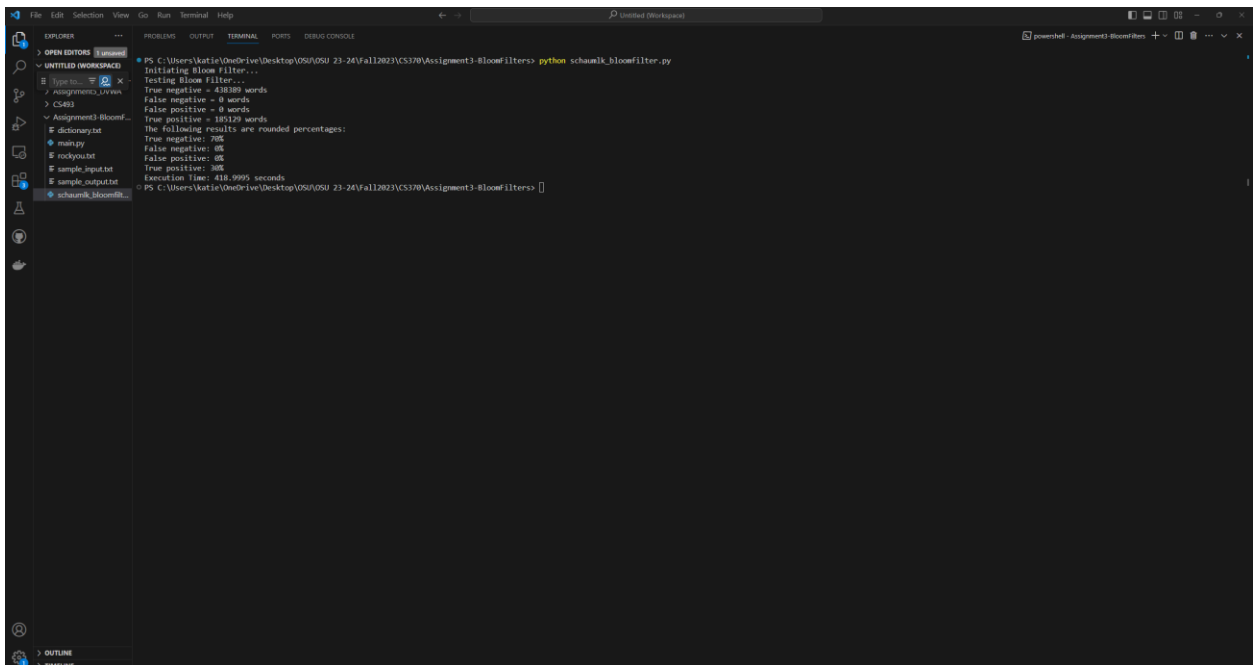


## Bloom Filter Write-Up

### 1. How long does it take to run your code?

To answer this question, I implemented a function that calculated the total running time of my code. Based on this screen shot from running my code:



```
PS C:\Users\katie\OneDrive\Desktop\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters> python schaumk_bloomfilter.py
Initiating Bloom Filter...
Testing Bloom Filter...
True negative = 438380 words
False negative = 0 words
False positive = 0 words
True positive = 185129 words
The following results are rounded percentages:
True negative: 70%
False negative: 0%
False positive: 0%
True positive: 30%
Execution Time: 418.9995 seconds
PS C:\Users\katie\OneDrive\Desktop\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters>
```

It took 418.9995 seconds to run.

### 2. How many bits are in your bit array and why was this appropriate for your implementation?

I tested a few different sizes for my bit array. I found that the higher the bit number, the longer the program takes to run because it uses more memory. I found that when I increased the number, the efficiency of the Bloom filter increased. When I used 134,198,017 bits, then the value of true negative was 70% and 30% for true positive. If I used a smaller bit array, then the program runs much faster, but with less accuracy, so I stuck with 134,198,017 bits.

### 3. How many hashing algorithms did you use and why was this appropriate for your implementation?

For this program, I used 10 hash function from the hashlib Library:

```
"MD5", "SHA-1", "SHA-224", "SHA-256", "SHA-384",  
"SHA-512", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-512"
```

This was appropriate for this implementation, because these hash functions contribute to the effectiveness of this Bloom filter by improving accuracy and distribution as well as reducing collisions. Since each hashing algorithm provides a different approach to hashing data, the more hashing algorithms used will make the Bloom filter more effective. I could use fewer hashing algorithms, but the program is less efficient when using fewer hashes.

**4. Describe the meanings of true positive, true negative, false positive, and false negative in relation to a Bloom Filter.**

In relation to Bloom Filters, a true positive indicates that an item does exist in the set. So, if the Bloom Filter determines that an element is present, and it is in the set, then it's considered a true positive.

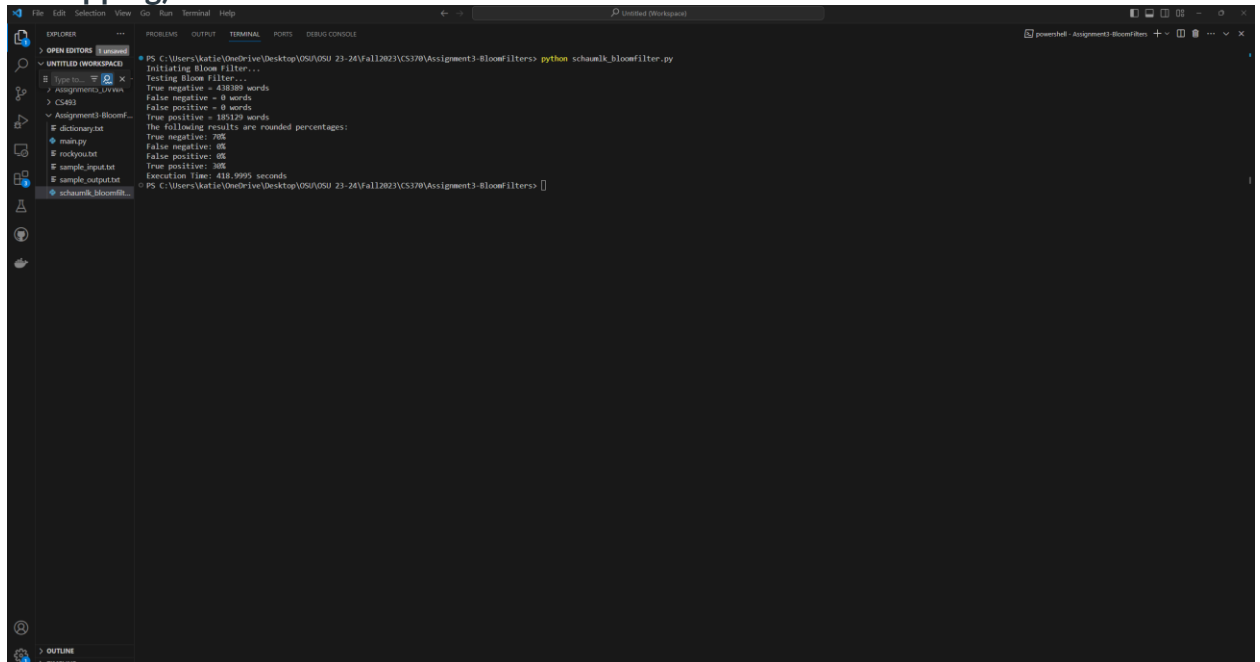
A true negative indicates that an item is not present in the set. If the Bloom Filter correctly reports that an element is not in the set, and it's not actually in the set, then it is a true negative.

A false positive is when the Bloom Filter incorrectly indicates that an item is in the set, but when it's not actually there. If the Bloom Filter reports that it is present, but it's not really in the set, then it's a false positive.

A False negative is when a Bloom Filter falsely indicates that an item is not present in the set. If the Bloom Filter reports that an element is not present, but the element is actually in the set, then it's considered a false negative.

**5. Provide statistics on true positive, true negative, false positive, and false negative for the dictionary.txt based on the rockyou.txt (i.e. preload rockyou.txt, test entries in dictionary.txt).**

- a. Include screenshot of output (show full screen / maximized terminal window - no clipping).



```
PS C:\Users\Katie\OneDrive\Desktop\OSU\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters> python schaumk_bloomfilter.py
Initiating Bloom Filter...
Testing Bloom Filter...
True negative = 438389 words
False negative = 0 words
False positive = 0 words
True positive = 185129 words
The following results are rounded percentages:
True negative: 70%
False negative: 0%
False positive: 0%
True positive: 30%
Execution Time: 418.9995 seconds
PS C:\Users\Katie\OneDrive\Desktop\OSU\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters>
```

6. Based on the design parameters of the Bloom filter, what was the projected false positive rate (see calculator link above) and how does that compare to your actual results?

When running my Bloom Filter, the calculator seems to be accurate for my program. I've calculated the number of items in the filter at 623,518 and the number of bits are at 134190817. When using 10 hash functions, this gives us a probability of 0 (1 in 26,868,589,280,690), which is accurate. If I use 3 hash functions, then the program runs much quicker, but less efficiently. I end up with 1% false positives when I use 3 hash functions, so I think it's appropriate to use 10 and get 0 false positives.

3 hash functions:

```
PS C:\Users\katie\OneDrive\Desktop\OSU\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters> python schaumk_bloomfilter.py
Initiating Bloom Filter...
Size of bit array: 134190817
Testing Bloom Filter...
True negative = 429175 words
False negative = 0 words
False positive = 9214 words
True positive = 185129 words
The following results are rounded percentages:
True negative: 69%
False negative: 0%
False positive: 1%
True positive: 30%
Number of hash functions used: 3
Execution Time: 107.3697 seconds
```

10 hash functions:

```
PS C:\Users\katie\OneDrive\Desktop\OSU\OSU 23-24\Fall2023\CS370\Assignment3-BloomFilters> python schaumlk_bloomfilter.py
Initiating Bloom Filter...
Testing Bloom Filter...
True negative = 438389 words
False negative = 0 words
False positive = 0 words
True positive = 185129 words
The following results are rounded percentages:
True negative: 70%
False negative: 0%
False positive: 0%
True positive: 30%
Execution Time: 418.9995 seconds
```