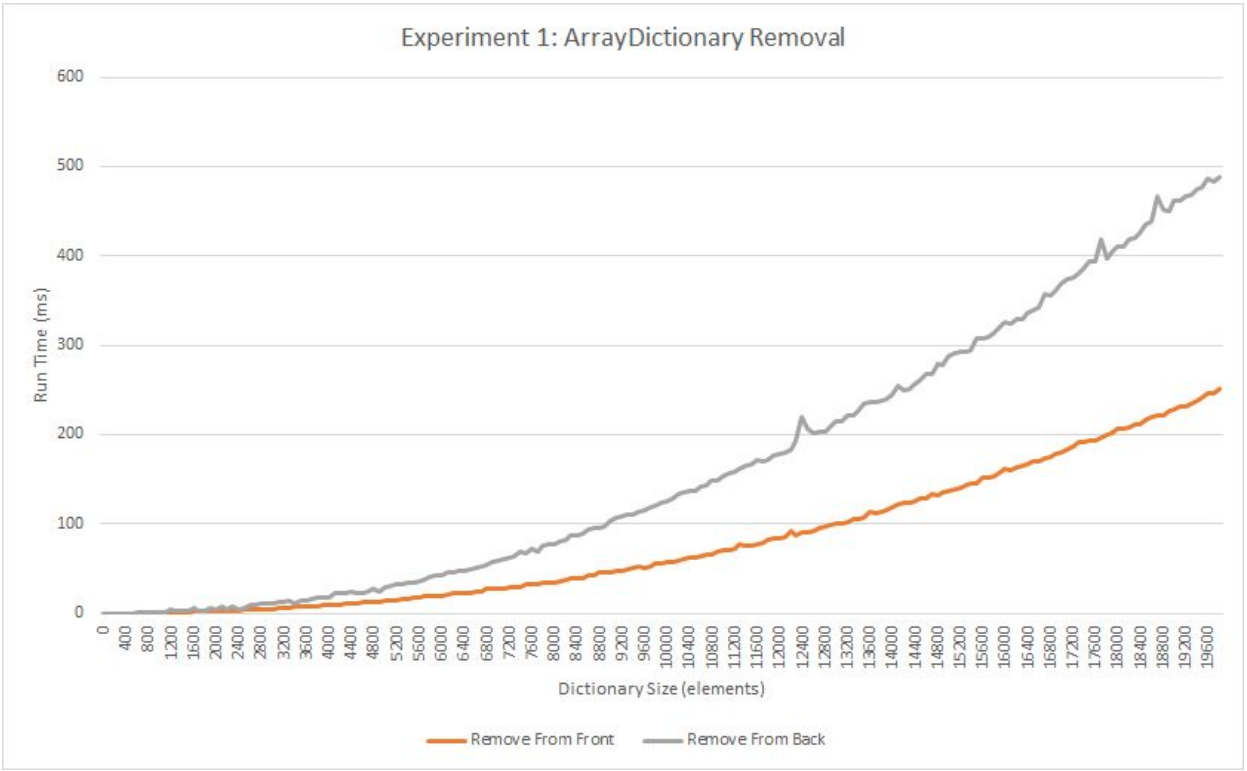
CSE 373 Homework 2 Part 2d: Group Write-Up

1) In order to deal with both null and non null entries we had to implement tests in our methods. One example of this is in our Array Dictionary methods. Many of them start out test whether the dictionary contains an input value by using the boolean containsKey method. We made our pair objects extend the comparable class to make it easier to compare them, no matter the object type (null or non null). containsKey is able to use the compareTo method to test for and handle null and non null entries.

2) ***Experiment 1 Summary/Hypothesis:*** This experiment will measure how fast our implementation of the array dictionary will remove Key-Value pairs. It will test and document the speed of removal from the front and back of the array dictionary for various sizes of array dictionaries.

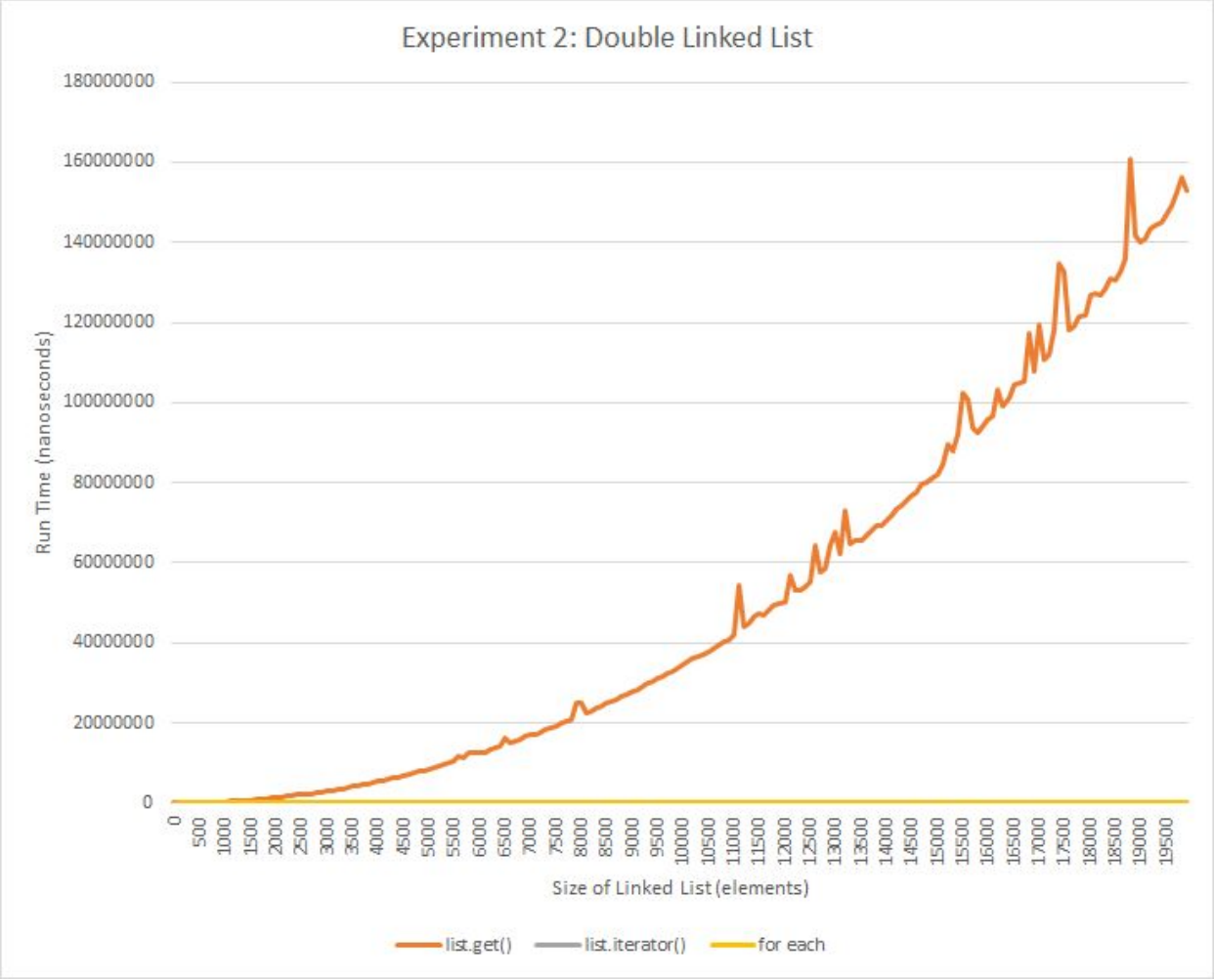
Test 1 seems to be testing removal from the front and test to seems to testing removal from the back of the array dictionary. I predict that removal from the front of the dictionary (results from test 1) will be faster than removal from the back of the dictionary (results from test 2). My reasoning for this is because in our implementation of the array dictionary, we start from the front of the array and compare values until we get to the end of the array. This lends our implementation to be faster at removing key-value pairs from the front of the array (test 1).



The results of the experiment show that our hypothesis was right. Removing from the front of our array dictionary was faster than removing from the back. This is probably due to the reasoning that is explained above in our experiment one hypothesis.

Experiment 2 Summary/Hypothesis: This experiment will measure how fast our implementation of the double linked list get method compares to our implementation of the double linked list iterator. Additionally it will test the speed of javas for each method on a double linked list.

In all three speed tests the methods are tested by how fast they can sequentially values from the linked list. For this kind of sequencing through the list, the iterator method should be fastest. It is set up to move sequentially from one item to the next- which is exactly what these tests are looking at.

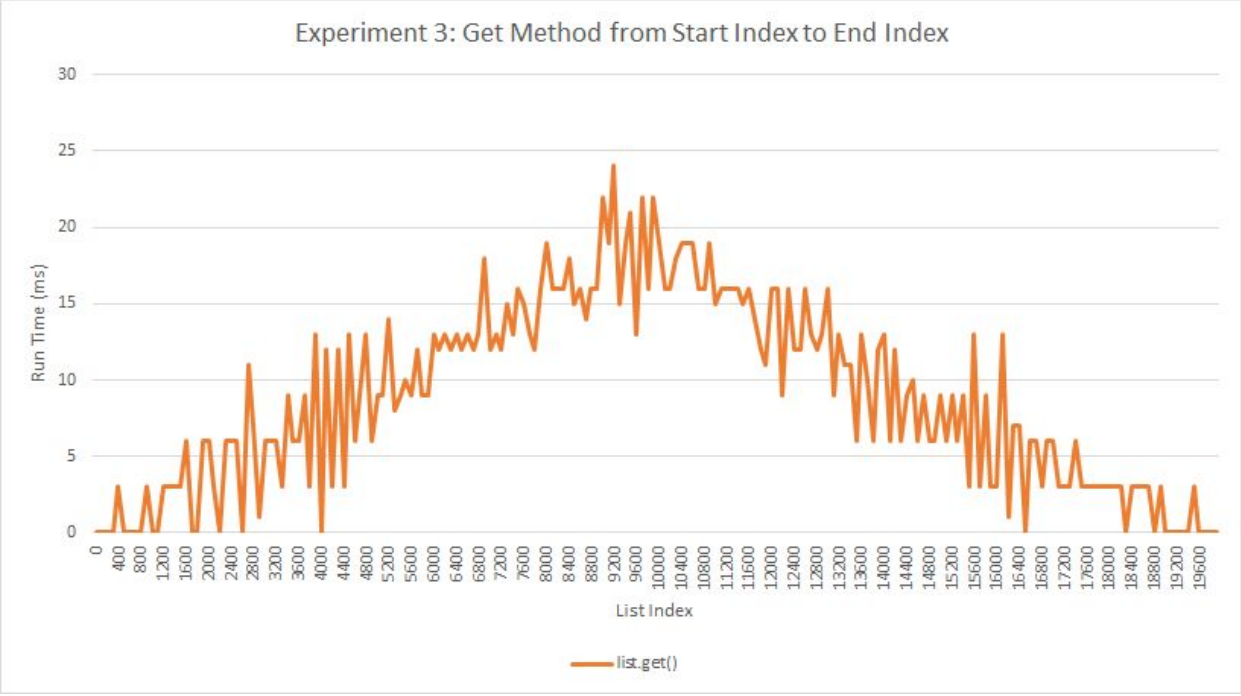


The results of this experiment show that the iterator and for each methods run in nearly the same amount of time and both those methods run much faster than the get method as the size of the list grows. Although this isn't exactly what I predicted these results make sense. The for each method essentially creates an iterator over the list which is the same as what the list.iterator method does.

The get method takes a lot longer because it is repeatedly staring at the front (or back) of the list and searching all the way through until it finds the right index. As the list gets bigger it has many more elements to search through. The iterator methods just keep track of where they are and pull the next item in the list which means they have to 'touch' a lot less elements in the list.

Experiment 3 Summary/Hypothesis: This experiment repeatedly tests our implementations get method at a specific index. It runs this test on list of increasing size. **This test actually tests the speed for the get method at indexes spread evenly from start to finish. The size of the list stays constant and the index for the get method goes from start to finish.***

In our get method we test whether the index is in the first or second half of the list and then start at either the front or back of the list and work or way in towards the index. The bigger the size of the list, the more list nodes our get method will have to pass through to get to the index. Therefore the speed of the get method should scale with the size of the list linearly.

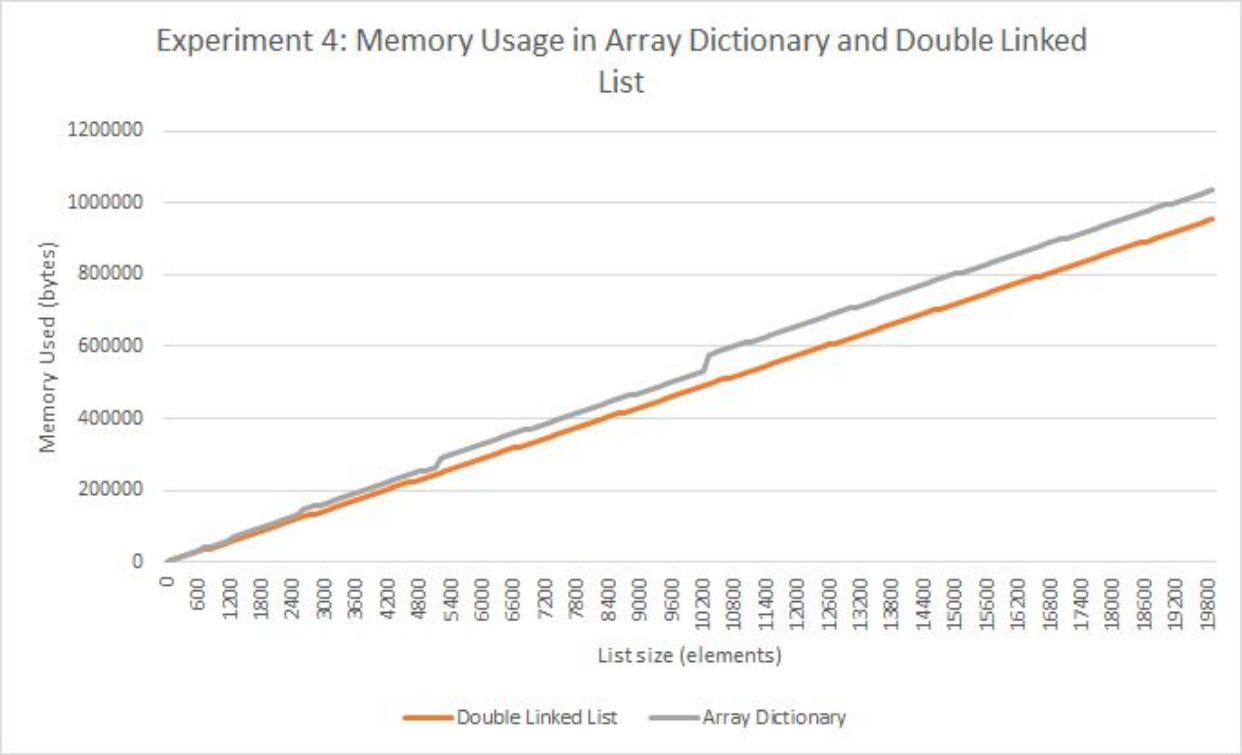


I misinterpreted the experiment code and that caused my hypothesis to be completely wrong. The size of the list didn't change. A large list was created and the get method was tested at indices from the start to the end of the list. The results of the experiment are exactly how one

would expect given the implementation of the get method. Initially the method test to see whether the index is in the front or the back of the list and then starts at whichever end is closer. So the closer the index is to the start or the end of the list the faster the get method will run. That is also to say the more centered the index is in the list the slower the get method will run.

Experiment 4 Summary/Hypothesis: This experiment find the approximate memory used in creating a double linked list and an array dictionary of sizes increasing to 20,000.

The Linked list will create only as many nodes as there are data points. The array dictionary will almost certainly have unused space at the end of the array as it has to create a bigger array and copy itself over everytime it runs out of space. Therefore it is my hypothesis that the array dictionary will have more memory used than the double linked list.



The results of this experiment are congruent with my hypothesis. The arrya dictionary does end up using more memory than the array list and in the plot there are ‘humps’ were it looks like the array dictionary had to increase its array size. On the other hand the linked list plot is relatively straight.