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.)
ArrayList would have re-sized automatically, implementation and running time is wasted in re-sizing the array. During insertion and deletion, running time is wasted on shifting elements in arrays, this would not be the case with ArrayLists.

What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?
If the set has n elements, the binary search will take at most $\log_2(n)$ steps to either find the element or conclude that it's not in the set. This is because each step reduces the problem size by half. Big-O notation is concerned with the upper bound, so the time complexity of the contains method will be O(log n).

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 the normalized range of execution time remains pretty constant, indicating the O(logN) behaviour predicted.

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)?

The binary search algorithm works by halving the search space with each iteration. This results in a logarithmic time complexity.Since the algorithm halves the search space with each comparison, the time complexity of the binarySearch method is **O(log n)**, where **n** is the number of elements currently in the set.