OOL 4

Group members:

Christian Hansen
Ben Delzer
Tim Dusek
Moses Ilunga
Mohamed Said

PHASE 1

-- COMPLETE --

PHASE 2

1) Absolutely! The data we see seems to be concurrent with the time and efficiency of every collection that was included with what we learned about in class.

2) There were a couple of outliers during the tests. One such time was a HashSet add time of 71685534.
This was roughly 20000000 nano seconds longer than the other HashSet adds. This could be caused by bad rng. The numbers may have just been inefficient
numbers for the HashSet to handle.

3) Individual{ we looked at multiple tables and tried to compare them in our head to find consistancies in top and bottom 3
add: top 3 - ArrayList, Vector, Hashtable
add: bottom 3 - Linked hash set, linked list, tree set
search top 3- hashset, linked hashset, treeset
search bottom 3 - Treemap, hashmap, hashtable
remove top 3 - hashmap, linked hashmap, hashset
remove bottom 3- Linked list, arraylist, vector
total top 3- hashset, treeset, linked hashmap
total bottom 3- arrayList, Vector, and linked hashset
}

Averages{
add: top 3 - Vector, hashtable, arrayList
add: bottom 3- Linked list,hashset, linked hashset
search top 3- hashset, linked hashset, treeset
search bottom 3-treemap, hashmap, hashtable
remove top 3- hashmap, linked hashmap, hashset
remove bottom 3- linked list, vector, arraylist
total top 3- hashset, linked hashset, tree set
total bottom 3- hashmap, hash table, Arraylist
}

4) Yes. For add, the only step for ArrayList is to add it to the end which is very quick. When it comes to the bottom of the
barrel of effieciency for add, treeset takes a while because you have to trace throughout the tree to figure out where you want to put the value.
For Search the top makes sense for treeset because you would just have to search left and right depending on the value until you either find the value if you dont.
when it comes to the least efficient for search, hashmap has keys that we have to run through and access before we can access the data.
This is effectively running two processes and adds significantly to the search time.
Hashmap is fast at removing because after you take a value out. You are done. You don't have to move anything else.
Linked list is bad at removing because after you remove a value, it has to go through and manually update all the pointers one by one.
Hashset is the quickest overall because you're feeding a value throguh and setting it to a key. There's no location you have to worry about.
ArrayList takes the longest thanks to removing values. It has to move everything over when it removes things from the middle of the list, and
when searching must go through every value in the list.

5) The prediction is that doubling searches won't do much for changing rankings, it'll just maybe add a bit of time to the average search time depending on
efficiency of the set.
RESULTS: It turns out it didn't change the times at all because it takes an average of all the search times and rankings were the same.

6) The prediction is that if searches were halved, the listed search times displayed would stay the same and the rankings should still stay the same.
RESULTS: We got the initial prediction correct. Search times and rankings stayed the same.

7) Tim- I'm extremely fascinated with how hashmaps get the hashes used to find values stored in places in the memory. I got that there was
keys values and pointers to the next set of keys and values but I had a hard time visualizing it. I found a credible source and checked it out and it helped me with
figuring out this project relating to hashmaps.

Ben - I was surprised specifically with the removing time the Treemap. Since whenever we removes something from a tree we have to rebuild the tree in able to keep some kind of structure. But even with this process the remove time for the Treemap was still one of the lowest.

Christian - I thought that it was interesting to see how consistent each operation was across each of the tests. This gives me an idea of what I can expect for real world performance for each of these data structures with different numbers.

Moses - I am generally surprised by the results. I observed that the top 3 times for adding will always be ArrayList, Vector, and Hashtable. The best times for searching will always be Linked Hash Set, and Linked List. When removing objects, the fastest ways to do so seems to always be hashmap, linked hashmap, and hashsets. So in general, the top time (performance-wise) when needing to add, search, or remove items from lists should be hashset.

Mohamed - I liked HashMap more. I tested many times just to see the difference between HashMap and TreeMap, and it seems that TreeMap is better if you're sorting, for example the Yellow Book which has a sorted/alphabetical order list. I think if you're working on applications, HashMaps will be better off as it doesn't sort but it has the best performance, whereas the slowest performing time general seems to be an ArrayList.

PHASE 3

We all used different sources throughout our observations. Here are our gathered sources individually:

Tim - https://www.quora.com/How-is-Hashmap-in-Java-implemented-internally-What-are-the-pros-and-cons-to-use-it-What-are-the-complexities-it-provides-for-insert-delete-and-lookup : Helped me see the pros and cons of the different methods.

Ben - http://stackoverflow.com/questions/20487619/complexity-of-treemap-insertion-vs-hashmap-insertion : Helped me understand the differences between a adding in a treemap vs a hashmap.

Christian - http://bigocheatsheet.com/ : General information about complexities along with a chart.

Moses - http://www.java67.com/2012/07/difference-between-arraylist-hashset-in-java.html : Made me understand the clear differences and similarities between arraylists and hashsets.

Mohamed - http://www.programcreek.com/2013/03/hashmap-vs-treemap-vs-hashtable-vs-linkedhashmap/ : Showed me a visual representation of the differences between hashmap, treemap, and hashtable.