**U1426365 JinChing Jeng**

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.)

Both in running time and in program development time, Java List might be more efficient than a basic array when sorting set because List is already a Collection that has several methods to call. To sort a set, we need to convert a set to List and then use Collections.sort() API and again convert back the list to a set. However, if I want to have backed the sorted set with a basic array, I can only use Collections.sort() to sort an array after creating an ArrayList of given array items. That is one more step to transfer it.

Ex.

Integer[] integers = *new* Integer[] {1,8,3};  
 Collections.sort(List.of(integers));

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

Best case: O(1), when the central index directly matches the desired value.

Worst case: O(log N), when the desired value is at either extremity or doesn’t exist in the list.

The interval is narrowed to half every time until the value is found or the interval is empty, so when the number of inputs is N, then it executes log N steps.

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 matches O(logN) that the rise of the curve decelerates as N increases, but sometimes it looks like O(N). It might depend on the laptop’s efficiency and background activities.

![Chart, line chart

Description automatically generated]()

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)?

Because the set is already sorted, I can use binary search to locate the correct position to insert the element, and the time complexity is O(N).

Time complexity explanation:

Step1: Using binary search to find position to insert is O(log N)

Step2: Using for loop to switch two elements is O(N)

Step3: O(log N) + O(N) can be simplified as O(N)

