Author: Zainab Hussein

Date: 2-2-2017

Unit Testing and Experimentation:

Analysis of Time Complexities of Add and Sort Methods on an ArrayList of Integers

Introduction

This lab aims to find the time complexity on an ArrayList data structure as it relates to inserting an unsorted element to the back versus front of the list, inserting while sorting versus sorting using insertion sort, and time difference in sorting a sorted versus unsorted list using the insertion sort method. An assumption made here is for the addSort method, I interpreted that as using the abstract class list sort method while adding an element to the back of the list.

Approach

There are two classes, the RandomIntegerContainer and ExperimentController classes whose method structures were defined in the lab manual given. The addToFront, addToBack and toArray methods were as specified in the API[1].The addSorted method utilized the sort method in the abstract class List, whereas the insertion sort algorithm was an adaptation of the class textbook[2]. In general, the RandomIntegerContainer class contains various add, sort and ArrayList to Array methods which are used in the ExperimentController class containing a main method, and various timing methods for the length of time taken by each of the methods in the latter class. A test class for the RandomIntegerContainer exists and functions properly to ensure all methods work as expected.

Methods

For credible results, the ExperimentController class is extensively examined with 9 different seeds for a given number of items. The average time is calculated for each number of items from the 9 times given by each seed. Both the seed and number of items range in an interval of a thousand, 1000 to 9000. The choice to use this wide range of inputs is to increase the probability of accuracy per time output. The outputs, timeAddToFront, timeAddToBack, timeAddSorted, timeSortOfUnsortedList and timeSortOfSortedList are all plotted against number of inputs, and analysis is done in the following section.

Data and Analysis

The graphs below are used to examine and analyze the time complexities of the various methods tested from the RandomIntegerContainer class.

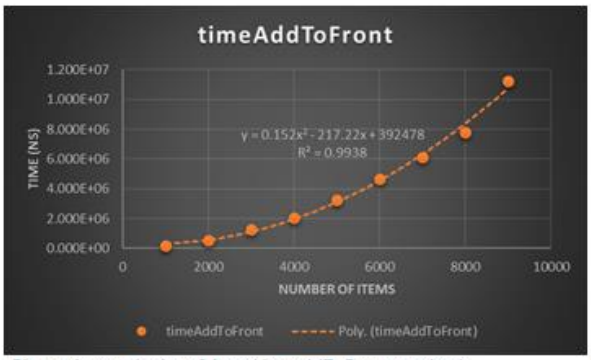

Figure 1 complexity of ArrayList addToFront method

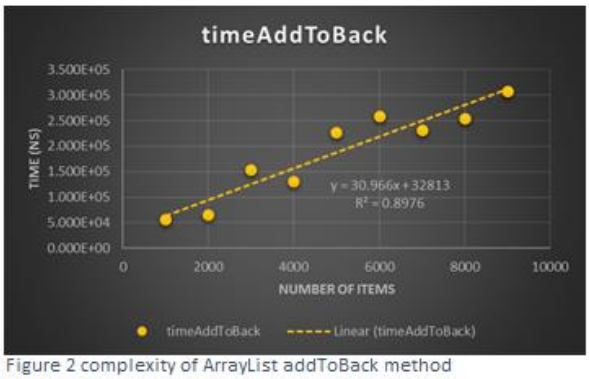

Figure 2 complexity of ArrayList addToBack method

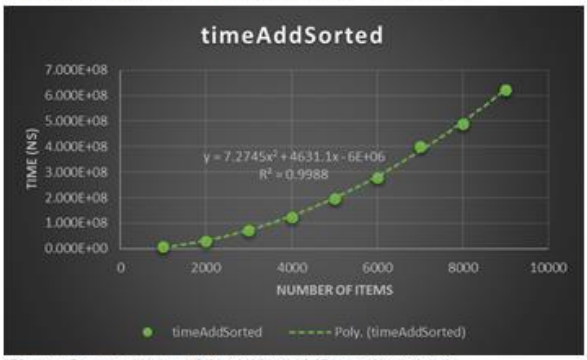

Figure 3 complexity of ArrayList addSorted method

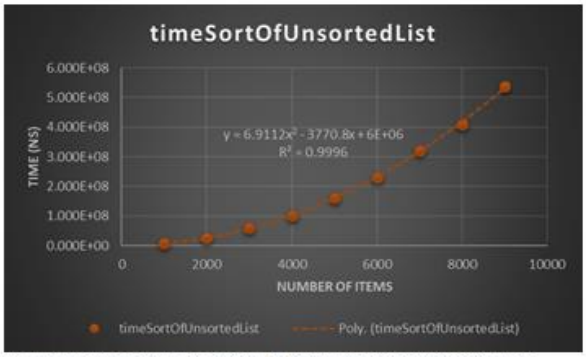

Figure 4 complexity of ArrayList insertionSort method on an unsorted list

The timeAddToFront shown in figure 1, timeAddToBack in figure 3 and timeSortOfUnsortedList in figure 4 fit to a quadratic curve suggesting a complexity of $O(N^2)$ whereas timeAddToBack in figure 2 and timeSortOfUnsortedList in figure 5 fit a linear graph suggesting a complexity of O(N). All these individual complexities are compared alongside one another in figures 6 to 8 below.

Comparison of the add methods in figure 6 show the addToBack being fairly constant as expected, even though the individual analysis gives it a linear time complexity. Both addSorted and addToFront show a quadratic complexity, with the addSorted having a much bigger coefficient constant than addToFront. This is expected as the addSorted adds and sorts the elements in the arraylist at the same time.

Comparison of sorting of unsorted elements using the addSorted and insertionSort methods in figure 7 show both methods having a time complexity of a quadratic nature. They start out similar for the first 3000 items sorted, then the addSorted method sorts slightly faster than insertion Sort.

Comparison of sorting a sorted versus unsorted list using insertionSort shown in figure 8 suggest sorting through a sorted list is much faster in a fairly linear time, which is the expected best case sort for insertion sort. The sorting through an unsorted list shows the average time complexity for insertionSort as expected

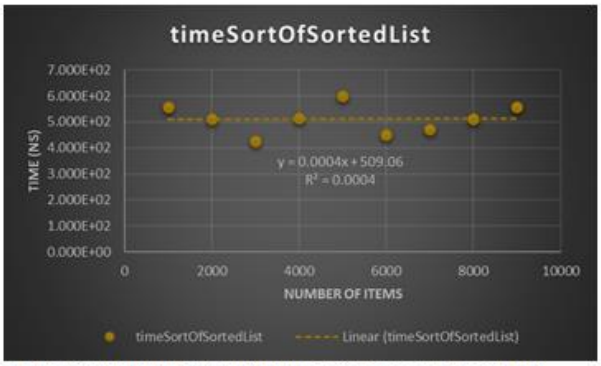

Figure 5 complexity of ArrayList insertionSort on a sorted list

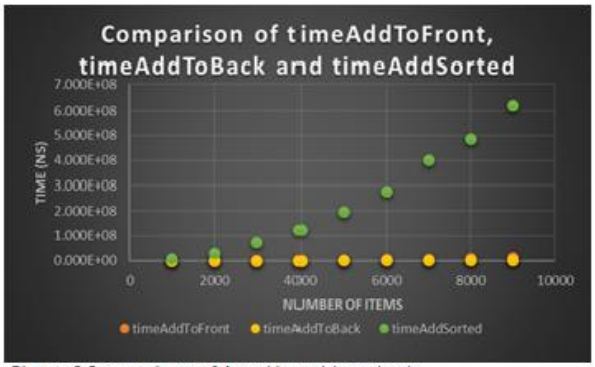

Figure 6 Comparison of ArrayList add methods

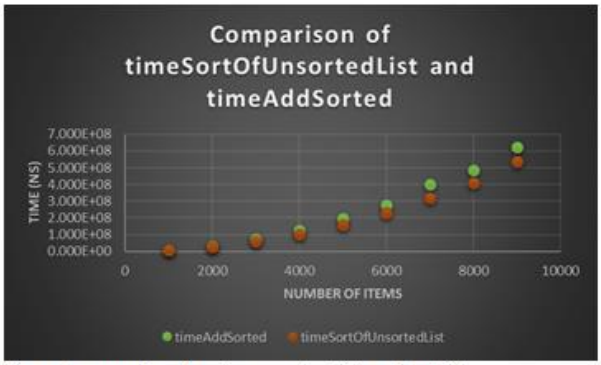

Figure 7 comparison of sorting unsorted lists on ArrayList

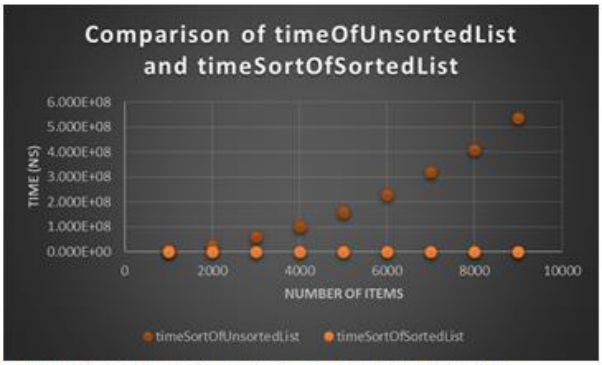

Figure 8 Comparison of sorting a sorted and unsorted list

Conclusion

The Add and sort methods worked as expected except for some discrepancies in time complexity of the addToFront and addToBack methods. The timeAddToBack method is a linear one from the experiment, but a constant time complexity was expected. The timeAddToFront complexity expected was O(N) but there was a quadratic one instead. These could be due to not running enough inputs or unexpected seed interaction with the various methods or other programs running on my pc when the experiment was run and data collected.

References

[1]List, ArrayList and Array API

[2] Data Structures and Problem solving using Java

Appendix

Average data calculated from the .csv file collected from experimentation.

numberItems	timeAddToFront	timeAddToBack	timeAddSorted	timeSortOfUnsortedList	timeSortOfSortedList
1000	1.986E+05	5.628E+04	8.125E+06	7.479E+06	5.559E+02
2000	5.478E+05	6.547E+04	3.097E+07	2.586E+07	5.130E+02
3000	1.252E+06	1.537E+05	7.293E+07	5.969E+07	4.277E+02
4000	2.050E+06	1.317E+05	1.245E+08	1.025E+08	5.134E+02
5000	3.246E+06	2.277E+05	1.962E+08	1.609E+08	5.987E+02
6000	4.649E+06	2.592E+05	2.797E+08	2.297E+08	4.503E+02
7000	6.111E+06	2.324E+05	4.010E+08	3.211E+08	4.707E+02
8000	7.800E+06	2.545E+05	4.880E+08	4.100E+08	5.131E+02
9000	1.122E+07	3.077E+05	6.231E+08	5.371E+08	5.558E+02