**Comparing the Performance of Cuckoo Hashing with java HashTable, HashMap & HashSet**

**Problem definition:**

In this project we compare the performance of my implementation of Cuckoo Hashing with java’s implementation of HashTable, HashMap & HashSet. The criteria for comparing the performance of the above data-structures is the time taken per million operations for add(), contains() and remove(). Smaller time values represent a better performance and vice-versa.

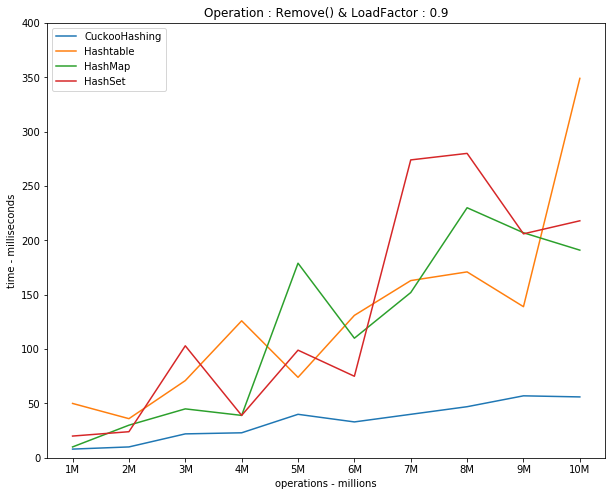
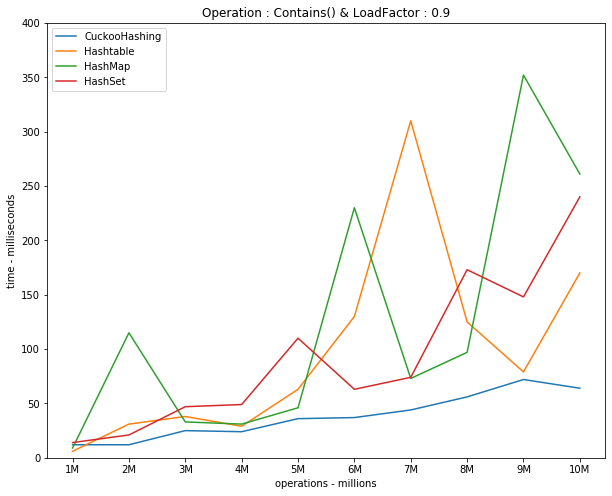
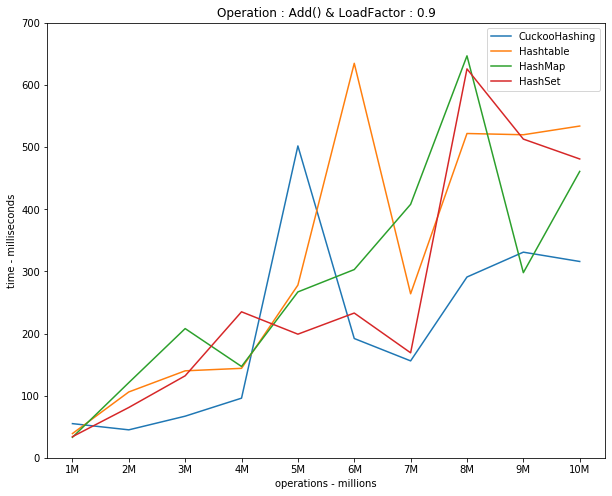
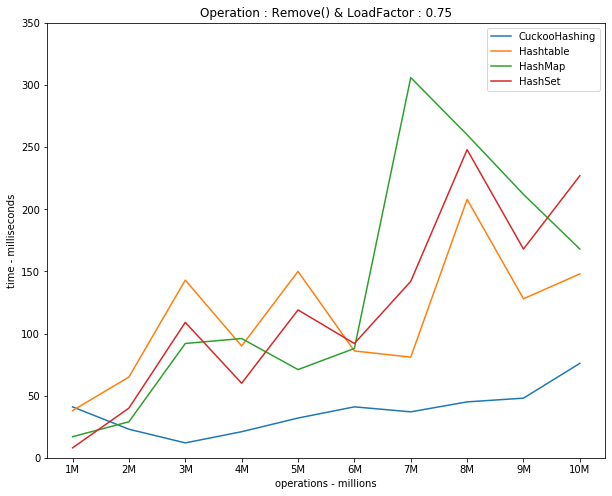
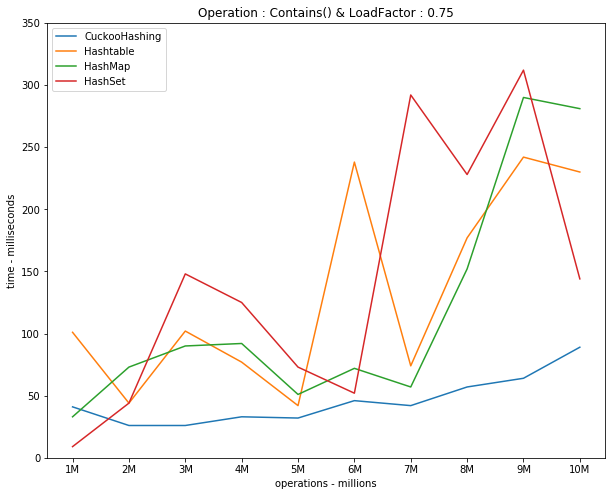
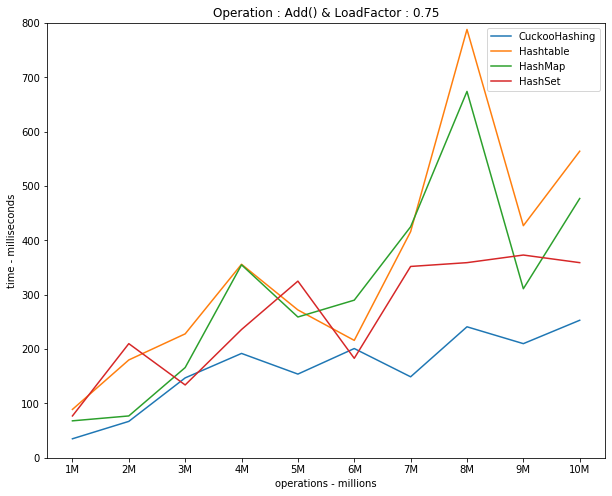
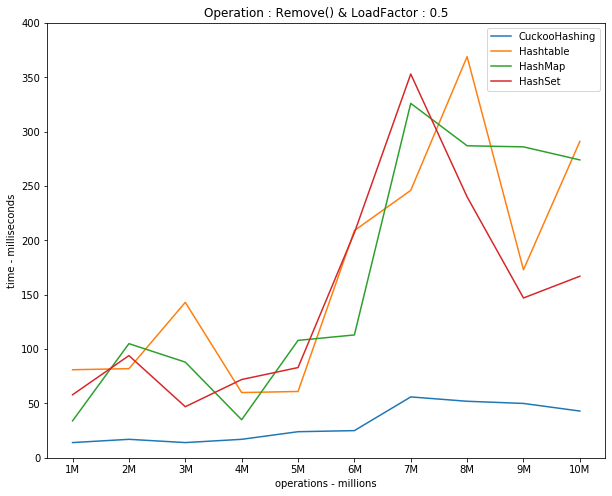
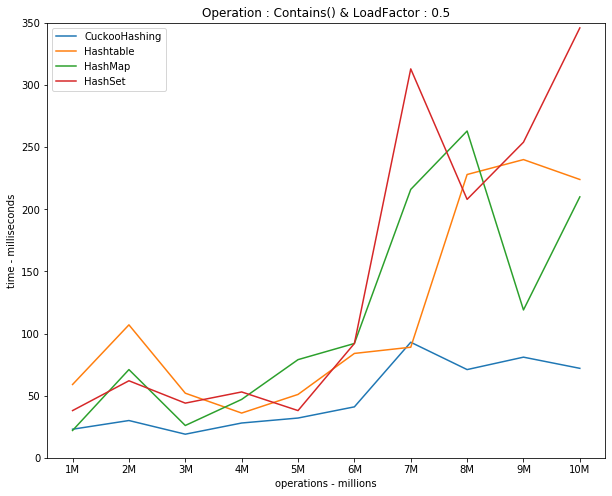
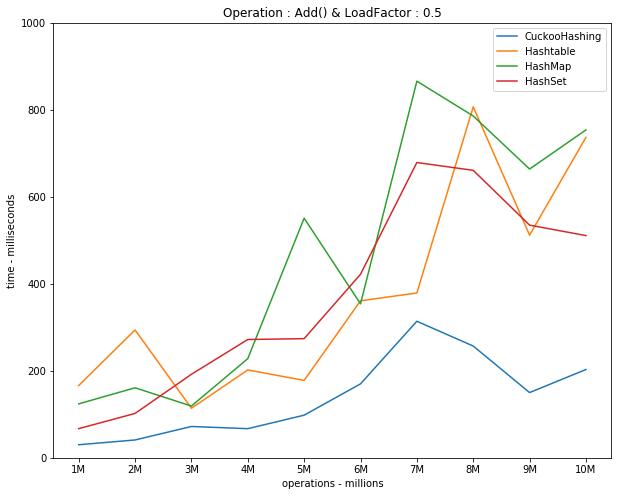
**Algorithm:**

The Cuckoo Hashing algorithm ensures a worst-case time complexity of O(1). It is able to achieve this performance by maintaining two hashtables with different hash functions to reduce the chances of collision. It ensures that the element will be present in either of the two locations and so we can search for it in constant time. We have also used a secondary HashTable to store elements in case the number of collisions exceed the threshold value when inserting. The hash functions use mod operation with the precalculated hash values which are distinct for each test case and are optimized for the number of elements to be inserted. While inserting we calculate the location of the element using one of the hash functions, if there is another element at that location it is replaces with the current element and we again search for the location of the replaced element using the other hash function. This replacement of elements is why the algorithm is called Cuckoo algorithm, because the cuckoo bird throws the eggs of other birds out of their nests and lays it’s own eggs in their place.

**Problem Setup:**

I have randomly generated 10 input files containing 1 – 10 million integers out of a range of 1 – 100 million. The algorithm is developed by testing on these input files. Since the size of the input files was too large to upload with the project I have created another method for generating sequential integers for each test case. For making the below report I have created 10 test cases with 1 million, 2 million till 10 million number. Then the hashtables are populated with these number by calling the add method and their total time in performing the operation for all the numbers is saved. The same step is done with contains and remove operations also and their respective times are saved. The entire process is repeated for Load\_factors of 0.5, 0.75 and 0.9. Then the below graphs are plotted for each operation and Load\_factor separately.

**Experimental Evaluation:**



**Summary:**

CuckooHashing is much faster than Hashtable, HashMap & HashSet, especially when the number of operations are in millions and the Load Factor is high. There is minor fluctuation due to processor speed and OS scheduling but in most of the cases above CuckooHashing has performed the fastest within the smallest period of time. There is another clear trend : while the time taken by Hashtable, HashMap & HashSet increases with the size of input, the time taken by CuckooHashing is almost contant although with minor variations. This proves that CuckooHashing algorithm ensures constant time, i.e. O(1).