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.)

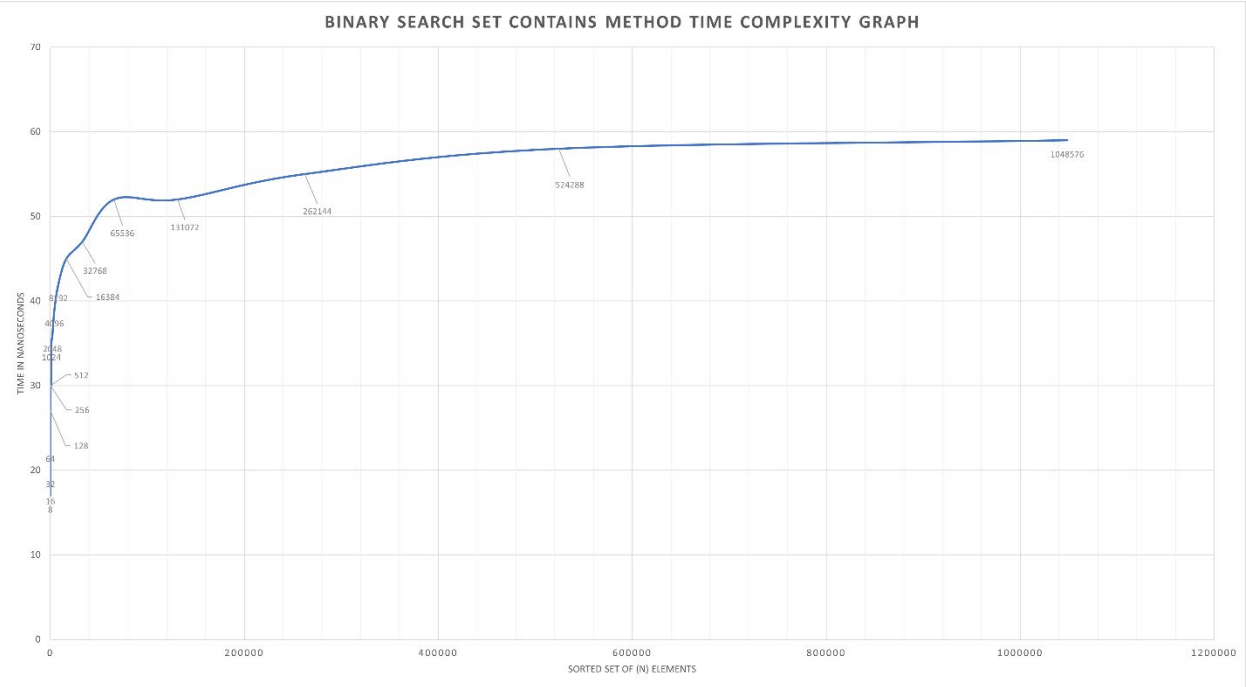
In program development time, I think it would have saved me a lot of time because ArrayList library has a lot of methods such as add that would have saved me a lot of time since it handles shifting the elements in the array. Also, I won't have to worry about the capacity of the array as the list library handles it automatically. It would be less complicated to deal with array lists compared to array by quite a lot.

In terms of efficiency, I think it would be more or less the same if not more efficient since array list also operates in a similar manner. I think memory management could be more efficient in the array list library.

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

The time complexity of the contains method is definitely O(Log N). That is because the it uses the binary search algorithm that splits the number of elements in half in every recursion. There are some edge cases in which the time complexity could be even O(1). For example if we search for the element that is in the middle index of the array, it will return the search within one iteration. Even in the worst case, the time complexity will be O(Log N).

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?

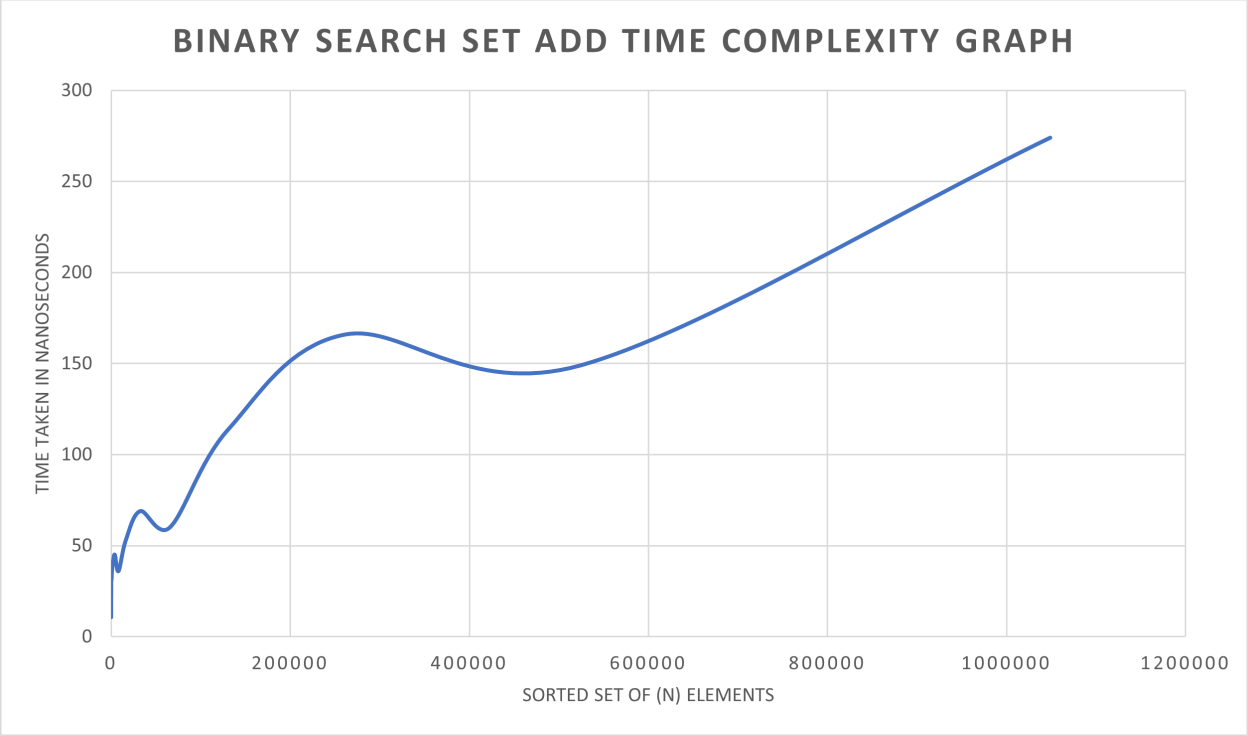


Yes it does match my expectations that I predicted in question 2.

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)?

a. It takes O(Log N) to find the index at which to insert the element.



c. In the worst case, the time complexity will be O(N).