ASSIGNMENT 3 ANALYSIS

Elijah Grubb u0894728

1. Tanner Barlow. Tanner submitted the program.

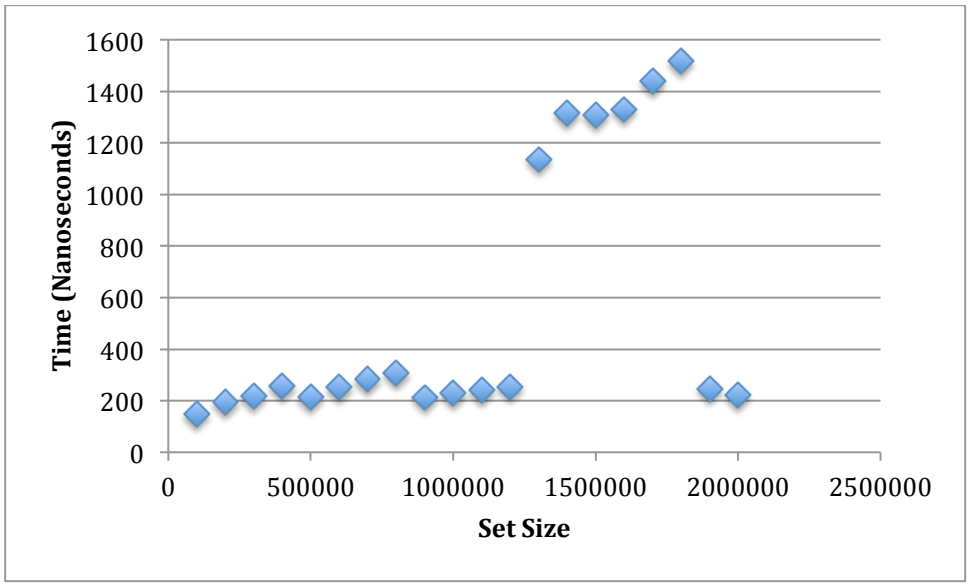
2. We switched about every 30 minutes. I think it was pretty good amount of time. Possibly switching less often if only because once you get going it's annoying to have to stop and start again.

3. Tanner was once again awesome and pulled more than his share of the weight. He's a very competent programmer and I very much plan on working with him again.

4. I think efficiency would have been very similar, with lists being just slightly more efficient. The main advantage that Java Lists have is that they grow automatically which would have removed some complexity that really doesn't happen that often. Otherwise, I still would have had to search, sort when adding and checked for duplicates by myself. Using a Java List also would have been helpful for a remove() method which would have helped in efficiency.

5. Since we used BinarySearch with its halfing principle I would expect the Big-O notation of the contains method to be O(log(N)).

6. By using a timesToLoop value of 1,000,000 I was able to get a pretty consistant basis on my data and viewed a curve that seemed relatively logistic. For some odd reason, however, you can see that the set sizes from 1,300,000 to 1,800,000 quadrupled in size in comparison to the other time lengths. No matter how many times I repeated my testing this was consistant for this set of numbers. However, outside of these outliers the curve appears very close to log(N) and makes our binary search look very efficient.



Set Size	Test 1	Test 2	Test 3	Average Time Length
100000	175	124	146	148.3333333
200000	187	196	203	195.3333333
300000	225	238	187	216.6666667
400000	229	314	232	258.3333333
500000	195	195	253	214.3333333
600000	252	210	296	252.6666667
700000	263	298	290	283.6666667
800000	305	343	275	307.6666667
900000	225	197	216	212.6666667
1000000	250	217	225	230.6666667
1100000	260	244	226	243.3333333
1200000	282	231	249	254
1300000	1230	1087	1088	1135
1400000	1500	1152	1290	1314
1500000	1257	1458	1209	1308
1600000	1493	1267	1233	1331
1700000	1521	1477	1320	1439.333333
1800000	1701	1432	1420	1517.666667
1900000	217	299	225	247
2000000	267	179	220	222

7. It takes about O(log(N)) to locate the index to be inserted since we utilize the binary search in out contains method to locate where it should go. Worst case scenario in would take O(log(N)) to locate the position to add an element into MySortedSet() since it utilizes the binary search.

timesToLoop	Test 1	Test 2	Test 3	Average Time Length
1000000	316	234	244	264.6666667

8. About 9 hours was spent working on this project.