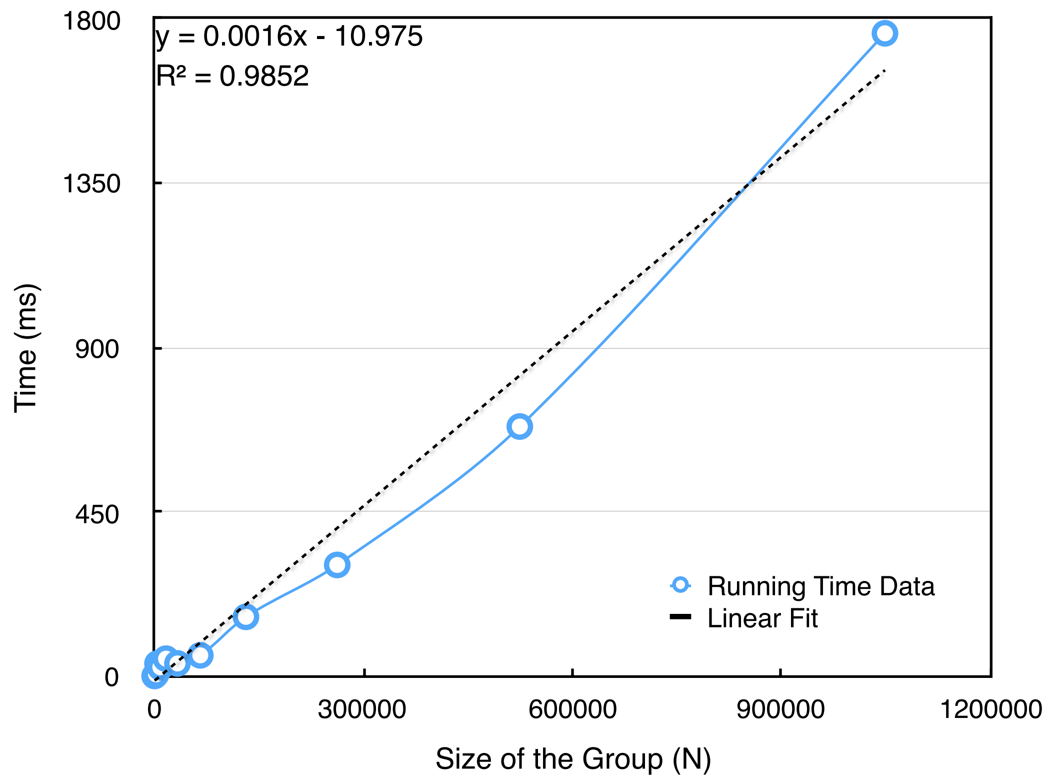


6. **What is the run-time performance of the getLargestAnagramGroup method if we use Java's sort method instead (http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html)? How does it compare to using insertion sort? (Use the same list of guiding questions as in #4.)**

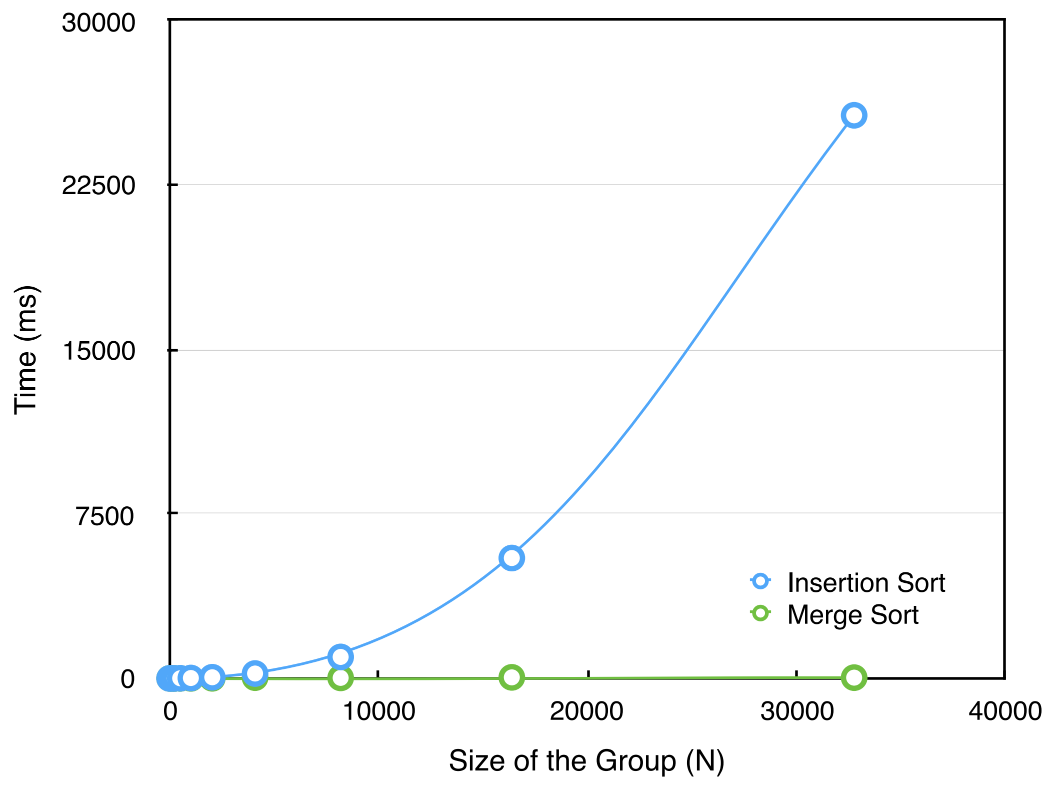
Using Java's sort method, the run time complexity of the getLargestAnagramGroup method is O(N*log(N)), where N is the number of words (Strings) contained in the list passed to the method as an argument. As determined in Question 5, the run-time performance of this method is dominated by the sorting of the list of already lexicographically sorted Strings. In the previous question, we used an insertion sort algorithm to accomplish this. Java's sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed N*log(N) performance. Mergesort is a divide and conquer algorithm which has much better run-time performance than insertion sort as shown in the figure below.

The growth rate of the plotted running times matches the Big-O behavior predicted. Additionally, the fit on the graph indicates that the complexity is increasing faster than linearly, suggesting a O(N*log(N)) complexity.

Running Time of AnagramUtilAlt's getLargestAnagramGroup Method With Merge Sort



Comparison of Insertion Sort and Merge Sort Running Times



7. **How many hours did you spend on this assignment?**

I spent about 12 hours in total working on this assignment.