Please note, I made the choice to construct the 3 pairs of sets of strings, K and K’, every time the main executable is run so that I did not have to store the sets of K and K’ in an additional file. However, to ensure these sets were the same for each run of the program, I utilized rand() to choose each letter of the constructed strings and used srand() to set the seed to a specified value before creating each K and corresponding K’. The first set of K and K’, which I named K1 and K1’, are constructed such that both K1 and K1’ each were made up of 10000 strings that were 31 characters in length. For this set, I constructed K1’ such that the first 5000 elements were contained within K1, and the last 5000 elements were newly constructed strings that were ensured not to be contained within K1. The second set of K and K’, which I named K2 and K2’, are constructed such that both K2 and K2’ each were made up of 20000 strings that were 31 characters in length. For this set, I constructed K2’ such that the last 5000 elements were contained within K2, and the first 15000 elements were newly constructed strings that were ensured not to be contained within K2. The third set of K and K’, which I named K3 and K3’, are constructed such that both K3 and K3’ each were made up of 30000 strings that were 31 characters in length. For this set, I constructed K3’ such that the first 12000 elements were contained within K3, and the last 18000 elements were newly constructed strings that were ensured not to be contained within K3. Therefore, the sets have various sizes and various mixtures of “positive” and “negative” keys. These sets of K and K’ are used throughout the following tasks. They are initially constructed during task 1, but saved for use in the other tasks.

**Task 1**

1. For task 1, I implemented bloom filters using the bloom filter library by Arash Patow found at <https://github.com/ArashPartow/bloom>. I constructed a total of 9 bloom filters, as there were 3 varying sized sets of strings (10000, 20000, and 30000 strings) and 3 different false positive rates to be tested. A bloom filter was created for each combination of these variables where the projected\_element\_count is set to the corresponding size of the set of strings used and false\_positive\_probability is set to the corresponding false positive rate. In the executable output, you will see that the bloom filters are numbered using double digits. The first digit indicates the chosen false positive rate (1 = (1/(2^7)), 2 = (1/(2^8)), and 3 = (1/(2^10))) and the second digit indicates the chosen size of the set of strings (1 = 10000, 2 = 20000, and 3 = 30000). After each bloom filter’s parameters were set as described above, each element of the corresponding K that matches the size specified to the bloom filter is inserted into the filter. Then, my process of querying involves looping through each element of the corresponding K’ that matches the K used and checking if the bloom filter claims the value from K’ is contained within it. If the value from K’ was from the original K (a “positive key”) and is not stated to be contained in the bloom filter, I increment a counter for false negatives as a sanity check. No bloom filter produced false negatives in my tests. If the value from K’ was not from the original K (a “negative key”) and is stated to be contained in the bloom filter, I increment a counter for false positives. I then calculate the false positive rate by dividing the false positive counter by the number of elements of the corresponding K’ that were not in the original K. I then return the false positive rate, time for query, and bloom filter size (calculated using the .size() function in the Arash Patow repo) to the terminal.
2. The programming of this task was fortunately not incredibly difficult due to our ability to use previous implementations of bloom filters. As such, most of the programming was rather repetitive as I ensured that every combination of size of sets and false positive rates was established.

**SAMPLE OUTPUT FROM EXECUTABLE**

--------------Task1: Bloom Filters--------------

Bloom filter 11. FP Rate entered (1/(2^7)). Sets size 10000. String size 31.

K1' is 50% keys present in K1 and 50% keys not in K1.

BF11 contains 0 false negatives

BF11 contains 43 false positives with rate 0.0086

Time taken for querying K1': 2014684 nanoseconds

Size of bloom filter: 100992 bits

Bloom filter 12. FP Rate entered (1/(2^7)). Sets size 20000. String size 31.

K2' is 25% keys present in K2 and 75% keys not in K2.

BF12 contains 0 false negatives

BF12 contains 150 false positives with rate 0.01

Time taken for querying K2': 2111629 nanoseconds

Size of bloom filter: 201984 bits

Bloom filter 13. FP Rate entered (1/(2^7)). Sets size 30000. String size 31.

K3' is 40% keys present in K3 and 60% keys not in K3.

BF13 contains 0 false negatives

BF13 contains 149 false positives with rate 0.00827778

Time taken for querying K3': 5891537 nanoseconds

Size of bloom filter: 302968 bits

Bloom filter 21. FP Rate entered (1/(2^8)). Sets size 10000. String size 31.

K1' is 50% keys present in K1 and 50% keys not in K1.

BF21 contains 0 false negatives

BF21 contains 26 false positives with rate 0.0052

Time taken for querying K1': 2248399 nanoseconds

Size of bloom filter: 115416 bits

Bloom filter 22. FP Rate entered (1/(2^8)). Sets size 20000. String size 31.

K2' is 25% keys present in K2 and 75% keys not in K2.

BF22 contains 0 false negatives

BF22 contains 62 false positives with rate 0.00413333

Time taken for querying K2': 3438705 nanoseconds

Size of bloom filter: 230832 bits

Bloom filter 23. FP Rate entered (1/(2^8)). Sets size 30000. String size 31.

K3' is 40% keys present in K3 and 60% keys not in K3.

BF23 contains 0 false negatives

BF23 contains 70 false positives with rate 0.00388889

Time taken for querying K3': 8715620 nanoseconds

Size of bloom filter: 346248 bits

Bloom filter 31. FP Rate entered (1/(2^10)). Sets size 10000. String size 31.

K1' is 50% keys present in K1 and 50% keys not in K1.

BF31 contains 0 false negatives

BF31 contains 3 false positives with rate 0.0006

Time taken for querying K1': 9607370 nanoseconds

Size of bloom filter: 144272 bits

Bloom filter 32. FP Rate entered (1/(2^10)). Sets size 20000. String size 31.

K2' is 25% keys present in K2 and 75% keys not in K2.

BF32 contains 0 false negatives

BF32 contains 14 false positives with rate 0.000933333

Time taken for querying K2': 4670956 nanoseconds

Size of bloom filter: 288544 bits

Bloom filter 33. FP Rate entered (1/(2^10)). Sets size 30000. String size 31.

K3' is 40% keys present in K3 and 60% keys not in K3.

BF33 contains 0 false negatives

BF33 contains 26 false positives with rate 0.00144444

Time taken for querying K3': 6993642 nanoseconds

Size of bloom filter: 432808 bits

--------------Task2: MPHF--------------

MPHF 1. Sets size 10000. String size 31. Gamma 2.

K1' is the same K1' as above, it is 50% keys present in K1 and 50% keys not in K1.

[Building BooPHF] 100 % elapsed: 0 min 0 sec remaining: 0 min 0 sec

Bitarray 43136 bits (100.00 %) (array + ranks )

final hash 0 bits (0.00 %) (nb in final hash 0)

boophf bits/elem : 4.313600

MPHF1 contains 0 false negatives

MPHF1 contains 4535 false positives with rate 0.907

Time taken for querying K1': 3715544 nanoseconds

MPHF 2. Sets size 20000. String size 31. Gamma 2.

K2' is the same K2' as above, it is 25% keys present in K2 and 75% keys not in K2.

[Building BooPHF] 100 % elapsed: 0 min 0 sec remaining: 0 min 0 sec

Bitarray 79936 bits (100.00 %) (array + ranks )

final hash 0 bits (0.00 %) (nb in final hash 0)

boophf bits/elem : 3.996800

MPHF2 contains 0 false negatives

MPHF2 contains 13638 false positives with rate 0.9092

Time taken for querying K2': 8092845 nanoseconds

MPHF 3. Sets size 30000. String size 31. Gamma 2.

K3' is the same K3' as above, it is 40% keys present in K3 and 60% keys not in K3.

[Building BooPHF] 100 % elapsed: 0 min 0 sec remaining: 0 min 0 sec

Bitarray 117248 bits (100.00 %) (array + ranks )

final hash 0 bits (0.00 %) (nb in final hash 0)

boophf bits/elem : 3.908267

MPHF3 contains 0 false negatives

MPHF3 contains 17442 false positives with rate 0.969

Time taken for querying K3': 12020059 nanoseconds

--------------Task3: Fingerprint Array--------------

MPHF + FP 11. Fingerprint size 7. Sets size 10000. String size 31. Gamma 2.

K1' is the same K1' as above, it is 50% keys present in K1 and 50% keys not in K1.

FP11 contains 0 false negatives

FP11 contains 1287 false positives with rate 0.2574

Time taken for querying K1': 4803750 nanoseconds

Size of fingerprint array alone: 70088 bits. Look at MPHF section for its size.

MPHF + FP 12. Fingerprint size 7. Sets size 20000. String size 31. Gamma 2.

K2' is the same K2' as above, it is 25% keys present in K2 and 75% keys not in K2.

FP12 contains 0 false negatives

FP12 contains 5351 false positives with rate 0.356733

Time taken for querying K2': 10561610 nanoseconds

Size of fingerprint array alone: 140104 bits. Look at MPHF section for its size.

MPHF + FP 13. Fingerprint size 7. Sets size 30000. String size 31. Gamma 2.

K3' is the same K3' as above, it is 40% keys present in K3 and 60% keys not in K3.

FP13 contains 0 false negatives

FP13 contains 7442 false positives with rate 0.413444

Time taken for querying K3': 14848462 nanoseconds

Size of fingerprint array alone: 210120 bits. Look at MPHF section for its size.

MPHF + FP 21. Fingerprint size 8. Sets size 10000. String size 31. Gamma 2.

K1' is the same K1' as above, it is 50% keys present in K1 and 50% keys not in K1.

FP21 contains 0 false negatives

FP21 contains 615 false positives with rate 0.123

Time taken for querying K1': 4949584 nanoseconds

Size of fingerprint array alone: 80072 bits. Look at MPHF section for its size.

MPHF + FP 22. Fingerprint size 8. Sets size 20000. String size 31. Gamma 2.

K2' is the same K2' as above, it is 25% keys present in K2 and 75% keys not in K2.

FP22 contains 0 false negatives

FP22 contains 2637 false positives with rate 0.1758

Time taken for querying K2': 10201768 nanoseconds

Size of fingerprint array alone: 160072 bits. Look at MPHF section for its size.

MPHF + FP 23. Fingerprint size 8. Sets size 30000. String size 31. Gamma 2.

K3' is the same K3' as above, it is 40% keys present in K3 and 60% keys not in K3.

FP23 contains 0 false negatives

FP23 contains 3722 false positives with rate 0.206778

Time taken for querying K3': 15184371 nanoseconds

Size of fingerprint array alone: 240072 bits. Look at MPHF section for its size.

MPHF + FP 31. Fingerprint size 10. Sets size 10000. String size 31. Gamma 2.

K1' is the same K1' as above, it is 50% keys present in K1 and 50% keys not in K1.

FP31 contains 0 false negatives

FP31 contains 155 false positives with rate 0.031

Time taken for querying K1': 4607923 nanoseconds

Size of fingerprint array alone: 100104 bits. Look at MPHF section for its size.

MPHF + FP 32. Fingerprint size 10. Sets size 20000. String size 31. Gamma 2.

K2' is the same K2' as above, it is 25% keys present in K2 and 75% keys not in K2.

FP32 contains 0 false negatives

FP32 contains 668 false positives with rate 0.0445333

Time taken for querying K2': 10349906 nanoseconds

Size of fingerprint array alone: 200072 bits. Look at MPHF section for its size.

MPHF + FP 33. Fingerprint size 10. Sets size 30000. String size 31. Gamma 2.

K3' is the same K3' as above, it is 40% keys present in K3 and 60% keys not in K3.

FP33 contains 0 false negatives

FP33 contains 914 false positives with rate 0.0507778

Time taken for querying K3': 33887502 nanoseconds

Size of fingerprint array alone: 300104 bits. Look at MPHF section for its size.