Xiaoshi Wang

CS150 Lab

February 6, 2017

Lab Report 1

Introduction

In this lab, our goal is to compare the time used for different ways to build up a series of numbers with various sizes and find some properties about those. There are three specific comparisons we want to talk about:

1. Compare the average run time of inserting from the front, back and sorted for different amounts of data,
2. Compare the average run time of adding the elements and the sorting the unsorted list vs. inserting the entries using the *addSorted* for different amount of data,
3. Compare the average run of sorting an unsorted array and sorting a sorted array for different amount of data.

Approach

To record and compare the run times of different methods, we need to first write a class named *RandomIntegerContainer* to define each way that we want to use to build up a series of numbers. We have totally four basic steps to build up a series of numbers in this experiment: insert numbers one by one from the front, back, sorted (which means inserting a certain number in its right place from small to large), and sort a list of numbers by selection sort. In *RandomIntegerContainer*, I first construct a ArrayList as a constructor. Then, I write four methods to achieve the four basic steps. Finally, I set up a method to return our ArrayList.

Before we begin write code for time comparison, we need to first make sure that *RandomIntegerContainer* works properly. Thus, I write a unit test. Four to five tests should run for each method. If there are no errors for all tests, we can write our second class, *ExperimentController*.

In *ExperimentController*, in each method, I record the average times of a certain type with different seeds of random numbers. Then, instead of out print all the values in the terminal, I create a new file and write all information in this file. By plugging varied inputs, the size of our sequences can be changed.

Methods

Most of my experimental setup is in *ExperimentController*. Since there are variances between random numbers with different seeds, I calculate the average of five run times with five different seeds. Inside each method which calculates the run time, I write a for loop to increase seeds by 100 for each time. Then, I add the time I get in each time to local variable sum and return sum/5 finally. Then, whenever I call any of the methods, I get an average run time of five different seeds as a return.

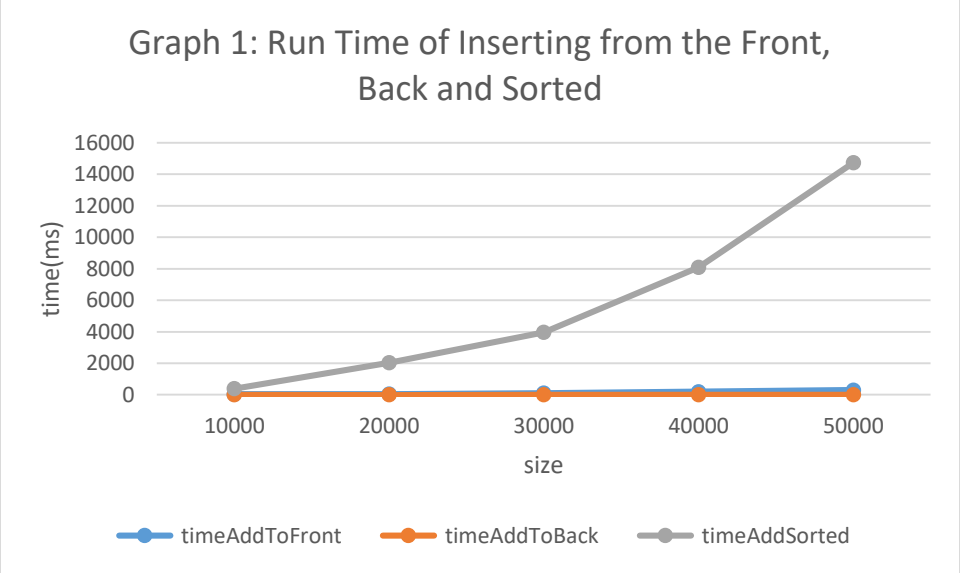
We also need to record run times for varied sizes. In main method, I use a for loop to write five lines of numbers. In each line, I write all six time down and then increase the input size by 10000. Then, I have the average run times of size 10000, 20000, 30000, 40000 and 50000.

Data and Analysis

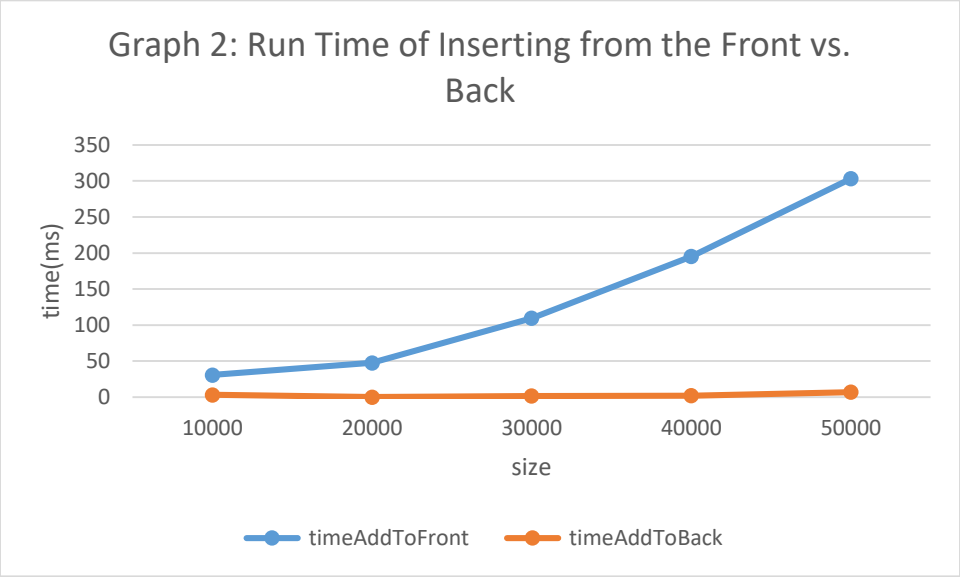
Table 1 the excel file I write in the main method of *ExperimentController*.

	timeAddToFront	timeAddToBack	timeAddSorted	timeAddAndSort	timeSortofUnsortedList	timeSortofSortedList
10000	30.6	3.2	390.4	766.2	654.2	0
20000	47.6	0	2036.4	3557.4	2868	4.8
30000	109.6	1.6	3969.6	6284.4	6172	1.8
40000	195.2	2	8090	12081.8	12144.4	1.2
50000	303.2	6.8	14725.8	19225.4	18557.8	8.8

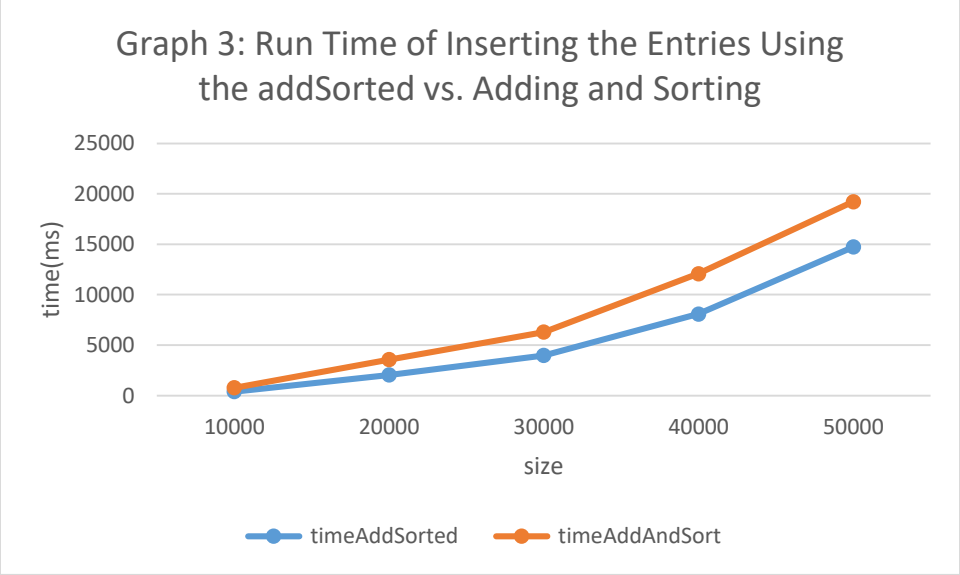
Table 1



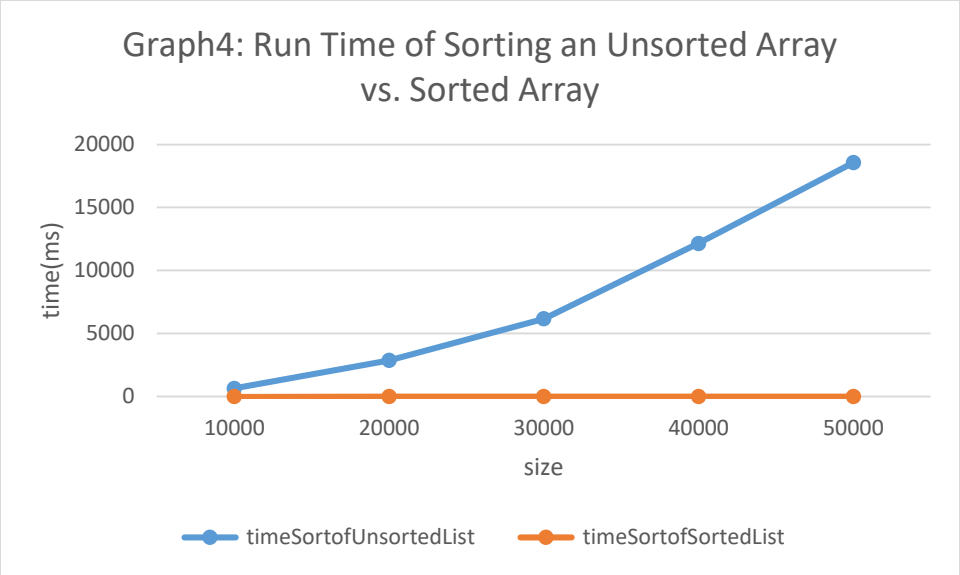
Graph 1 illustrates the run time of inserting form the front, back and sorted. We can conclude from the graph that almost all the run times are increasing as the size increases. But the increasing rate of time for add sorted is faster than any of the other two, which is quadratic.



Since the difference between time for adding to front and back is too small in graph 1, I add graph 2 for better analyzing front and back. The time for back is always smaller than back, and the run time for front is almost linear but that for back is constant.



For run time of inserting by addSorted and adding then sorting, time for adding then sorting is always larger than addSorted regardless the size of inputs. However, the increasing rates of those two are similar: their time both increase when the size increases.



The last comparison is between sorting an unsorted array and sorted array. Clearly, the time for unsorted array is much larger than that for sorted one. Also, the time for sorted array is almost constant as the size of input increases.

Conclusion

Run times of building up a sequence depends on both the size and methods we use. Adding elements one by one to the back is the fastest way for all sizes. Although addSorted is the slowest comparing with adding back and front, it is faster than the method which first adding then sorting. Also, the time used for sorting an unsorted array is much longer that for sorted array.

References

Kölling, Michael, and Mærsk Institute. *Unit Testing in BlueJ*. 1.0st ed. N.p.: n.p., n.d. Web. 6 Feb. 2017. <http://www.bluej.org/tutorial/testing-tutorial.pdf>.
"ArrayList (Java Platform SE 7)." *ArrayList (Java Platform SE 7)*. N.p., n.d. Web. 06 Feb. 2017.
Sadovnik, Amir. "Objective." *CS150: Lab 2*. N.p., n.d. Web. 06 Feb. 2017.
Weiss, Mark Allen. *Data structures and problem solving using Java*. Reading, MA: Addison-Wesley, 2002. Print.