Lab 1 Report

CS150 Lab – Section 2

Oanh Doan

September 10, 2017

1. Introduction

The goal of lab 2 is to compare the theoretical complexity versus the actual running time of the program. In addition, we compare the sorting time of unsorted list versus that of a sorted list. Our hypothesis of the methods' complexity is that:

- AddToBack: O(n)
- AddToFront: $O(n^2)$
- AddSorted: $O(n^2)$
- SortOfUnsortedList: $O(n^2)$
- SortOfSortedList: $O(n^2)$

2. Approach

2.1. Design of the program

a. The RandomDoubleContainer class stores data (ArrayList) and different methods whose complexity and running time will be measured and compared. The class includes the following methods
   - addToFront(double num)
   - addToBack(double num)
   - addSorted(double num)
   - swap(int x1, int x2) – swap the positions of 2 numbers
   - selectionSort()
   - convertToArray()
   - toString()
   - getArrayList()

b. The RandomDoubleContainerTest class contains methods to test the methods above:
   - testAddToFront1()
   - testAddToFront2()
   - testAddToBack1()
   - testAddToBack2()
   - testSwap1()
   - testSwap2()
   - testSelectionSort1()
   - testSelectionSort2()

c. The ExperimentController class contains methods to measure the running time of different methods in RandomDoubleInteger class.
    - timeAddToFront()
    - timeAddToBack()
    - timeAddSorted()
    - timeSortOfUnsortedList()
    - timeSortOfSortedList()

d. The HelloWorld class contains a method to print a message a number of times specified by user.

2.2. Choice of algorithm and data structures

- ArrayList ("ArrayList (Java Platform SE 8)," n.d.) is used to store data and implement the methods
- Selection Sort ("11.06.01 - Selection Sort," n.d.) is the algorithm used to sort an array of numbers in ascending order. This algorithm searches for the largest number in the unsorted array, places at the end and searches for the next largest value.

3. Methods

Testing the methods:

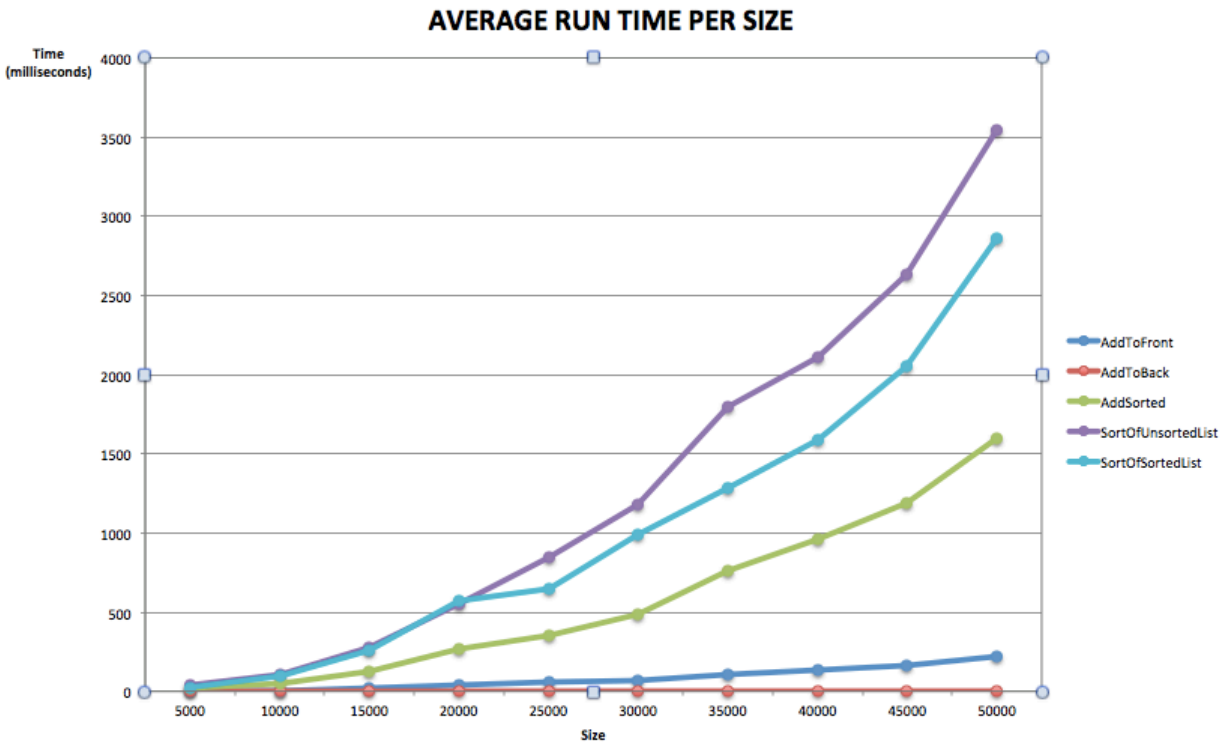
- Run the program with 10 different input sizes to observe changes in complexity with respect to changes in amount of data.
- With each input size, try 3 different seeds to ensure randomness and take the average of their sorting time.
- Final output is reformatted on a separate Excel sheet for data visualization.

4. Data and Analysis

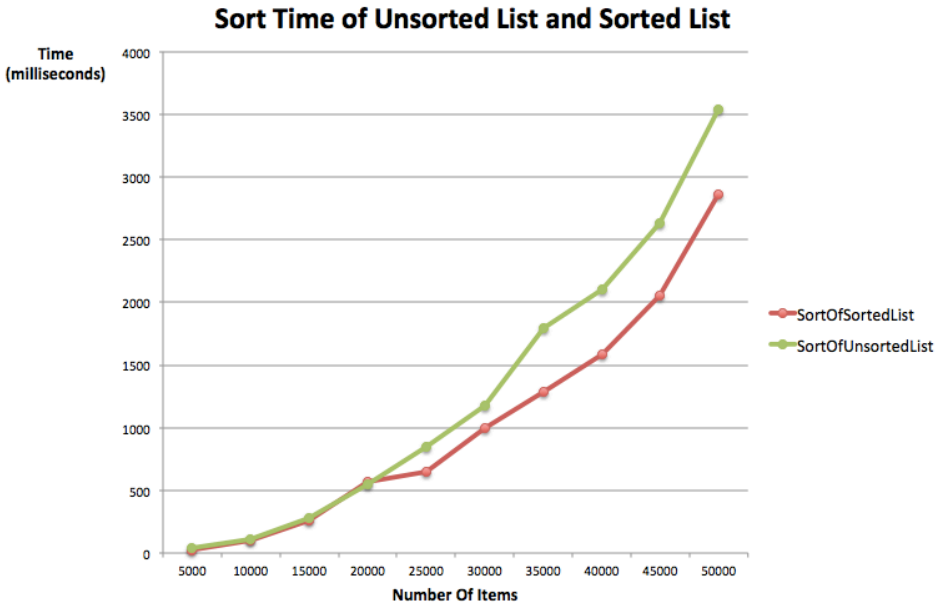
4.1. Analyze run time

Size	Run Time (milliseconds)				
	AddToFront	**AddToBack**	**AddSorted**	**SortOfUnsortedList**	**SortOfSortedList**
5000	3	0	16	36	25
10000	6	1	45	108	101
15000	18	1	122	279	259
20000	42	1	269	550	568
25000	56	1	357	846	648
30000	68	1	489	1175	993
35000	108	1	761	1797	1283
40000	133	1	965	2106	1587
45000	160	1	1193	2637	2057
50000	223	1	1595	3541	2857

Table 1. Average sorting time per size of input



Graph 1. Average run time per size



Graph 2. Average run time of SortOfSorted List and SortOfUnsortedList

Table 1 and Graph 1 present the average run time of each method, using 3 trials with 3 different seeds. From Table 1 and Graph 1, in terms of run time, we can see that:

SortOfUnSortedList > SortOfSortedList > AddSorted >AddToFront > AddToBack

The results match with our hypothesis of the methods' complexity. However, we notice that even though AddToFront, SortOfUnSortedList, and SortOfSortedList share the same complexity,

their run time differ significantly. We attempt to explain these differences below:

- AddToFront: $O(n^2)$
  Each time the method is called, *n-1* elements are moved to the right. Calling the method *n* times causes complexity to be $O(n^2)$
- AddSorted: $O(kn) + O((n-k)n) \sim O(n^2)$ where *k* is the position of the inserted number
  Every time the method is called, it searches for the appropriate position to insert the number and all elements to its right one position up. Depending on the array's randomness, *k* can be close to *n* or 0. If *k* is close to *n*, the main part of the process is to look for the appropriate position. This procedure involves constantly calling array's size, comparing the index with the array's size and comparing the current value with the input value. Therefore, even though this method has the same complexity as *AddToFront*, it may end up taking more time to finish.
- SortOfUnsortedList: Every time the second loop runs, it finds the largest value of *(n-i)* elements by constantly comparing the first element with the following elements and updating the index of largest value found. Thus, there are *(n-i)* comparisons. Therefore, the number of comparisons done is

  $$(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$$

  So the complexity is $O(n^2)$. In addition, a comparison involves many smaller steps, including checking the conditions for i,j, retrieving elements from the array, updating indices, swapping numbers, etc. Because of these steps, SortOfUnsortedList is the most time-consuming method.
- SortOfSortedList has the same complexity as SortOfUnsortedList and involves the same number of steps, except that we never need to make any swaps because the numbers are already in the right order. With a small dataset, we do not need to make many switches. Therefore, both methods have approximately the same run time. However, with a big dataset, we may need to make many switches, causing the run time of two methods to differ.

4.2. Compile and run HelloWorld()

```
Last login: Sun Sep 10 00:55:03 on ttys000
Oanhs-MacBook-Air:~ oanhdoan$ cd desktop
Oanhs-MacBook-Air:desktop oanhdoan$ cd cs150l
Oanhs-MacBook-Air:cs150l oanhdoan$ cd lab2
Oanhs-MacBook-Air:lab2 oanhdoan$ cd Lab2
Oanhs-MacBook-Air:Lab2 oanhdoan$ ls
ArrayListTest.class             RandomDoubleContainer.java
ArrayListTest.ctxt              RandomDoubleContainerTest.class
ArrayListTest.java              RandomDoubleContainerTest.ctxt
ExperimentController.class      RandomDoubleContainerTest.java
ExperimentController.ctxt       RandomExplore.class
ExperimentController.java       RandomExplore.ctxt
HelloWorld.class                RandomExplore.java
HelloWorld.java                 output2.csv
README.TXT                      output2_withgraph.xls
RandomDoubleContainer.class     package.bluej
RandomDoubleContainer.ctxt
Oanhs-MacBook-Air:Lab2 oanhdoan$ javac HelloWorld.java
Oanhs-MacBook-Air:Lab2 oanhdoan$ java HelloWorld
Sorry you don't want to say hello
Oanhs-MacBook-Air:Lab2 oanhdoan$ java HelloWorld 2 3 4
Sorry you don't want to say hello
Oanhs-MacBook-Air:Lab2 oanhdoan$ java HelloWorld 5
Hello World
Hello World
Hello World
Hello World
Hello World
Oanhs-MacBook-Air:Lab2 oanhdoan$ java HelloWorld a
java.lang.NumberFormatException: For input string: "a"
Oanhs-MacBook-Air:Lab2 oanhdoan$
```

Image 1. Compile and run HelloWorld()

5. Conclusion

We confirm that our hypothesis of the methods' complexity was correct:

- AddToBack: O(n)
- AddToFront: $O(n^2)$
- AddSorted: $O(n^2)$
- SortOfUnsortedList: $O(n^2)$
- SortOfSortedList: $O(n^2)$

However, there's a significant difference between theoretical complexity and actual run time, causing methods with the same complexity to differ in run time.

6. References

11.06.01 - Selection Sort. (n.d.). Retrieved September 10, 2017, from https://canvas.instructure.com/courses/1171720/assignments/6432392?module_item_id=12945299

ArrayList (Java Platform SE 8). (n.d.). Retrieved September 10, 2017, from https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html