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.)

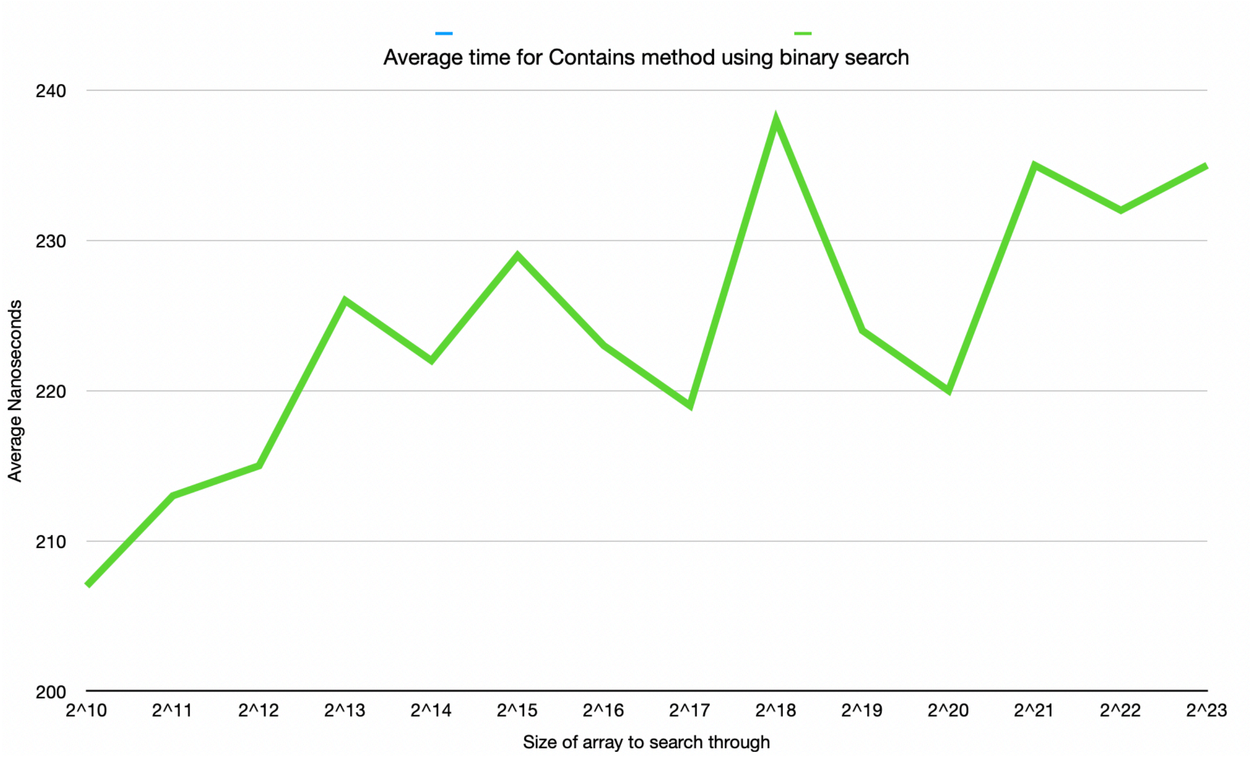
We could have used the built in functions for the list we used, allowing us to not worry about growing the list ourselves
We would have been able to code much quicker, but the runtime would most likely not change much, possible going slightly slower

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

O(lg(N))
It runs constant time functions in a loop that runs lgN times because we cut it in half each time

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?

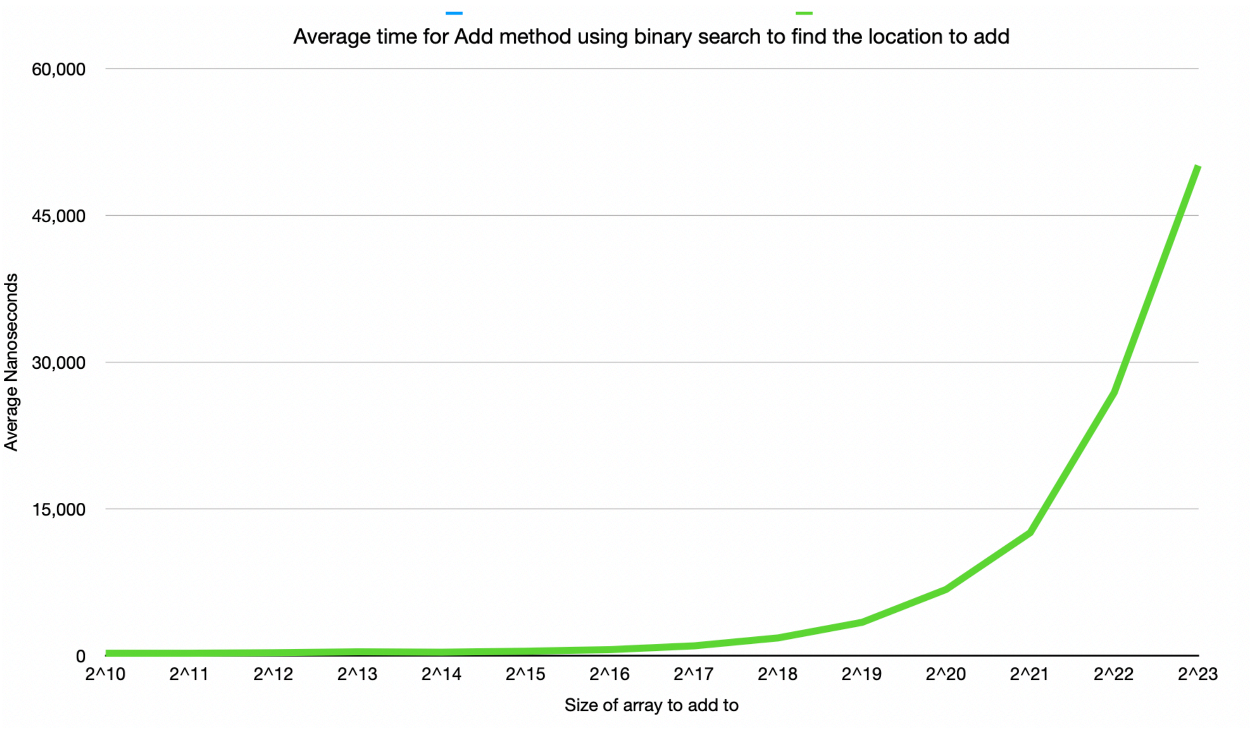
The average slowly goes up with a lot of noise, and seems to match the timing of lg(N) from the question above



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)?

It takes N^2 time to find and insert a random location in the array
O(N^2)


Average time for Add method using binary search to find the location to add
Average Nanoseconds
2^10
2^11
2^12
2^13
2^14
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
Size of array to add to