Harry Kim
U1226472
9/23/2020
CS 2420

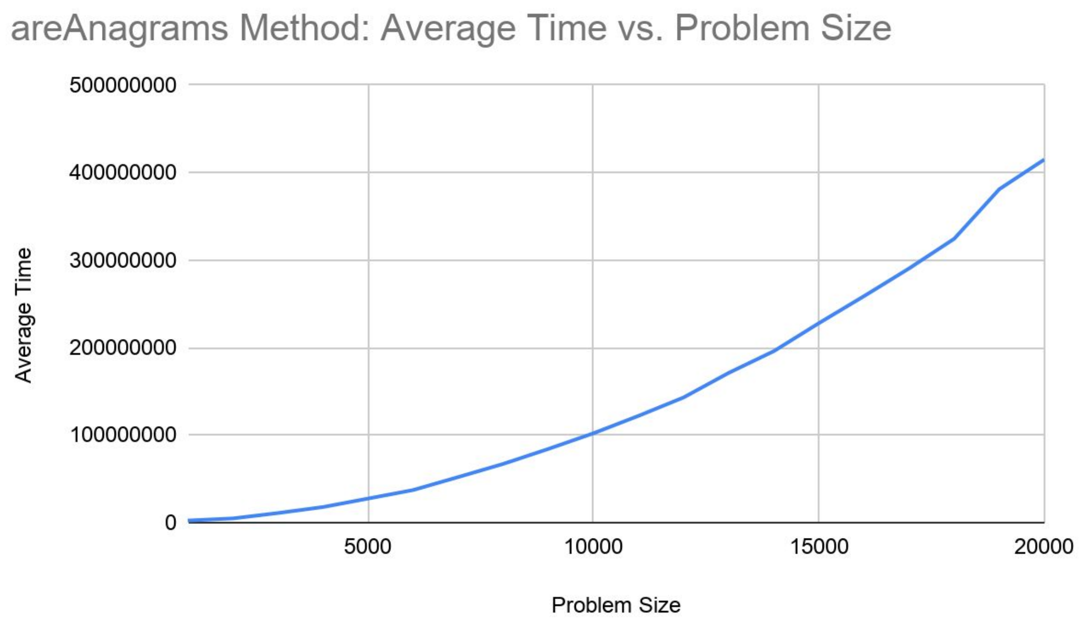
Analysis Document — Assignment 4

1. I am **not** invoking one of my three exemptions for pair programming due to an extenuating circumstance. My programming partner is Braden Morfin, and I submitted the program to Gradescope.

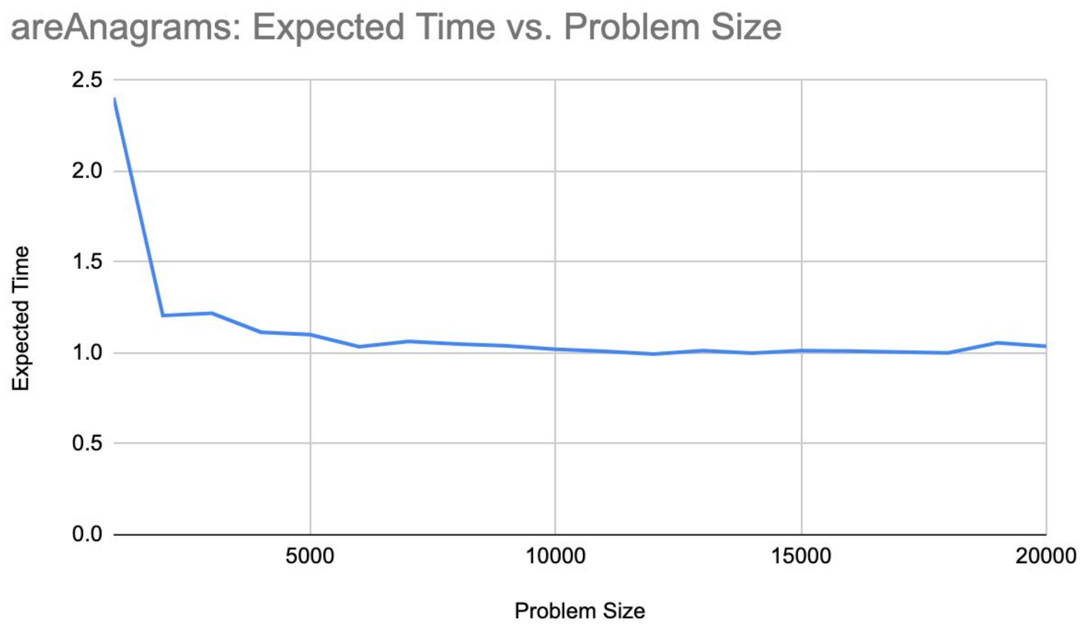
2. Pair programming experience:
    - I think my partner and I applied the techniques of pair programming to this assignment very well. I think we both learned from each other a lot.
    - Together, my partner and I developed the program for this assignment efficiently. We got a little stuck on the *getLargestAnagramGroup* method for a bit but we were able to resolve that after working some things out and stepping away from the code for a bit.
    - My partner and I spent about two hours each day for 4 days to complete the assignment and create tests for the assignment for a total of 8 hours.
    - Braden's very smart and is a nice person, he's good at communication, and he codes very well. I would plan to work with Braden again.

3. The reason why the signature of the method *String sort(String)* is not *void sort(String)* is because strings, unlike arrays, are immutable and cannot be changed, and so we must return a new string that has the sorted letters from the original. Also, the reason why the signature of method *void insertionSort(T[], Comparator<? super T>)* is not *T[] insertionSort(T[], Comparator<? super T>)* is because arrays, unlike primitives such as int and boolean, are objects in Java and are of reference type, and so the objects within the array can change but the reference for the array never changes. This is why methods can change the contents of the arrays and may have a void return type.

4. In order to collect the data, I plotted the running time of the *areAnagrams* method for problem sizes 1,000 to 20,000 by steps of 1000. The big O of the *areAnagrams* method should be O(n^2), n being the input size, because our *areAnagrams* method makes use of the sort method which contains a nested for loop for strings.

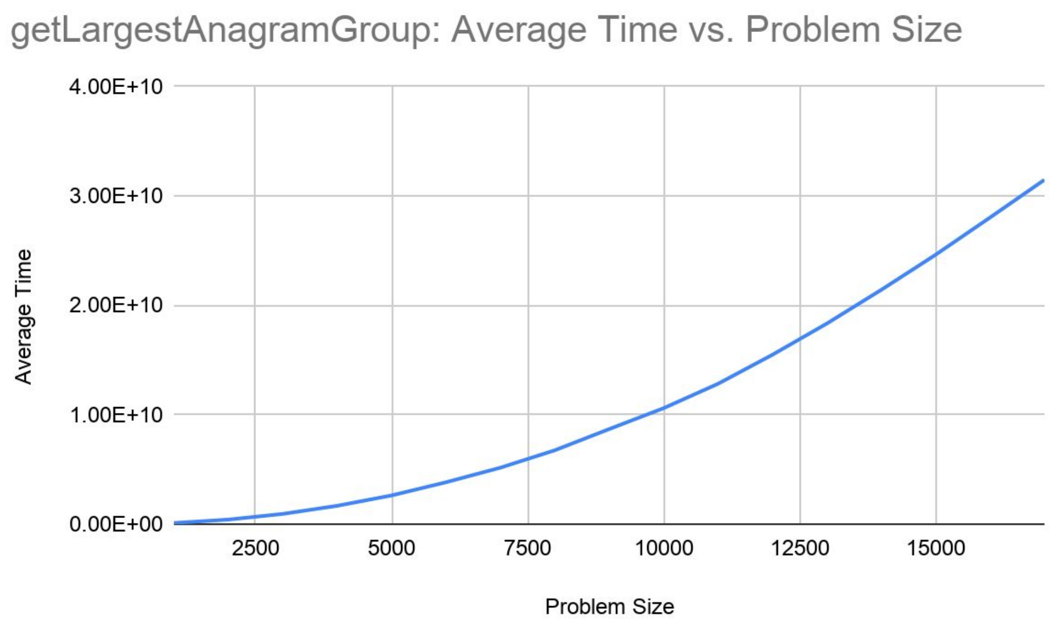


The growth rate of the plotted running times matches the Big-O behavior we predicted.

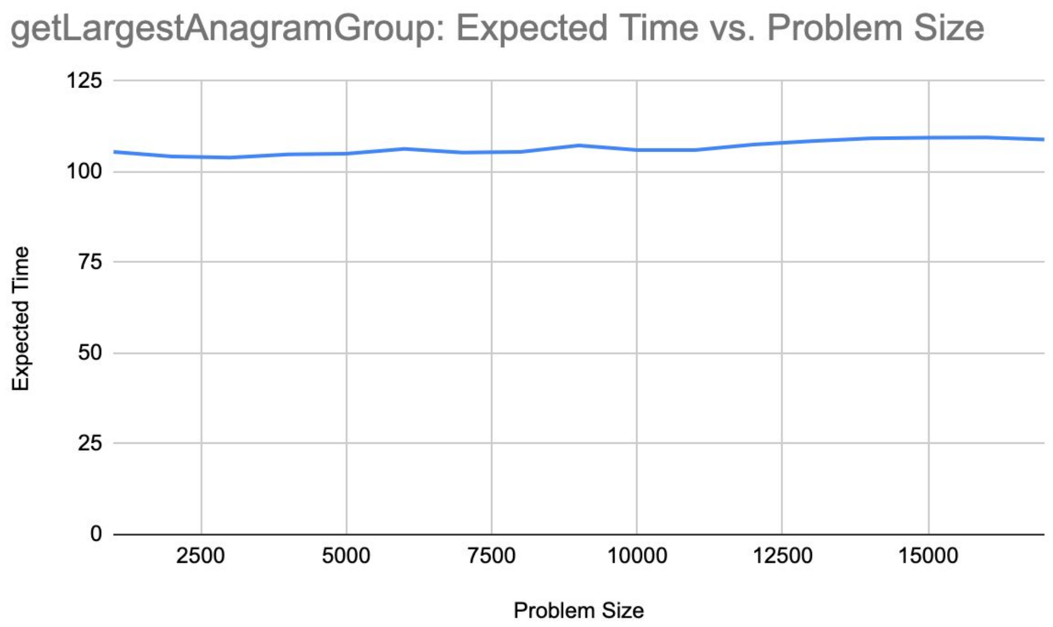


The expected times, using the "CheckAnalysis" technique, of the *areAnagrams* method also checks out since the numbers converge to a positive number which indicates that the empirically observed running times match the predicted running times.

5. In order to collect the data, I plotted the running time of the *areAnagramsGroup* method for problem sizes 1,000 to 17,000 by steps of 1,000. The big O of the *areAnagramsGroup* method should be O(n^2), n being the input size, because our *areAnagramsGroup* method makes use of the *insertionSort* method which uses insertion sort.

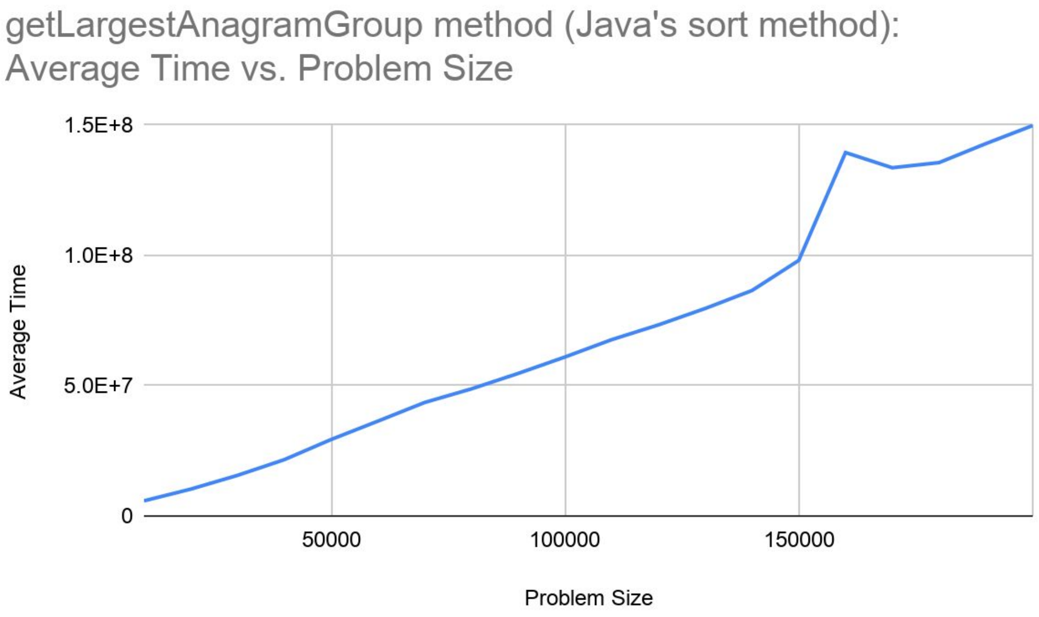


The growth rate of the plotted running times matches the Big-O behavior we predicted.

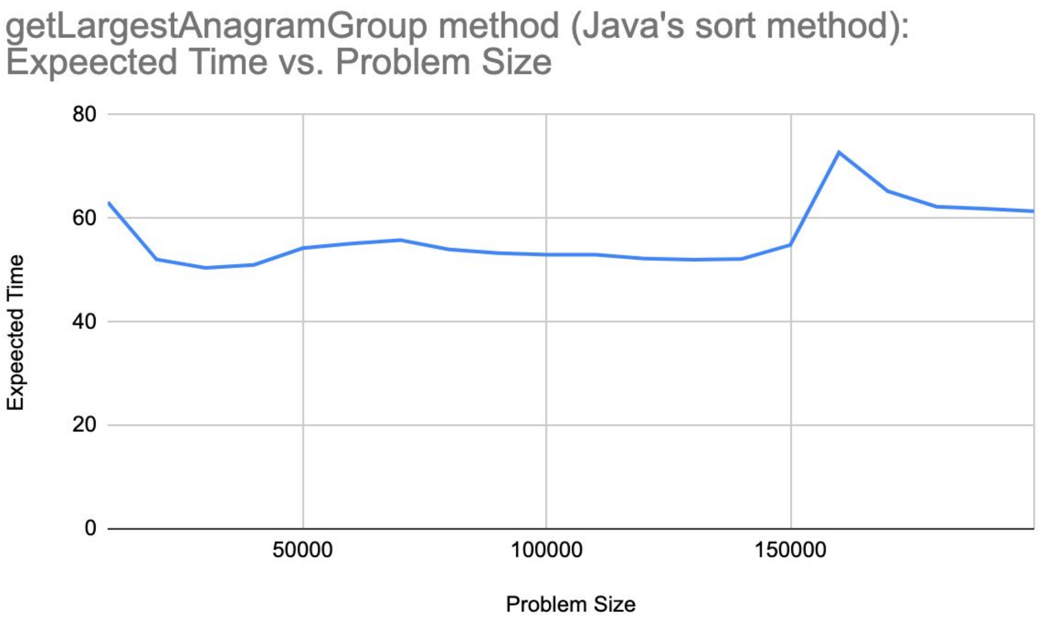


The expected times, using the "CheckAnalysis" technique, of the *areAnagramsGroup* method also checks out since the numbers converge to a positive number which indicates that the empirically observed running times match the predicted running times.

6. In order to collect the data, I plotted the running time of the *areAnagramsGroup* method that uses Java’s sort method for problem sizes 10,000 to 200,000 by steps of 10000. The big O of the *areAnagramsGroup* method that uses Java’s sort method should be O(nlogn), n being the input size, because our *areAnagramsGroup* method makes use of Java's sort method which uses merge sort.



The growth rate of the plotted running times matches the Big-O behavior we predicted.



The expected times, using the "CheckAnalysis" technique, of the *areAnagramsGroup* method that uses Java’s sort method also checks out since the numbers converge to a positive number which indicates that the empirically observed running times match the predicted running times. Compared to using insertion sort, it seems that Java’s sort method, which uses merge sort, is faster, and this makes sense because the runtime complexity of O(n^2) is much more taxing than O(nlogn) as input sizes increase.