Rian Brumfield Analysis_

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.)

Program development time would have been significantly reduced if we backed Sorted Set with an ArrayList because we could have taken advantage of methods like add(index, value), contains(), and remove(), that are already coded. As far as program running time, the time would be more or less the same because the heart of the Sorted Set class is the Binary Search which has a time complexity of O(LogN).

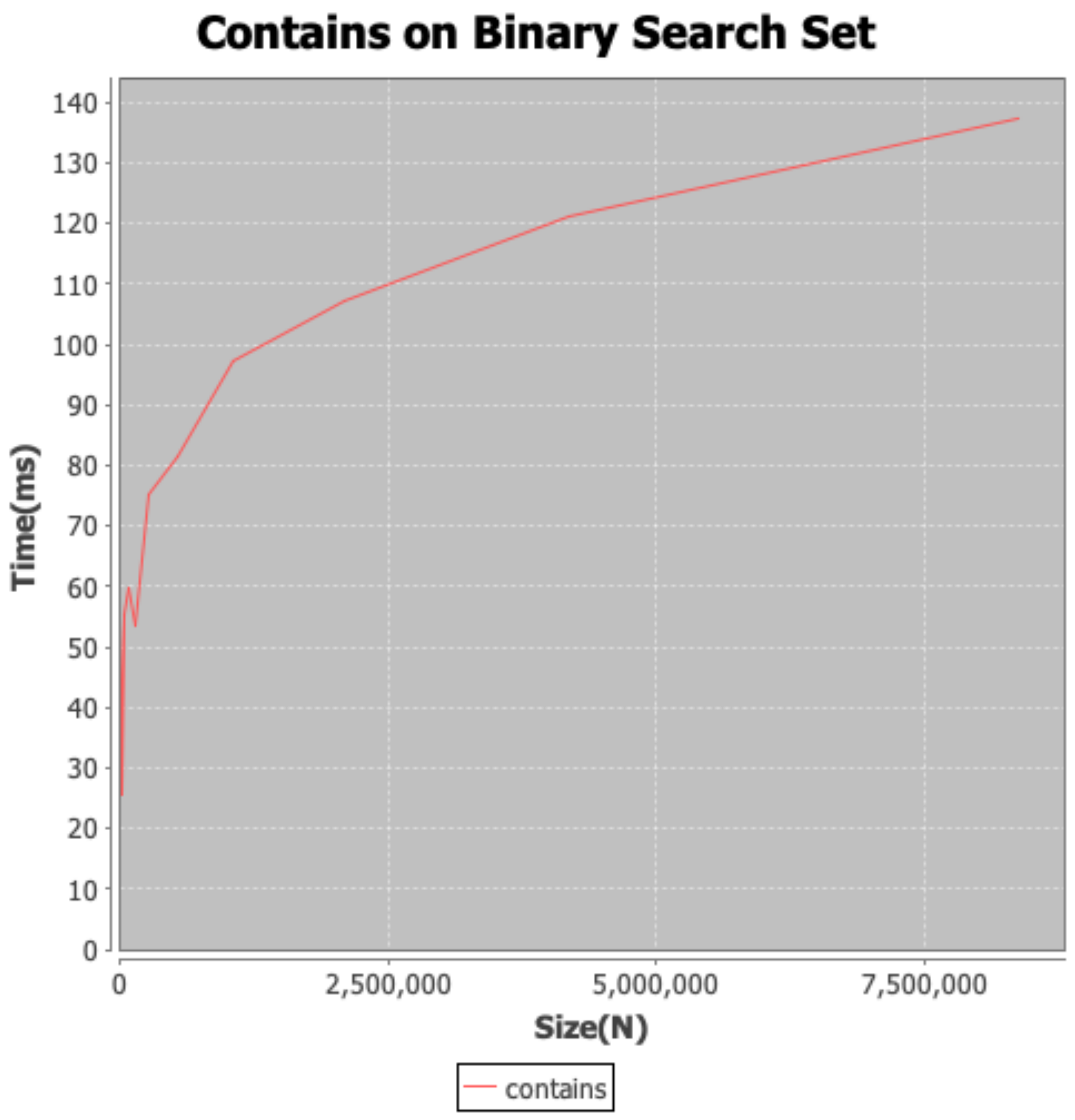
For example, to take advantage of the add(index, value), whose time complexity is O(N) method from JavaList, we would have to run the binary search to find the index, whose time complexity is O(LogN). Therefore, the total time complexity to add a value would be O(NlogN).

Similarly, the add function I wrote calls binary search O(LogN) , and then shifts every other element down one in the array, O(N), so my add time complexity is still O(NlogN).

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

My contains method has Big-O complexity of O(Log(N)) because it relies on Binary search to find the element in the set and other small, O(1) complexity for other operations. Binary search has O(Log(N)) complexity because half of the data set is removed for each layer of the 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?



According to the graph above, the contains() method appears to be O(logN) in complexity, as predicted. If the complexity were O(NlogN), O(N^2), or O(2^N), then the rate of change between would be increasing as N tends towards infinity. Furthermore, if the complexity were O(N), the rate of change between points would be constant. O(LogN) is the only potential function whose rate of change decrease as N tends toward infinity.

Therefore, in order to confirm the above graph models a logarithmic function, I calculated the average rate of change over three intervals.

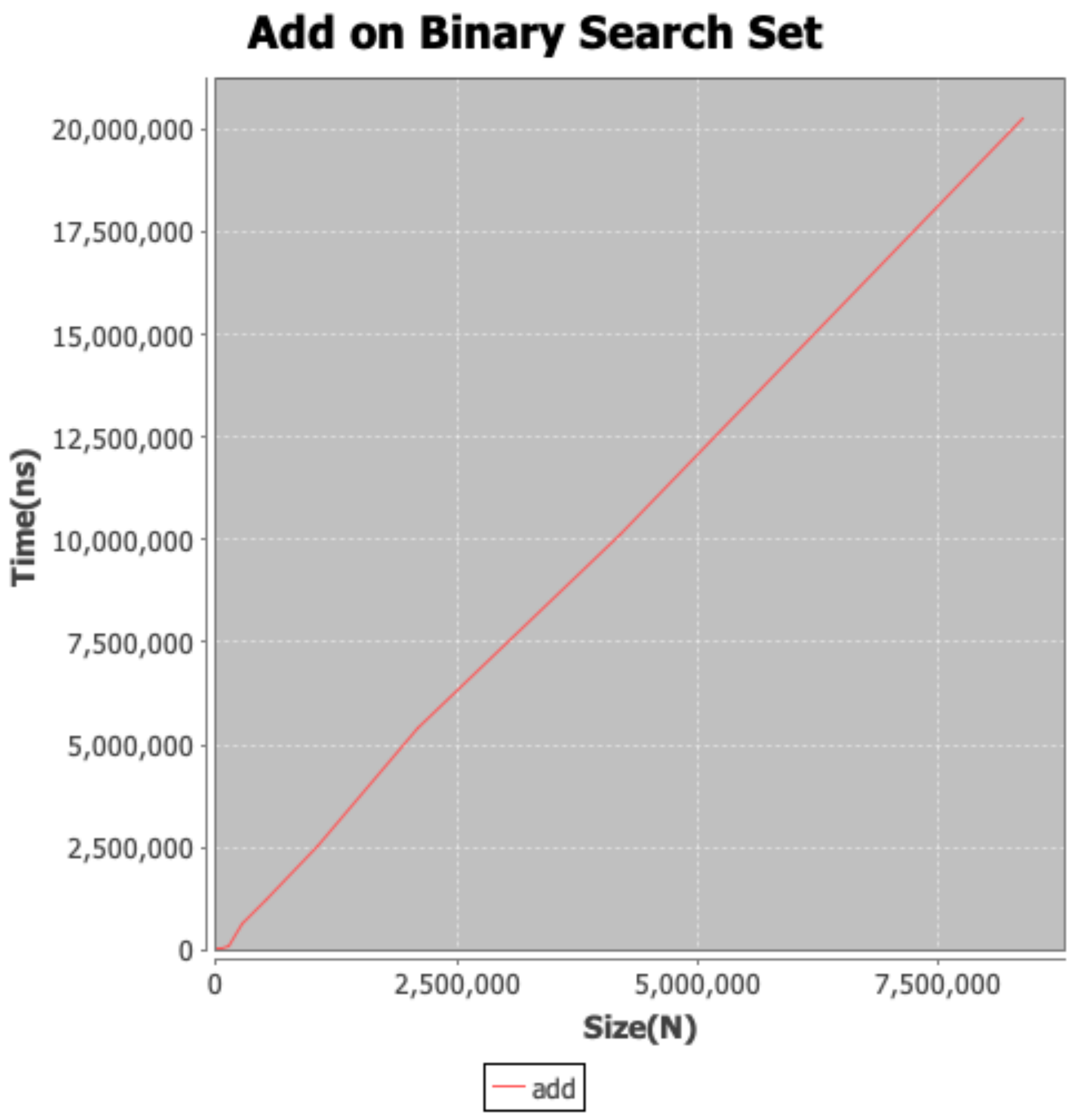
For the first interval, [1634, 2768], the rate of change on this interval calculated to approximately 953.58E^-6.

For the second interval, [31072, 262144], the rate of change on this interval calculated to approximately 128.55E^-6.

For the final interval, [2097152, 4194304], the rate of change on this interval calculated to approximately 7.75E^-6.

The rate of change between points is approaching 0 as N tends towards infinity, which implies the function is indeed logarithmic.

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 worst case for finding the index to add an element to the Binary Search Set is O(LogN) because the binary search which finds the index is O(LogN) complexity. Moving the elements in the array over one to make room for the new element is at worst O(N). Therefore, the complexity of the adds method is at worst O(N + Log(N)). Because N is more dominant than logN, the complexity would be O(N), which is consistent with the seemingly linear graph above.

To prove the above data represents a linear function, I calculated the average rate of change on three intervals.

**For interval one, [8192, 16384], the average rate of change was 0.6866.
For interval two, [16384, 32768], the average rate of change was 0.6696.**

For the final interval [32768, 65536], the average rate of change was 0.6340.

The rates of change over the three intervals are more or less constant, which implies the modeled function is linear.

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