Performance Analysis

Mihai Jiplea - Yangfan Zhang

Introduction

The purpose of this exercise was to implement and compare the efficiency of three specific data containers (Skip Lists, Bloom Filters and Randomized Binary Search Tree) that are built with randomized algorithms.

In practice, Skip Lists and Randomized Binary Search Trees can yield logarithmic complexity in all the three basic operations (add, delete and find) while Bloom Filters give a constant time for these operation, but impose certain correctness trade-offs.

All the implemented data structures do not allow duplicate entries
.

Performance overview

Because of the randomization factor, the listed complexities are the expected values (which can be deduced in theory by means of probabilities) rather than exact the empirical data.

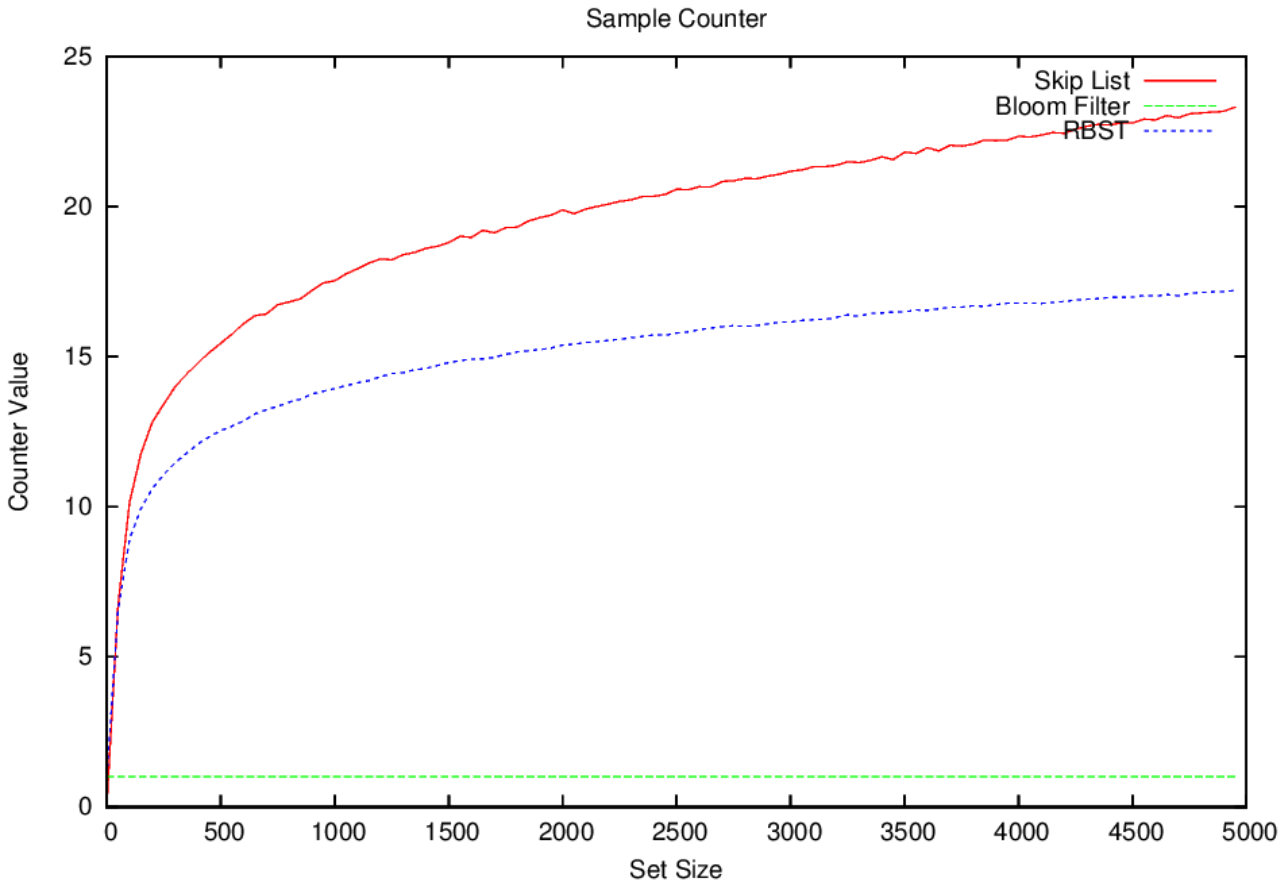
Add

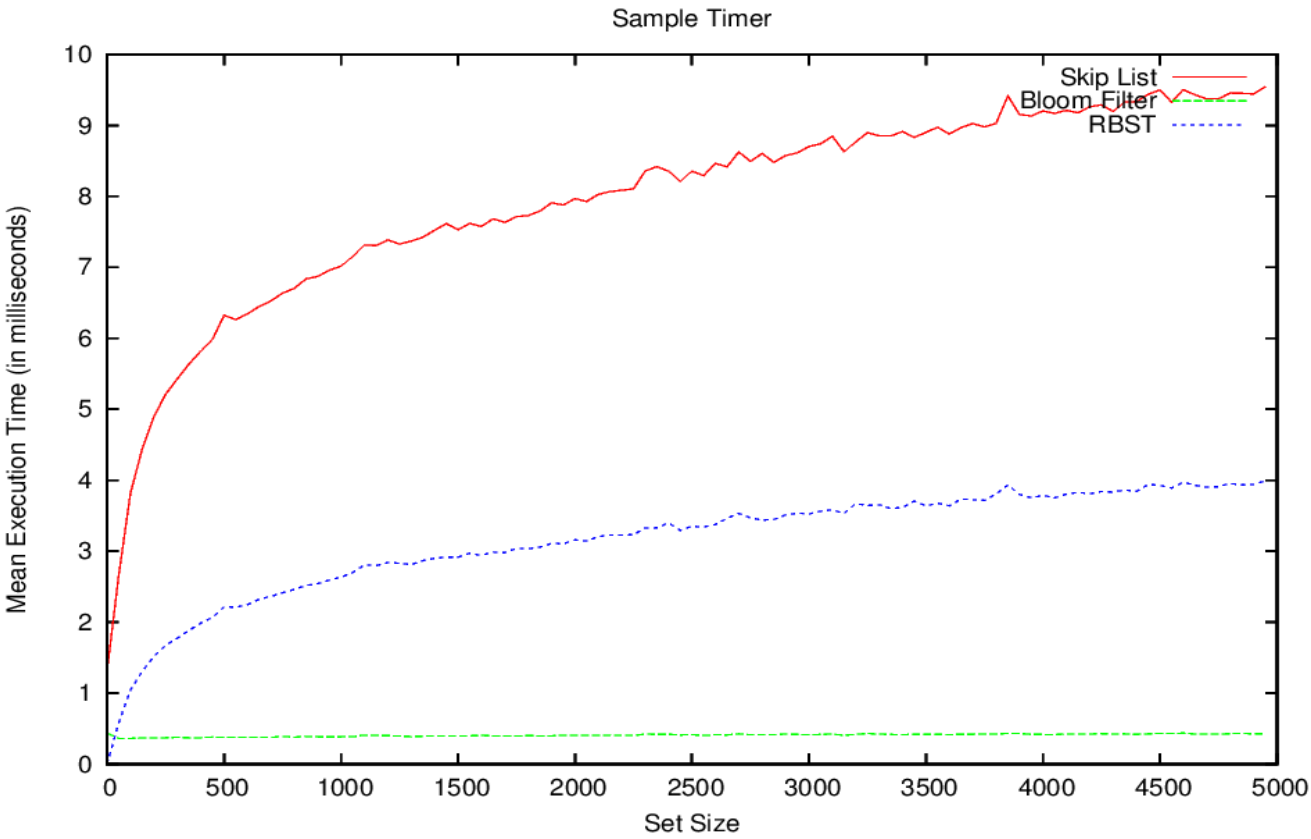
The add method inserts an element into a data structure in its ordered position.

Expected complexities for add methods:

Data Structure	Skip List	Bloom Filter	RBST
Complexity	$O(\log_2 n)$	$O(1)$	$O(\log_2 n)$

The above complexities are seen in the following graphs:





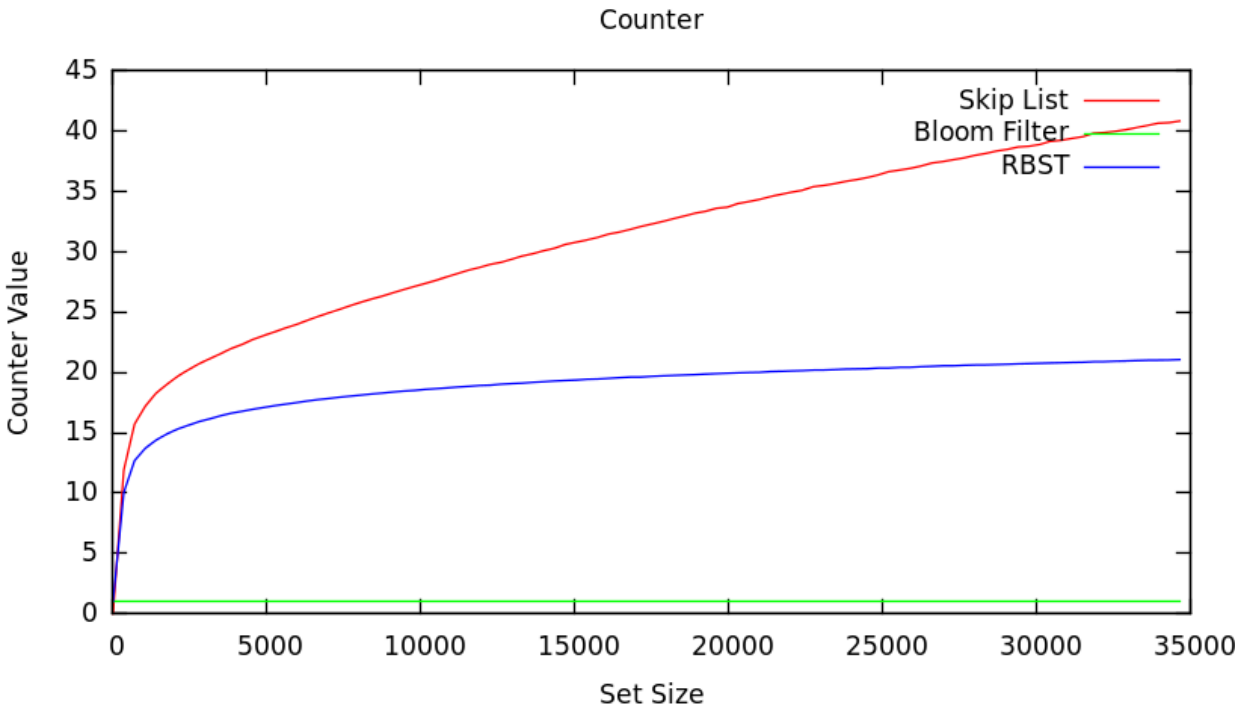
As we can very well see, bloom filter is the most efficient among the three. Adding an element takes constant time regardless of the size of the existing set.

The logarithmic complexity is just an overall tendency of the curvature of the graph of the Skip Lists and RBST. This is a result of the randomization factor.

We can also observe that, although both maintain a logarithmic tendency, RBST is more efficient than the Skip Lists. This could be because RBST only adds the new element to its correct position once, while Skip Lists may need to add the element to multiple levels in the list.

In fact, the randomization factor will disturb the efficiency of the Skip Lists further when the data set is large. We can have more nodes added at unfortunate positions or with unfavorable heights which will require more calls to add. As a result, the Skip Lists will no longer maintain the binary logarithmic behavior and tend towards a liniar complexity. This is shown in the following diagram.

In the following diagram, when we set the data set size to be 35000 instead of 5000, Skip Lists no longer have $O(\log_2 n)$ complexity. The gradient of the graph does not approach zero as the set size increases. RBST still has $O(\log_2 n)$ performance complexity while bloom filters still have constant time complexity.



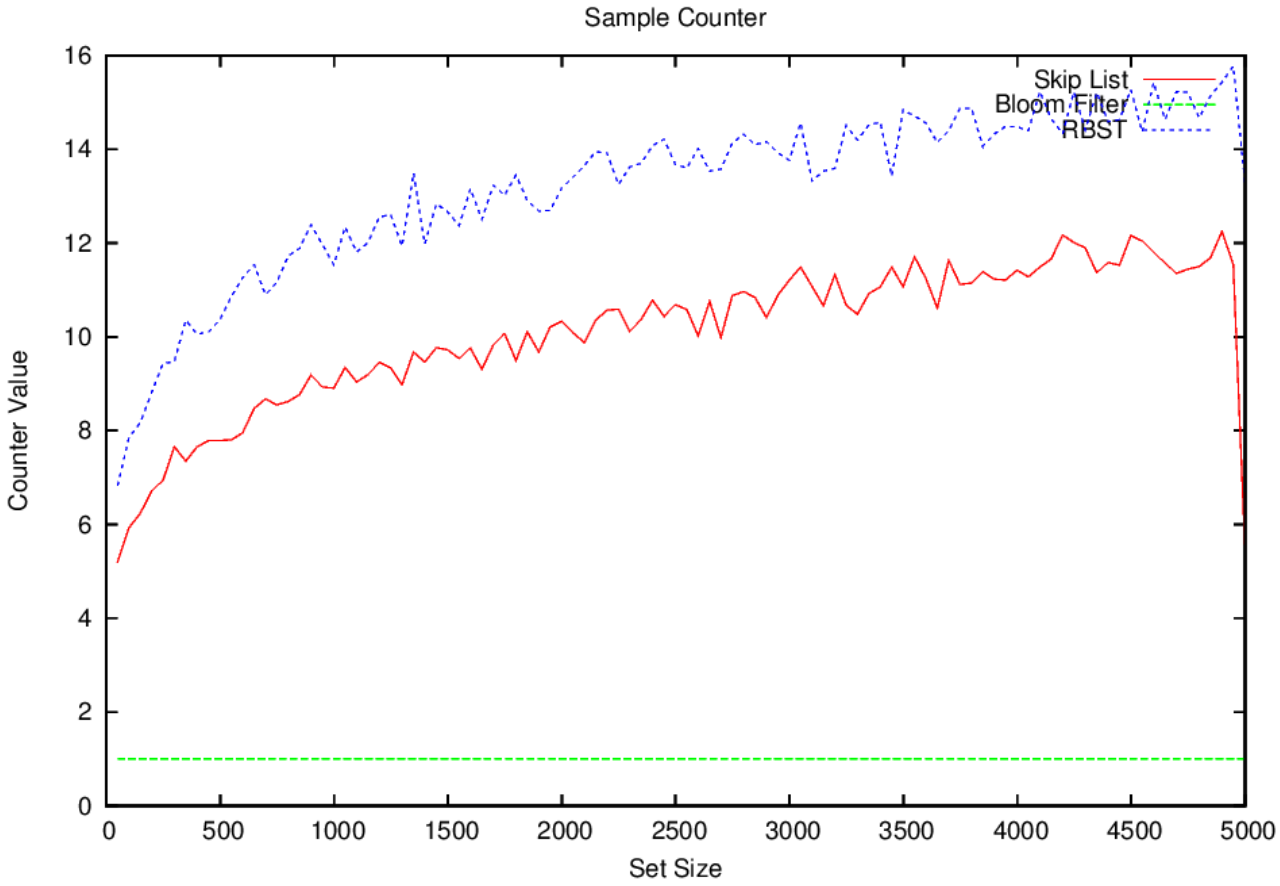
Delete

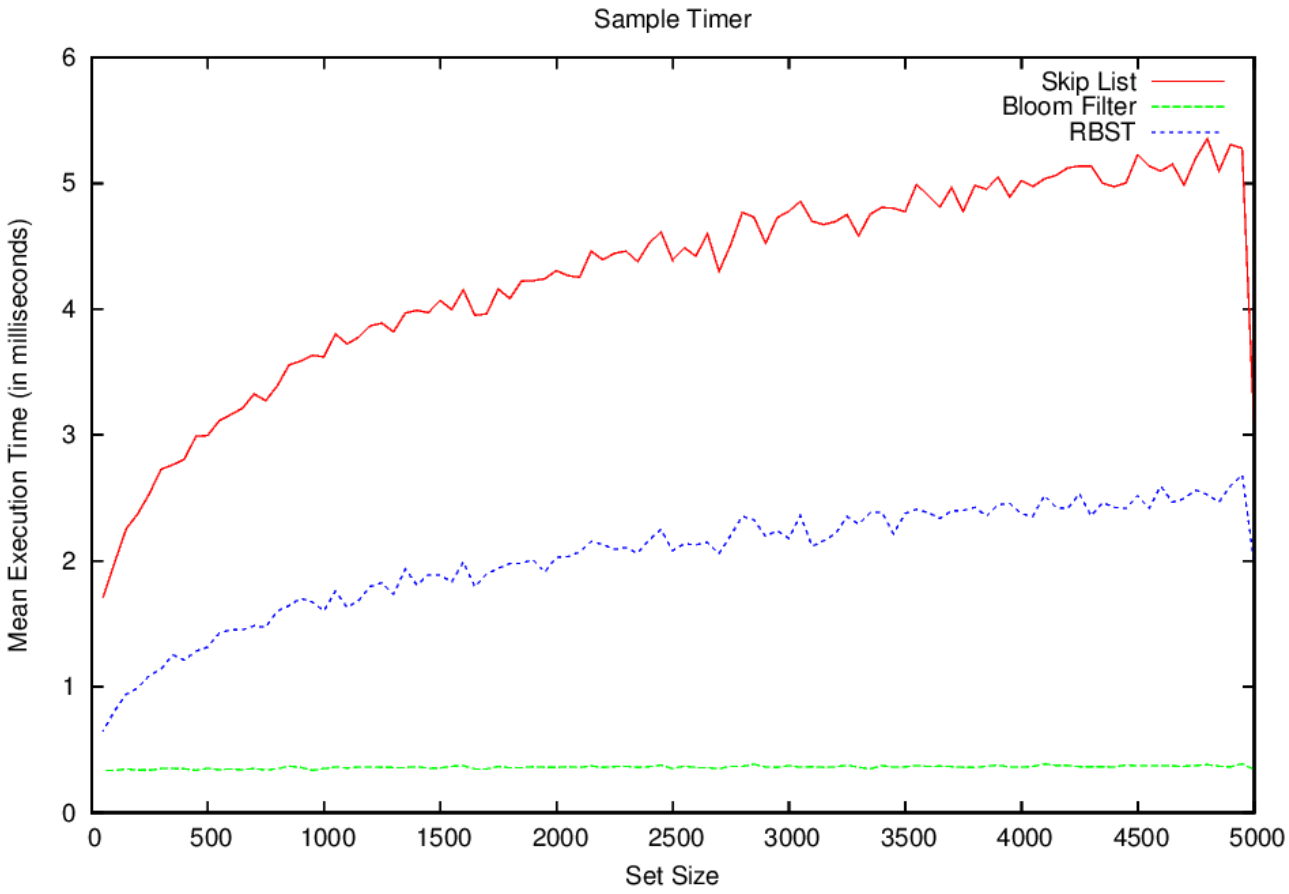
The delete method removes a certain element from the data structure.

The expected complexities are the following:

Data Structure	Skip List	Bloom Filters	RBST
Complexity	$O(\log_2 n)$	$O(1)$	$O(\log_2 n)$

As it can be easily seen, the complexity of this method is also illustrated in the graph below:



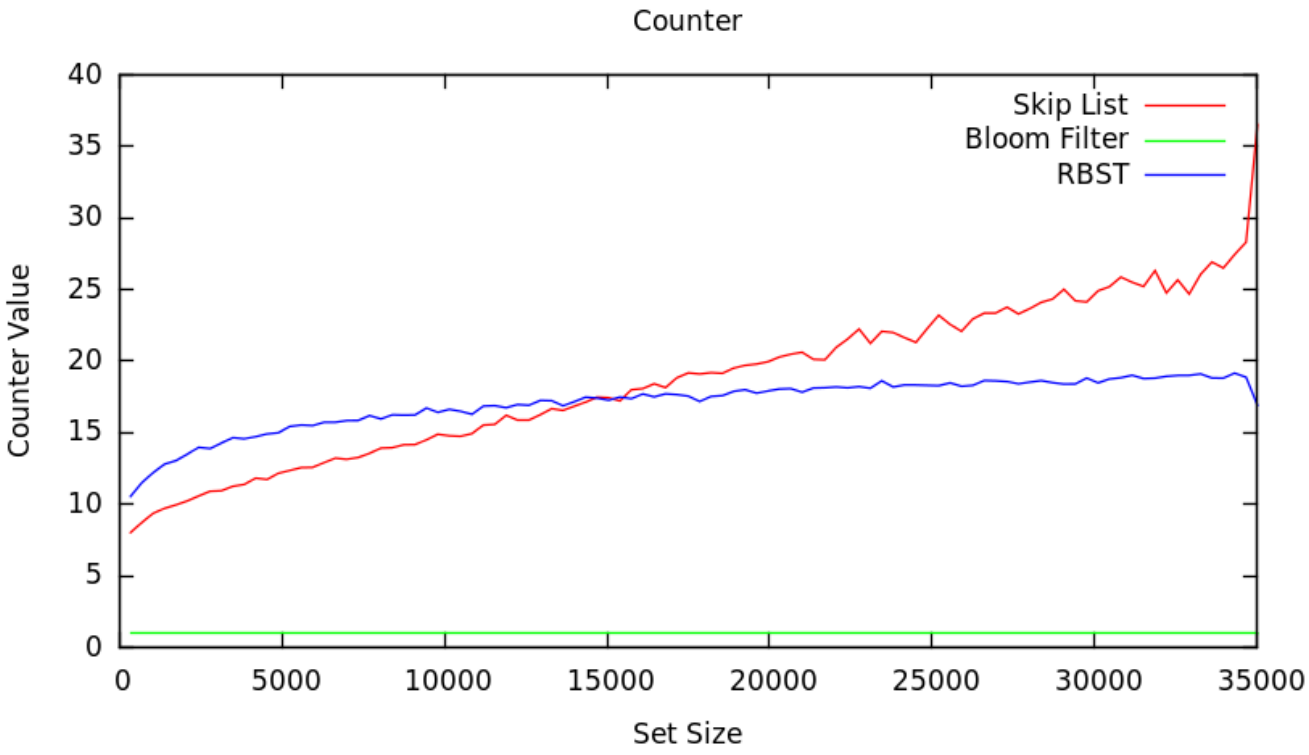


An interesting observation here is that the graphs of the Skip List and RBST tend to have more spikes than the add method do. This might be due to the fact that sometimes deletion can just take constant time. For example, if a delete query is ran on an RBST for deleting an element which happens to be the root, the running time will be much better than when trying to delete leaves at deeper levels.

Regarding Bloom Filter deletion, we have not implemented a counting bloom filter and the current one will violate the data structure's rules and give false negatives in the event of hash collisions. A counting Bloom Filter will reduce this risk significantly (by permitting up to 2^k hash collisions, depending on an arbitrary chosen k), but it will also require more memory space. The current bloom filter implementation gives constant time complexity.

When we have changed the number of deletions to 35000, we obtained the following graph.

The RBST still behaves logarithmically while skip list takes increasing counts as the data set size increases:



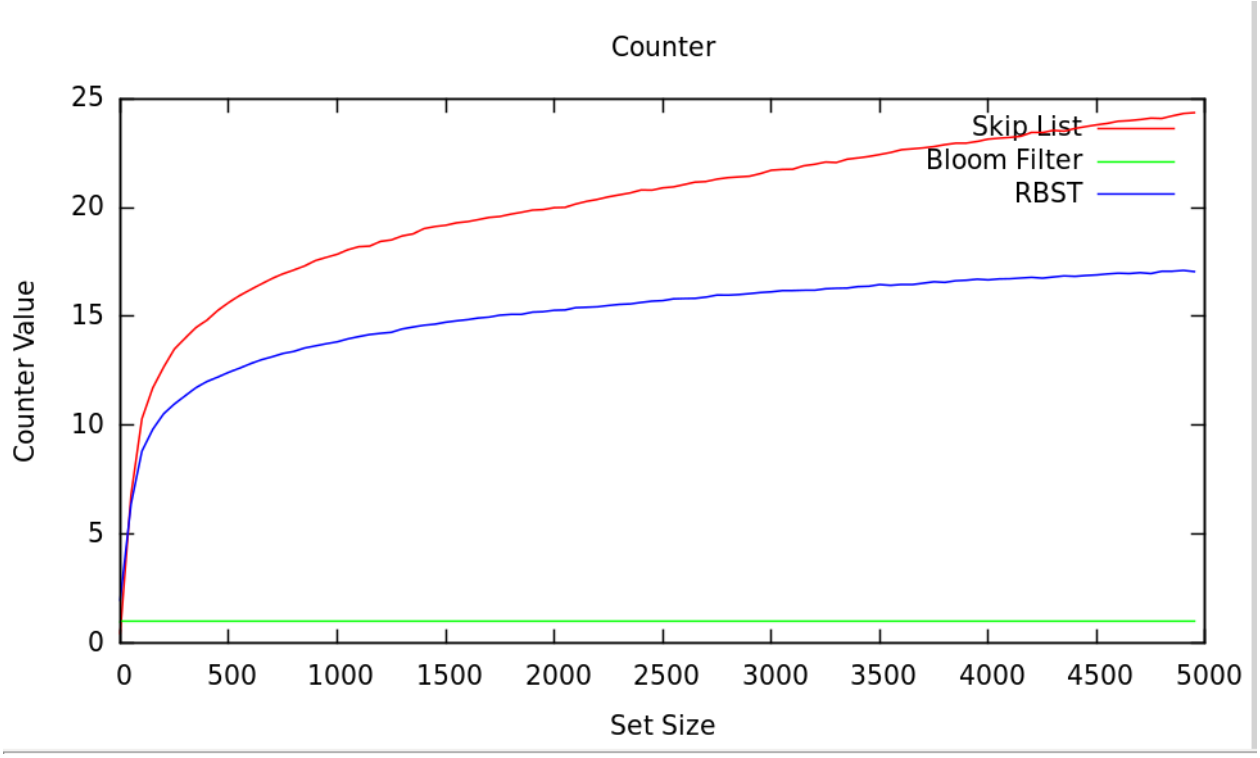
Find

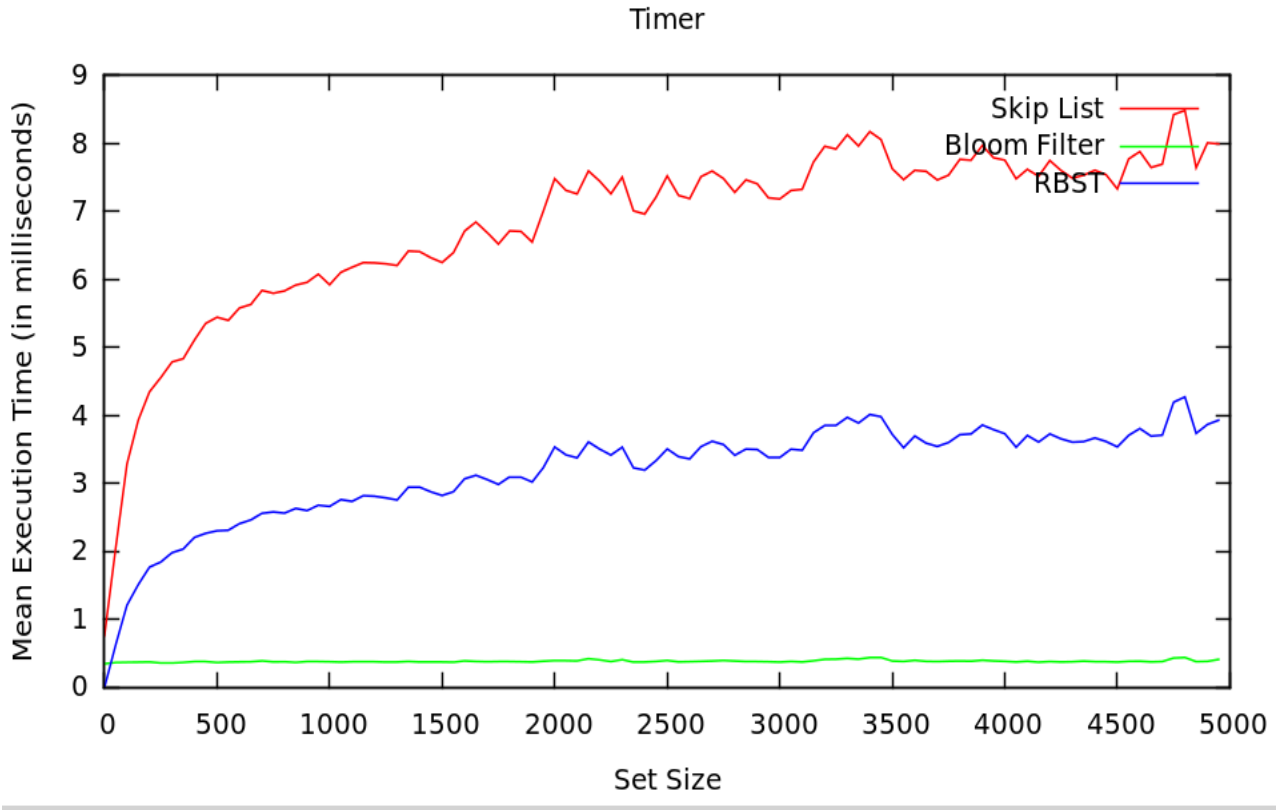
The find method will check if a certain key is contained in the data structure.

The expected complexities are the following:

Data Structure	Skip List	Bloom Filter	RBST
Complexity	$O(\log_2 n)$	$O(1)$	$O(\log_2 n)$

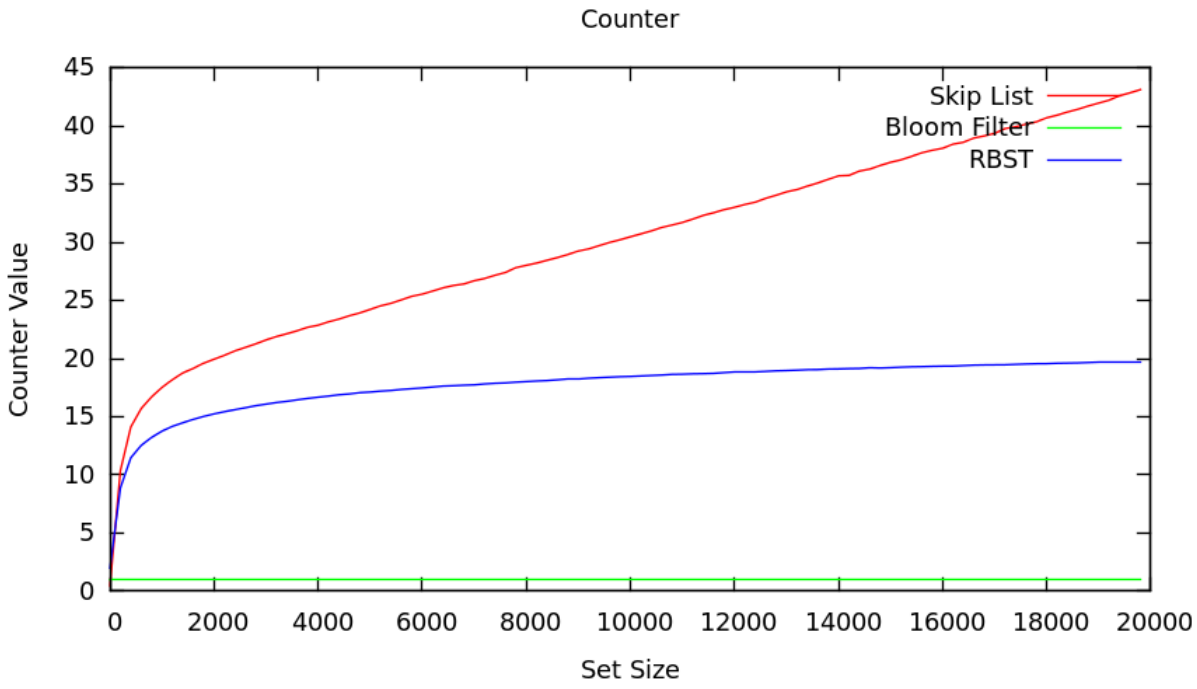
The graphical output again illustrates the previous mentioned complexities for each data structure:





The RBST and the Skip Lists still maintain a logarithmic tendency while the Bloom Filter takes constant time to return an answer. Although Bloom Filters are the most efficient out of the three data containers, there is a certain probability that it will return false positives.

When we increase the data set size to 20000, the graph is similar to the graph generated by add method with a large data set size. The reason is similar as discussed above.



Conclusion

Data containers that use randomized algorithms can be very efficient in practice and yield logarithmic or constant time complexities in most cases for their required operations.

Bloom Filters yields constant time performance and are the most efficient among the three. However, it might give false positives. The probablility of have false positives can be reduced by either using better hash functions, enlarging the bit vector or implementing a counting Bloom Filter.

Overall, RBST is more efficient than Skip Lists, especially when dealing with large data set.