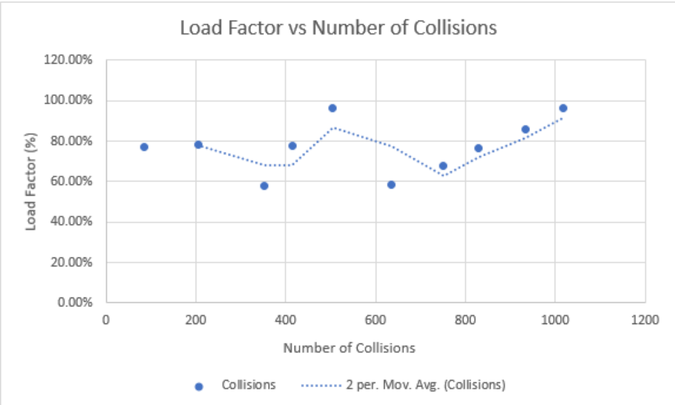
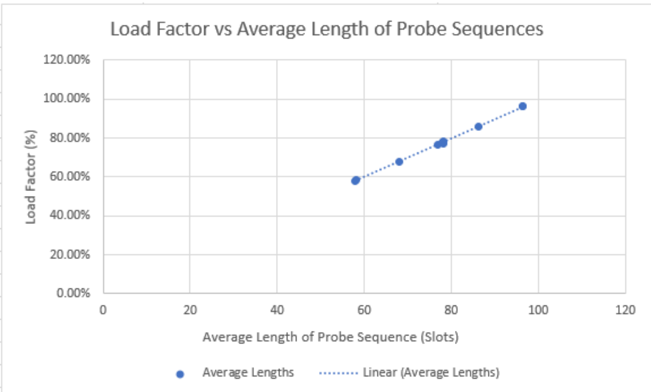
Part 4

Several insertions were performed in intervals of 100 till 1000. The statistics for size of the table, the load factor, the number of collisions and the average length of probe sequences per trial were recorded. Each individual trial was run a few extra times to ensure accuracy and reliability in the data. The first comparison was



made with the Load Factor (Percentage) against the Number of Collisions. The results yielded showed a non-linear trend. The number of collisions seem to increase with the number of insertions rather linearly. The load factor however, increases for a set interval of collisions. Afterwards, it would seem like the load factor drops when the table doubles when it's found to be full. As the size of the table doubles each time, the rate of increase of the load factor in relation to the number of collision decreases with the same insertions intervals. As the table size increases to by 2^n, the decrease in the rate of increase of load factor seems to be exponential.



The Average Length of Probe Sequences appears to have an almost perfect linear correlation with the load factor. The numbers are only about a percent or less off from one another. An increased load in the table would increase the chances of an inserted key encountering a collision and time spent linearly stepping through the table to find a free slot. Several tests were conducted on individual inserts and it produced values that had a wide range (almost that of zero to total number of insertions). As the length observed in the table is averaged with the total number of insertion, it would be a number like the percentage of the load of the table.

The following table shows the statistics recorded as well as the show that the table size increases in relations to 2^n.

Insertions	Size	Load Factor	Collisions	Average Probe Sequences
100	128	77.30%	85	78.13

200	256	78.13%	205	78.13
300	512	58.08%	353	58.01
400	512	77.93%	414	77.93
500	512	96.12%	505	96.48
600	1024	58.30%	636	58.03
700	1024	67.96%	751	67.97
800	1024	76.76%	829	76.76
900	1024	86.13%	933	86.13
1000	1024	96.39%	1017	96.39

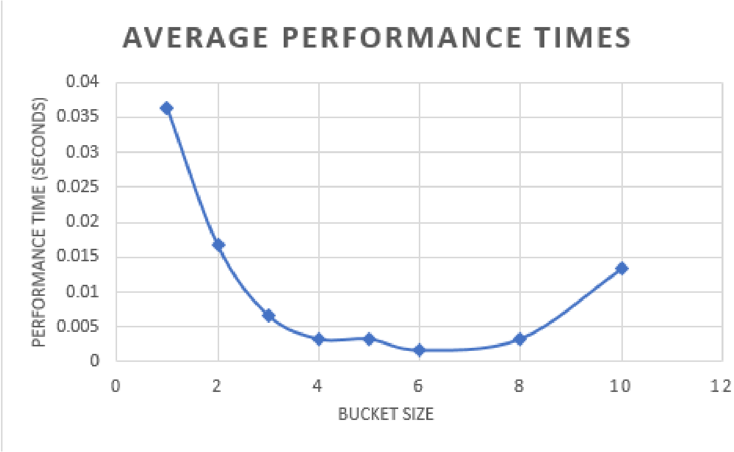
Part 5

The table shows a clear inverse relation with table size and bucket size in terms of 2^n for the increase in size of the extendible table. 10000 inserts and lookups were run through the dimefox server compiler for each designated bucket size. As bucket sizes from 1 to 10 showed a unique trend compared to larger numbers, they were examined in detail.

Bucket Size	Insertions	Lookups	Average Performance (seconds)	Table Size
1	10000	10000	0.0363	524288
2	10000	10000	0.0167	262144
3	10000	10000	0.0067	16384
4	10000	10000	0.0033	16384
5	10000	10000	0.0033	8192
6	10000	10000	0.0017	8192
8	10000	10000	0.0033	4096
10	10000	10000	0.0133	4096
50	10000	10000	0.0200	256
100	10000	10000	0.0233	128
200	10000	10000	0.0300	64
500	10000	10000	0.0367	32
1000	10000	10000	0.0633	16

The bucket sizes of 4 to 8 produces the most efficient average CPU performance. This proves the relationship between bucket size and performance time of multi-key extendible hash tables to be non-linear. Bucket sizes 1-3 would be too small and hence enlarge the table too much, making it slower to run. Bucket sizes up to one tenth of the insertion or lookup sizes are shown to be slow and inefficient, having close to O(n) lookup time rather than O (1).

The following graph shows the parabolic, non-linear relationship of up to 10 bucket size.



The results prove there that would be optimal bucket sizes for each insertion/ lookup size given to the multi-key extendible hash function. A bucket size near 1 would just be similar to the less efficient single-key extendible hashing and a bucket size of a big fraction of the insertion/ lookup size would just be no different from a O(n) search through an array due to the smaller table size.