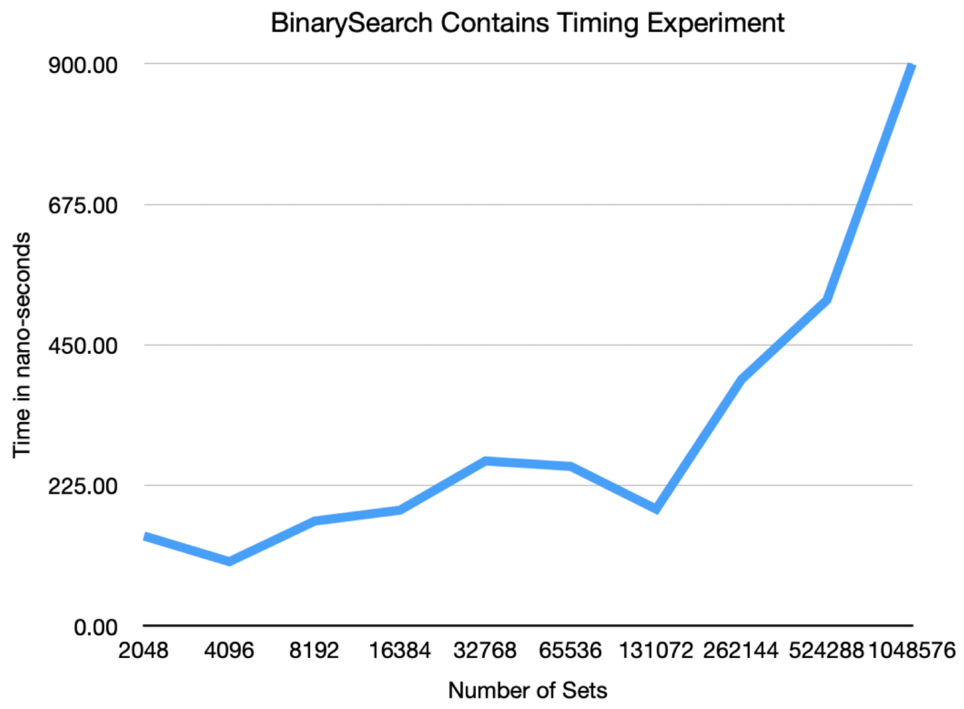
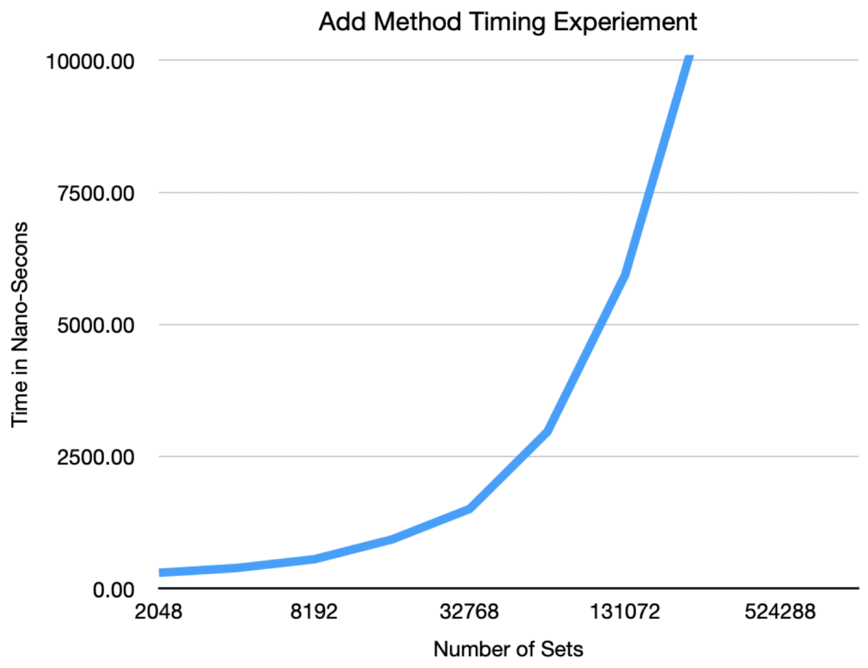
1. Dynamic sizing would be different. With a List, there's no need for a separate method like growArray() to dynamically increase the capacity of the underlying data structure. Lists in Java, such as ArrayList, automatically handle resizing when needed. Operations that involve array manipulation, such as System.arraycopy, would not be necessary with a List. Lists automatically handle the resizing and shifting of elements when new elements are added or removed. The iterator implementation might be simpler with a List. You can directly use the iterator provided by the List interface (List.iterator()), eliminating the need for a custom iterator class. Using a List may result in more efficient insertions and deletions, especially when the size of the set changes frequently. Lists handle resizing and shifting more efficiently than manual array manipulation.Random access to elements in a List is generally O(1), similar to array access.
2. The contains method in the BinarySearchSet class exhibits a Big
O of O(log n) due to its implementation of binary search. This efficiency arises from the algorithm's ability to consistently halve the search space, whether utilizing a provided Comparator or relying on the natural ordering of elements. As the size of the set grows, the number of iterations required for binary search increases logarithmically with the number of elements (n).



3.

The growth rate of the running times in the timing results does align with the expected Big-O behavior predicted above (O(log n)). As the set size (n) increases, the running times don't grow linearly but increase at a rate that corresponds to a logarithmic growth.



4.

In the worst-case, the time complexity for locating the position to add an element in a binary search set is O(log n), where n is the size of the set. This is because binary search involves repeatedly dividing the search space in half until the correct position is found, leading to a logarithmic growth in the number of comparisons.