DS 221 - Introduction to Scalable Systems
Assignment 2 - B

Shriram R.
M Tech (CDS)
06-02-01-10-51-18-1-15763

October 14, 2018

1 Searching

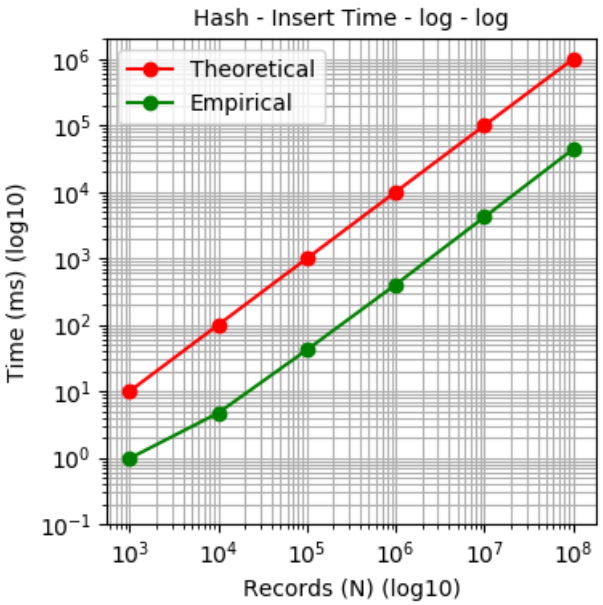
The dictionary abstract data type has been implemented by hash map, unsorted arrray and sorted array. Time complexities for Insert and Lookup performance have been analyzed and empirically verified. The following sections will cover the analysis and empirical results in detail.

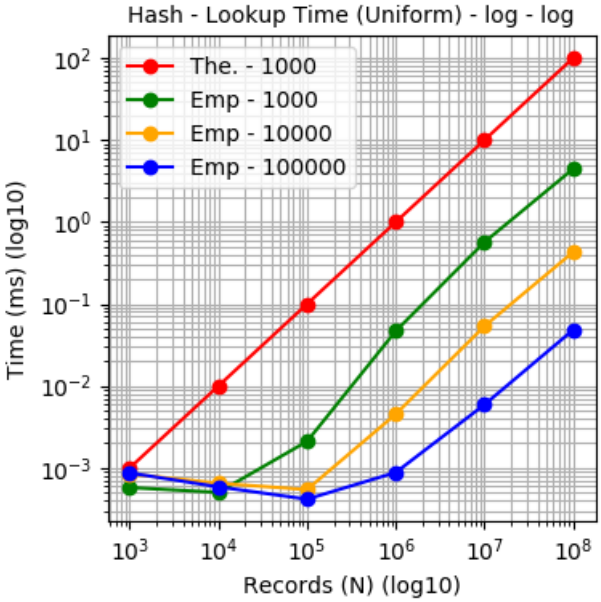
1.1 Hash Table

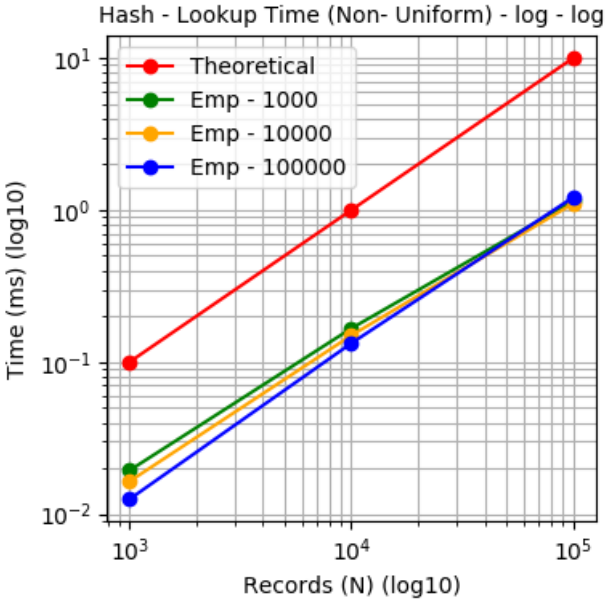
Let C denote the capacity or number of buckets in the hash table and N denote the total number of records in the hash table. Collisions are resolved by chaining of key-value records. K mod C is the hash function used where K is the key value.

In the case of insertion, a new record would be added to the end of the chain (STL list) of its bucket. The time complexity for this insertion is O(1). Hence, insertion of N records take O(N) time. This is independent of the key distribution.

For uniform distribution of keys, each bucket would have $\frac{N}{C}$ records on average. Hence, the time complexity to search a record is $O(\frac{N}{C})$. For non-uniform ditribution, the worst case time complexity is O(N). This happens when all the records are present in a single bucket. However, if the maximum number of records in a bucket is bounded by M (<N), then the worst case time complexity is O(M).







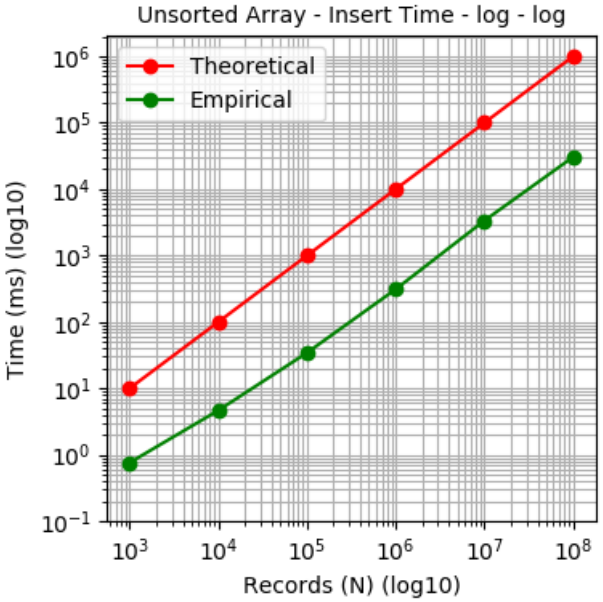
The X axis in lookup plots denote the number of records present in hash table during the lookup. Each line in the plot is for different capacity. The time plotted is the average value for 1 record.

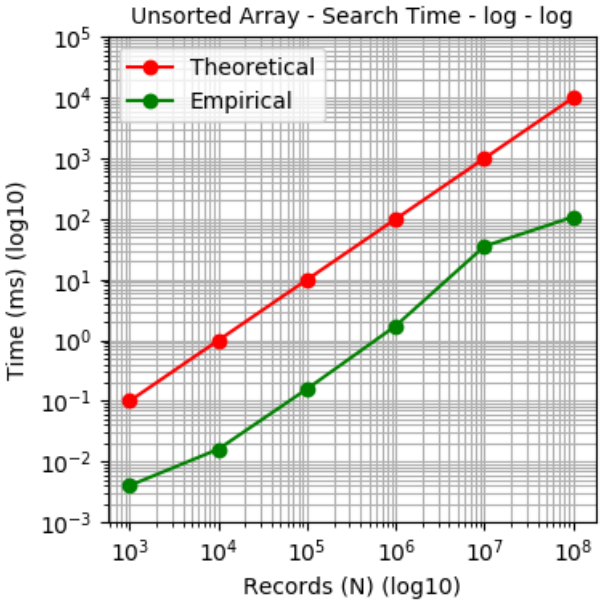
It is observed that the insert time is consistent with theoretical analysis. The lookup time is consistent for large values of N but deviates from the analytical model for lower N values for uniform distribution. This is could be due to caching and locality that are not captured in the theoretical model. For non-uniform, all the keys generated map to a single hash bucket to simulate a worst case.

1.2 Unsorted Array

Insertion for each record can be done in O(1) time since a new record is added to the end of the array. Hence, the time complexity to insert N records would be O(N).

For lookup, linear scan is performed over the entire array till the last record since the records are not sorted by key. Hence, the worst case time complexity for lookup is O(N). Note that the insert and lookup time is independent of key distribution.



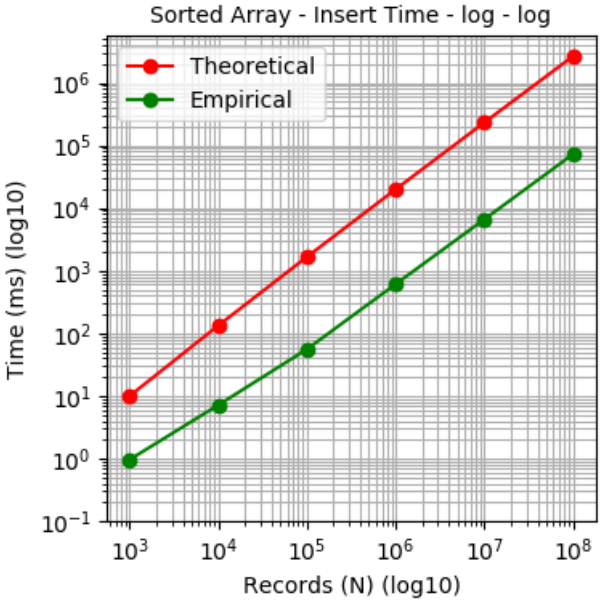


From the emprical results, it can be observed that the time taken generally agrees with the analysis. There is a minor deviation of lookup time from the theoretical value which could be due to cache effects and inaccuracies in measuring the lookup time.

1.3 Sorted Array

Insertion of records takes O(N) time similar to that of unsorted array but there is an additional step at the end of insertion which sorts the records by key. This takes an additional O(N log N) time. So, the total time complexity is O(N log N).

For lookup, binary search of the lookup key is performed on the sorted array. The time complexity for this binary search is O(log N). Similar to sorted array, the lookup and insert time is independent of key distribution.



It is observed that the insert time is consistent with theoretical analysis. The lookup time marginally deviates from the theoretical value for lower N values. This could be due to caching and inaccuracies in capturing the time for lookup.

1.4 Comparison

The table given below show the theoretical complexities of the above implementations. It can be inferred that unsorted array is the worst choice in terms of lookup time. Hash table is better than sorted array in terms of lookup if the bucket count C is greater than $\frac{N}{logN}$. Hash table is also faster for inserts than sorted array.

Implementation	**Insert**	**Lookup**
Hash Table	O(N)	O($\frac{N}{C}$)
Unsorted Array	O(N)	O(N)
Sorted Array	O(N log N)	O(log N)

The following table shows the emprical results for $N = 10^6$ (Uniform distribution) and $C = 10^4$ (Hash Table). The lookup time is the average for 1000 records. Sorted array took longest time time to insert the records while unsorted array took longest to lookup a key value. Hash table lies in between in terms of both insert and lookup. If capacity (C) is increased for hash table, the lookup time should be less than the sorted array.

Implementation	**Insert (ms)**	**Lookup (ms)**
Hash Table	437.963	4.531
Unsorted Array	311.291	1733
Sorted Array	611.142	0.775

1.5 Note on Empirical Analysis

The empirical testing was done on a compute node in the Turing cluster having a 8 core processor clocked at 2.6 GHz and 32GB RAM. Test data was generated using custom scripts available at [4].

1.6 Note on Code

The IDictionary.h file contains the definition of the given interface. The DictImpl.cpp file contains three different implementation of the given interface. The Runner.cpp contains the driver code for the implementation. The Makefile can be used to generate the Runner.o file.

1.7 References

1. https://en.cppreference.com/
2. DS 221 Course lecture notes
3. https://www.geeksforgeeks.org/
4. /home/shriramr/ds221/part_3/code/test
5. /home/shriramr/ds221/part_3/data/