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.)

Using a list rather than an array would have been more efficient in the growing and adding/removing from a list. Because of how the elements are stored in memory, lists can be added to and removed more easily. When we grow our array, we have to allocate new memory and copy everything over, which is inefficient. Using a list you can manipulate pointers(or references) to add/remove. The issue is access; you cannot use binary search on a list, because they are accessed through the head, then iterating all the way down the list until the element in question is found, giving it a big O(N). Using an array gives us random access to elements in the array, which is O(1).

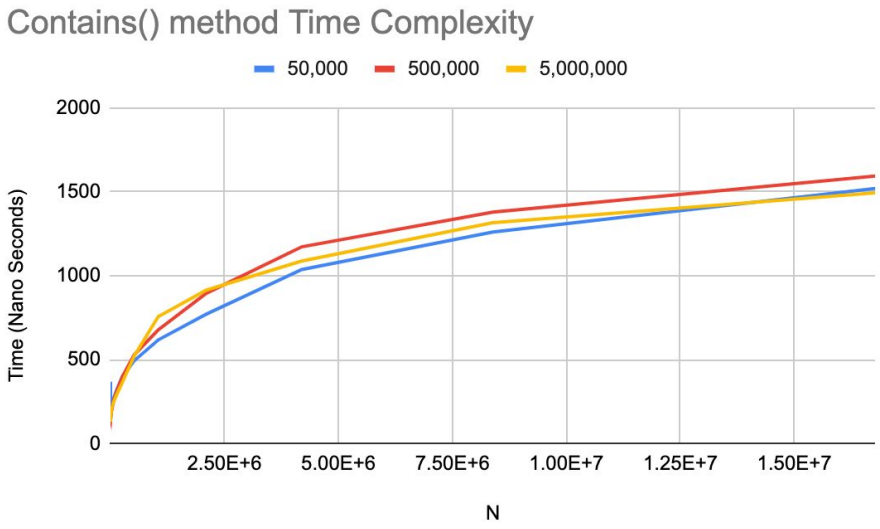
2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?

My contains method calls by binary search method which has a O (log n). Other steps involved are only single operations, therefore the contains method should also have a O of (log n).

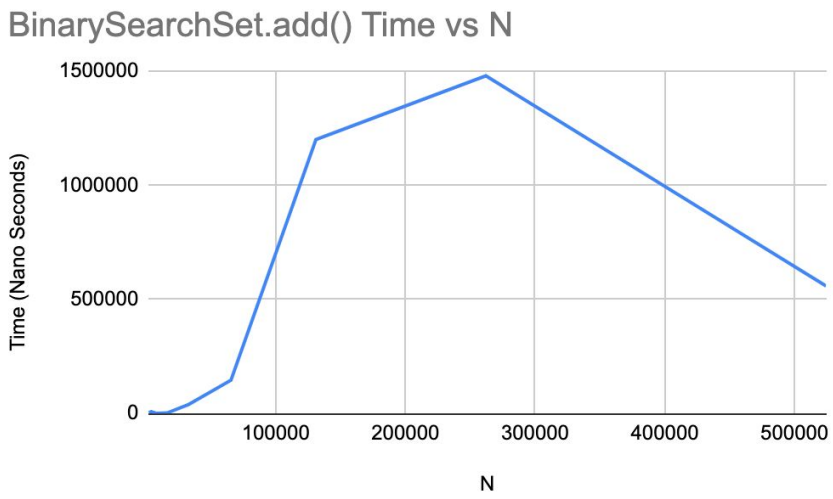
3. Plot the running time of BinarySearchSet's contains method, using the timing techniques demonstrated in previous labs. Be sure to use a decent iteration count to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match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

Yes, it becomes a O(log(n)) plot.

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)?



Big O of add is O(N log N). It calls binary search to find the correct position in which to insert the element(log(N)). Once it has the position it needs to add to the array, which in the worst case means shifting every element in the array which is O (N). Therefore, the time complexity of add is O(N log N).



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