Craig Lombardo

CS 150 Lab 2

Professor Liew

2/8/15

1. Introduction

The purpose, or goal, of this lab was to compare ArrayLists with LinkedLists in hopes of finding out which list is more beneficial for different applications. Part a introduces us to class inheritance and polymorphism, while part b asks us to implement our new found knowledge and extend those principles in order to compare search algorithms with list type. For this lab the problem included figuring out how to implement an AbstractList through the use of ArrayLists and LinkedLists and figuring out how to create methods that worked for both types. For this lab I assumed that the methods created not only worked for all List types, but also had the same complexity, I quickly found out I was wrong.

2. Approach:

For the purposes of this lab I have designed 6 classes for both, parts a and b. There is one parent IntContainer class which will contain the methods that may be implemented by its two children, as well as a protected AbstractList called data. The methods in the parent class for part a will allow the user to add to the front of the list, add at a sorted position or sort an unsorted list. The methods in the parent class for part b will allow its children to add to the front of a list or find the second largest number in a list. The children of the parent IntContainer class are ArrayListIntContainer and LinkedListIntContainer. It is important to note that in the constructor method of the child classes, the protected AbstractList data is instantiated in the form of an ArrayList or LinkedList, respectively. There is a ListExperimentController class, which handles all of the simulation and data output. The next class is Wheel, which generates pseudorandom numbers based on a seed value and an upper bound limit; numbers produced here will be within the range 1-upper. The final class in both situations is a test class, which will be used to ensure all methods function properly. Both part a and part b will be testing ArrayLists and LinkedLists only.

3. Methods

The setup of this program was broken down into different parts. A key aspect was the creation of methods that ran the experimental work. Each test was run in two stages, the first included a small test to ensure things worked as expected on a small scale, and then the second part was a more extensive test. The first test gave me a general sense of how long the program would take to run given the time taken for n elements, and the complexity of whatever it was that I was testing at that time. This time around, instead of trying to match a best-fit line, or quadratic to my data, I checked to ensure that my best-fit line matched the predetermined complexity. I decided to stick with the model I used from lab 2, in that my ListExperimentController classes had a main method which allows the user to specify what test to run, so not all tests must be run at the same time. This also gives the user, me in this case, the ability to easily change the number of trials executed. The parameters passed in, for both, followed the format of the first argument, args[0], corresponding to the test I wanted to run, and the second argument, args[1], corresponding to the number of tests to conduct. It is important to note that I did not put in exceptions that will determine whether or not a valid string was passed in for the first argument, nor a check for non-integer values passed in for the second argument.

4. Data and Analysis

This graph represents the arrayTimeAddFromFront test. The complexity of the addToFront method for ArrayLists is O(n), this is because to do so there is an element added to the front of the array, shifting n elements up one position. The graph shows complexity O(n^2) as it is data collected for n elements, thus O(n\*n). The R^2 value achieved for this data was 0.99973, which is a fairly conclusive correlation, assuming the sample size I have tested with is large enough.

This graph represents the arrayTimeAddSorted test. The complexity of the addSorted method for ArrayLists is O(n). The reason behind this is because the method is constructed in two parts, the first part adds the element to the end of the array (which is O(1)), the second part is a call to the insertionSort method. In it’s worst case scenario, insertion sort will have to move n elements up one position to make way for the element to be placed in the correct place. The graph shows complexity O(n^2) as it is data collected for n elements, thus O(n\*n). The R^2 value achieved for this data was 1, which is a conclusive correlation, assuming the sample size I have tested with is large enough.

This graph represents the linkedTimeAddFromFront test. The complexity of the addToFront method for LinkedLists is O(1). Adding an element to the front of a LinkedList is constant time; however, the graph shows complexity O(n) as it is data collected for n elements, thus O(1\*n). The R^2 value achieved for this data was 0.89108, which is not a very a conclusive correlation; however, a linear fit matches the trend the best. There are a few fluctuations, which are very high above the best-fit line; however, there are also many fluctuations below the line, which even things out. Although the correlation it not very high, the trend of the data is linear.

This graph represents the linkedTimeAddSorted test. The complexity of the addSorted method for LinkedLists is O(n^3). The reason behind this is because first we add the number to the end of the list O(1) then we must do some setting and getting of the values. For a LinkedList there is a compexity O(n) because in it’s worst case it must traverse through the entire list to find the value it needs/the node it is trying to set. In the insertion sort method I have created (which is used to sort the array after adding it to the end) there is a set action performed within a while loop, which worst case scenario means O(n^2). This entire process is encompassed in a for loop which means that is another complexity of O(n). Total everything up and we get a complexity of O(n^3). The graph shows complexity O(n^4) as it is data collected for n elements, thus O(n^3\*n). The R^2 value achieved for this data was 1, which is a conclusive correlation, assuming the sample size I have tested with is large enough.

This graph represents the arrayTimeFindSecond test. The complexity of the findSecondLargest method for ArrayLists should be O(n). Finding an element in an ArrayList is constant time; however, that is for n elements. The graph shows complexity O(n) as it is matching the best-fit line; however, it is possible more testing was needed or I incorrectly calculated complexity. The R^2 value achieved for this data was 0.96188, which is not a very a conclusive correlation; however, a linear fit matches the trend the best. It is important to now the major fluctuation at the beginning of the data; the data point seems to be an outlier however, every test conducted had the same outlier which led me to believe it was not as strange as initially anticipated. Although the correlation it not very high, the trend of the data is mostly linear.

This graph represents the linkedTimeFindSecond test. The complexity of the findSecondLargest method for LinkedLists is O(n^2). The reason behind this is that we must get a few values from the list. For a LinkedList getting the value of a node is of complexity O(n) because in it’s worst case we must traverse through the entire list to find the value it needed. Since the get method is nested in a for loop we get complexity O(n^2); however, the graph shows complexity O(n^3) as it is data collected for n elements, thus O(n^2\*n). The R^2 value achieved for this data was 0.99262, which is a conclusive correlation, assuming the sample size I have tested with is large enough.

5. Conclusion

Through my experimental data I have determined that ArrayLists are superior when trying to traverse an array, such as to sort or compare data; however, LinkedLists are superior when trying to add elements to the front of an array. Should I want to make a queue, or something of this nature, I would most likely use a queue; however, if I needed to run computational processes I would use an ArrayList as they are superior in getting setting values.

6. References

<http://www.mycstutorials.com/articles/sorting/insertionsort>

<http://www.cs.lafayette.edu/~liew/courses/cs150/lab/labs/lab03g/>

<http://www.cs.lafayette.edu/~liew/courses/cs150/lab/labs/lab03f/>

Data Structures and Problem Solving Using Java by Mark Allen Weiss