Craig Lombardo

Lab 6

3/8/15

1. Introduction:

In today’s society, efficiency and speed are some of the most important aspects of any design. The purpose of this lab was to learn about two new sorting algorithms, Shell Sort and Insertion Sort and use them to sort data stored in two different containers. This lab will also teach us about inheritance and using it to create two parent Sort classes, each with two children. It was important to make a good set of interface methods, which will allow my class to be both flexible and testable. For this lab we are trying to figure out which sorting algorithm is faster for sorting, Shell Sort or Insertion Sort. The sorting of both is compared based on their performances in sorting ArrayLists and LinkedLists. For this lab I assumed that the sorting algorithms had roughly the same complexity and would take roughly the same amount of time to finish, I was wrong (on part of that).

2. Approach:

For this lab I made the two parent classes, which inherit the SortedList interface and each have two children. For each sorting parent class (Shell and Insertion Sort) I made an ArrayList class and a LinkedList class. The reasoning behind this was to make testing the different sorts on different data structures easier and more efficient. This was also done to reduce the amount of repeated code as through the use of inheritance and polymorphism between parent and child, I could make a set of rules (interface) and methods that a parent has, and the child will automatically conform to these rules and will have these methods. The parent class was also designed to have AbstractLists, so the parent classes cannot be instantiated and must be one of their children. There is also an ExperimentController class which handles all of the testing of the methods found within the parent Classes. Although the parent classes only differ by the sorting algorithm they use, given that the only tests we are conducting are based upon sorting algorithm and data structure type, the other methods are irrelevant. There is also a Wheel class used that will generate pseudorandom numbers which will be used in the data collection process by filling the different list types with random numbers before sorting them.

3. Methods:

For the purposes of this lab I decided to continue with the same structure as previous labs. The setup of this program was broken down into different parts. A key aspect was the creation of methods that ran my experimental work for me. Each test was run in two stages, the first included a small test to ensure things worked as expected on a small scale and did not give me any errors, 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. This small test also helped me get a general sense on what to expect for my more extensive tests. These tests proved to be extremely effective as I very quickly learned that the time for the LinkedList was far greater than that of the ArrayList for both sorting algorithms. For the purposes of this lab I tested everything multiple times with different seeds. The experiment is set up to use the seed to generate random numbers and conduct the test 5 times, averaging the results to present me with data that is relatively consistent, or at least I can hope. The tests are designed to minimize the amount that is changed between tests to ensure that I receive conclusive results; the only part of the experiment that really changes is the seed value used and the pseudorandom numbers that are inserted into the list. Beyond that everything else is the same, from the number of elements added to the list, to the sorting algorithm used, for that specific sorting algorithm. The tests were run through the ExperimentController main method. I decided to also stick with my convention of allowing the user to choose which test they want to conduct; this is done so the user has more flexibility in testing and does not need to conduct all tests at once. I have decided however to limit the number of parameters passed in, I have switched to a general rule of ten data points covering a large range of data values rather than having a crazy amount of data points with a very low interval. For the selection of test to run, args[0] is the test type, either insertArray, insertLinked, shellArray or shellLinked. It is important to note that I have not put a system in place to ensure that there were no incorrect inputs passed in. Should an invalid argument be passed, i.e. an argument that is not a String listed above, a java error will occur.

4. Data and Analysis:

This graph represents the time taken to sort n random elements in a LinkedList using insertion sort (Average time(ms) VS Number of). The complexity of this graph is O(n^2).

This graph represents the time taken to sort n random elements in an ArrayList using insertion sort (Average time(ms) VS Number of). The complexity of this graph is O(n^2).

This graph represents the time taken to sort n random elements in a LinkedList using shell sort (Average time(ms) VS Number of). The complexity of this graph is O(n^2).

This graph represents the time taken to sort n random elements in an ArrayList using shell sort (Average time(ms) VS Number of). The complexity of this graph is O(n^2).

This graph is a plot of Average Time(ms) VS Number Of elements for both LinkedLists. The purpose of this plot is to compare insertion sort and shell sort in regards to their performance with a LinkedList. It can be seen that shell sort is much faster than insertion sort. Insertion sort time gets to about half of shell sorts max time with a tenth of the elements. Although they have the same complexity, it is very easy to see the difference between the two.

This graph is a plot of Average Time(ms) VS Number Of elements for insertion sort on LinkedLists and ArrayLists. The purpose of this plot is to compare insertion sort and in regards to it’s performance on different lists. It can be seen that ArrayLists are much faster than LinkedLists. The max LinkedList sort time was about a quarter of the max ArrayList sort time and that was for a significantly smaller sample size, thus ArrayList insertion sort is much faster.

This graph is a plot of Average Time(ms) VS Number Of elements for shell sort on LinkedLists and ArrayLists. The purpose of this plot is to compare shell sort in regards to it’s performance on different lists. It can be seen that ArrayLists are much faster than LinkedLists. The max LinkedList sort time was about half the max ArrayList sort time and that was for a significantly smaller sample size, thus ArrayList shell sort is much faster.

This graph is a plot of Average Time(ms) VS Number Of elements for shell sort and insertion sort on ArrayLists. The purpose of this plot is to compare the two sorts in regards to their performance on ArrayLists. It can be seen that shell sort is much faster than insertion sort. The max shell sort time was about 3/4 the max insertion sort time and that was for a significantly larger sample size, thus shell sort is much faster.

5. Conclusion

It can be seen through the experimental data that shell sort is much faster than insertion sort. In both cases, ArrayLists and LinkedLists, shell sort was much faster than its insertion sort counterpart. Generally speaking ArrayLists were faster, which is expected as getting and setting values in ArrayLists is O(1), whereas for LinkedLists, getting and setting values has a worst case complexity of O(n). Although all sort types resulted in O(n^2), the coefficients in front of the ArrayList tests were much smaller than those of the LinkedList tests. The final conclusion is that shell sort on ArrayLists is the fastest of the 4 options.

6. References:

Gootooru, Nataraja. "Program: Write a Program for Insertion Sort in Java." *Write a Program for Insertion Sort in Java.* Nataraja Gootooru, 2015. Web. 03 Mar. 2015.

"Java Platform SE 7." *Java Platform SE 7*. N.p., n.d. Web. 06 Mar. 2015.

"Sorting Algorithms/Shell Sort." *Sorting Algorithms/Shell Sort*. RosettaCode.org, 4 Feb. 2015. Web. 3 Mar. 2015.

Weiss, Mark Allen. *Data Structures & Problem Solving Using Java*. Boston: Pearson/Addison Wesley, 2010. Print.

http://www.cs.lafayette.edu/~liew/courses/cs150/lab/labs/lab05g/