1. 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 had the sorted set with a java List instead of basic array, then there would be no need for a separate method like growArray() to dynamically increase the capacity of mine data structure. In operations that requires array manipulation, such as System.arraycopy would not be required with a list. Using a list instead of the method that I used would be more efficient insertions and deletions specially when the size of the set changes. Lists overall, handle resizing and shifting more efficiently.

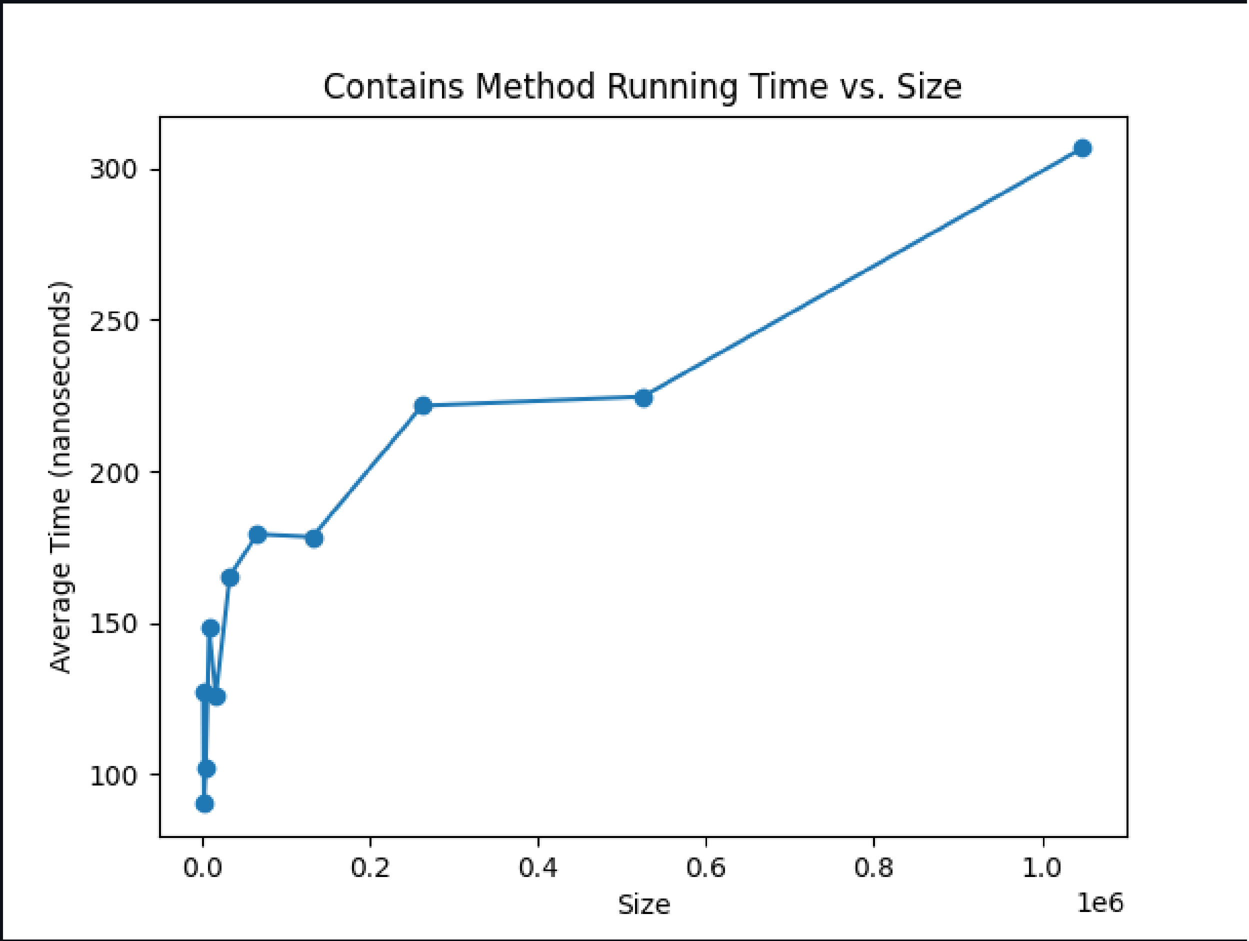
2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?

The Big-O behavior of BinarySearchSet contains shows a O (log n) due to its method implementation of always dividing by 2. As the size of the set grows, the number of iterations required for the binary search increases logarithmically with number n number of elements.

3. 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?

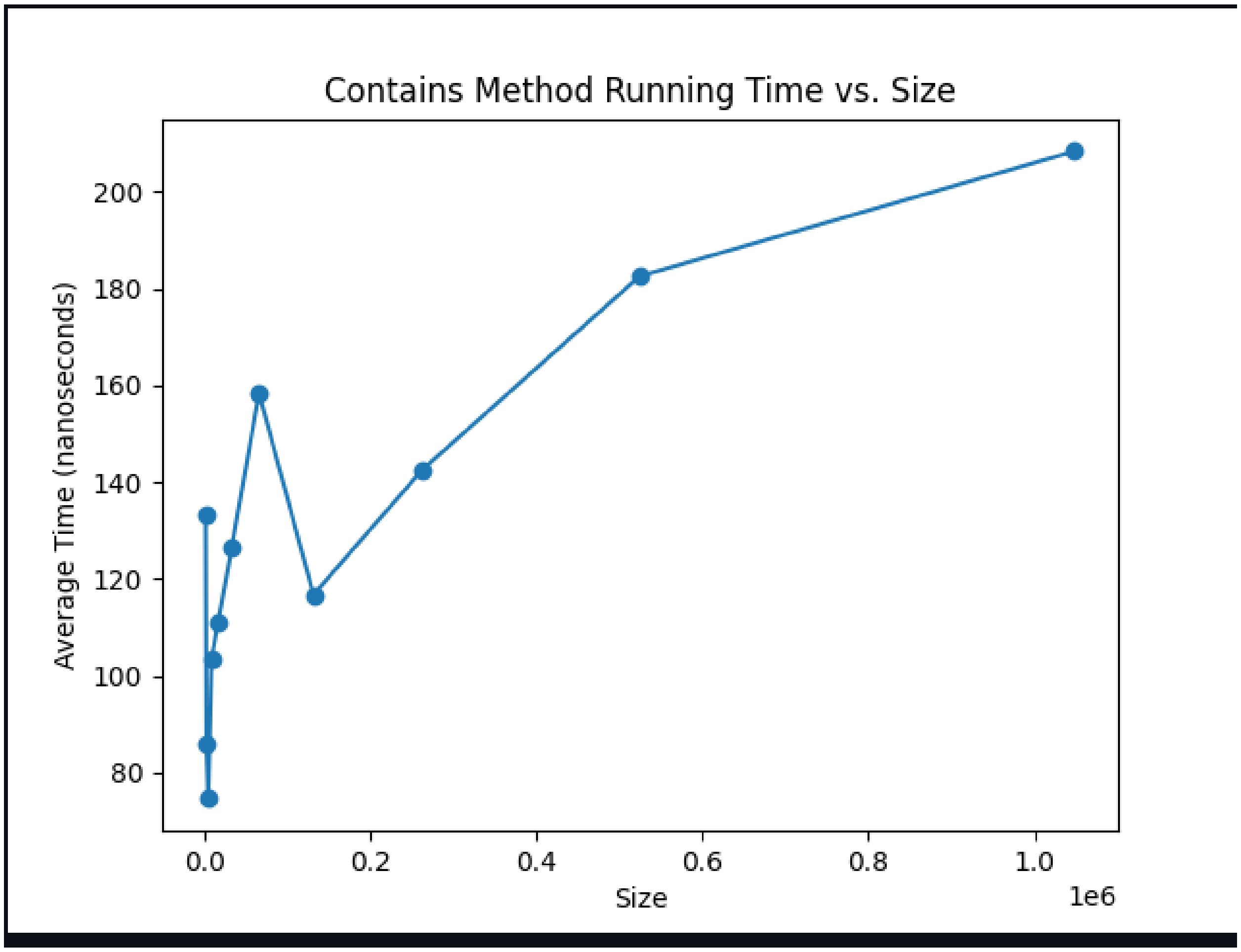
The way that my experiment went, the growth rate of these running times matches the big O behavior of O (log(n)). As the size increased, the time did not increased linearly but instead to a rate that matches logarithmic growth.

I think it actually looks like O (sqrt(n) instead of Log at least for my data. Because from sizes to 0.9 to 1.0 from all range the values did not match a log of n pattern but instead of square root of n pattern with yields a higher spike in the average time.



4. 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)?

In worst case, the time is O (log n) since it's algorithm does it's binary search repeatedly always cutting search space byDid half until finds the correct position which I think it is the pattern exhibited by my plot experiment.


Contains Method Running Time vs. Size
Average Time (nanoseconds)
80
100
120
140
160
180
200
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Size