Assignment 3 Q&A

Ray Ding

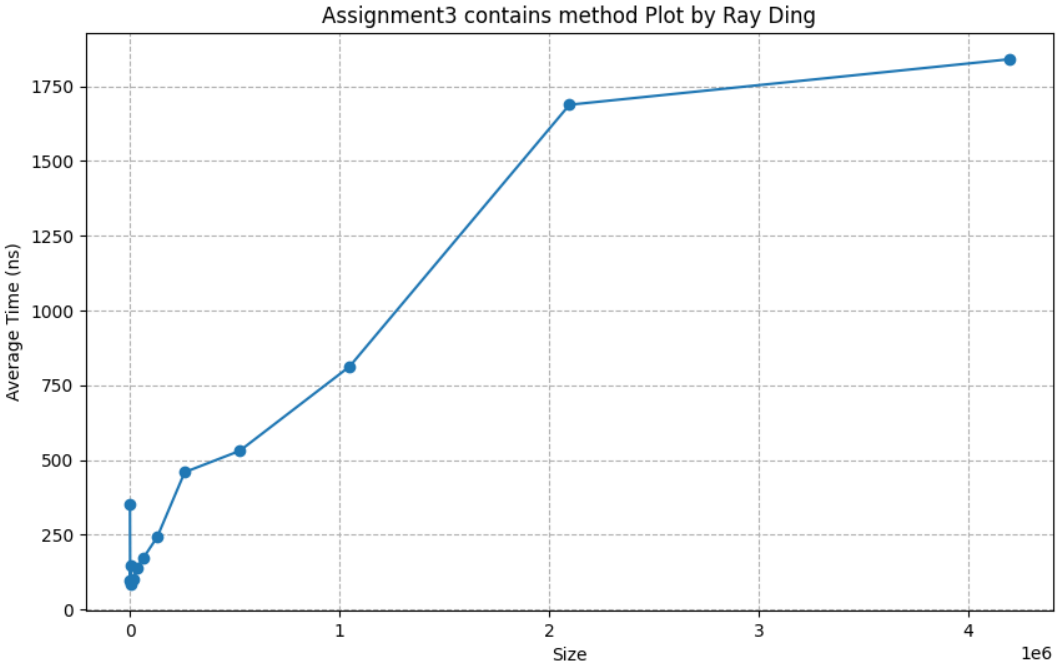
1. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)
Implementation Differences:

Dynamic Sizing: With a Java List, the sorted set would dynamically adjust its size. This means less manual resizing logic compared to using a basic array.
API Methods: Java Lists offer a rich set of methods for insertion, deletion, and searching, which could simplify the implementation.
Memory Management: Using a List abstracts away the lower-level memory management required with arrays.
Efficiency Considerations: Running Time Efficiency: For certain operations like adding or removing elements, a List might be less efficient than an array, especially if a LinkedList is used. However, for ArrayList, the efficiency would be comparable.
Program Development Time: Using a List would likely reduce development time due to easier handling of dynamic sizing and the availability of high-level methods.
Space Efficiency: Lists generally consume more memory than arrays due to additional object overhead.

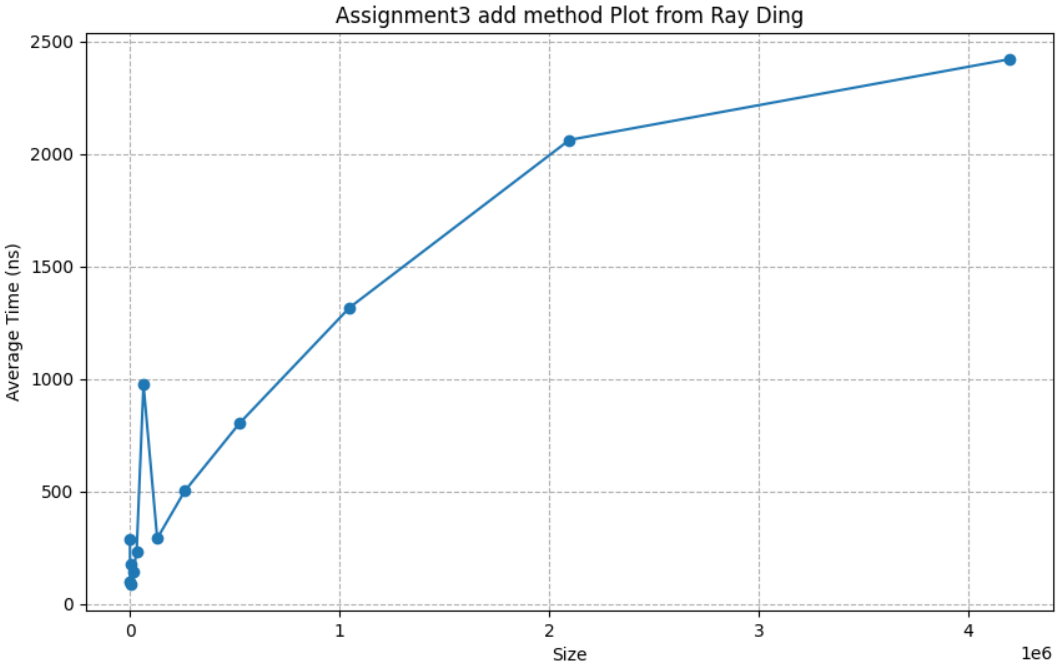
2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?
The contains method in a BinarySearchSet, which uses binary search, would have a time complexity of O(log n). This is because binary search divides the search space in half with each step, significantly reducing the number of elements to be checked compared to linear search.

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?

Iteration count is 100.
As shown by the figure below, the growth rate does align with the O(log n) complexity.



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)?



The add method performs a binary search similar to the contains method. This also has a Big-O of O(log N), as shown by the figure above.

In the worst case (inserting at the beginning of the array), as displayed by the figure below, this requires shifting all N elements, which has a time complexity of O(N).