Programming Assignment 1

CS738 – Algorithms in the Real World
by Cole Lamers
2/22/2025

Table of Contents

Bloom Filter Report

The general intent of the Bloom Filter tests that I wanted to achieve was to just see how modifying the data, universe, and load factors affects the Bloom Filter. The control is the base run upon the first time generating the dataBloom.txt file. From there generateBloomData() was commented out and no longer used to keep the results similar. From here, I simply halved or doubled the value of a single argument. Using this to compare against both the control and the other test related to the same value altered would give me three total results to verify the impact altering the input arguments. The “All Doubled” and “All Halved” tests are also were conducted to see how the breadth of all being changed in tandem can affect the outcome.

Experiments for Bloom Filter

1.1.1 Control Test

The purpose of this test was to establish a baseline for evaluating other experiments. It was expected to serve as a reference point. There were really no expectations of this one overall aside from serving as the baseline control item to compare everything else to.

1.1.2 Half Data Size

This experiment aimed to assess the effect of reducing the data size on accuracy and performance. The hypothesis was that a smaller dataset would lead to lower false positives and improved search speed. However, the false positive rate increased significantly to 61.73%, while search times showed worse results. To me this contradicted my expectations, suggesting that a smaller dataset may have increased hash collisions because of the smaller universe.

1.1.3 Double Data Size

The goal of this test was to see whether doubling the dataset size would enhance or degrade performance. From the previous test, I estimated that since the universe had more slots available to it, the there would be less false positives decreasing collisions. The results showed this, but also showed how drastically it affected successful search times. Insertions and Hash/BF Searches were very fast however. This indicates that a larger dataset improves accuracy but at a negative detriment to successful searches.

1.1.4 Half Universe

This experiment investigated how reducing the universe size affects search accuracy. My hypothesis was that a smaller universe would lead to a higher false positive rate due to increased density in the Bloom Filter. The outcome supported this assumption, as the false positive rate rose to 60.61%, demonstrating a significant drop in accuracy. Searches and insertions were similar to the control, but overall this confirmed that a reduced universe size leads to more hash collisions and poorer performance.

1.1.5 Double Universe

The purpose of this test was to determine whether increasing the universe size would improve accuracy. From all previous tests, I assumed that a larger universe would decrease false positives and but decrease search reliability. The results validated this as the false positive rate dropped to 0.2278%, indicating a significant improvement. Although the successful search was half of the control and the Insertions and Search times were slightly or more than 2x better than the control. Which makes sense because of the larger universe minimizing collisions.

1.1.6 Half Load Factor/Alpha Size

This test examined the impact of reducing the load factor on performance and accuracy. The hypothesis was that lowering the load factor would decrease accuracy and increase false positives because of the fewer hash functions that occurred. This was the most surprising to me because I would’ve figured less hash functions would’ve increased collisions and Insertions/Searches would’ve only slightly increased in speed due to fewer hash functions. The results did not show this where the Successful search stayed the same, but the false positive rate dropped to 3.67%. My only guess is fewer hash functions decreases the probability that two different hash functions calculations will collide.

1.1.7 Double Load Factor/Alpha Size

The objective of this experiment was to test the effects of increasing the load factor. Because of the previous test, I assumed the false positive rate would rise, and it did, to 39.77%, significantly higher than the control, although Insertions did better, Searching was worse. If I recall correctly, we discussed the Birthday Paradox, and my assumption is related to how that works, because as the number of hash functions increase, the hashes increase the probability of potential collisions. The more comparisons you make, the more likely you are to encounter a collision.

1.1.8 All Half

This experiment tested the combined effect of halving the data size, universe size, and load factor. My hypothesis was that the accuracy would degrade significantly due to higher element density. This was the only one I felt like made sense based off of the results, as the false positive rate reached 66.45%, the highest among all tests. This confirms that reducing all factors simultaneously leads to a severe drop in accuracy and performance.

1.1.9 All Double

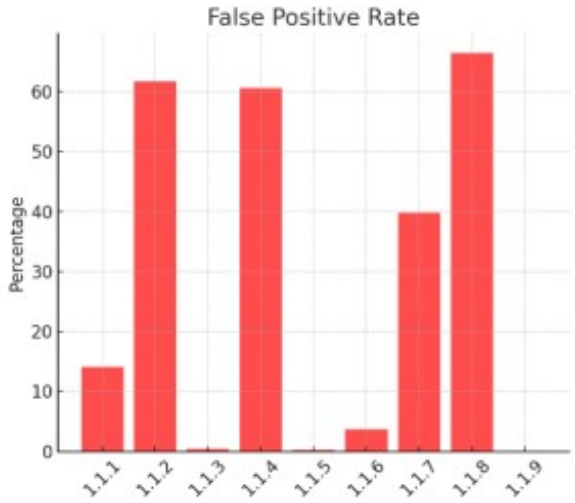
The final test examined the combined effect of doubling the data size, universe size, and load factor. I expected that accuracy would improve, the false positives would drop, and the Successful search would drop. The results confirmed these as the false positive rate fell to 0.0003%, the lowest in all tests. This suggests that increasing all three parameters together significantly enhances accuracy and reduces hash collisions, but is not as accurate in terms of a successful search.

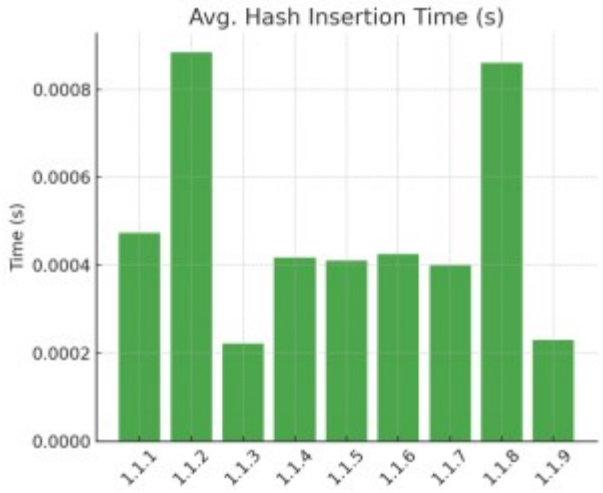
Data Collected for Bloom Filter

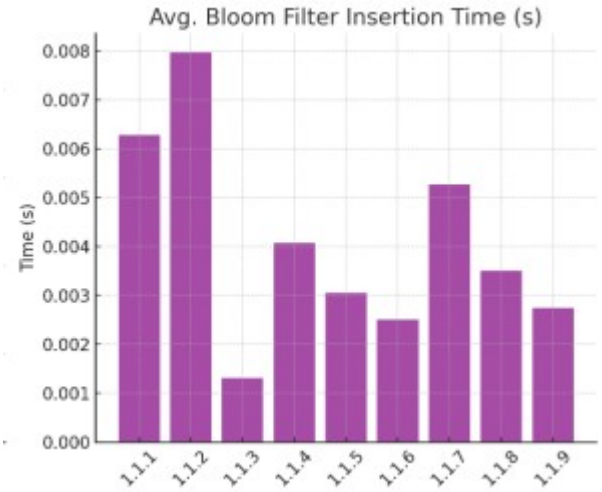
Number	Test	Data Size	Universe Size	Load Factor	# of Hash Funcs.	% Successful Search	False Positive Rate	False Negative Rate
1.1.1	Control	1000000	4000000	16	12	22.110800%	14.083825%	0.000000%
1.1.2	Half Data Size	500000	2000000	16	12	22.099450%	61.731150%	0.000000%
1.1.3	Double Data Size	2000000	8000000	16	12	0.1105540%	0.3631500%	0.000000%
1.1.4	Half Universe	1000000	2000000	16	12	22.099450%	60.618350%	0.000000%
1.1.5	Double Universe	1000000	8000000	16	12	11.055400%	0.2278000%	0.000000%
1.1.6	Half Load Factor/Alpha Size	1000000	4000000	8	6	22.110800%	3.6712000%	0.000000%
1.1.7	Double Load Factor/Alpha Size	1000000	4000000	32	23	22.110800%	39.769125%	0.000000%
1.1.8	All Half	500000	1000000	8	6	22.074900%	66.451900%	0.000000%
1.1.9	All Double	2000000	16000000	32	23	5.5277000%	0.0003188%	0.000000%

Number	Test	Avg. Hash Insertion Time	Avg. BF Insertion Time	Avg. Hash Search Time	Avg. BF Search Time
1.1.1	Control	0.0004730	0.00627300	0.0000895000	0.002150000
1.1.2	Half Data Size	0.0008840	0.00796600	0.0000950000	0.002552500
1.1.3	Double Data Size	0.0002215	0.00130250	0.0000808750	0.000762500
1.1.4	Half Universe	0.0004170	0.00407500	0.0000930000	0.002735500
1.1.5	Double Universe	0.0004110	0.00305000	0.0000758750	0.000759750
1.1.6	Half Load Factor/Alpha Size	0.0004250	0.00250200	0.0000602500	0.000626250
1.1.7	Double Load Factor/Alpha Size	0.0004000	0.00526800	0.0000987500	0.003851250
1.1.8	All Half	0.0008600	0.00349800	0.0000920000	0.001137000
1.1.9	All Double	0.0002300	0.00273900	0.0000769375	0.000695625

Charts for Bloom Filter







1 Tests

1.1 Bloom Filter Test Results

1.1.1 Control

```
Data size = 1000000
Universe size = 4000000
Load factor = 16
Number of hash functions = 12

% of Successful Searches = 22.1108% (884432 out of 4000000)
False Positive Rate = 14.083825%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 4.73E-4; BF = 0.006273
Average Search Time: Hash = 8.95E-5; BF = 0.00215
```

1.1.2 Half Data Size

```
Data size = 500000
Universe size = 2000000
Load factor = 16
Number of hash functions = 12

% of Successful Searches = 22.09945% (441989 out of 2000000)
False Positive Rate = 61.73115%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 8.84E-4; BF = 0.007966
Average Search Time: Hash = 9.5E-5; BF = 0.0025525
```

1.1.3 Double Data Size

```
Data size = 2000000
Universe size = 8000000
Load factor = 16
Number of hash functions = 12

% of Successful Searches = 11.0554% (884432 out of 8000000)
False Positive Rate = 0.36315%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 2.215E-4; BF = 0.0013025
Average Search Time: Hash = 8.0875E-5; BF = 7.625E-4
```

1.1.4 Half Universe Size

```
Data size = 1000000
Universe size = 2000000
Load factor = 16
Number of hash functions = 12

% of Successful Searches = 22.09945% (441989 out of 2000000)
False Positive Rate = 60.61835%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 4.17E-4; BF = 0.004075
Average Search Time: Hash = 9.3E-5; BF = 0.0027355
```

1.1.5 Double Universe Size

```
Data size = 1000000
Universe size = 8000000
Load factor = 16
Number of hash functions = 12

% of Successful Searches = 11.0554% (884432 out of 8000000)
False Positive Rate = 0.2278%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 4.11E-4; BF = 0.00305
Average Search Time: Hash = 7.5875E-5; BF = 7.5975E-4
```

1.1.6 Half Load Factor/Alpha Size

```
Data size = 1000000
Universe size = 4000000
Load factor = 8
Number of hash functions = 6

% of Successful Searches = 22.1108% (884432 out of 4000000)
False Positive Rate = 3.6712%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 4.25E-4; BF = 0.002502
Average Search Time: Hash = 6.025E-5; BF = 6.2625E-4
```

1.1.7 Double Load Factor/Alpha Size

```
Data size = 1000000
Universe size = 4000000
Load factor = 32
Number of hash functions = 23

% of Successful Searches = 22.1108% (884432 out of 4000000)
False Positive Rate = 39.769125%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 4.0E-4; BF = 0.005268
Average Search Time: Hash = 9.875E-5; BF = 0.00385125
```

1.1.8 All Half

```
Data size = 500000
Universe size = 1000000
Load factor = 8
Number of hash functions = 6

% of Successful Searches = 22.0749% (220749 out of 1000000)
False Positive Rate = 66.4519%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 8.6E-4; BF = 0.003498
Average Search Time: Hash = 9.2E-5; BF = 0.001137
```

1.1.9 All Doubled

```
Data size = 2000000
Universe size = 16000000
Load factor = 32
Number of hash functions = 23

% of Successful Searches = 5.5277% (884432 out of 16000000)
False Positive Rate = 3.1875E-4%
False Negative Rate = 0.0%

Average Insertion Time: Hash = 2.3E-4; BF = 0.002739
Average Search Time: Hash = 7.69375E-5; BF = 6.95625E-4
```

Count-Min Sketch (CMS) Report

With these tests, they require a little bit more thought because of the complexity of the CMS compared to the Bloom Filter. To start off, my primary goal was to test with only 1 CHUNK_SIZE of data for the majority of the tests. This is the Control test for everything. Later on do I adjust the amount of CHUNK_SIZE data utilized because I have to rerun the generateCMSData() function and get all new data. My adjustment for this independent of all other tests helps ensure the integrity of the Control and other test when being compared against one another. I conducted similar tests from what I did with the Bloom Filter, where I had a control, and modified individual values by double or by half, and tracked the results. The only additional actions I did this time was I added 3 more, where I reran the generateCMSData() function and regenerated the data, but I generated it with 4 chunks and 8 chunks respectively. I was under the impression that part of the reason my doubling and halving of “k” wasn’t producing any significant results was because the data set was too small for it to be impacted. I then decided to choose to do separate tests of 4 and 8 total chunks because I wanted more data to see if the results would vary. The boundaries in these tests did begin to show, however, they were not as significant as I expected. I thought the number of heavy hitters would affect the results, but in reality, it is not that significant in regards to the CMS.

Experiments for CMS

Test 1.2.1: Control

This test served as a baseline to compare with the other setups. The hypothesis was that the standard setup would give accurate estimates with minimal variation. The outcome confirms this with the actual average sitting at 1000 and only a minor change in estimates of the average difference of 19.415. The estimates remained close to the actual values, supporting the initial hypothesis. No underestimates were recorded and the heavy hitter count was exactly as expected at 24.

Test 1.2.2: Double Universe

Here my goal was to see how doubling the universe size would impact estimation accuracy. This hypothesis suggests that it would improve estimates and produce more heavy hitters. The result was actually no deviation in accuracy and the estimates remained spot on with an average difference of zero. The heavy hitter count stayed the same again at 24. While the estimates were perfect, the heavy hitter count remained unchanged which actually contradicted my hypothesis.

Test 1.2.3: Half Universe

In this test I reduced the universe size by half. I was hypothesizing that this would decrease accuracy. Interestingly though the estimates were still accurate with an average difference of 0. The heavy hitter count dropped to 16. While the accuracy remained intact, the decrease in heavy hitters also occurred. My assumption is that it's because with fewer values, it's easier for fewer items to become "heavy hitters," leading to a lower count of them.

Test 1.2.4: Double ϵ and δ

By increasing both error tolerance (ϵ) and confidence level (δ), this test aimed to introduce more variation in estimates while keeping the heavy hitter count consistent. My thought with this test was to see how if they would significantly alter the heavy hitters and estimates negatively. The outcome showed some variation where the average difference jumped up drastically to 88.687, though the heavy hitter count stayed the same at 24, but with an estimate of 29. My guess is that the additional variations in tolerances caused the differences, but still kept the heavy hitters in check.

Test 1.2.5: Half ϵ and δ

The idea behind halving ϵ and δ was to improve accuracy and reduce the number of heavy hitters. I hypothesized that it would have the opposite effect of 1.2.4. To compare, the estimates remained accurate and the heavy hitter count stayed steady at 24. Because it's capturing the heavy hitters in a similar fashion, I believe it's safe to assume reducing these values does not affect accuracy detection.

Test 1.2.6: Double k Heavy Hitters

Doubling the k-value was expected to increase the number of detected heavy hitters. That was at least my thought. However the outcome didn't align with that at all. The heavy hitter count stayed at 24, with no significant changes. The accuracy remained strong (average difference of 0), but the number of heavy hitters didn't increase. My guess was that it was similar to the test of 1.2.5 where the CMS system was set up well enough that these adjustments did not affect it much.

Test 1.2.7: Half k Heavy Hitters

Here was another test on k, halving it's value. Since the previous one did not show much of a difference compared to the control, I considered that half might affect it. What came out actually was the same results again with the heavy hitter count remained the same at 24, despite halving k. The accuracy was still perfect as well. The k heavy hitter value then to me seems like it might not be affected here because the data size is too small.

Test 1.2.8: All Double

In this test, both the universe size and k were doubled, which was expected to increase both estimate accuracy and heavy hitter count. The estimates were accurate (average difference of zero), but the heavy hitter count didn't increase as predicted, staying at 24. So, the hypothesis didn't hold up entirely, as the heavy hitter count didn't change.

Test 1.2.9: All Half

Halving both the universe and k-values was expected to reduce accuracy and heavy hitter detection. These estimates remained accurate with an average difference of 0 and the heavy hitter count decreases to 16. This result contradicted my hypothesis as neither accuracy nor heavy hitter count changed significantly. I can't exactly explain why, as I figured with a consistent reduction in everything would've garnered similar results to the doubled one almost as if the variable changes could be reflected in a consistent ratio.

Test 1.2.10: 4x Data Points of Control

In this test I quadruped the data points and expected this to improve both accuracy and heavy hitter detection. The results showed an increase in heavy hitters to 69, which supported my hypothesis. However, the average difference shot up to 256.975, indicating that there was more variability with the added data points. So while the heavy hitter count increased, the accuracy was impacted by the larger data set.

Test 1.2.11: 4x Data Points Double k

Here both the data points were bumped up to 4 times the amount and k was doubled. My expectation was that this would improve estimates and heavy hitter detection. The number of heavy hitters did rise to 77, but the accuracy took a hit, with the average difference growing to 149.994. Although from the previous tests 1.2.6 and 1.2.7, it did confirm for me that the alteration of k was more reflective on the data size.

Test 1.2.12: 4x Data Points Half k

This test aimed to reduce both heavy hitters and accuracy by increasing the data points while halving k. The heavy hitter count did drop to 69, supporting the hypothesis, but the accuracy remained strong with an average difference of 58.852. Again, like from test 1.2.11, this confirmed that k required a larger data set to take effect.

Test 1.2.13: 8x Data Points of Control

In this test, I increased the data points by eight times and expected this to improve both accuracy and heavy hitter detection. The heavy hitter count rose to 147, which was in line with my expectations, but the average difference jumped to 437.18. This showed that while more heavy hitters were detected, the larger dataset introduced more variability and negatively affected accuracy.

Test 1.2.14: 8x Data Points Double k

Here, I increased the data points by eight times and doubled k, thinking this would improve accuracy and heavy hitter detection. The number of heavy hitters rose to 158, which confirmed my hypothesis, but accuracy suffered with the average difference increasing to 286.734. The increased heavy hitter count was as expected, but the accuracy declined, similar to the previous tests where data size played a larger role.

Test 1.2.15: 8x Data Points Half k

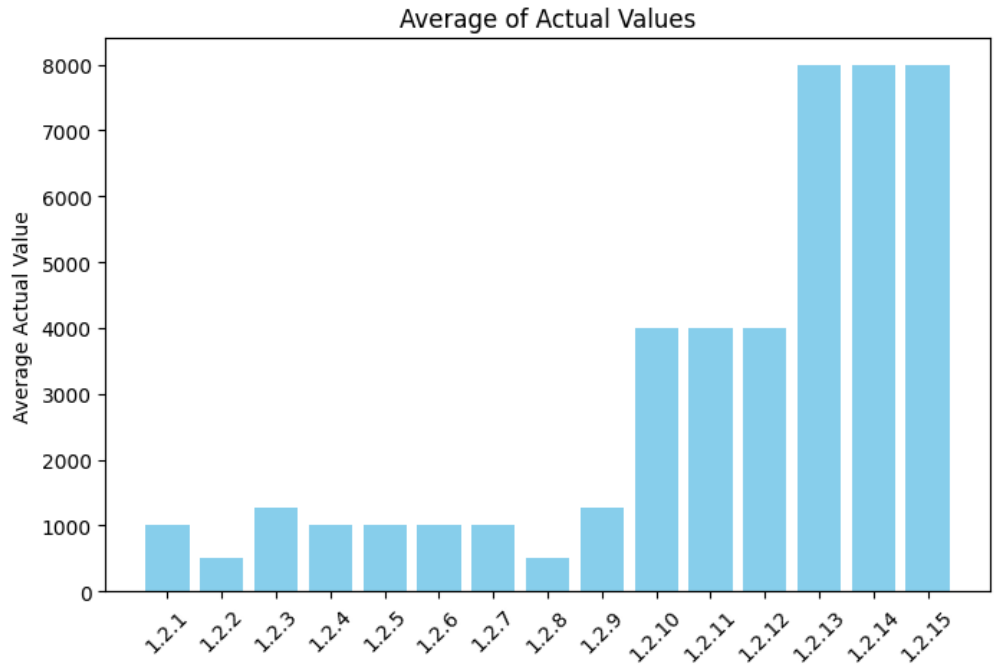
For this test, I increased the data points by eight times and halved k, expecting fewer heavy hitters and reduced accuracy. The heavy hitter count dropped slightly to 147, supporting my hypothesis, but the accuracy remained unexpectedly high with an average difference of 823.131. Despite detecting fewer heavy hitters, the accuracy did not decrease as anticipated, which suggests that the effect of halving k may need a larger data set to become noticeable.

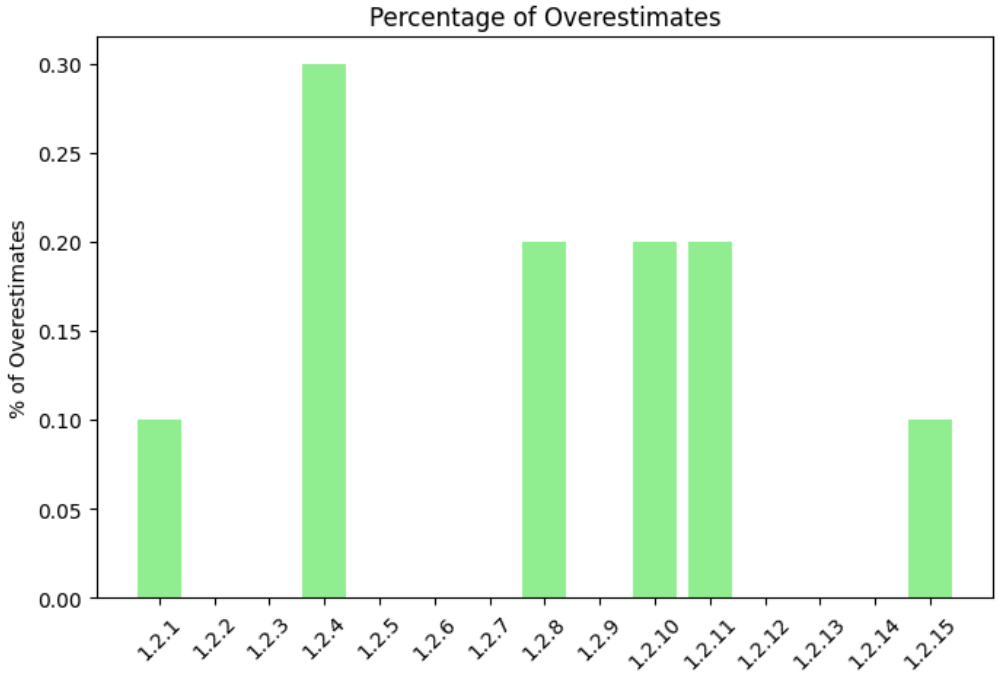
Data Collected for CMS

Test Number	1.2.1	1.2.2	1.2.3	1.2.4	1.2.5	1.2.6	1.2.7	1.2.8	1.2.9	1.2.10
Test	Control	Double Universe	Half Universe	Double ε and δ	Half ε and δ	Double k Heavy Hitters	Half k Heavy Hitters	All Double	All Half	4x Data Points of Control
Number of data points (n)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	4000000
Universe	1000	2000	500	1000	1000	1000	1000	2000	500	1000
Number of columns (B)	272	272	272	136	544	272	272	136	544	272
Number of rows (hash functions)	4	4	4	3	5	4	4	3	5	4
Epsilon	0.01	0.01	0.01	0.02	0.005	0.01	0.01	0.02	0.005	0.01
Delta	0.01	0.01	0.01	0.02	0.005	0.01	0.01	0.02	0.005	0.01
n/B	3676	3676	3676	7352	1838	3676	3676	7352	1838	14705
n*epsilon	10000	10000	10000	20000	5000	10000	10000	20000	5000	40000
Number of underestimates	0	0	0	0	0	0	0	0	0	0
Average of actual	1000	500	1274.796	1000	1000	1000	1000	500	1274.796	4000
Average of (estimate - actual)	0	0	0	88.687	0	0	0	86.9185	0	256.975
% of cases where (estimate - actual >= n*epsilon)	0.00%	0.00%	0.00%	0.30%	0.00%	0.00%	0.00%	0.20%	0.00%	0.20%
% of cases where (estimate >= actual + n/B)	0.00%	0.00%	0.00%	0.40%	0.00%	0.00%	0.00%	0.30%	0.00%	0.60%
Average difference when we compute actual as (estimate - n/B)	86.629	43.3145	115.006	223.243	44.112	86.629	86.629	146.8445	58.816	1115.373
k for k-heavy hitters	10000	10000	10000	10000	10000	20000	5000	20000	5000	10000
Number of actual heavy-hitters	24	24	16	24	24	24	24	24	16	69
Number of estimated heavy-hitters	24	24	16	29	24	24	24	31	16	77

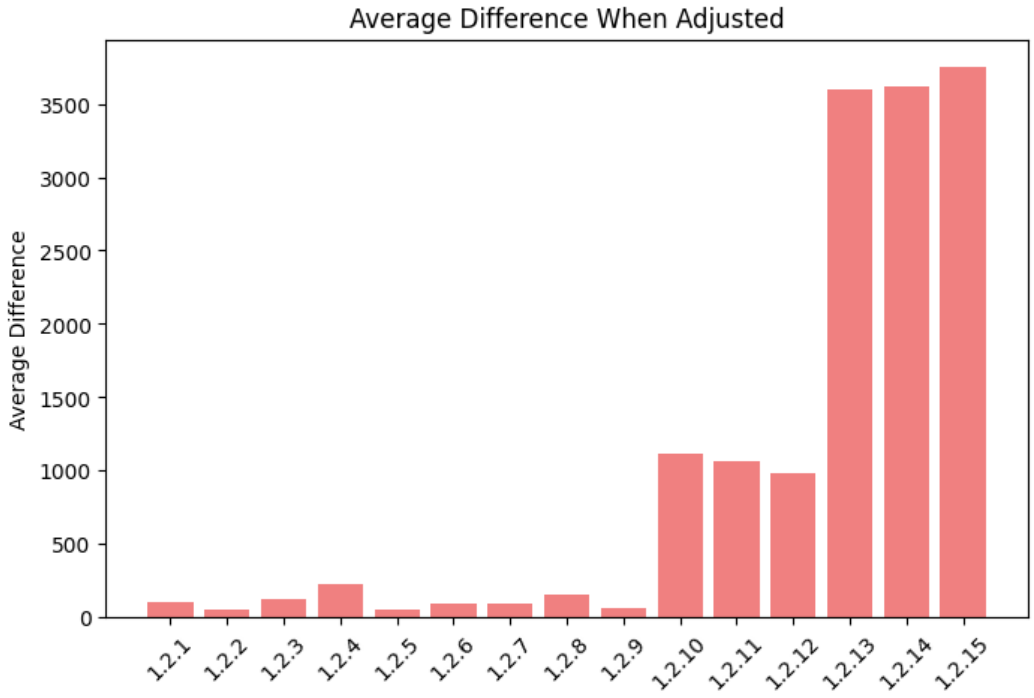
Test Number	1.2.10	1.2.11	1.2.12	1.2.13	1.2.14	1.2.15
Test	4x Data Points of Control	4x Data Points Double k	4x Data Points Half k	8x Data Points of Control	8x Data Points Double k	8x Data Points Half k
Number of data points (n)	4000000	4000000	4000000	8000000	8000000	8000000
Universe	1000	1000	1000	1000	1000	1000
Number of columns (B)	272	272	272	272	272	272
Number of rows (hash functions)	4	4	4	4	4	4
Epsilon	0.01	0.01	0.01	0.01	0.01	0.01
Delta	0.01	0.01	0.01	0.01	0.01	0.01
n/B	14705	14705	14705	29411	29411	29411
n*epsilon	40000	40000	40000	80000	80000	80000
Number of underestimates	0	0	0	0	0	0
Average of actual	4000	4000	4000	8000	8000	8000
Average of (estimate - actual)	256.975	149.994	58.852	437.18	286.734	823.131
% of cases where (estimate - actual >= n*epsilon)	0.20%	0.20%	0.00%	0.00%	0.00%	0.10%
% of cases where (estimate >= actual + n/B)	0.60%	0.30%	0.20%	0.30%	0.10%	0.90%
Average difference when we compute actual as (estimate - n/B)	1115.373	1058.6	983.041	3594.883	3622.743	3751.603
k for k-heavy hitters	10000	20000	5000	10000	20000	5000
Number of actual heavy-hitters	69	69	69	147	148	147
Number of estimated heavy-hitters	77	73	71	168	158	172

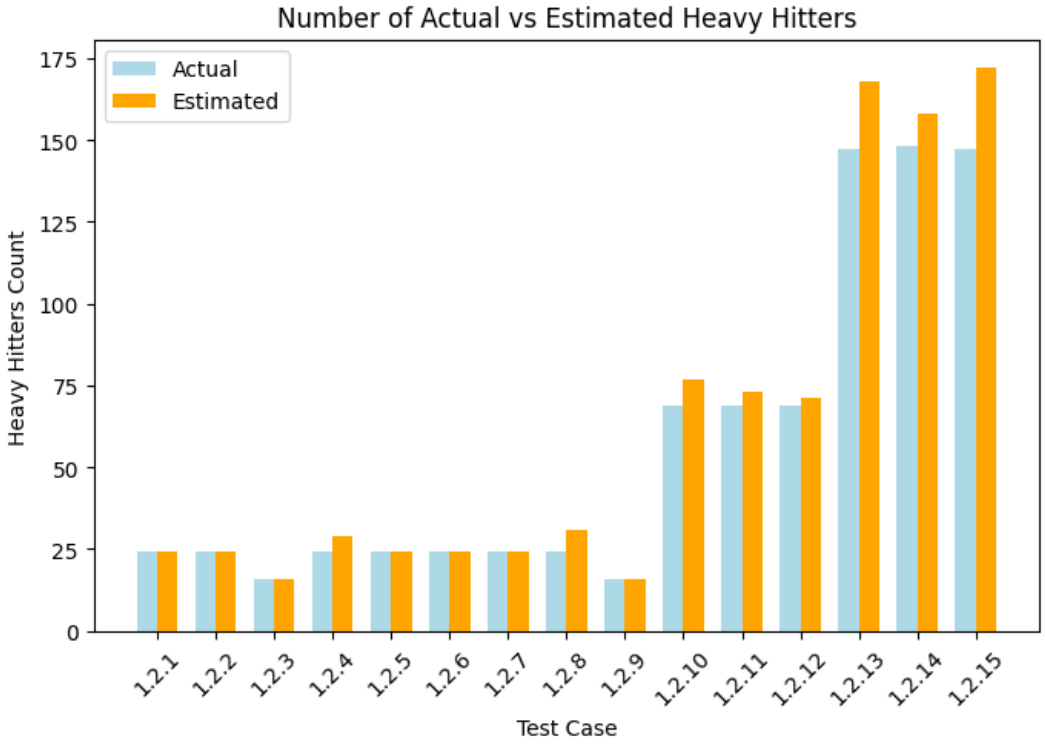
Charts for CMS





Charts for CMS cont'd.





1.2 CMS Test Results

1.2.1 Control

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 3676
n*epsilon: 10000

Number of underestimates: 0
Average of actual: 1000.0
Average of (estimate - actual): 19.415
% of cases where (estimate - actual >= n*epsilon): 0.1%
% of cases where (estimate >= actual + n/B): 0.1%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 24
```

1.2.2 Double Universe

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 2000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 3676
n*epsilon: 10000

Number of underestimates: 0
Average of actual: 500.0
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 24
```

1.2.3 Half Universe

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 500
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 3676
n*epsilon: 10000

Number of underestimates: 0
Average of actual: 1274.796
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 16
Number of estimated heavy-hitters: 16
```

1.2.4 Double ϵ and δ

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 1000
Number of columns (B): 136
Number of rows (hash functions): 3
Epsilon: 0.02
Delta: 0.02
n/B: 7352
n*epsilon: 20000

Number of underestimates: 0
Average of actual: 1000.0
Average of (estimate - actual): 88.687
% of cases where (estimate - actual >= n*epsilon): 0.3%
% of cases where (estimate >= actual + n/B): 0.4%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 29
```

1.2.5 Half ϵ and δ

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 1000
Number of columns (B): 544
Number of rows (hash functions): 5
Epsilon: 0.005
Delta: 0.005
n/B: 1838
n*epsilon: 5000

Number of underestimates: 0
Average of actual: 1000.0
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 24
```

1.2.6 Double k Heavy Hitters

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 3676
n*epsilon: 10000

Number of underestimates: 0
Average of actual: 1000.0
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 20000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 24
```

1.2.7 Half k Heavy Hitters

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 3676
n*epsilon: 10000

Number of underestimates: 0
Average of actual: 1000.0
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 5000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 24
```

1.2.8 All Double

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 2000
Number of columns (B): 136
Number of rows (hash functions): 3
Epsilon: 0.02
Delta: 0.02
n/B: 7352
n*epsilon: 20000

Number of underestimates: 0
Average of actual: 500.0
Average of (estimate - actual): 86.9185
% of cases where (estimate - actual >= n*epsilon): 0.2%
% of cases where (estimate >= actual + n/B): 0.3%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 20000
Number of actual heavy-hitters: 24
Number of estimated heavy-hitters: 31
```

1.2.9 All Half

```
Read 1000000 values.

Number of data points (n): 1000000
Universe: 500
Number of columns (B): 544
Number of rows (hash functions): 5
Epsilon: 0.005
Delta: 0.005
n/B: 1838
n*epsilon: 5000

Number of underestimates: 0
Average of actual: 1274.796
Average of (estimate - actual): 0.0
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.0%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 5000
Number of actual heavy-hitters: 16
Number of estimated heavy-hitters: 16
```

1.2.10 4x Data Points of Control

```
Read 1000000 values.
Read 2000000 values.
Read 3000000 values.
Read 4000000 values.

Number of data points (n): 4000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 14705
n*epsilon: 40000

Number of underestimates: 0
Average of actual: 4000.0
Average of (estimate - actual): 256.975
% of cases where (estimate - actual >= n*epsilon): 0.2%
% of cases where (estimate >= actual + n/B): 0.6%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 69
Number of estimated heavy-hitters: 77
```

1.2.11 4x Data Points Double k

```
Number of data points (n): 4000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 14705
n*epsilon: 40000

Number of underestimates: 0
Average of actual: 4000.0
Average of (estimate - actual): 149.994
% of cases where (estimate - actual >= n*epsilon): 0.2%
% of cases where (estimate >= actual + n/B): 0.3%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 20000
Number of actual heavy-hitters: 69
Number of estimated heavy-hitters: 73
```

1.2.12 4x Data Points Half k

```
Read 1000000 values.
Read 2000000 values.
Read 3000000 values.
Read 4000000 values.

Number of data points (n): 4000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 14705
n*epsilon: 40000

Number of underestimates: 0
Average of actual: 4000.0
Average of (estimate - actual): 58.852
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.2%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 5000
Number of actual heavy-hitters: 69
Number of estimated heavy-hitters: 71
```

1.2.13 8x Data Points of Control

```
Read 1000000 values.
Read 2000000 values.
Read 3000000 values.
Read 4000000 values.
Read 5000000 values.
Read 6000000 values.
Read 7000000 values.
Read 8000000 values.

Number of data points (n): 8000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 29411
n*epsilon: 80000

Number of underestimates: 0
Average of actual: 8000.0
Average of (estimate - actual): 437.18
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.3%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 10000
Number of actual heavy-hitters: 147
Number of estimated heavy-hitters: 168
```

1.2.14 8x Data Points Double k

```
Read 1000000 values.
Read 2000000 values.
Read 3000000 values.
Read 4000000 values.
Read 5000000 values.
Read 6000000 values.
Read 7000000 values.
Read 8000000 values.

Number of data points (n): 8000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 29411
n*epsilon: 80000

Number of underestimates: 0
Average of actual: 8000.0
Average of (estimate - actual): 286.734
% of cases where (estimate - actual >= n*epsilon): 0.0%
% of cases where (estimate >= actual + n/B): 0.1%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 20000
Number of actual heavy-hitters: 148
Number of estimated heavy-hitters: 158
```

1.2.15 8x Data Points Half k

```
Read 1000000 values.
Read 2000000 values.
Read 3000000 values.
Read 4000000 values.
Read 5000000 values.
Read 6000000 values.
Read 7000000 values.
Read 8000000 values.

Number of data points (n): 8000000
Universe: 1000
Number of columns (B): 272
Number of rows (hash functions): 4
Epsilon: 0.01
Delta: 0.01
n/B: 29411
n*epsilon: 80000

Number of underestimates: 0
Average of actual: 8000.0
Average of (estimate - actual): 823.131
% of cases where (estimate - actual >= n*epsilon): 0.1%
% of cases where (estimate >= actual + n/B): 0.9%
Average difference when we compute actual as (estimate - n/B):

k for k-heavy hitters: 5000
Number of actual heavy-hitters: 147
Number of estimated heavy-hitters: 172
```

Citations

https://en.wikipedia.org/wiki/Bloom_filter
https://en.wikipedia.org/wiki/Count%E2%80%93min_sketch
https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/
https://www.geeksforgeeks.org/count-min-sketch-in-java-with-examples/

I used ChatGPT to verify my code was correct and that the implementation satisfied the conditions. I also used ChatGPT to provide the Python code to help generate the charts.