# CS150 Lab 3

Oanh Doan

September 17, 2017

# 1 Introduction

The goal of lab 3 is to compare time complexity and run time of various methods using different data structures, particularly LinkedList and ArrayList. Our assumption is that most methods using LinkedList will take more time than using ArrayList because it is easier to access elements by position in ArrayList.

# 2 Approach

## 2.1 Design of program

The program includes 4 main classes as follow:

- DoubleContainer: store methods
- ArrayListDoubleContainer (a subclass of DoubleContainer): store data
- LinkedListDoubleContainer (a subclass of DoubleContainer): store data
- ExperimentController: generate data, apply the methods and record run time

The program also contains 2 unit test classes:

- ArrayListDoubleContainerTest
- LinkedListDoubleContainer

## 2.2 Choice of Algorithm and Data Structures

- AbstractList (*AbstractList (Java Platform SE 7)*, n.d.), an ancestor of ArrayList and LinkedList, is used in the parent class because of its higher implementation. Methods that work on AbstractList also work on the other two data structures.
- ArrayList (*ArrayList (Java Platform SE 8)*, n.d.) and LinkedList (*LinkedList (Java Platform SE 7)*, n.d.) are used to store data and implement the methods.

- Selection Sort (*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.
- Binary search (*Binary Search*, 2014) is the algorithm used to search for an element in a sorted list by repeatedly dividing the list in half. If the searching value is smaller than the midpoint of the list, we only search from the first half and ignore the second half of the list, and vice versa.
- Linear search is the algorithm used to search for an element in a sorted/unsorted list. The algorithm compares the searching value with every element in the list, starting from left to right, until the value is found or it reaches the end of the list.

# 3 Methods

- With each data structure, run the methods using 4 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.
- Compare the average run time of each method on 2 data structures as input size change.
- Final output is reformatted on a separate Excel sheet for data visualization.

# 4 Data and Analysis

Table 1 displays the average run time of 6 methods on the 2 different data structures, using various input sizes.

| Size | Data Structure | AddToFront | AddToBack | SortOfUnsorted | SortOfSorted | linearSearch Findable | binarySearch Findable | linearSearch NotFindable | binarySearch NotFindable |
|---|---|---|---|---|---|---|---|---|---|
| 100 | ArrayList | 0.18 | 0.08 | 0.73 | 0.32 | 0.02 | 0.00 | 0.01 | 0.01 |
| 1000 | | 0.28 | 0.17 | 9 | 2.47 | 0.09 | 0.01 | 0.07 | 0.01 |
| 2000 | | 0.42 | 0.90 | 8 | 10 | 0.11 | 0.01 | 0.14 | 0.01 |
| 5000 | | 1.95 | 0.37 | 57 | 61 | 0.31 | 0.02 | 0.52 | 0.02 |
| 100 | LinkedList | 0.16 | 0.07 | 1.57 | 0.53 | 0.03 | 0.01 | 0.02 | 0.00 |
| 1000 | | 0.22 | 0.16 | 371 | 444 | 0.48 | 0.02 | 0.53 | 0.02 |
| 2000 | | 0.16 | 0.15 | 3244 | 3830 | 3 | 0.05 | 2.01 | 0.02 |
| 5000 | | 0.40 | 0.38 | 54967 | 70323 | 13 | 0.18 | 14 | 0.07 |

Table 1: Average run time per input size (milliseconds)

## 4.1 AddToFront

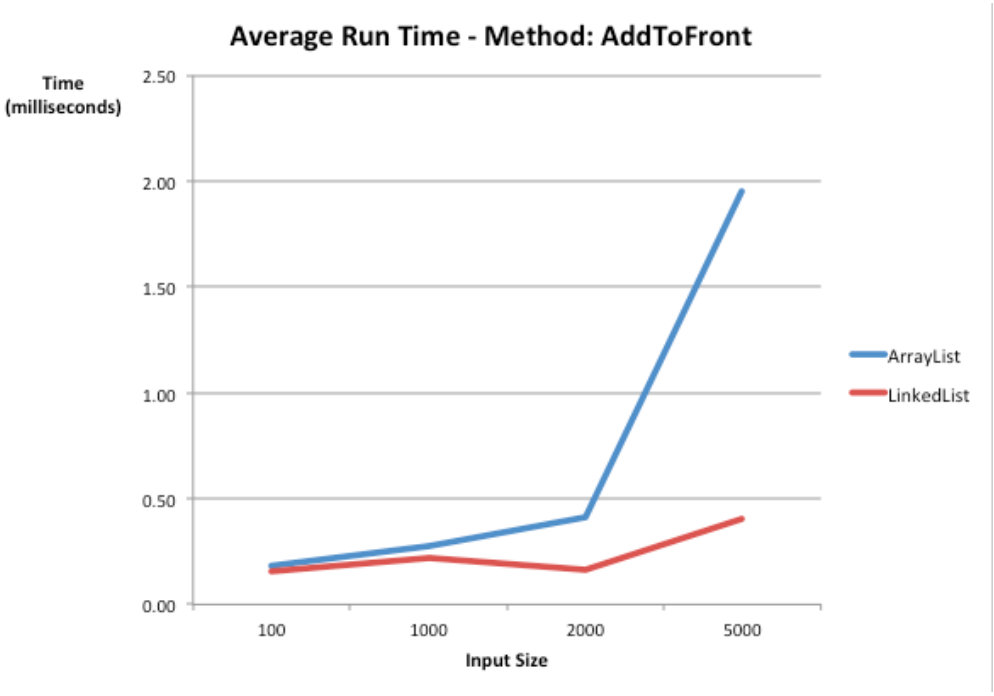


Figure 1: Average time adding numbers to the front of a list

Figure 1 displays the average run time of the AddToFront method using different data structures. From Lab 2, we know that the program's time complexity using ArrayList is $O(n^2)$. However, it drops to $O(n)$ if we use LinkedList instead. The reason is that as soon as we move away from the list head, we already reached the position where the number should be inserted.

## 4.2 AddToBack

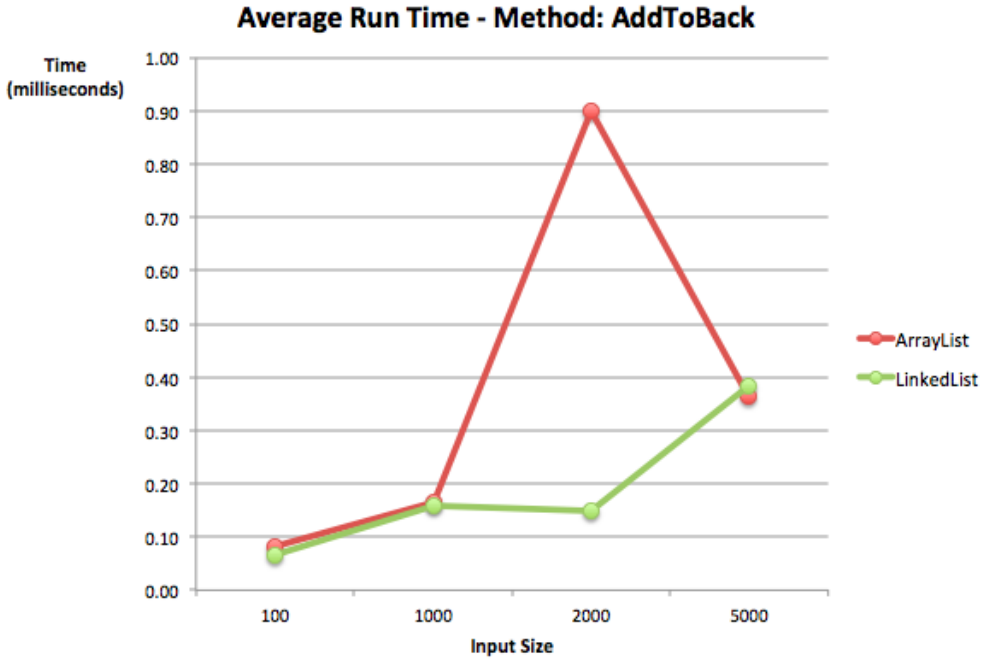


Figure 2: Average time adding numbers to the back of a list

From Lab 2, we know that time complexity of the AddToBack method grows linearly as the amount of data in ArrayList increases. In this lab, we learn that time complexity of this method and the size of LinkedList is also a linear relationship.

## 4.3 Sort a unsorted list

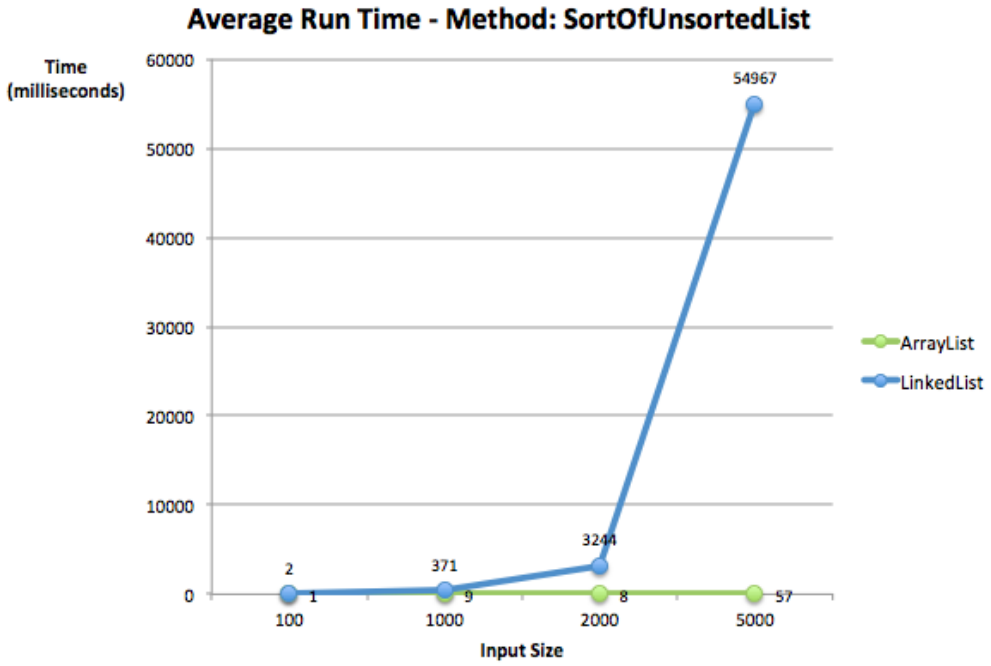


Figure 3: Average sort time of an unsorted list

The time complexity of the selection sort algorithm using ArrayList is $O(n^2)$. However, when applied to LinkedList, the actual run time ends up much greater because accessing list elements by position is not easy in LinkedList.

## 4.4 Sort a sorted list

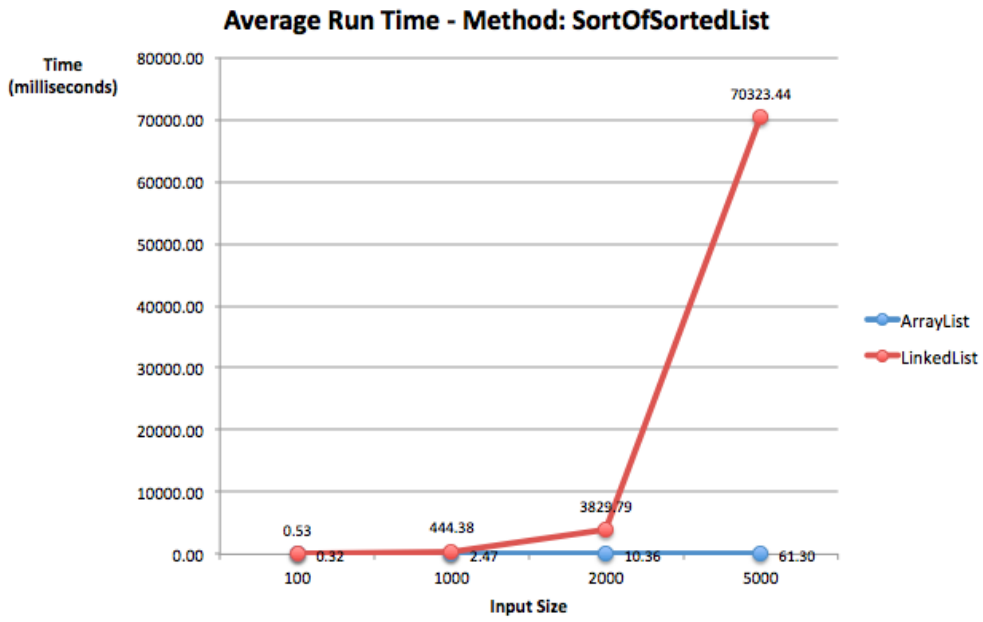


Figure 4: Average sort time of sorted list

Similar to the previous method, sorting a sorted LinkedList is much more timeconsuming than an ArrayList because we cannot easily access the list elements.

## 4.5 Linear search

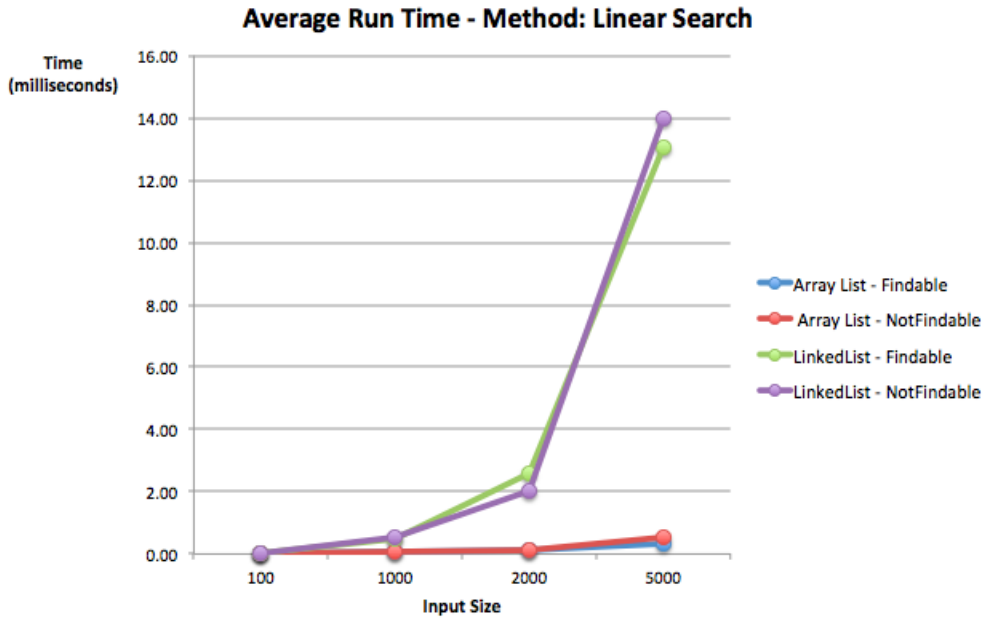


Figure 5: Average search time using Linear Search

Figure 5 presents the average time searching for an in-range and an out-of-range number in sorted ArrayList and LinkedList. Because ArrayList is faster to access by position, the program runs more quickly. There's no significant difference between searching for an in-range or an out-of-range number probably because the number chosen is close to the right end of the interval $[0, 1]$. To find this number, the program needs to go from left end to its position, which is close to the list right end.

## 4.6 Binary search

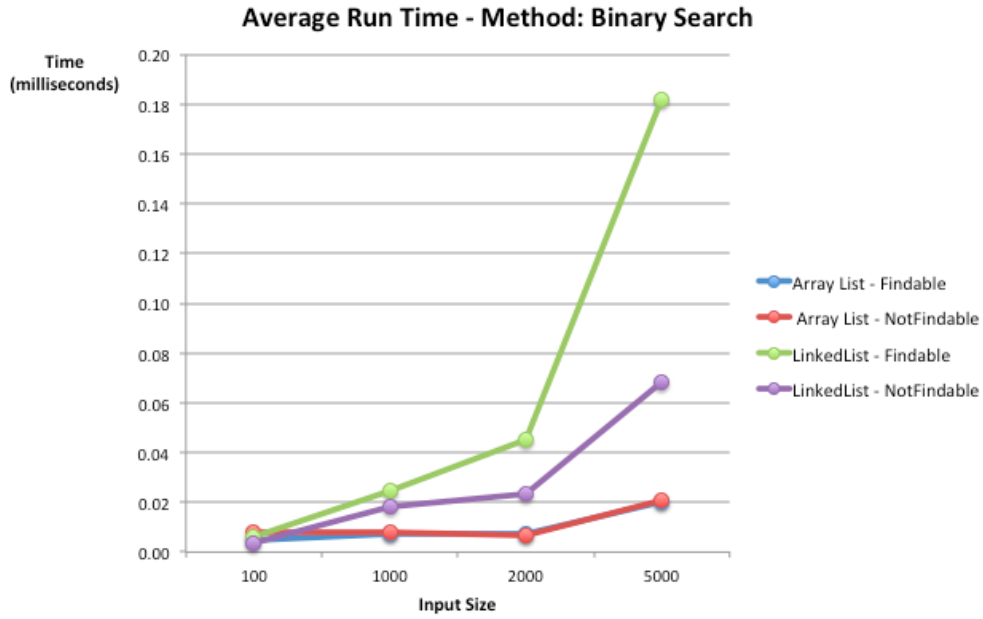


Figure 6: Average search time using Binary Search

Unlike linear search, we can see a significant difference in searching for an in-range and an out-of-range number in LinkedList. This is because the algorithm significantly reduced the number of times we need to access list elements and make comparison, which is time-consuming and costly in LinkedList.

### 4.7 Modify ChildClass1

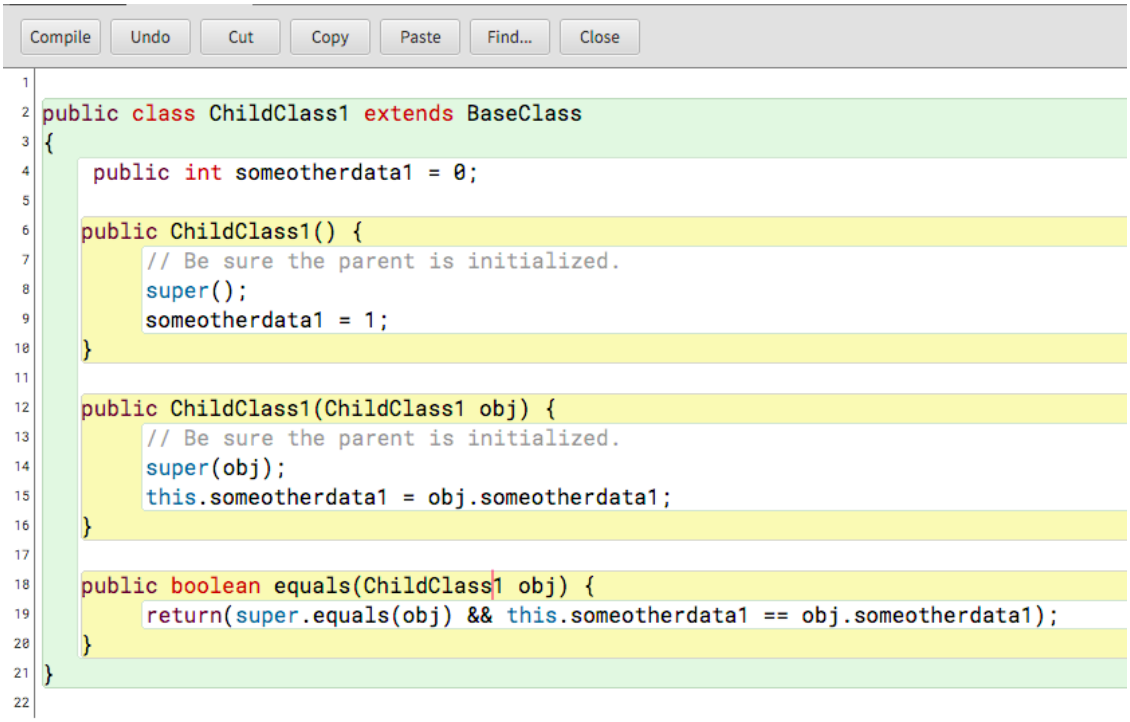


Figure 9: Modified equals() method of ChildClass1

The method takes advantage of the equals() method in the superclass. **super.equals(obj)** compares the variable *somedata* of *obj* with that of another ChildClass1 object.

# 5 Conclusion

- With methods that repeatedly need to access list elements by position, using ArrayList is much more time efficient. Examples are the sorting a list and searching for an element from the list.
- With *insertion* of numbers at list head or list tail, using LinkedList is at least as good as ArrayList.

# References

Abstractlist (java platform se 7). (n.d.). Oracle. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/AbstractList.html

Arraylist (java platform se 8). (n.d.). Oracle. Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Binary search. (2014). Retrieved from http://www.geeksforgeeks.org/binary-search/

Linkedlist (java platform se 7). (n.d.). Oracle. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Selection sort. (n.d.). Retrieved from https://canvas.instructure.com/courses/1171720/assignments/6432392?module_item_id=12945299