RAD Implementeringsprojekt

Emil Nielsen - rjq923, Daniel Kjeldsen - vsq693

June 8, 2022

1 Introduction

The goal of this project was to practice implementing hash functions via. a hash table chaining or and using count-sketch as a method of estimating the second moment of frequencies. We measured running times experimenting with different variables and will analyze our results below.

Part 1

We would expect the run-time of multiplyShift to be faster than multiplyModPrime because the data types we use for multiplyModPrime have bigger type values (BigInteger vs. ulong), and we perform more expensive operation in multiplyModPrime. We tested the run time for different n and got the following.

n	multiplyShift	multiplyModPrime
1000	1ms	2ms
10000	5ms	11ms
100000	14ms	77ms
1000000	83ms	550ms
10000000	482ms	5239ms

As expected we see that the run-time for multiplyShift is significantly smaller than multiplyModPrime. However, some of the variation might be caused by an unoptimal implementation of multiplyModPrime.

Part 3

We have implemented our square sum and have picked n = 16777216 and then pick different l.

l	shift	prime
4	1297ms	11688ms
8	1462ms	10375ms
12	1424ms	10269ms
16	1509ms	10057ms
20	3389ms	12190ms
22	6173ms	14951ms
22	13259ms	27805ms
24	13311ms	Out of memory.

As we see the time increases as we increase l. At l = 24 we ran out of memory. Before running out of memory the time for multiplyShift was beginning to approach that of multiplyModPrime as it went from a factor of 9 to almost 2.

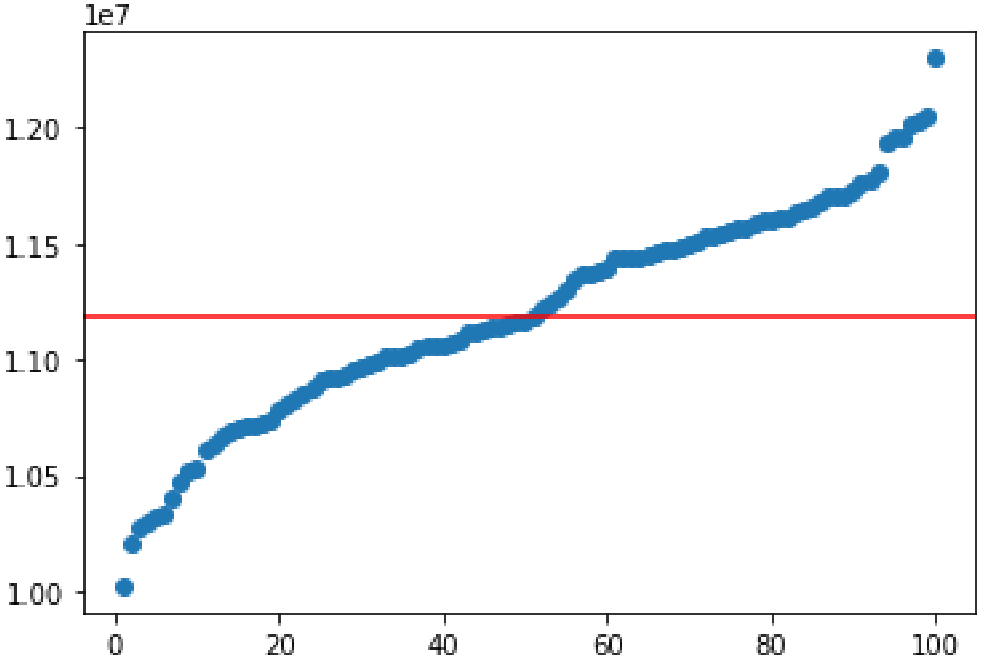
Part 7 and 8

First run: $m = 2^{10}$

100 Trials

The correct sum of squares is 11,184,810 and the expected variance is 549,755,813,888. In the first case, where we have $m = 2^{10}$, we get the following plot and values:

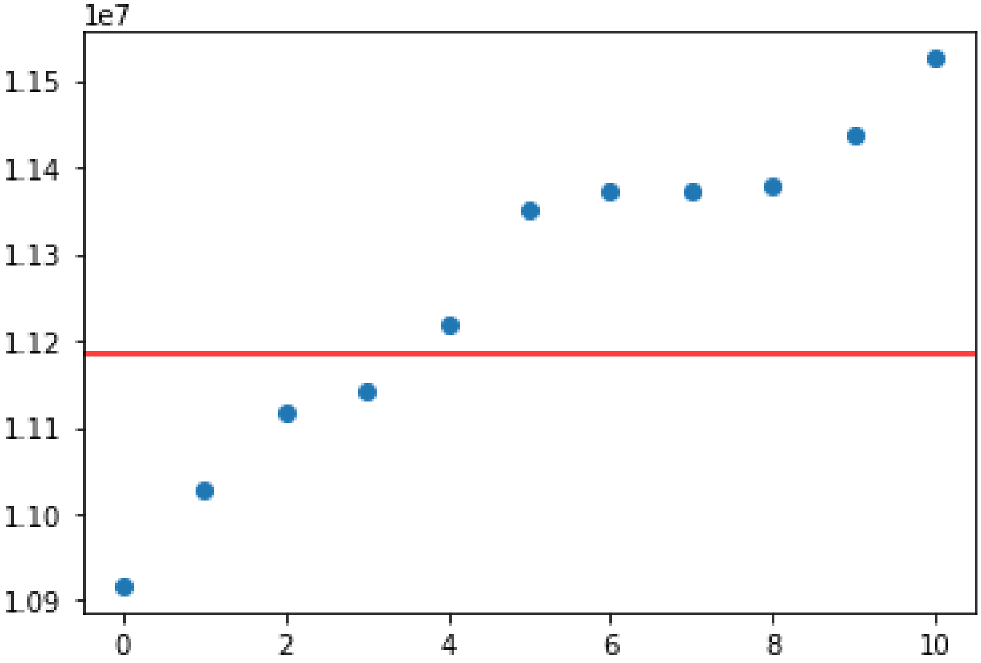
m	2^{10}
MSE	218,241,435,299
mean	11,203,634



Median trials

And here is our results if we divide the trials into eleven groups of 9 and take their medians.

m	2^{10}
MSE	38,165,304,417
mean	11,259,604

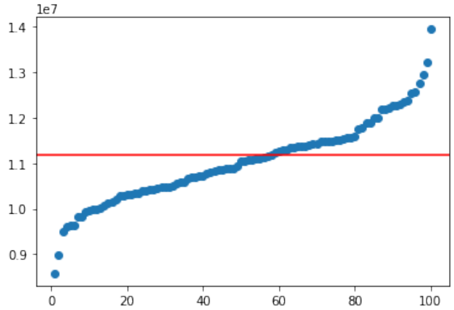


This gives us a much smaller mean squared error, and - as you can see from the labels on the y-axis in the plots - our results are closer to the correct sum of squares.

Second run: $m = 2^8$

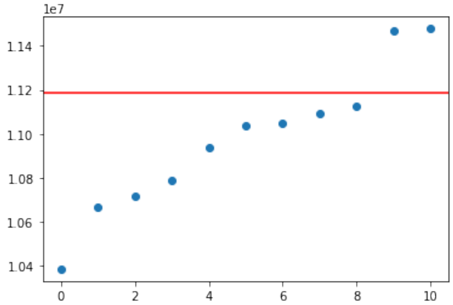
100 Trials

m	2^8
MSE	847,504,786,360
mean	11,034,735



Median Trials

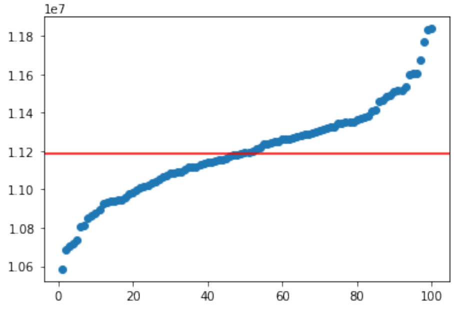
m	2^8
MSE	142,537,632,940
mean	10,976,042



Third run: $m = 2^{12}$

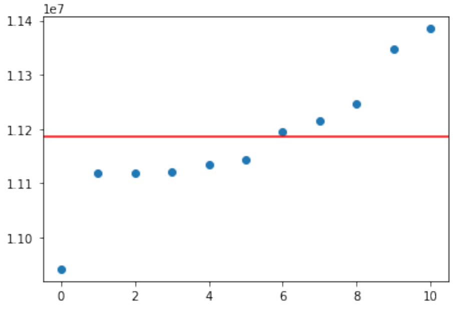
100 Trials

m	2^{12}
MSE	60,302,250,980
mean	11,196,762



Median Trials

m	2^{12}
MSE	13,442,973,339
mean	11,178,633



Our running time is as follows:

m	2^8	2^{10}	2^{12}
HashTable w. multiplyShift	37643ms	8923ms	8555ms
HashTable w. multiplyModPrime	17250ms	17509ms	21270ms
CountSketch method	19246ms	19303ms	19996ms

The results of our running time tests are rather counter-intuitive because it takes longer to come up with an estimate of the second moment rather than deterministically calculating it. However, we do see the running time of hashtable with chaining using multiplyModPrime rise above that of the CountSketch method when m becomes large enough.

A larger m also leads to smaller MSE and a mean closer to S concluded from our above trials. It improves our results even further when taken medians from groups from our trials. The best result we got was when $m = 2^{12}$ and we sampled the medians. In that case the MSE was way lower than in any other case and the estimated second moment was less than 7000 or 0.06% off.

Naturally some of our result are heavily affected by the way we implemented them and we are very surprised by the running time discoveries we did. If we were to do further experiments, we would probably try increasing the m in our CountSketch method to see if the runtime in our implementation would ever beat that of hashtable with chaining using multiply-shift hashing.