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.)
Most of our functionality would have already been implemented if we had utilized a Java List. Java Lists already contain add, contains, remove, an iterator, sort, etc.

It looks like the Java list generally uses a linear search which will be dramatically less efficient than our binary search. Since we are guaranteed to have a sorted array utilizing binary search will speed things up dramatically (O(log(n)) -> O(n^2)). Program development would have been substantially faster though since only small changes would have had to have been made to the already provided functionality.

2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?
My method runs a binary search on our already sorted array and so I expect the behavior to be O(log(n)). Each iteration of our binary search loop cuts the remaining indices to search in half and should therefore approximate log_2_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?
It does! See figure 1 on the last page for details

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)?
Finding the correct location to insert the element is relatively quick - .add() run .contains() on the array which we've already seen is O(log(n)). Even in the worst case (where the element we're searching for is on one end of the array) we will only need to loop through log(n) times to find it, and the average and best cases will be even better.

The reason .add() ends up being fairly slow is that once that position is found all of the elements up to that position have to be viewed and shifted - an O(n) operation. For large n the cost to shift position dwarfs the cost to find that position so the overall behavior approximates O(n). See figure 2 on the last page for the graph!

Figure 1 -

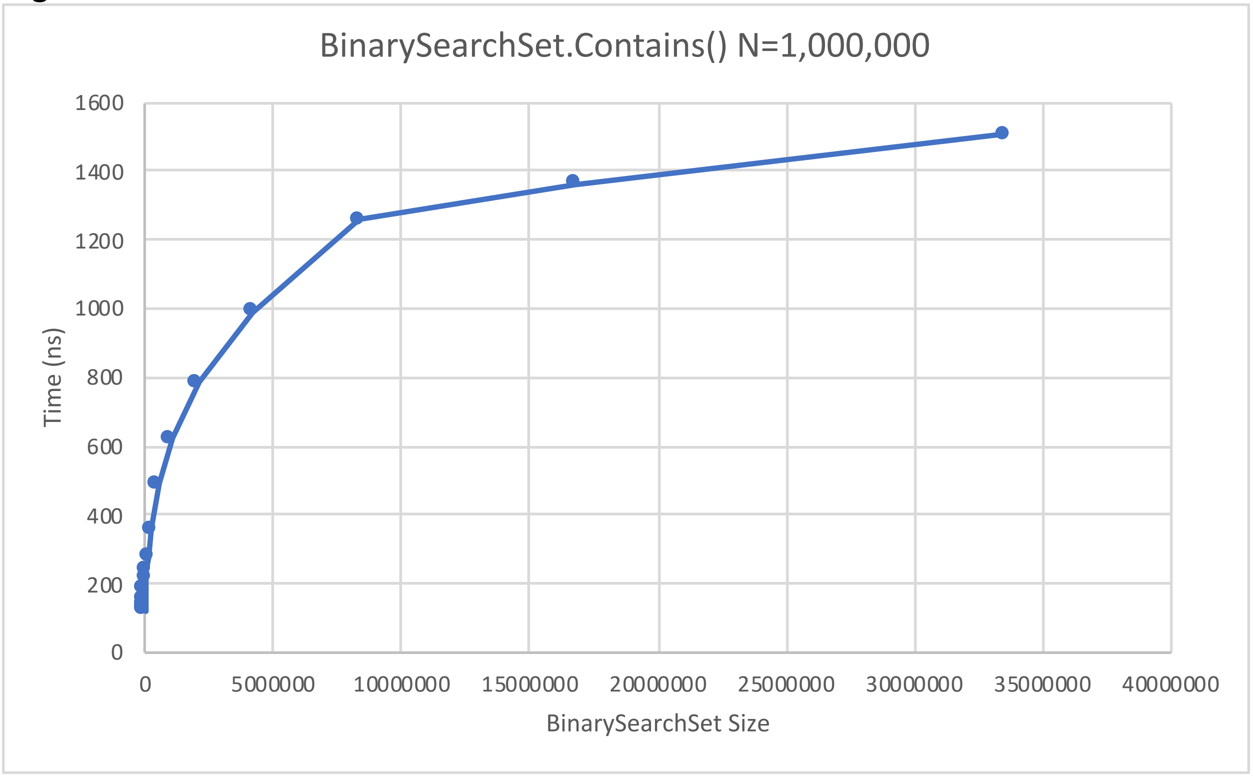
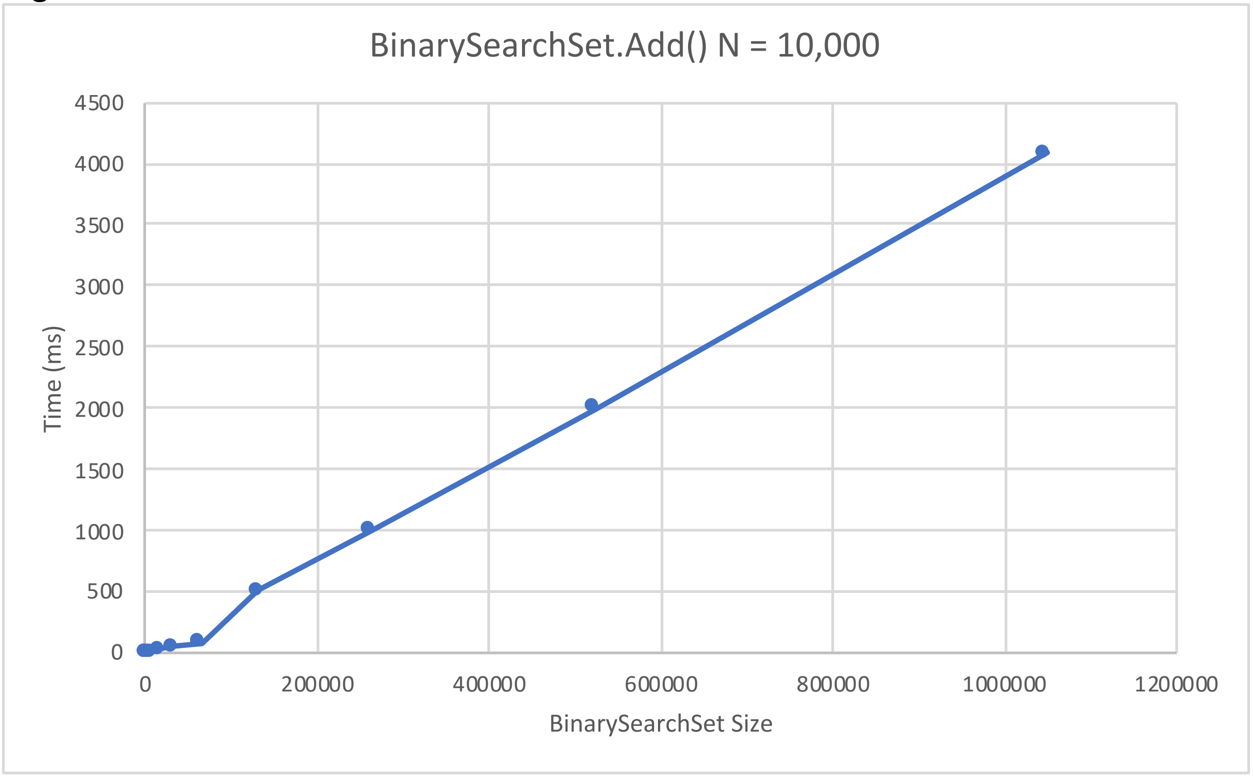


Figure 2 -


BinarySearchSet.Add() N = 10,000
Time (ms)
BinarySearchSet Size