Overall algorithm time and space computational complexity comparison:

	Time complexity (worst case)	Space computational complexity	NOTE (n is the number of element/login)
Linear algorithm	O(n)	O(1)	
Binary algorithm	O(log n)	O(1)-iterative O(logn)-recursive	
Hashing	O(n)	O(n)	
Bloom filter	O(K)	$1.44\log\left(\frac{1}{fpp}\right)$	n: estimated number of element. m: bit array of m bits *k: number of hash functions. *fpp = false positive probability
	O(1)	$1.05\left[\frac{\log\left(\frac{1}{fpp}\right)+2}{load}\right]$	*load : the percentage of the filter it's currently using.

Bloom filter :

*fpp: is the fp/fp + tn, where fp is the number of falsepositives and tn is the number of true negatives.

*m: size of bit array. (k bits will be set in the bloom filter of size m)

Choosing a Bloom filter parameter m and k:

Given n and p, the better number of bits can be defined as follow:

$$m = \mathrm{ceil}\left(\frac{(\mathrm{n} * \log(\mathrm{p}))}{\log\left(\frac{1}{power(2, \log(2))}\right)}\right)$$

Given m and n, the better nubmer of hash function can be defined as follow:

$$k = round\left(\left(\frac{m}{n}\right) * \log(2)\right)$$

Experimental algorithm comparison:

The value in the cell is the running time(sec) of each algorithm, and it is implemented with the algorithm's section, defined in "Assignment1_Script.m".

This running time here including creating the object, insert, hashing….

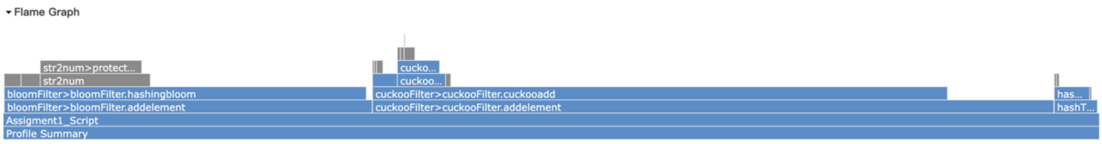
Element size	10	100	1000

Linear search	0.00256	0.002157	0.001694
Binary search	0.00082	0.00636	0.00673
Hashing	0.0063	0.02	0.201
Bloom filter	0.0194	0.185	1.49
Cuckoo filter	0.0319 seconds	0.0365	0.048054

This running time here only include searching for one item.

Element size	10	100	1000	10000
Linear search	0.006452	0.005857	0.005512	0.014198
Binary search	0.008891	0.008371	0.007554	0.001659
Hashing	0.003531	0.030540	0.007325	0.003826
Bloom filter	0.007192	0.011982	0.009521	0.001528
Cuckoo filter	0.004153	0.005069	0.007185	0.001528

The searching time for each algorithm didn't seems to change a lot as the element size goes up. However, the code did take longer to execute when we increase the element size to 10,000. As a result, I took a look at the "run and time", which is a more professional way to calculate the executing time…haha.



Here is the visualization from the "run and time" report.
And you can notice that cuckoo filter took the most time to execute compared to bloom filter and the hashing is the third is the "hashing" algorithm.
From the "Profiler.pdf" report, we can see most of the time is most of the time is spend on "adding element" in both bloom filter and cuckoo filter algorithm rather than "searching item".

Therfore, I just want to validate the profiler result by the most fundamental way of recording execution runtime, which is "tic toc" , so I experiment the executing time of the action of "adding element".
The running time here only include the adding element

Element size	100	10000
Linear search	0.001265	0.004311
Binary search	0.003496	0.002374
Hashing	0.033039	1.709875
Bloom filter	0.277816	13.559416
Cuckoo filter	0.024673	27.485878

We can see the both bloom filter and cuckoo filter really take lots of time to add element when we increase the size of elements.
Although the search time between all the algorithm almost the same (pretty well), cuckoo filter perform mostly better than other whatever the element size is in term of “searching”.
Both bloom filter and cuckoo filter take so many time to add element in. Theoretically, cuckoo filter should be better than bloom filter, but I think it this result is mostly because of my not so efficient implementation. (haha…)

On top of that, I think my implementation of the assignment justifies the professor saying that “the simpler the algorithm is the better” and having a simple algorithm isn’t always the bad thing. For example, linear and binary search acutally aren’t perform badly in my experiment. As a result, there is no certain best or worst algorithm, It all depend on what problem you trying to solve.

Reference :

1. https://www.upgrad.com/blog/linear-search-vs-binary-search/
2. https://iq.opengenus.org/time-complexity-of-binary-search/
3. https://iq.opengenus.org/time-complexity-of-hash-table/
4. https://arxiv.org/pdf/1903.12525.pdf
5. https://brilliant.org/wiki/bloom-filter/ XXXXbloom filter
6. https://octo.vmware.com/bloom-filter/ XXXX bloom filter
7. https://bdupras.github.io/filter-tutorial/
8. https://brilliant.org/wiki/cuckoo-filter/
9. https://www.split.io/glossary/false-positive-rate/
10. https://www.pdl.cmu.edu/PDL-FTP/FS/cuckoo-conext2014.pdf
11.