Xiaoshi Wang

CS150 Lab

February 13, 2017

**Lab Report 2**

**Introduction**

In this lab, our goal is to compare running time of seven different tasks used by ArrayList and *LinkedList*. Our seven tasks are: *addToFront*, *addToBack*, *addSorted*, *InsertionSort* (both sorted and unsorted arrays), *linearSearch* and *binarySearch*. We want to build up a series of numbers with various sizes and find some properties about those. While comparing the running time between *ArrayList* and *LinkedList,* we should ensure the way we proceed each task should be exactly the same.

**Approach**

To record and compare the run times of different methods between *ArrayList* and *LinkedList*, we need to first write two class named *ArrayListIntegerContainer* and *LinkedListIntegerContainer* to define each way that we want to use to build up a series of numbers. Notice that the methods we need for *ArrayListIntegerContainer* are and should be the exactly the same as those for *LinkedListIntegerContainer*. Thus, we need to first create a parent class for both *ArrayListIntegerContainer* and *LinkedListIntegerContainer.* We name this class as *IntegerContainer*.

We have totally six basic steps to build up a series of numbers as an *abstractList* 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), sort a list of numbers by selection sort, linear search and binary search. In *IntegerContainer*, I write six methods to achieve the six basic steps. Finally, I set up a method to return our *ArrayList* for the unit tests later.

Before we begin write code for time comparison, we need to first make sure that *ArrayListIntegerContainer* and *LinkedListIntegerContainer* work properly. Thus, I write unit tests for both of them. 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. I do this for both of the classes. 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. If there are some running time for some methods much larger than others. I would redo the experiment with larger value for those with relatively short time.

**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 and for *ArrayList* and *LinkedList* respectively. At first, I use a for loop to write five lines of numbers in main method. In each line, I write all seven time down and then increase the input size by 500 for both *ArrayList* and *LinkedList*. Then, I have the average run times of size 500, 10000, 1500, 2000 and 2500. Then, I notice that the running time for *addSorted* and *Sortion* are much larger than other method. And some running time is even 0. Thus, I do the experiment again without *addSorted* and *Sortion* by increasing the input size by 5000. I repeat this process until I can compare the values between *ArrayList* and *LinkedList*. In conclusion, I increase the size by 10000 for *AddToFront*, *AddToBack* and *BinarySearch*. And I increase the size by 100000 for *AddToBack* and *BinarySearch.*

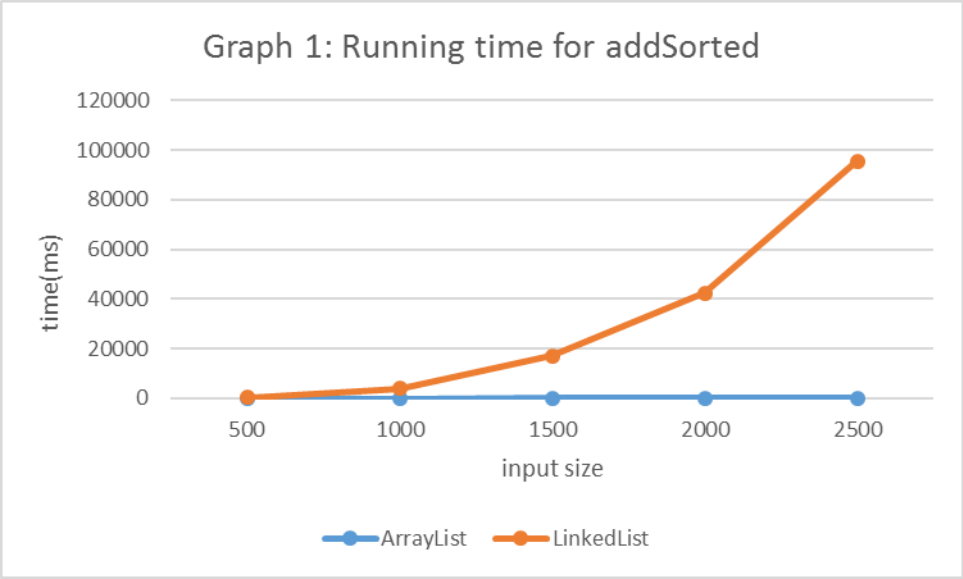
**Data and Analysis**

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

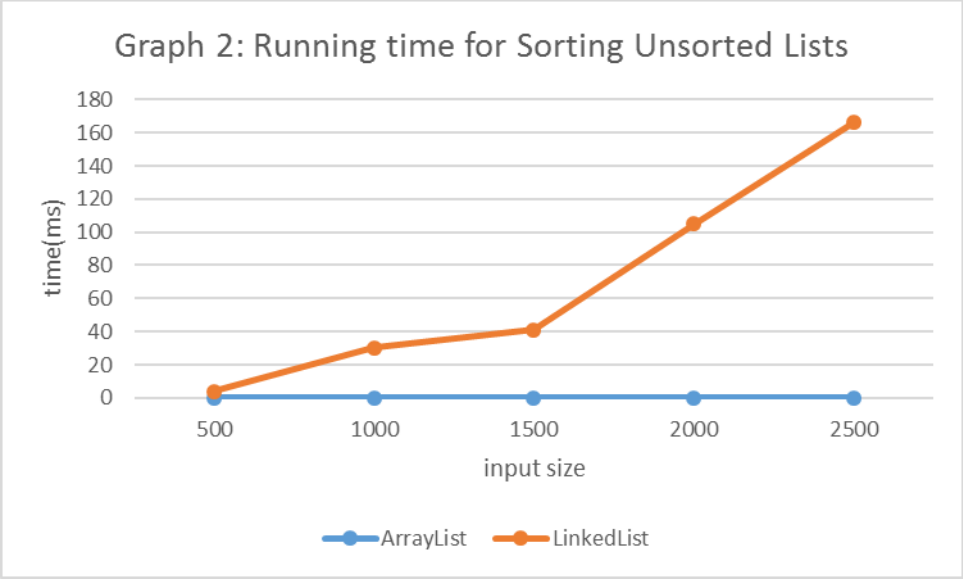
| size | | Front | Back | Sorted | SortofUnsortedList | SortedList | LinearSearch | BinarySearch |
|---|---|---|---|---|---|---|---|---|
| 500 | A | 0 | 0 | 0 | 6.4 | 0 | 0 | 0 |
| 500 | L | 0 | 0 | 378 | 1061.6 | 4 | 0 | 0 |
| 1000 | A | 0 | 0 | 6.2 | 17 | 0 | 0 | 0 |
| 1000 | L | 0 | 0 | 3815.8 | 10438.8 | 30.4 | 6 | 0 |
| 1500 | A | 0.6 | 0 | 26 | 71.8 | 0 | 0 | 0 |
| 1500 | L | 0.2 | 0 | 17170.6 | 35253.8 | 41.2 | 13.8 | 0 |
| 2000 | A | 0.8 | 0.2 | 26.6 | 68.4 | 0 | 0.2 | 0 |
| 2000 | L | 0 | 0.2 | 42270.4 | 82568.4 | 105 | 25 | 0 |
| 2500 | A | 3 | 0 | 40.6 | 104.2 | 0 | 0 | 0 |
| 2500 | L | 0 | 0 | 95560.6 | 156749.4 | 166.2 | 39.6 | 0 |

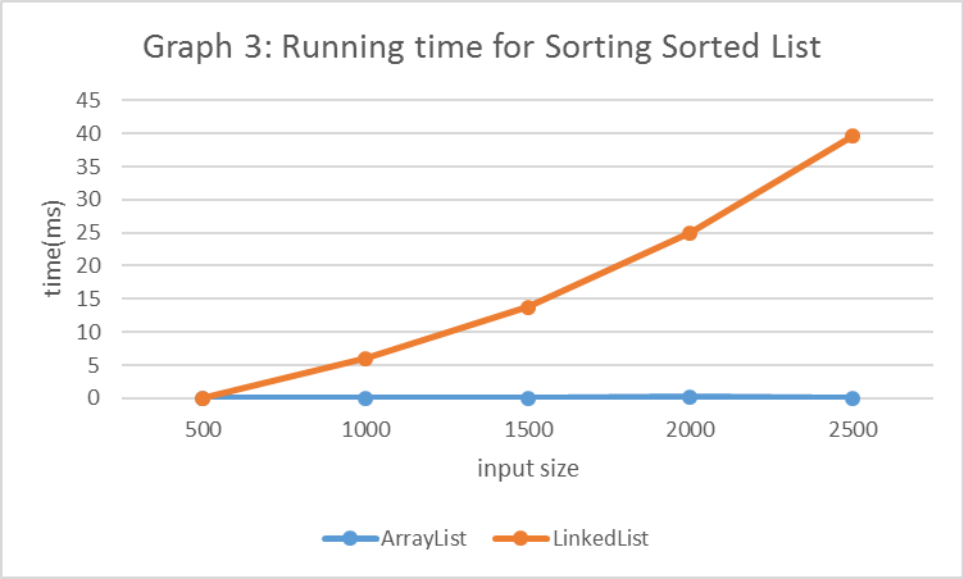
Table 1

Table 1 shows running time of all seven tasks for both ArrayList and LinkedList. For the second column, A means the time for *ArrayList,* and L means for *LinkedList*. For the table, we can clearly notice that the running time for *AddToSoted*, S*ortofUnsortedLis*t and *SortofSortedList* are much more longer than others. This implies these three methods are more complex. Here are the three graphs and analysis about these three tasks according to Table 1.




From graph 1 above, we can clearly notice that the running time of LinkedList is always longer than that of ArrayList in the task of addSorted. Also, the time for ArrayList is constant but that for LinkedList is quadratic. Thus, the difference between times of ArrayList and LinkedList becomes larger and larger as the input size increases.



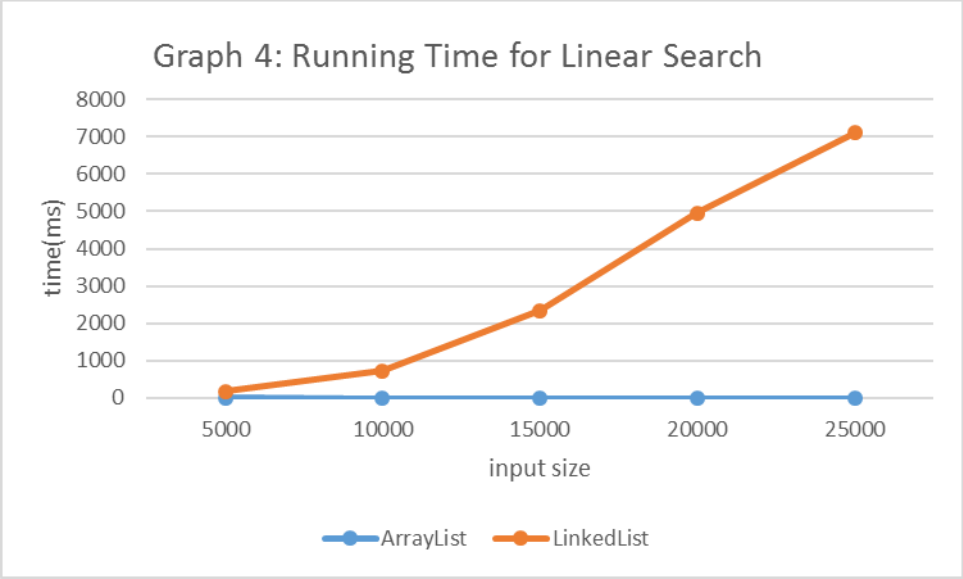



From Graph 2, we can see that the time of ArrayList is also shorter than that of LinkedList. The time for ArrayList is constant. However, the time of LinkedList is linear this time. This property can be illstrated in Graph 3, running time of Sorting Sorted List, as well.

Since, the time is so small in Table 1 for other tasks, I do the experiment again without addSorted and Sortion by increasing the input size by 5000. Here is Table 2 from myConclusion2.csv.

| | | timeAddToFront | timeAddToBack | timeLinearSearch | timeBinarySearch |
|---|---|---|---|---|---|
| 5000 | Array | 3.2 | 0 | 0.4 | 0 |
| 5000 | Linked | 0 | 0 | 173 | 0 |
| 10000 | Array | 20 | 0 | 0 | 0 |
| 10000 | Linked | 0 | 0 | 718.6 | 0 |
| 15000 | Array | 60 | 3.2 | 0 | 0 |
| 15000 | Linked | 3 | 3.4 | 2336.4 | 0 |
| 20000 | Array | 53.8 | 0 | 0 | 0 |
| 20000 | Linked | 0 | 3 | 4964.6 | 2 |
| 25000 | Array | 119.6 | 1.2 | 0 | 0 |
| 25000 | Linked | 3 | 0 | 7104.4 | 0 |

Table 2

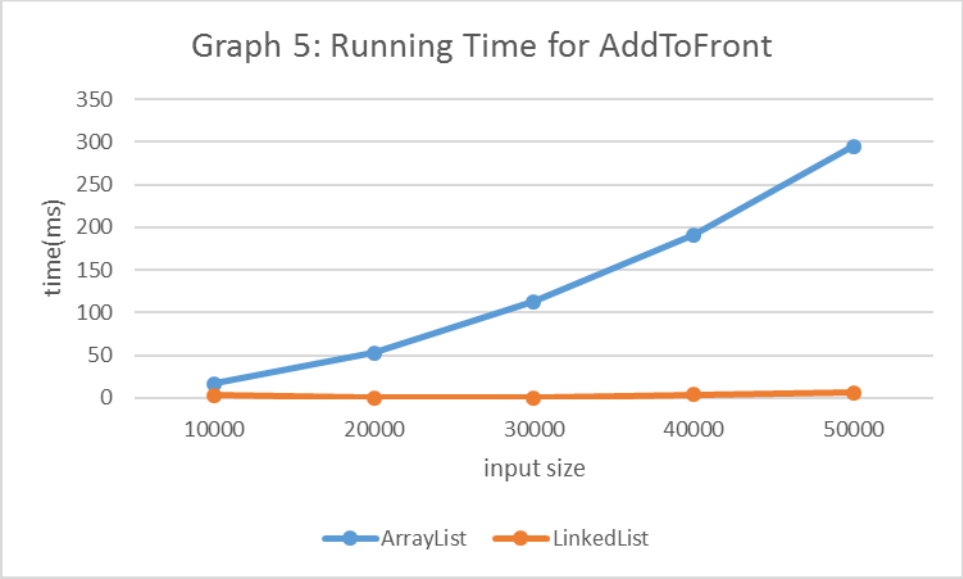



From Graph 4, we can see that its property is similar with the Graph 2 and 3. The time for ArrayList is shorter and constant. But that for LinkedList is longer and linear.

Then, I increase the size by 10000 for AddToFront, AddToBack and BinarySearch. Here is a table from myConclusion3.csv.

| | | timeAddToFront | timeAddToBack | timeBinarySearch |
|---|---|---|---|---|
| 10000 | Array | 17 | 0 | 0 |
| 10000 | Linked | 3 | 0 | 0 |
| 20000 | Array | 53 | 1 | 0 |
| 20000 | Linked | 0 | 3 | 0 |
| 30000 | Array | 112.8 | 0 | 0 |
| 30000 | Linked | 0.2 | 6.2 | 3 |
| 40000 | Array | 191.2 | 3 | 0 |
| 40000 | Linked | 4 | 9.8 | 7 |
| 50000 | Array | 295.4 | 3.2 | 0 |
| 50000 | Linked | 6.2 | 3 | 6.2 |

Table 3

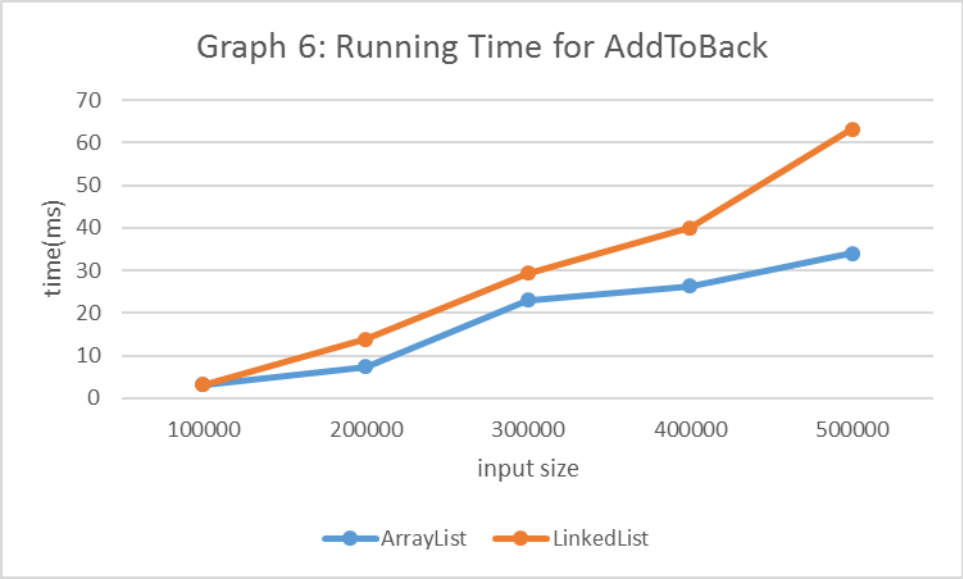



Graph 5 is what I get for AddToFront task. Notice that the property is different in this time: time used by ArrayList is larger than LinkedList. Also, time for ArrayList is linear and that for LinkedList is constant.

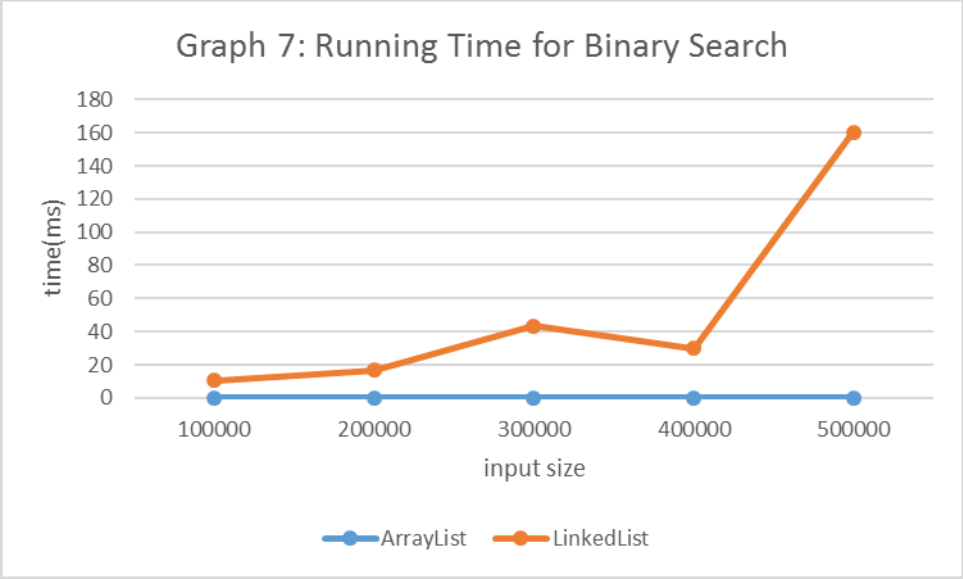
Finally, I increase the size by 100000 for AddToBack and BinarySearch in myConclusion.csv.

| | | timeAddToBack | timeBinarySearch |
|---|---|---|---|
| 100000 | Array | 3.2 | 0 |
| 100000 | Linked | 3.2 | 10.4 |
| 200000 | Array | 7.4 | 0 |
| 200000 | Linked | 13.8 | 16.8 |
| 300000 | Array | 23 | 0 |
| 300000 | Linked | 29.4 | 43.4 |
| 400000 | Array | 26.4 | 0 |
| 400000 | Linked | 40 | 30 |
| 500000 | Array | 34 | 0 |
| 500000 | Linked | 63.2 | 160.4 |

Table 4




From Graph 6, the running time for addToBack has the same increasing rate for ArrayList and LinkedList. And the value fo LinkedList is always longer than that for ArrayList.




For Binary Search, Graph 7 tells that the Arraylist runs more quickly and constantly. And the running time for LinkedList is linear.

**Conclusion**

Except running time for addToFront, using ArrayList is always faster than using LinkedList. In these six tasks, the running time for ArrayList is all constant except addToBack, which is linear. For LinkedList, the running time is quadratic in addSorted and linear in other cases. AddToFront

is special comparing with other tasks. Running time for LinkedList is constant and much shorter than that for ArrayList.

**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>
Sadovnik, Amir. "Objective." *CS150: Lab 3*. N.p., n.d. Web. 13 Feb. 2017.
Weiss, Mark Allen. *Data structures and problem solving using Java*. Reading, MA: Addison-Wesley, 2002. Print.