ASSIGNMENT 8

Random seed: 23

Grow Experiment:

```
PS C:\Users\ajitb\OneDrive\Desktop\DS1\assignments-master-09-hash_experiment-cpp\09-hash_experiment\cpp> .\a.exe grow-ms-low 23
128 1.699 1.701
256 2.532 4.108
512 1.557 0.881
1024 2.583 6.120
2048 2.440 3.664
4096 1.624 0.731
8192 1.735 1.252
16384 1.526 0.567
32768 2.001 3.324
65536 1.581 0.822
131072 13.133 71.026
262144 1.846 1.489
524288 1.703 1.501
1048576 2.106 2.051
2097152 2.555 6.522
```

Fig: grow-ms-low

```
PS C:\Users\ajitb\OneDrive\Desktop\DS1\assignments-master-09-hash_experiment-cpp\09-hash_experiment\cpp> .\a.exe grow-ms-high 23
128 1.661 0.960
256 3.385 9.565
512 2.201 3.575
1024 1.760 1.312
2048 2.440 3.664
4096 1.627 0.969
8192 2.304 3.321
16384 1.574 0.624
32768 2.415 4.293
65536 5.290 20.790
131072 2.147 3.405
262144 1.605 0.839
524288 2.053 1.976
1048576 2.204 3.530
2097152 51.290 310.822
```

Fig: grow-ms-high

```
PS C:\Users\ajitb\OneDrive\Desktop\DS1\assignments-master-09-hash_experiment-cpp\09-hash_experiment\cpp> .\a.exe grow-poly-1 23
128 1.685 1.050
256 1.632 0.738
512 3.456 6.540
1024 1.511 0.502
2048 1.905 2.559
4096 2.595 3.806
8192 2.056 1.744
16384 1.906 1.685
32768 1.669 1.569
65536 1.477 0.502
131072 2.122 1.838
262144 1.422 0.375
524288 4.597 17.940
1048576 2.749 6.950
2097152 1.498 0.435
```

Fig: grow-poly-1

```
PS C:\Users\ajitb\OneDrive\Desktop\DS1\assignments-master-09-hash_experiment-cpp\09-hash_experiment\cpp> .\a.exe grow-poly-2 23
128 2.102 0.345
256 2.135 0.217
512 2.182 0.255
1024 2.148 0.156
2048 2.182 0.123
4096 2.200 0.102
8192 2.205 0.078
16384 2.234 0.083
32768 2.212 0.034
65536 2.216 0.031
131072 2.215 0.029
262144 2.219 0.016
524288 2.217 0.024
1048576 2.218 0.008
2097152 2.217 0.007
```

Fig: grow-poly-2

```
PS C:\Users\ajitb\OneDrive\Desktop\DS1\assignments-master-09-hash_experiment-cpp\09-hash_experiment\cpp> .\a.exe grow-tab 23
128 2.014 0.245
256 2.127 0.278
512 2.222 0.279
1024 2.184 0.170
2048 2.195 0.123
4096 2.227 0.128
8192 2.192 0.061
16384 2.225 0.078
32768 2.209 0.034
65536 2.226 0.034
131072 2.219 0.025
262144 2.218 0.019
524288 2.220 0.015
1048576 2.218 0.011
2097152 2.219 0.006
```

Fig: grow-tab

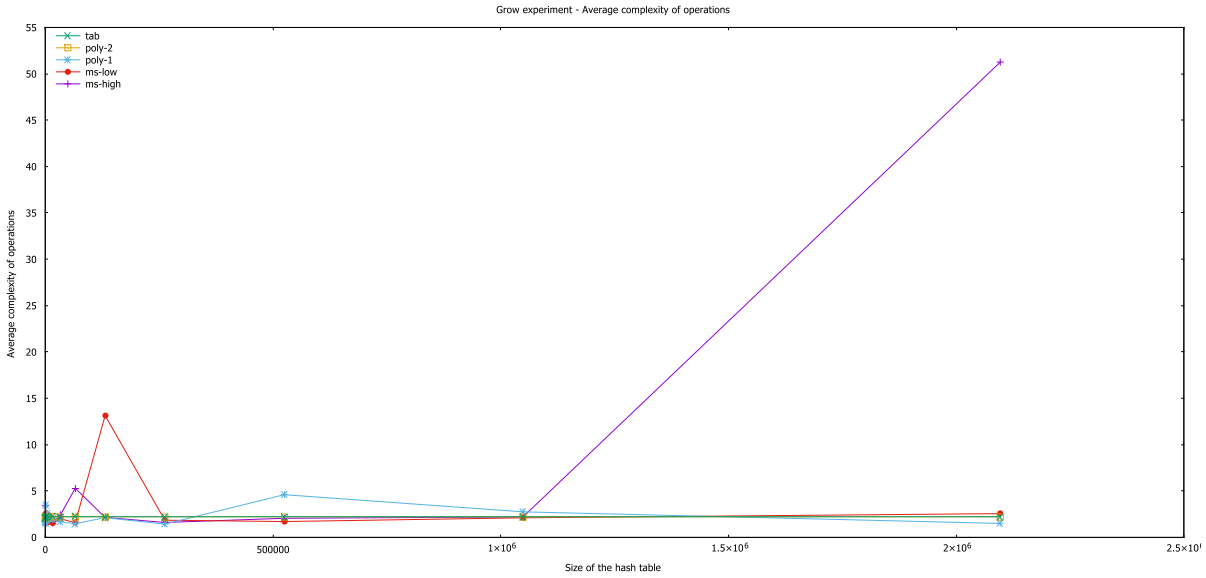

Fig: Graph with no logarithmic scale on either axis.

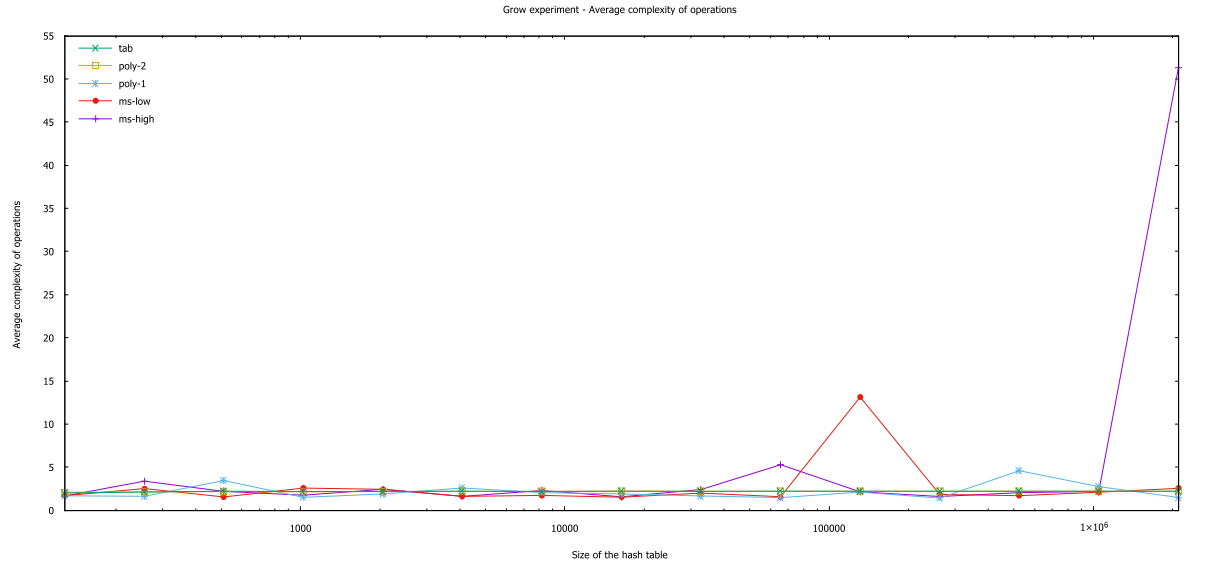

Fig: x axis has logarithmic scale.

Linear probing is the simplest form for collision handling in open-addressing, where each bucket contains at most one item. Linear probing has a major disadvantage that it tends to produce clusters, because of continuous intervals of cells occupied by items.

From the graph presented above, I find the graph with logarithmic scale on x axis is more fitting than the graph with no logarithmic scale. As the graph with logarithmic scale displays with distinct plots between 0 to 500000 intervals on x-axis, while graph with no logarithmic scale displayed not so clear plots. The graph pictures that the plots between intervals 0 to 60000 showed a good sign of performance with low step size. An interesting observation is that, ms-high shows a sudden raise at the end while it was being in oscillation with small rise and fall before. Also, the ms-low and poly-1 where in oscillation with small rise and fall, while ms-low has a sudden rise of around 13 after the interval 100000, and then it turned back to it's previous state. The most interesting part of the plot is that, poly-2 and tab hash overlapped and showed no display of oscillation in performance while it remained stable from start till end. Therefore, the lookup operation was done efficiently by poly-2 and tab hashes, while ms-high, ms-low and poly-1 hashes showed poor performance with too many oscillations. This can be cited from the theorem from the lectures, that when a hash function is truly or completely random functions, then it is the expected number of probes when searching for x is bounded by a constant independent of n, m, h, and x, where m -> table size, n -> the number of items, x -> an item and h being hash function. So, as per the theorem and tab hash being truly random (as stated from the lecture) and poly-2 hash shows similar states because, 2-independent hashing is useful that it leads to a small number of direct collisions, while the sensitivity of a hash function has caused much oscillation on other hash functions.

● Theorem: Using 2-independent hash functions, we can prove an O(n1/2) expected cost of lookups with linear probing, and there's a matching adversarial lower bound.

● Theorem: Using 3-independent hash functions, we can prove an O(log n) expected cost of lookups with linear probing, and there's a matching adversarial lower bound. Multiply hash can be sited as an example as stated from lectures.

● Theorem: Using 5-independent hash functions, we can prove an O(1) expected cost of lookups with linear probing.

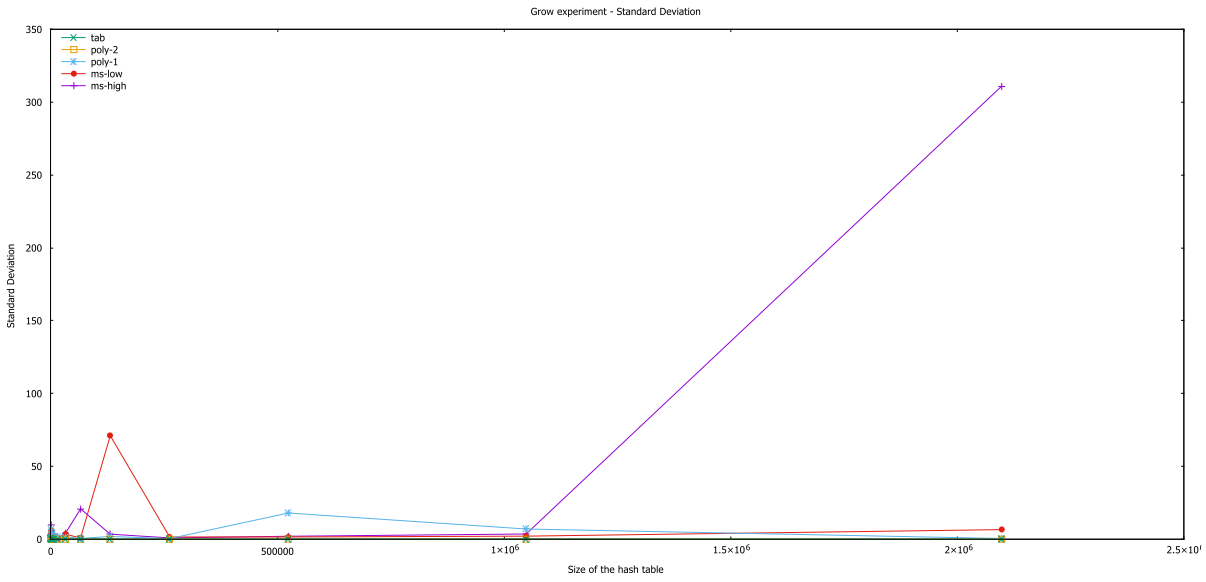

Fig: Graph with no logarithmic scale on either axis.

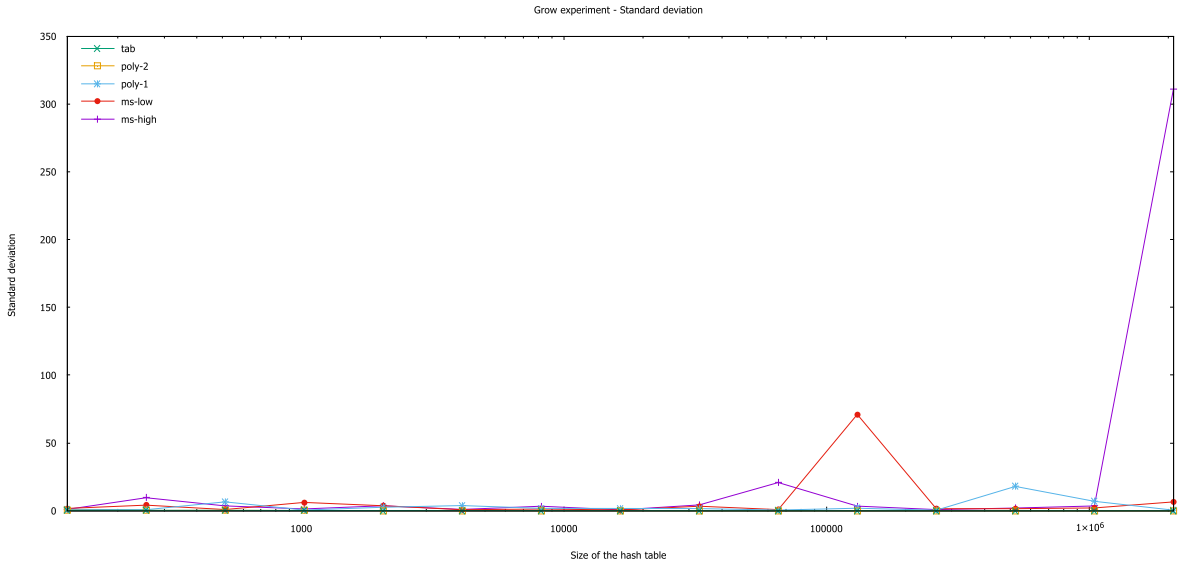

Fig: x axis has logarithmic scale.

From the graph presented above, I find the graph with logarithmic scale on x axis is more fitting than the graph with no logarithmic scale. As the graph with logarithmic scale displays with distinct plots between 0 to 500000 intervals on x-axis, while graph with no logarithmic scale displayed not so clear plots. Similar to the graphs from 'Grow experiment - Average complexity of operations' above, the graphs in 'Grow experiment – Standard Deviation' portraits similar graphs, with ms-high showing massive increase at the end, and ms-low and poly-1 showing plots with oscillation, while poly-2 and tab hashes were overlapped and stable from start till the end. This says that truly random hash function shows less collision than other functions and results in stable variance. From the graphs I can see that, number of steps is directly proportional to the Standard deviation, that the number of steps increases, the more the block is bad making critical. Increasing the degree of independence lets us control the variance of the distribution. With 2-independent hashing, we use one degree of independence to condition on knowing where some specific key lands. At that point, we only have one more degree of independence – not enough to control the variance. With 3-independent hashing, we use one degree of independence to condition on knowing where the key lands. We can then use the two remaining degrees of independence to control the variance and use Chebyshev's inequality. Small increases to the independence of a hash function can dramatically tighten concentration bounds.

Usage Experiment:

(x-axis to be in %)

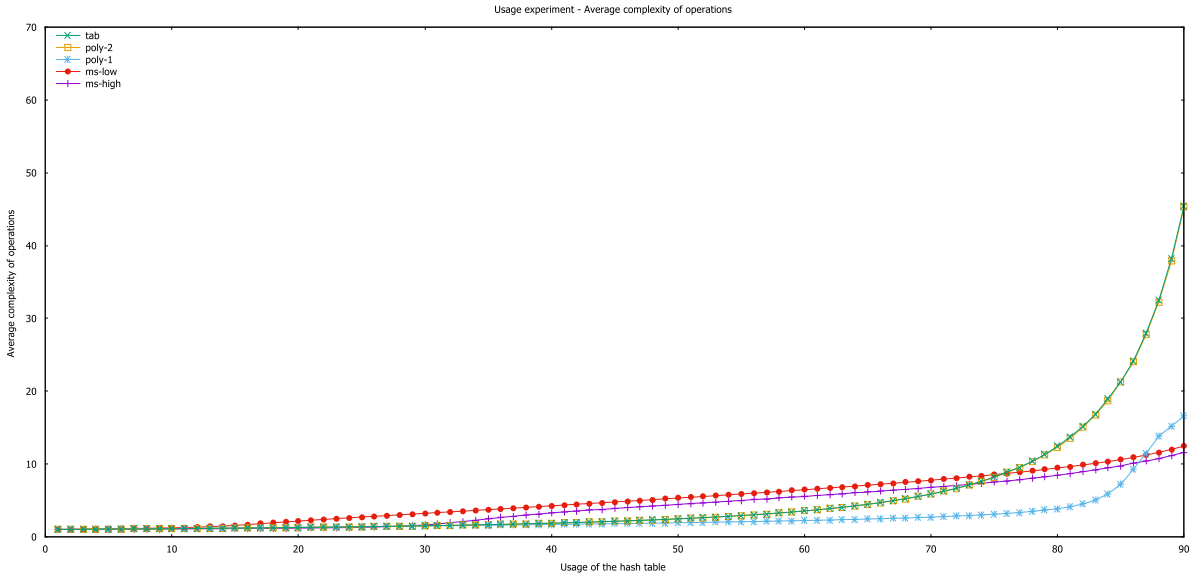

Fig: Graph with no logarithmic scale on either axis.

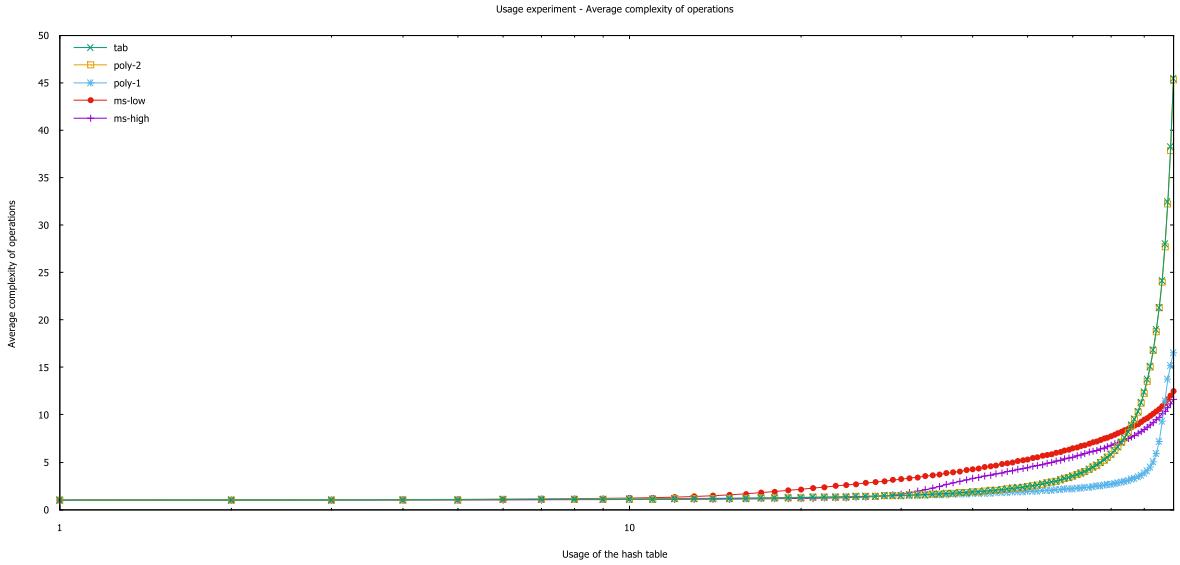

Fig: x axis has logarithmic scale.

From the graph presented above, I find the graph with no logarithmic scale is more fitting than the graph with logarithmic scale. As the graph with no logarithmic scale displays with distinct plots from the start till the end of the intervals on x-axis, while graph with logarithmic scale displayed not so clear plots. The plot of tab and poly-2 hashes overlapped and started from

stable but after 60% of usage, it gradually started increasing maximum and showed bad performance till 90% usage. While ms-low and ms-high started stable and raised slowly showing good performance till the end of 90%. And poly-1 showed good performance till 80% usage, but at the end of 90% usage, it finished slightly above ms-low and ms-high. Notice that this is O(1) for any fixed α, but as α grows, this runtime gets progressively worse, as expected. This statement can be cited from the lectures, from the theorem 'Chernoff bound for the right tail', where our ultimate goal is to estimate size of runs. In order to do this, we prove that a long run must contain at least one large critical block.

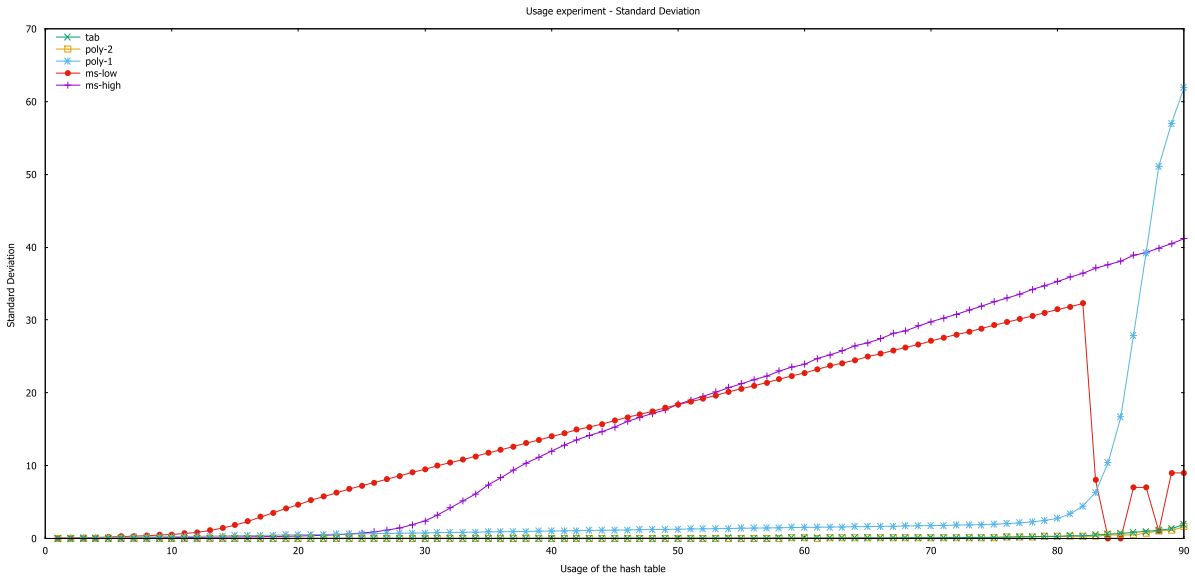

Fig: Graph with no logarithmic scale on either axis.

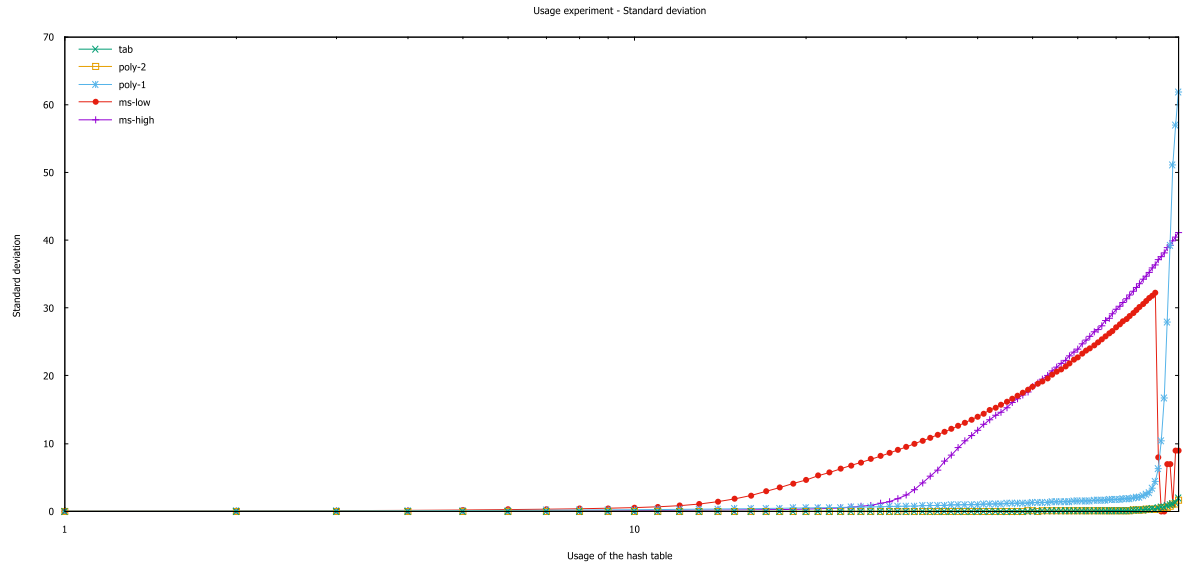

Fig: x axis has logarithmic scale.

From the graph presented above, I find the graph with no logarithmic scale is more fitting than the graph with logarithmic scale. As the graph with no logarithmic scale displays with distinct plots from the start till the end of the intervals on x-axis, while graph with logarithmic

scale displayed not so clear plots. The plots of ms-low and ms-high hashes showed a massive rise from the start of insertion and kept rising, while ms-low managed to drop at the end. Theorem (Mitzenmacher and Vadhan): Using 2- independent hash functions, if there is a reasonable amount of entropy in the distribution of the keys, linear probing takes time O(1). Unlike Grow experiment, the Standard Deviation plots is inversely proportional to the operations performed plots by usage experiment. As the tab and poly-2 hashes seemed to have low Standard deviation while they performed poor in average operations, and vice versa with other hash functions.

As conclusion, we can see from the plots that tab and poly-2 hashes showed good performance with lookup operation, while they drastically showed bad performance in insertion operations. Like wise ms-high and ms-low showed bad performance in lookup operation but was stable with insertion operations. Poly-1 seems to be the average performer of the 5 hash functions.