Raisa Litchfield
830782

Design of Algorithms

Assignment 2: Dynamic Hash Tables Report

Part 4: Load Factor and Collisions

Experiments using the linear probing hash table were designed to best answer the following research questions:

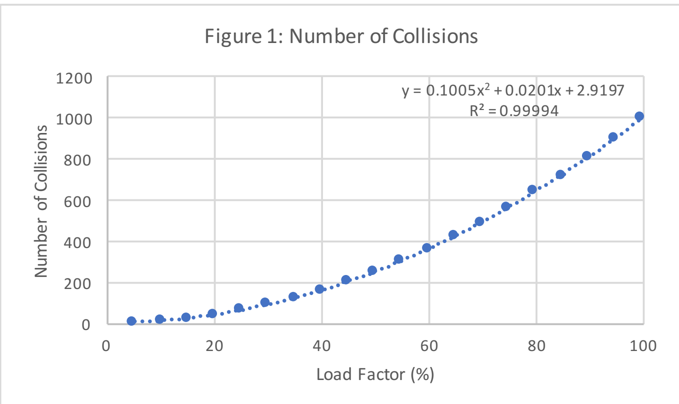
- What is the relationship between load factor and the number of collisions in a linear hash table?
- What is the relationship between load factor and the average probe sequence length in a linear hash table?

The hash table's load factor is the number of keys in the table divided by the table size and multiplied by 100. Therefore, to test the effect of load factor, experiments with varying numbers of key insertions were conducted (20 tests ranging from 100 to 2,000 keys with increments of 100). Additionally, the number of collisions is defined as 'the number of keys for which the first address they hash to is already occupied'; and the average probe sequence length as 'the average number of slots which must be checked before a key is inserted into a free space'. It was hypothesised that as load factor increased, the number of collisions and the average probe sequence length would also increase.

To ensure reliability of the data, three trials for each number of key insertions were conducted with separate randomly-generated key insertion datasets. Furthermore, to minimise any other factors affecting the results, the starting hash table size was set to 2,000. As the maximum number of key insertions was 2,000, setting the linear hash table to this size would ensure it is not doubled.

Three text files containing 2,000 key insert commands along with a statistics table request, and exit request, were created for the three trials (Appendix 4.1). To reduce the variability between the 20 tests, text files containing the first 100, 200, etc. keys were created using the first n insert commands from these files (Appendix 4.2). Note that n is the number of keys to be inserted.

Using the 20 files created for each trial as input, the assignment's program was executed using a linear hash table with a starting table size of 2,000 (Appendix 4.3). Output from these executions were saved to three separate files corresponding to each trial. On completion, the data for load factor, number of collisions, and average probe sequence length were extracted from the statistics tables in the three text files (Appendix 4.4) and copied into an Excel spreadsheet (Appendix 4.5). The mean of the three trials for each variant was then calculated and two graphs were created from the data. These graphs showing the relationship between load factor and number of collisions (Figure 1) as well as load factor and average probe length (Figure 2) are shown below.



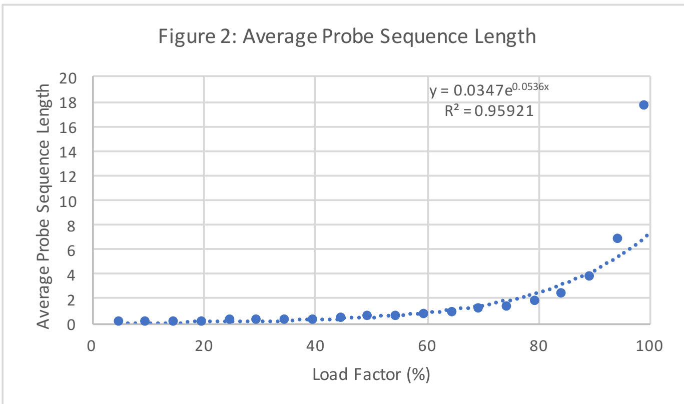


Figure 1 exhibits a quadratic trend for the number of collisions. That is, as the load factor of a table increases, the number of collisions also increases at a quadratic rate. This was consistent with the hypothesis and seems logical. As the load factor increases, it is more and more likely that a key being inserted will have another key in the address it hashes to resulting in a collision. Figure 2 reveals a seemingly exponential trend. Again, this is consistent with the hypothesis that as load factor increases, the average probe sequence length also increases. Like the previous relationship, this makes logical sense as a greater load factor would result in a greater chance of a key colliding each time an insertion attempt is made, after the initial collision.

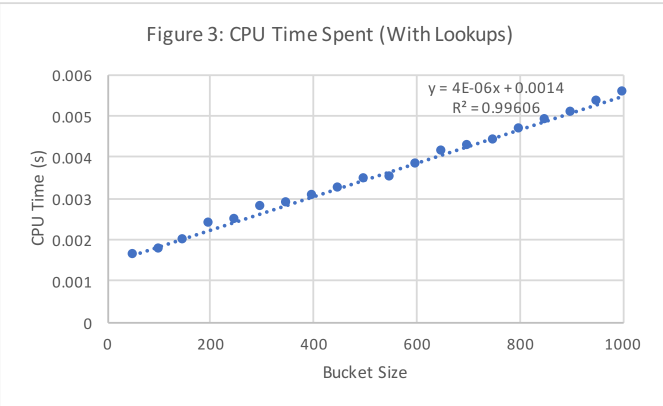
Part 5: Keys Per Bucket

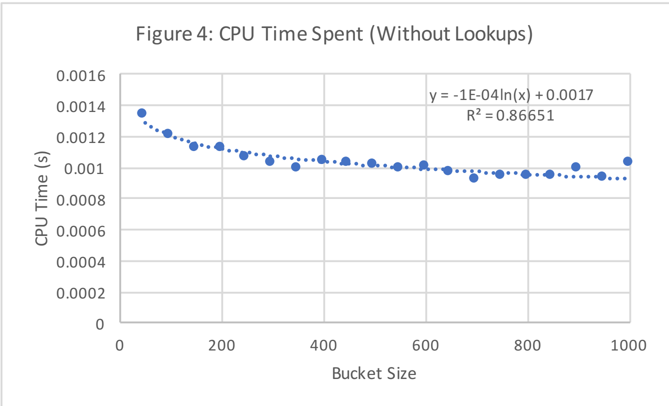
An experiment using the <u>multi-key extendible hash table</u> was designed to best answer the following research question:

- What is the relationship between the number of keys per bucket (i.e. bucket size) and the performance of insert and lookup operations in a multi-key extendible hash table?

To test the effect of bucket size on the performance of insert and lookup operations in a multi-key extendible hash table, three experiments were created. The <u>first experiment</u> was designed to test the effect of bucket size on both insert and lookup operations simultaneously. As lookup commands are inbuilt into the insertion function, only insertion requests were used for the programs input. For each insertion, a lookup is conducted to ensure no duplicates are inserted into the hash table. <u>The second experiment</u> was designed to only test the effect of bucket size on insertions. To quickly do this, the program was recompiled with the key lookup within the insertion function commented out. <u>The third experiment</u> was designed to test only lookups. This was quickly achieved by commenting out the three lines used for recording the CPU time in the insertion function and recompiling the program.

For the first two experiments, three key insertion files were created for three different trials using the cmdgen program. These text files included 2,000 key insertion request, a statistics table request and an exit request. Bucket sizes ranging from 50 to 1,000 were tested with increments of 50 resulting in a total of 20 data points for each trial of each experiment. The terminal commands used to create these files and execute the experiment are shown in Part 5 of the Appendix. The three graphs created from calculating the mean of the trials from each experiment are shown below (Figures 3 – 5).





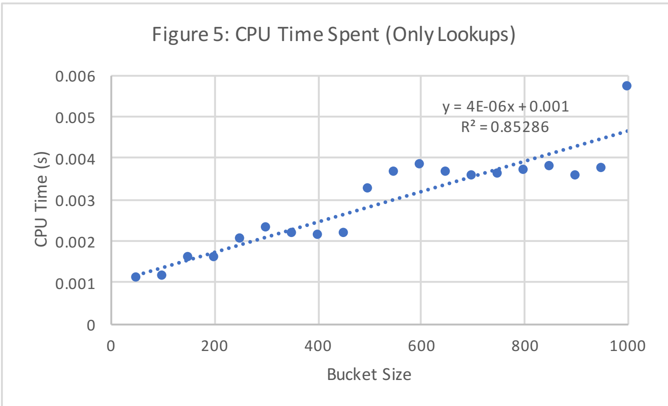


Figure 3 shows a positive linear relationship between bucket size and CPU time spent on insertions and lookups. This relationship is understood better with an analysis of the underlying code. When inserting a key into the hash table, a lookup is conducted. This lookup has a worst case of O(n), where n is the number of keys in the bucket. This worst case occurs very often as the key being looked up is often not in the table. A more efficient algorithm could've been implemented here, however for simplicity the linear search algorithm was used. This linear time complexity from the code is reflected in this Figure.

Figure 4 shows a negative logarithmic relationship between bucket size and the CPU time spent on insertions only. This relationship may be a result of doubling the hash table. Note that for this experiment, the lookup section within the insert key function was commented out and therefore the O(n) time complexity of a key lookup is not reflected in the graph. Each time a hash table doubles, the following time it doubles will take a greater amount of time. This is because the table is larger and more pointers to buckets must be copied down the table. Despite this however, since the table is larger, it will fill up less quickly and thus will be doubled less and less often. As bucket size increases, the number of times the hash table must double to accommodate more keys decreases at a logarithmic rate. This in turn causes the CPU time spent to also decrease at a logarithmic rate thus resulting in the negative logarithmic trend displayed. With these discussed facts in mind, it is likely that there is a bucket size that seems to optimise the hash table's insert and lookup operations. This is something that could be explored further with more experiments.

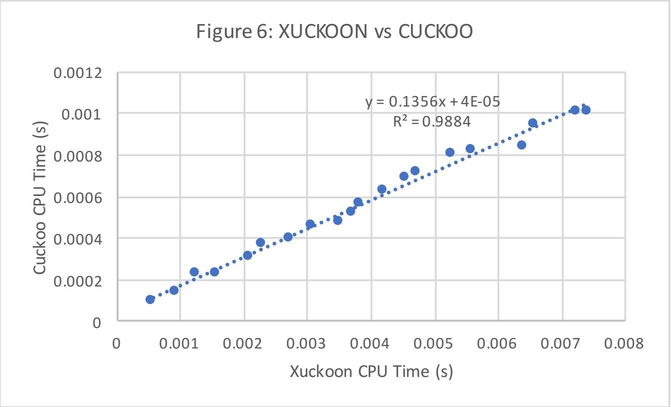
Figure 5 shows a positive linear relationship between bucket size and CPU time spent on only lookups. The data points on this graph seem to fluctuate more than the two previous experiments which mostly followed their trends. The overall relationship is similar to the trend shown in Figure 3 and has a similar explanation. The time complexity for a lookup is O(n) where n is the number of keys in the bucket. Therefore, as bucket size increases, the number of keys potentially in a bucket also increases. Moreover, this causes the time it takes for a lookup operation to again, increase resulting in the trend seen in the above Figure.

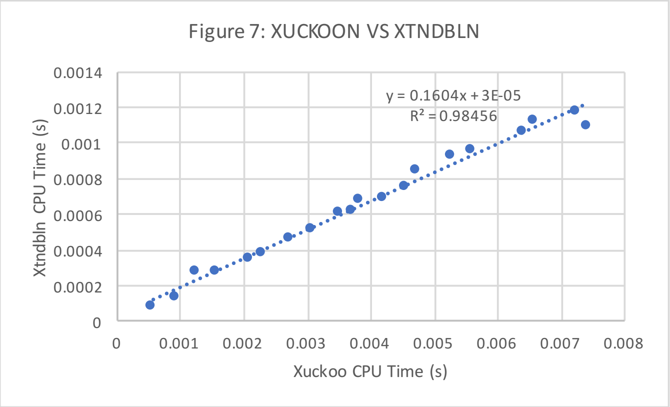
Bonus Challenge: Multi-key Extendible Cuckoo Hashing

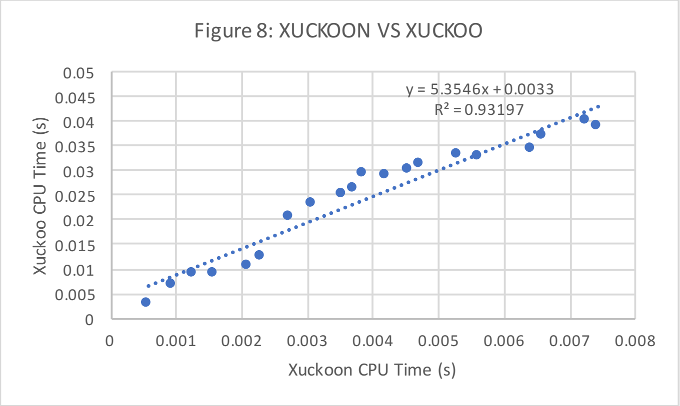
An experiment using the multi-key extendible cuckoo, single-key extendible cuckoo, multi-key extendible, and cuckoo hash tables was designed to best answer the following questions:

- What is the relationship between the CPU time spent on insert and lookup operations in multi-key extendible cuckoo hashing and:
    - ❖ Single-key extendible cuckoo hashing?
    - ❖ Multi-key extendible hashing?
    - ❖ Cuckoo hashing?

To explore these three relationships, each of the four hash table variants were executed 20 command text files. These files included key inserts ranging from 100 to 2,000 with increments of 100. Bucket sizes of 5 keys were used for the two versions of multi-key extendible hashing for consistency in the experiments. They also contained a statistics table request and an exit request. Furthermore, three trials for each of the four hash tables was also conducted and the mean of them calculated to produce the following graphs.







Figures 6 to 8 exhibit very similar trends. Cuckoo hashing, multi-key extendible hashing and single-key extendible hashing have positive linear trends. CPU time for single-key extendible cuckoo hashing seems to increase at the fastest rate (gradient=5.3546), followed multi-key extendible hashing (gradient=0.1604) and cuckoo hashing (gradient=0.1356). These relationships were expected and seem to be logical since, as CPU increases for multi-key extendible hashing, CPU in all three of the other hash table variants tested were also expected to increase. Figure 8 shows more fluctuation and variance from the linear trend. Further experiments and analysis would be needed to determine why this is the case.

Additional graphs were created to better see the differences in the CPU time of each hash table variant against the number of keys inserted. These are shown below.

Figure 9: Number of Keys Inserted vs CPU Time Spent

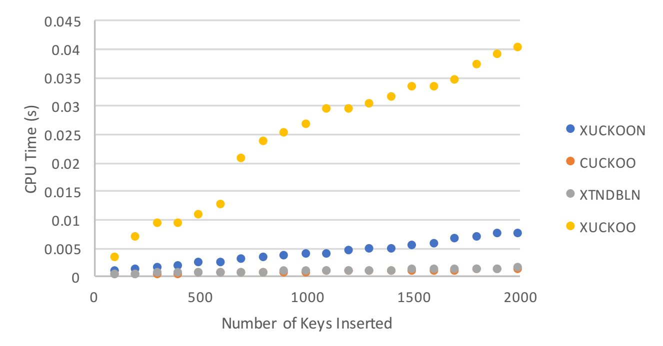


Figure 10: Number of Keys Inserted vs CPU Time Spent

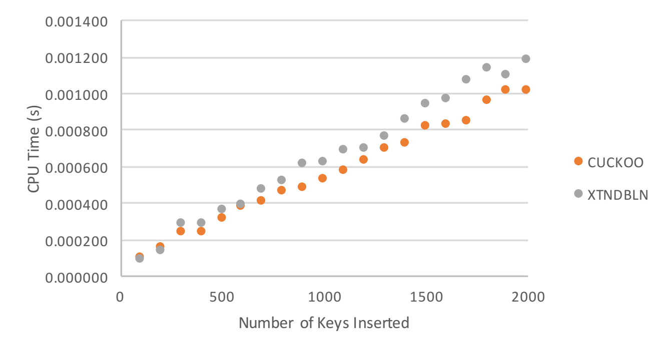


Figure 9 shows the relationship between the number of keys inserted and the CPU time spent for all four of the hash table variants. This graph provides an easy visualisation of which hash tables are most efficient. Clearly, single-key extendible cuckoo hashing exhibits the worst efficiency as it has the greatest slope. This is followed by the multi-key extendible cuckoo hashing. The two most efficient hashing tables with the tested bucket sizes and other variables seem to be the multi-key extendible hashing and cuckoo hashing. In Figure 9 the differences between the trends for these two hash tables is difficult to see so Figure 10 was created, showing data for only these two hash tables. Figure 10 reveals that the multi-key extendible hashing is less efficient than regular cuckoo hashing for this experiment. It is important to note that these trends may be different with other bucket sizes and therefore further experiments are recommended to discover the most optimised hash table.

Appendix

The appendix has not been counted in the page count as all materials are supplementary and only provided for ease when and if the experiments are repeated.

Part 4: Load Factors and Collisions

4.1 Sample terminal command for creating text file with 2,000 key inserts

```
./cmdgen 2000 0 - > experiments/t1/2000.txt
```

4.2 Sample terminal command for creating text files from 100 to 2,000 keys with increments of 100

```
for ((n=101; n<2000; n+=100));
do sed "$n,2000d" experiments/t1/2000.txt > experiments/t1/$((n-1)).txt;
done;
```

4.3 Sample terminal command for running the program with the 20 input files (linear, starting size = 2,000)

```
for ((n=100; n<=2000; n+=100));
do ./a2 -t linear -s 2000 < experiments/t1/$n.txt >> experiments/linear1/output.txt;
done;
```

4.4 Sample terminal commands for extracting data from statistics table (load factor, number of collisions and average probe sequence length)

```
grep -wE "(load\ factor)" experiments/linear1/output.txt |
sed 's/[^0-9|.]*//g' > experiments/linear1/load_factor.txt

grep -wE "(collisions)" experiments/linear1/output.txt |
sed 's/[^0-9|.]*//g' > experiments/linear1/collisions.txt

grep -wE "(probe)" experiments/linear1/output.txt |
sed 's/[^0-9|.]*//g' > experiments/linear1/probe_length.txt
```

4.5 Table of all data for part 4 with calculated mean values

Number of Keys	Load factor				Collisions				Probe Length			
	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean
100	5.000	5.000	5.000	5.000	4.000	2.000	2.000	2.667	0.040	0.020	0.020	0.027
200	10.000	10.000	10.000	10.000	14.000	11.000	7.000	10.667	0.080	0.065	0.035	0.060
300	15.000	15.000	15.000	15.000	30.000	29.000	21.000	26.667	0.123	0.120	0.073	0.105
400	20.000	20.000	20.000	20.000	49.000	44.000	39.000	44.000	0.155	0.142	0.102	0.133
500	25.000	24.950	25.000	24.983	76.000	69.000	64.000	69.667	0.196	0.178	0.152	0.175
600	29.900	29.900	29.950	29.917	105.000	105.000	92.000	100.667	0.239	0.239	0.214	0.231
700	34.900	34.850	34.900	34.883	133.000	132.000	117.000	127.333	0.265	0.270	0.229	0.255
800	39.900	39.800	39.900	39.867	170.000	166.000	147.000	161.000	0.305	0.322	0.302	0.310
900	44.800	44.800	44.900	44.833	212.000	206.000	194.000	204.000	0.358	0.379	0.406	0.381
1000	49.800	49.800	49.900	49.833	254.000	257.000	242.000	251.000	0.414	0.525	0.515	0.485
1100	54.800	54.750	54.900	54.817	304.000	310.000	300.000	304.667	0.489	0.598	0.615	0.567
1200	59.750	59.700	59.900	59.783	372.000	367.000	357.000	365.333	0.644	0.770	0.785	0.733
1300	64.600	64.650	64.900	64.717	429.000	427.000	416.000	424.000	0.766	0.896	0.945	0.869
1400	69.600	69.650	69.750	69.667	491.000	497.000	485.000	491.000	0.941	1.075	1.222	1.079
1500	74.550	74.600	74.750	74.633	562.000	573.000	558.000	564.333	1.119	1.320	1.702	1.380
1600	79.550	79.600	79.750	79.633	639.000	648.000	639.000	642.000	1.452	1.695	2.204	1.784
1700	84.550	84.600	84.650	84.600	717.000	731.000	717.000	721.667	2.074	2.262	2.906	2.414
1800	89.550	89.500	89.600	89.550	808.000	821.000	801.000	810.000	3.149	3.449	4.676	3.758
1900	94.450	94.350	94.450	94.417	898.000	914.000	899.000	903.667	5.665	5.515	9.254	6.811
2000	99.450	99.350	99.350	99.383	990.000	1009.000	997.000	998.667	19.416	12.527	21.141	17.695

Part 5: Keys per Bucket

5.1 Sample terminal command for creating text file with 2,000 key inserts

```
./cmdgen 2000 0 - > experiments/t1/2000.txt
```

5.2 Sample terminal command for running the program with the key inserts file with bucket sizes ranging from 50 to 1,000 with increments of 50 (xtndbln)

```
for ((n=50; n<=1000; n+=50));
do ./a2 -t xtndbln -s $n < experiments/t1/2000.txt >> experiments/xtndbln1/output.txt;
done;
```

5.3 Sample terminal command for extracting CPU time spent from the statistics table

```
grep -wE "(CPU)" experiments/xtndbln1/output.txt |
sed 's/[^0-9|.]*//g' > experiments/xtndbln1/CPU_time.txt
```

4.5 Table of all data for part 5 with calculated mean values

Bucket Size	CPU time (with lookups)				CPU time (without lookups)				CPU time (only lookups)			
	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean
50	0.001674	0.001683	0.001479	0.001612	0.001567	0.001290	0.001183	0.001347	0.001085	0.001252	0.000996	0.001111
100	0.001779	0.001770	0.001619	0.001723	0.001350	0.001182	0.001106	0.001213	0.001152	0.001107	0.001158	0.001139
150	0.001966	0.002002	0.001856	0.001941	0.001219	0.001158	0.001006	0.001128	0.001764	0.001507	0.001476	0.001582
200	0.002544	0.002360	0.002235	0.002380	0.001246	0.001080	0.001070	0.001132	0.001850	0.001501	0.001493	0.001615
250	0.002454	0.002436	0.002392	0.002427	0.001058	0.001034	0.001112	0.001068	0.002525	0.001878	0.001676	0.002026
300	0.002907	0.002760	0.002578	0.002748	0.001018	0.001005	0.001062	0.001028	0.002380	0.002047	0.002478	0.002302
350	0.002902	0.002870	0.002737	0.002836	0.000998	0.001014	0.000971	0.000994	0.002206	0.002100	0.002264	0.002190
400	0.003091	0.003029	0.003025	0.003048	0.001079	0.001075	0.000973	0.001042	0.002148	0.002115	0.002148	0.002137
450	0.003235	0.003209	0.003232	0.003225	0.001036	0.001066	0.000975	0.001026	0.002111	0.002132	0.002322	0.002188
500	0.003470	0.003471	0.003422	0.003454	0.001043	0.001007	0.001001	0.001017	0.003706	0.003168	0.002820	0.003231
550	0.003435	0.003493	0.003446	0.003458	0.001095	0.000970	0.000911	0.000992	0.003546	0.003437	0.004033	0.003672
600	0.003680	0.003997	0.003682	0.003786	0.001056	0.000996	0.000959	0.001004	0.003475	0.003638	0.004326	0.003813
650	0.003953	0.004129	0.004253	0.004112	0.001011	0.000989	0.000919	0.000973	0.003547	0.003644	0.003739	0.003643
700	0.004158	0.004149	0.004404	0.004237	0.000891	0.000964	0.000904	0.000920	0.003536	0.003561	0.003655	0.003584
750	0.004399	0.004349	0.004440	0.004396	0.000940	0.000940	0.000952	0.000944	0.003611	0.003473	0.003699	0.003594
800	0.004629	0.004620	0.004646	0.004632	0.000947	0.000974	0.000929	0.000950	0.003504	0.003648	0.003880	0.003677
850	0.004805	0.004814	0.004927	0.004849	0.000930	0.001030	0.000893	0.000951	0.003795	0.003668	0.003867	0.003777
900	0.005044	0.005080	0.005071	0.005065	0.001059	0.000973	0.000947	0.000993	0.003503	0.003595	0.003622	0.003573
950	0.005313	0.005337	0.005268	0.005306	0.000880	0.001028	0.000916	0.000941	0.004023	0.003590	0.003662	0.003758
1000	0.005519	0.005524	0.005614	0.005552	0.001047	0.001065	0.000977	0.001030	0.00634	0.006352	0.004385	0.005694

Bonus Challenge: Multi-key Extendible Cuckoo Hashing

B.1 Table of all data for bonus challenge with calculated mean values

Number of Keys	XUCKOON				CUCKOO				XTNDBLN				XUCKOO			
	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean	Trial 1	Trial 2	Trial 3	Mean
100	0.000531	0.000503	0.000662	0.000565	0.000099	0.000100	0.000097	0.000099	0.000073	0.000089	0.000097	0.000086	0.002996	0.002891	0.003217	0.003035
200	0.000786	0.000986	0.001045	0.000939	0.000116	0.000149	0.000178	0.000148	0.000126	0.000142	0.000145	0.000138	0.008334	0.005552	0.006414	0.006767
300	0.001081	0.001236	0.001448	0.001255	0.000220	0.000236	0.000241	0.000232	0.000319	0.000306	0.000215	0.000280	0.008921	0.009192	0.008893	0.009002
400	0.001338	0.001665	0.001672	0.001558	0.000228	0.000233	0.000253	0.000238	0.000251	0.000329	0.000274	0.000285	0.009954	0.008592	0.008878	0.009141
500	0.001892	0.002183	0.002225	0.002100	0.000312	0.000303	0.000317	0.000311	0.000326	0.000422	0.000323	0.000357	0.011318	0.009778	0.011059	0.010718
600	0.002018	0.002438	0.002413	0.002290	0.000346	0.000389	0.000379	0.000371	0.000385	0.000414	0.000369	0.000389	0.012568	0.011692	0.012841	0.012367
700	0.002441	0.002932	0.002784	0.002719	0.000383	0.000422	0.000411	0.000405	0.000450	0.000452	0.000492	0.000465	0.022526	0.019602	0.019565	0.020564
800	0.002925	0.003007	0.003283	0.003072	0.000427	0.000465	0.000495	0.000462	0.000492	0.000573	0.000487	0.000517	0.023195	0.024407	0.022605	0.023402
900	0.003250	0.003784	0.003513	0.003516	0.000490	0.000524	0.000424	0.000479	0.000664	0.000629	0.000534	0.000609	0.025211	0.026839	0.023196	0.025082
1000	0.003436	0.003903	0.003785	0.003708	0.000537	0.000554	0.000478	0.000523	0.000620	0.000605	0.000641	0.000622	0.027177	0.027256	0.024648	0.026360
1100	0.003661	0.003805	0.004042	0.003836	0.000593	0.000604	0.000515	0.000571	0.000670	0.000751	0.000638	0.000686	0.029020	0.032894	0.025648	0.029187
1200	0.003795	0.004494	0.004309	0.004199	0.000638	0.000650	0.000600	0.000629	0.000731	0.000635	0.000725	0.000697	0.029966	0.030717	0.026449	0.029044
1300	0.004607	0.004827	0.004192	0.004542	0.000726	0.000742	0.000611	0.000693	0.000803	0.000715	0.000769	0.000762	0.029196	0.031147	0.029511	0.029951
1400	0.004688	0.004985	0.004470	0.004714	0.000731	0.000749	0.000676	0.000719	0.000817	0.000850	0.000889	0.000852	0.030380	0.033267	0.030225	0.031291
1500	0.005199	0.005529	0.005088	0.005272	0.000784	0.000825	0.000827	0.000812	0.000934	0.000974	0.000898	0.000935	0.030759	0.033395	0.034916	0.033023
1600	0.005411	0.005892	0.005461	0.005588	0.000825	0.000864	0.000790	0.000826	0.000968	0.000941	0.000969	0.000959	0.031532	0.034336	0.033113	0.032994
1700	0.006383	0.006471	0.006346	0.006400	0.000827	0.000921	0.000772	0.000840	0.001037	0.001026	0.001123	0.001062	0.033867	0.034455	0.034708	0.034343
1800	0.006624	0.006733	0.006374	0.006577	0.000946	0.000941	0.000974	0.000954	0.001071	0.001251	0.001075	0.001132	0.038575	0.035777	0.036304	0.036885
1900	0.006988	0.006958	0.008314	0.007420	0.000960	0.000958	0.001108	0.001009	0.001178	0.001029	0.001086	0.001098	0.040826	0.037730	0.038147	0.038901
2000	0.006523	0.007117	0.008090	0.007243	0.001038	0.001040	0.000950	0.001009	0.001224	0.001180	0.001141	0.001182	0.037034	0.039142	0.043535	0.039904