When you are satisfied that your program is correct, write a brief analysis document. Ensure that your analysis document addresses the following.

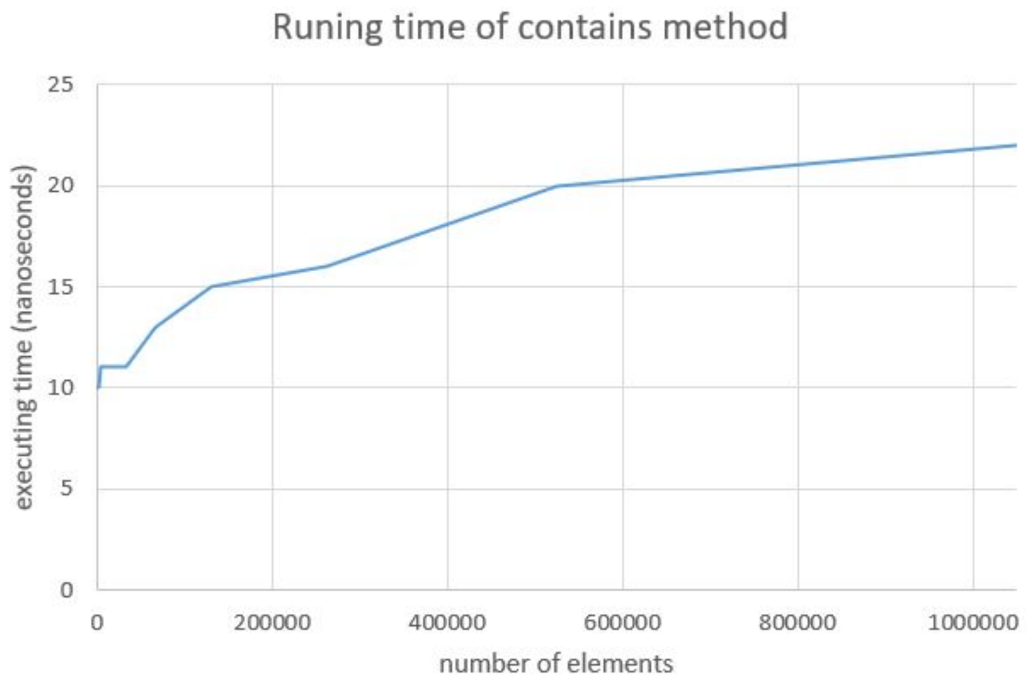
1. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development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

- If I use a List instead of a basic array, I need to used the .get() method to retrieve the element stored in the list. While for an array, I can use the [] operator to do so. In this respect, using a basic array will have a better running time efficiency.
- Also, for List, there is a method .add(), which can insert element to the end of the List or after a specific position. For adding an element into the basic array, I need to create a new array and copy the data before and after the insertion location separately into the new array, which takes extra space.
- List also already have the methods like .addAll(), .remove() and .removeAll(). Therefore, the program development time is less for using a Java List.

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

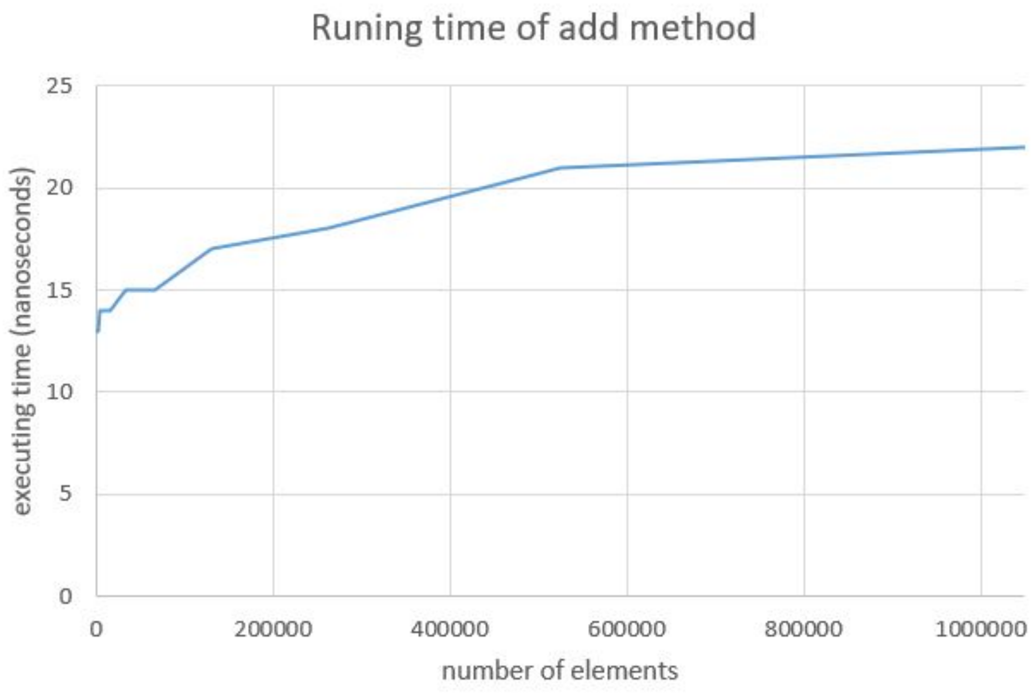
Since the array in the BinarySearchSet should already be sorted before performing .contains() method, we use binary search to retrieve the value. Therefore, the Big-O of the contains method will be O(log(N)), where N is the number of elements in the array. Also, the best case time will be that the value we want to check is at the middle of the array.

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?



As shown in the graph on the last page, I tested the method for Sets that contain 1024, 2048… to 2^20 elements. The running times in nanoseconds are shown in the graph. The growth rate of these running times are logarithmic thus match the Big-oh behavior.

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)?



As shown in the above picture, I again used the same samples used in question 3. In the worst case, the time complexity for adding an element would be O(N^2).

Upload your analysis document addressing these questions as a PDF document with your solution.