Lindsay Haslam

If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)

If I were to have a JavaList, I wouldn't have needed the growArray() method because it would automatically resize. Currently, I have to do it manually. With a JavaList, I could also use the add(index, element) and remove(index) methods which handle shifting behind the scenes, whereas I'm attempting to take care of shifting manually with "System.arraycopy." In my code, I have to check for index bounds errors, whereas that wouldn't be an issue with a Java List because the size would adjust dynamically. My iterator class would be much more simple, because I would be able to use the iterator that is already provided by the List class. I think using a Java List would have made me code cleaner and it probably would have taken less time for me to build.

What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?

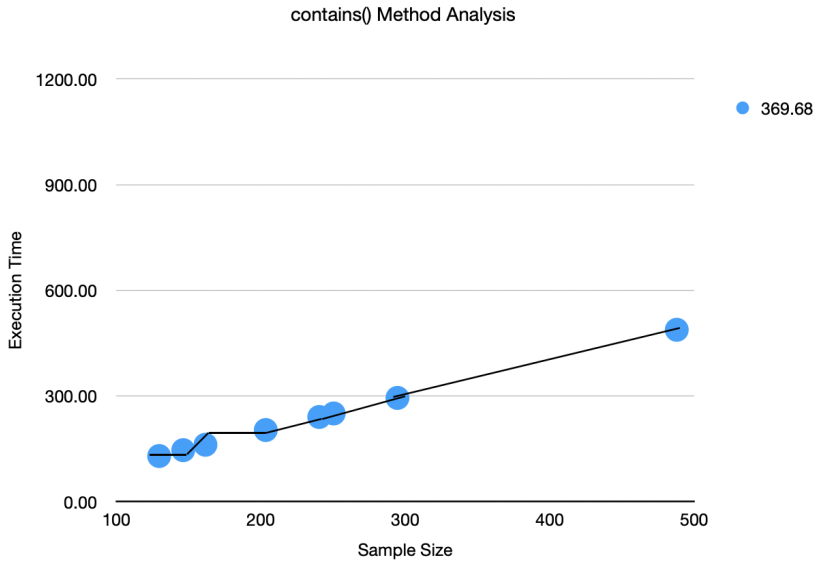
I would expect the behavior to be similar to an O(log n). I think it would take a lot of time at first, but then level out as time passes. We are using binary search in our contains(), and so I would assume that it would take less time and effort as the specific area (the middle) it searches through gets smaller with each iteration.

Plot the running time of BinarySearchSet's contains method, using the timing techniques demonstrated in previous labs. Be sure to use a decent iteration count to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 2?

Kind of, overall the closest BigO behavior it matches is O(log n), but it isn't a perfectly curved line like the examples I saw online and in class.

containsAnalysis

1024	**369.68**
2048	161.61
4096	129.57
8192	146.23
16384	203.37
32768	294.55
65536	240.4
131072	250.38
262144	508.41
524288	487.91
1048576	1195.02



Consider your add method. For an element not already contained in the set, how long does it take to locate the correct position at which to insert the element? Create a plot of running times. Pay close attention to the problem size for which you are collecting running times. Beware that if you simply add N items, the size of the sorted set is always changing. A good strategy is to fill a sorted set with N items and time how long it takes to add one additional item. To do this repeatedly (i.e., iteration count), remove the item and add it again, being careful not to include the time required to call remove() in your total. In the worst-case, how much time does it take to locate the position to add an element (give your answer using Big-oh)?

The first time, it takes over 200 seconds to locate the correct position. The worst-case would be an O(log n), with n being the size of the set.

addMethodAnalysi

1024	**226.68**
2048	327.49
4096	405.41
8192	753.72
16384	1421.2
32768	2805.49
65536	5808.72
131072	11126.7
262144	24577.44
524288	68579.15
1048576	155252.89