Author: Zainab Hussein

Date: 2-13-2017

Inheritance and Polymorphism:

Analysis of Time Complexities of Add, Sort and Search Methods between an ArrayList and LinkedList of Integers

Introduction

This lab aims to show inheritance of the abstract class' methods by the child classes ArrayListIntegerContainer and LinkedListIntegerContainer, and polymorphism as the child class references the abstract parent class. It also aims to compare the time complexity of an ArrayList and LinkedList 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, time difference in sorting a sorted versus unsorted list using the insertion sort and linear and binary search methods.

Approach

There are four classes, the abstract IntegerContainer class, ArrayListIntegerContainer and LinkedListIntegerContainer classes and ExperimentController class whose method structures were defined in the lab manual given. The addToFront, addToBack, toArray and linearSearch methods were as specified in the API[1].The addSorted method utilized the sort method in the abstract class List, whereas the insertion sort and binary search algorithm were an adaptation of the class textbook[2]. In general, the abstract IntegerContainer class contains various add, sort, search and toArray 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 ArrayListIntegerContainer and LinkedListIntegerContainer exists and functions properly to ensure all methods work as expected.

Methods

For credible results, the ExperimentController class is extensively examined with 5 different seeds for a given number of items. The average time is calculated for each number of items from the 8 times given by each seed. Both the seed and number of items range in an interval of a thousand, 1000 to 8000. The choice to use this wide range of inputs is to increase the probability of accuracy per time output. The outputs are ArrayList and LinkedList timeAddToFront, timeAddToBack, timeAddSorted, timeSortOfUnsortedList, timeSortOfSortedList, linearSearch and binarySearch 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 child ArrayListIntegerContainer and LinkedListIntegerContainer, inherited from the abstract IntegerContainer class.

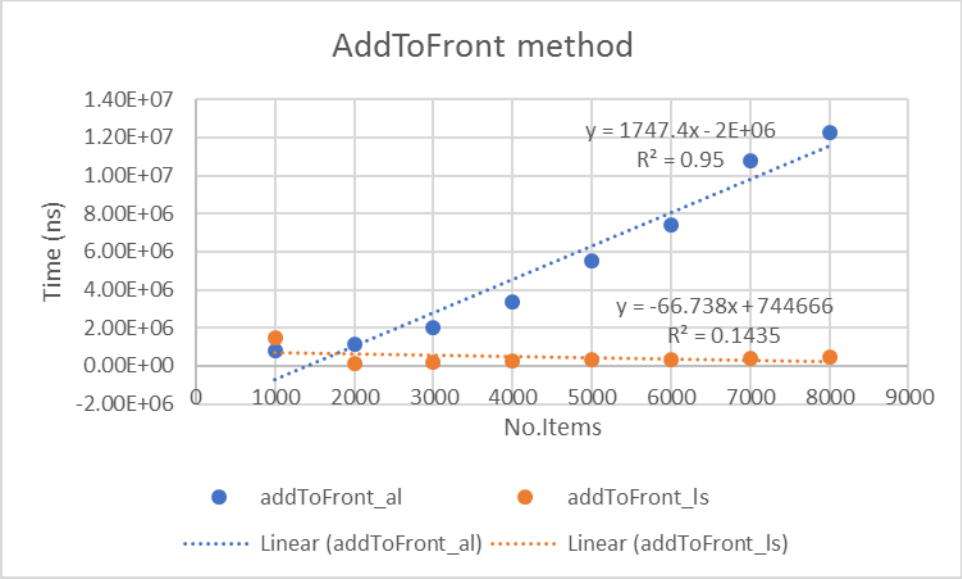

Figure 1 addToFront comparison

Time addToFront for ArrayList is greater than LinkedList, both have a time complexity of O(n). The linkedlist fit has a very small R squared suggesting the coefficient should be higher.

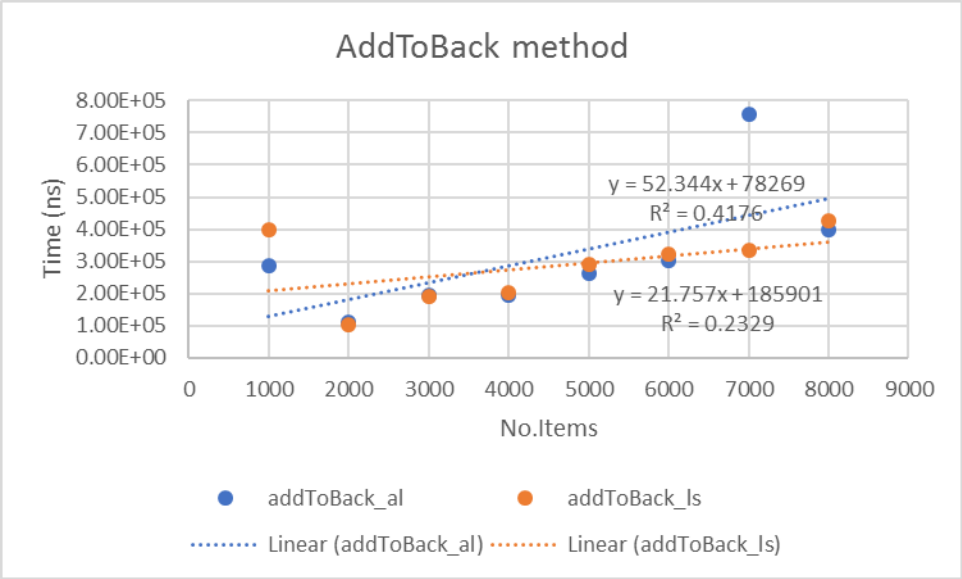

Figure 2 addToBack comparison

Time addToBack for ArrayList starts of lower than LinkedList then after around 3600 items, both have a time complexity of O(n). The fit for both have a very small R squared suggesting the coefficient are very small, which is consistent with the expected O(1) time complexity.

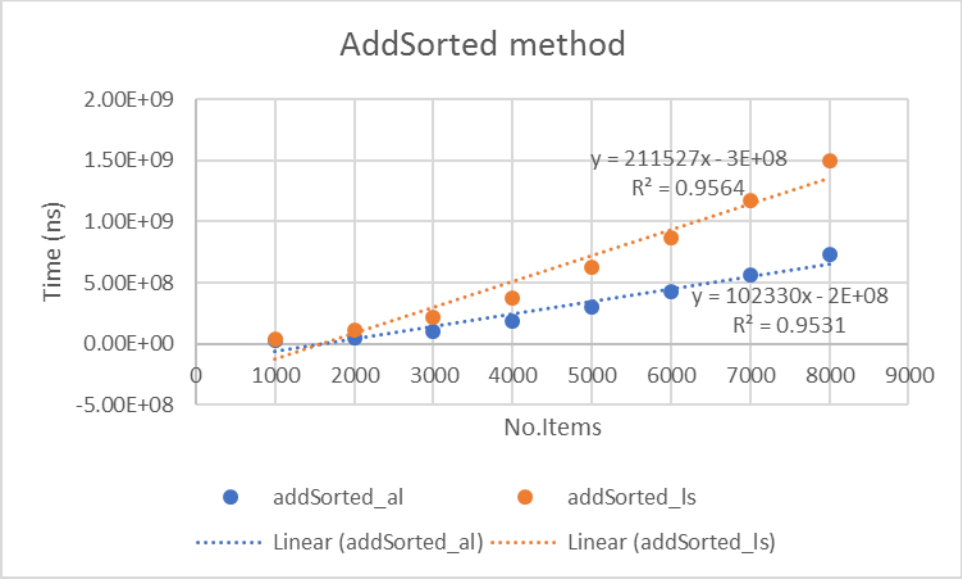

Figure 3 addSorted comparison

Time addSorted for ArrayList and LinkedList both have a time complexity of O(n), consistent with expected complexity. The fit for both have a very high R squared suggesting assuring the correctness of the fits.

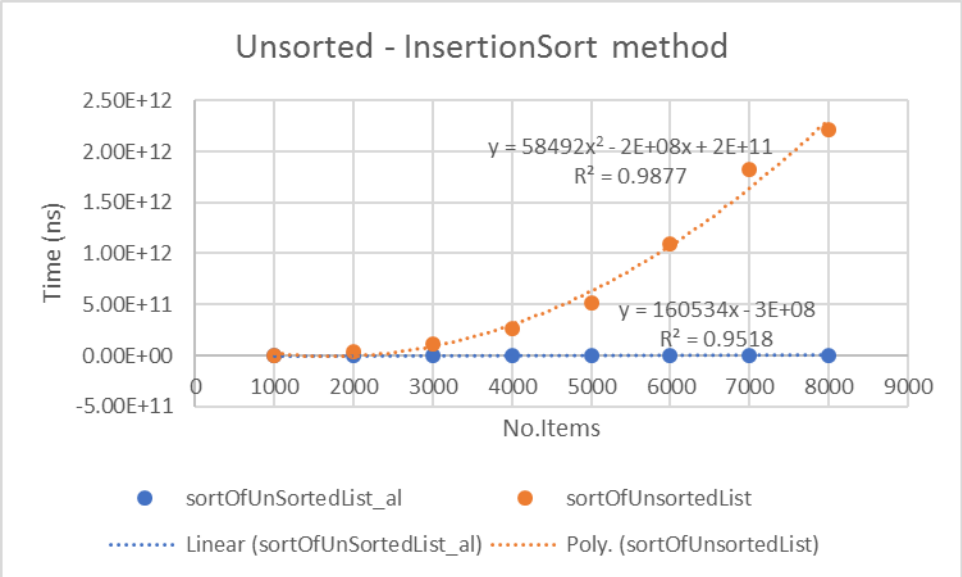

Figure 4 unsorted_InsertionSort comparison

Time sortOfUnsortedList for ArrayList is higher than LinkedList. The fit for both have a very high R squared suggesting assuring the correctness of the fits, attributing a $O(n^2)$ to arraylist and O(n) on linkedlist, expected insertion sort is $O(n^2)$ average case and O(n) best case.

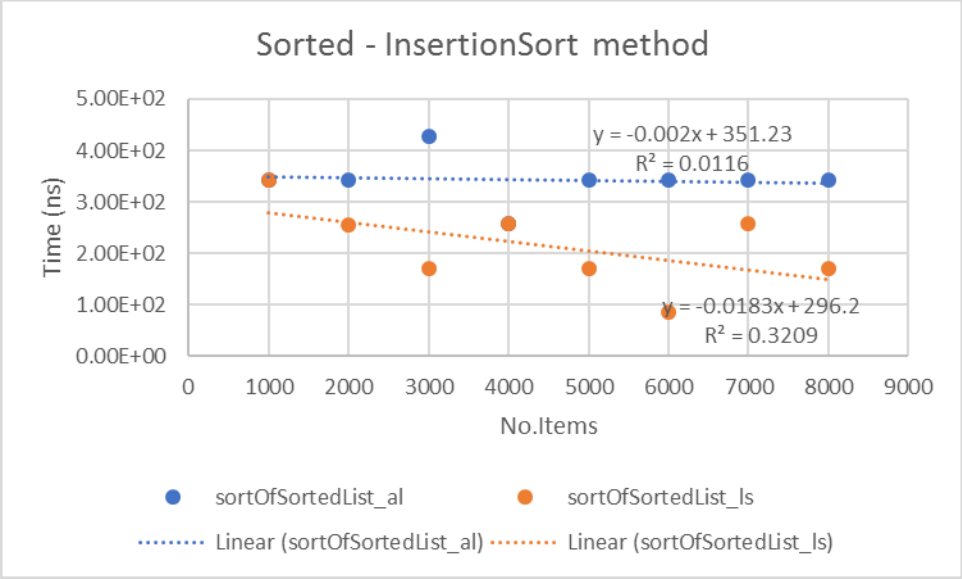

Figure 5 sorted_InsertionSort comparison

Time sortOfSortedList for ArrayList is higher than LinkedList, both have a very small coefficient translating to a time complexity of O(n). The fit for both have a very small R squared suggesting the coefficient are very small, which is consistent with the expected O(1) time complexity.

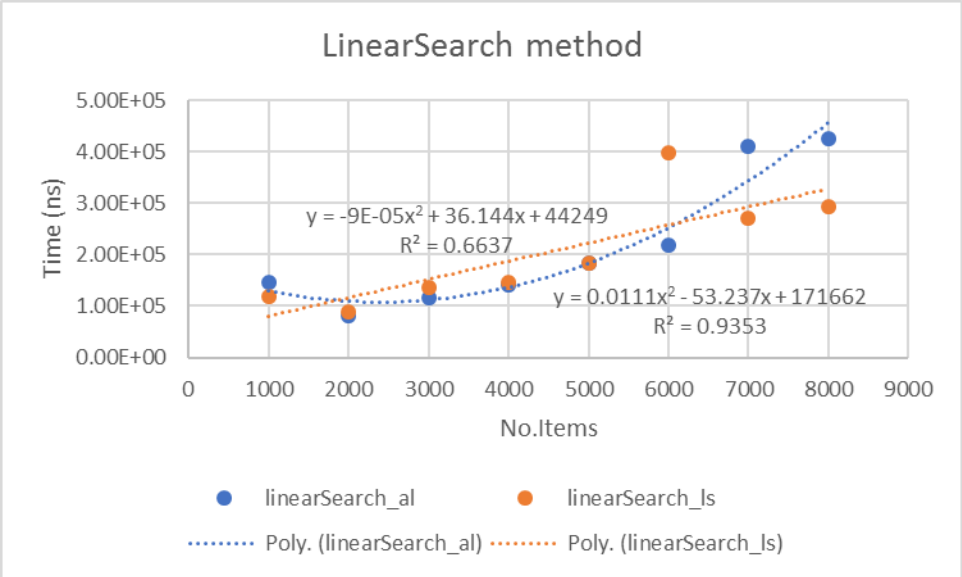

Figure 6 linearSearch comparison

Time linearSearch for ArrayList has a time complexity of is $O(n^2)$ and has a high R squared value suggesting correctness of the fit. The LinkedList fit of O(n) is not consistent with expected $O(n^2)$, this discrepancy is shown by low R squared.

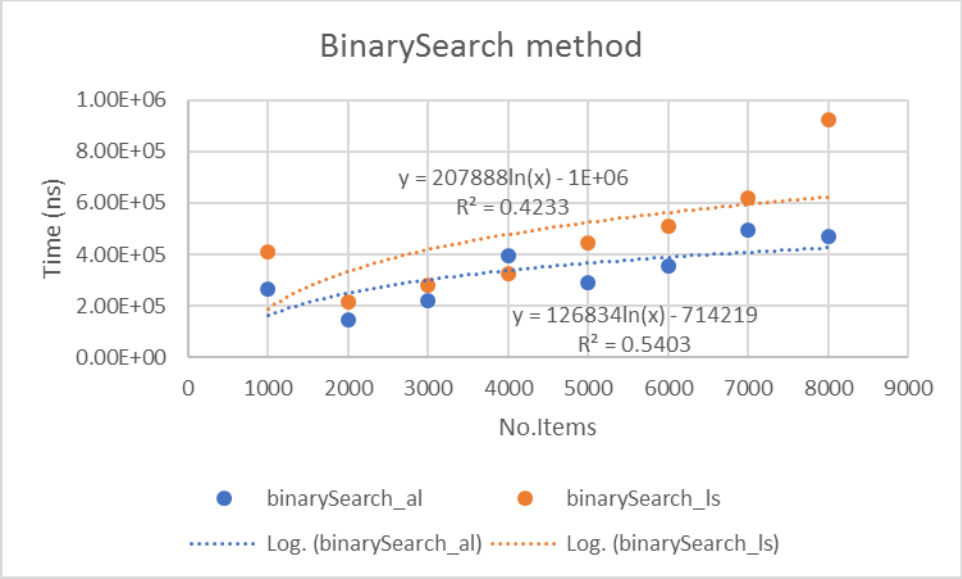

Figure 7 binarySearch comparison

Time binarySearch for ArrayList and LinkedList both have a time complexity of O(logn), consistent with expected complexity. However, the linkedlist binary search is higher than arraylist, the fit for both have a relatively low R squared suggesting there could be better fits.

Conclusion

The add, sort and search methods for the arraylist and linkedlist results worked as expected, with an overall O(n) complexity for the addToFront and addSorted methods, O(1) for addToBack, O(n^2) for unsorted insertion sort, O(1) for sorted insertion sort, O(n^2) for linear search and O(logn) for binary search methods.

References

[1] AbstractList, ArrayList and LinkedList API

[2] Data Structures and Problem solving using Java

Appendix

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

	No.Items	ArrayList	LinkedList
AddToFront	1000	7.97E+05	1.48E+06
	2000	1.12E+06	1.30E+05
	3000	2.05E+06	2.05E+05
	4000	3.37E+06	2.64E+05
	5000	5.50E+06	3.29E+05
	6000	7.40E+06	3.18E+05
	7000	1.08E+07	3.78E+05
	8000	1.23E+07	4.48E+05
AddToBack	1000	2.87E+05	3.99E+05
	2000	1.13E+05	1.04E+05
	3000	1.96E+05	1.91E+05
	4000	1.93E+05	2.02E+05
	5000	2.64E+05	2.92E+05
	6000	3.04E+05	3.21E+05
	7000	7.57E+05	3.33E+05
	8000	3.98E+05	4.28E+05
AddSorted	1000	2.45E+07	3.67E+07
	2000	4.96E+07	1.11E+08
	3000	1.07E+08	2.19E+08
	4000	1.83E+08	3.72E+08
	5000	3.00E+08	6.31E+08
	6000	4.24E+08	8.73E+08
	7000	5.67E+08	1.17E+09
	8000	7.31E+08	1.50E+09
SortOfUnsorted	1000	2.83E+07	4.61E+09
	2000	7.67E+07	3.61E+10
	3000	1.59E+08	1.18E+11
	4000	2.71E+08	2.64E+11
	5000	4.81E+08	5.21E+11
	6000	6.48E+08	1.09E+12
	7000	8.71E+08	1.83E+12
	8000	1.15E+09	2.21E+12
SortOfSorted	1000	3.42E+02	3.42E+02
	2000	3.42E+02	2.56E+02
	3000	4.28E+02	1.71E+02
	4000	2.57E+02	2.57E+02
	5000	3.42E+02	1.71E+02
	6000	3.42E+02	8.56E+01
	7000	3.42E+02	2.57E+02
	8000	3.42E+02	1.71E+02
LinearSearch	1000	1.47E+05	1.18E+05
	2000	8.24E+04	8.84E+04
	3000	1.16E+05	1.37E+05
	4000	1.42E+05	1.47E+05
	5000	1.83E+05	1.84E+05
	6000	2.19E+05	3.98E+05
	7000	4.11E+05	2.72E+05
	8000	4.24E+05	2.94E+05
BinarySearch	1000	2.66E+05	4.09E+05
	2000	1.45E+05	2.15E+05
	3000	2.21E+05	2.81E+05
	4000	3.94E+05	3.27E+05
	5000	2.92E+05	4.47E+05
	6000	3.57E+05	5.08E+05
	7000	4.96E+05	6.17E+05
	8000	4.68E+05	9.23E+05