Streaming Data Systems Assignment 2

Chinmay Parekh

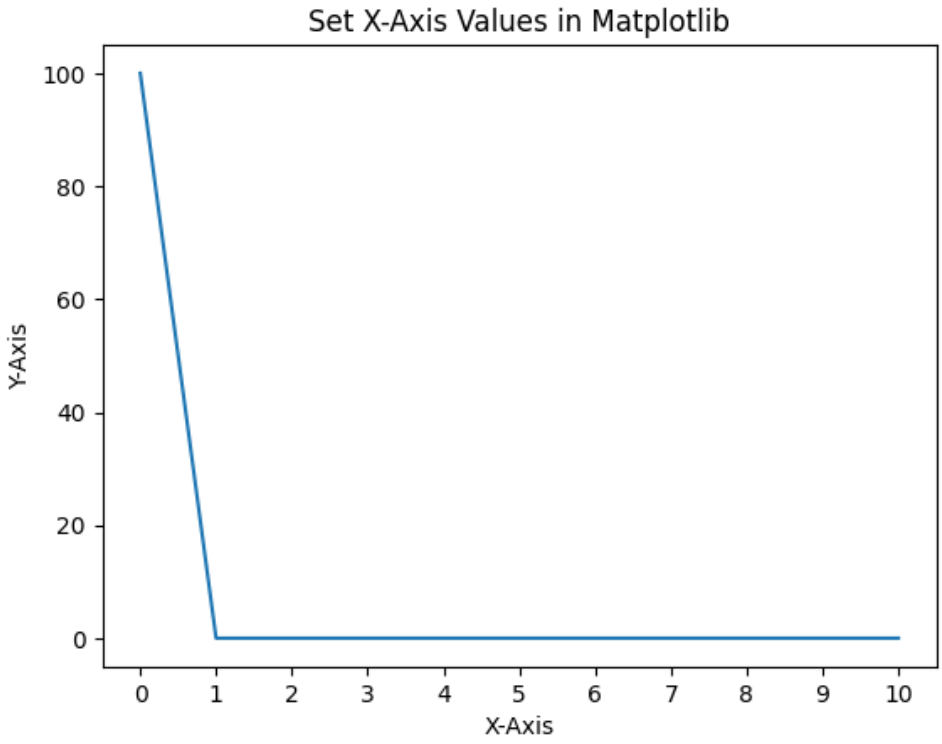
IMT2020069

Code

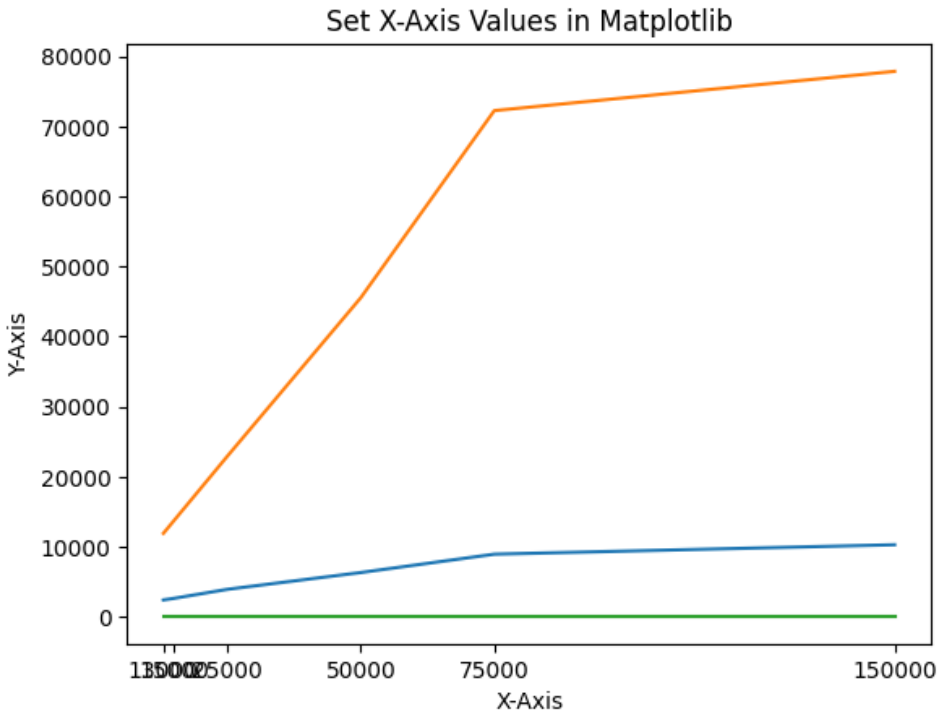
Baseline:

Driver function calls the producer to produce all the data and then the consumer function is called to process the data.

Latency plot:



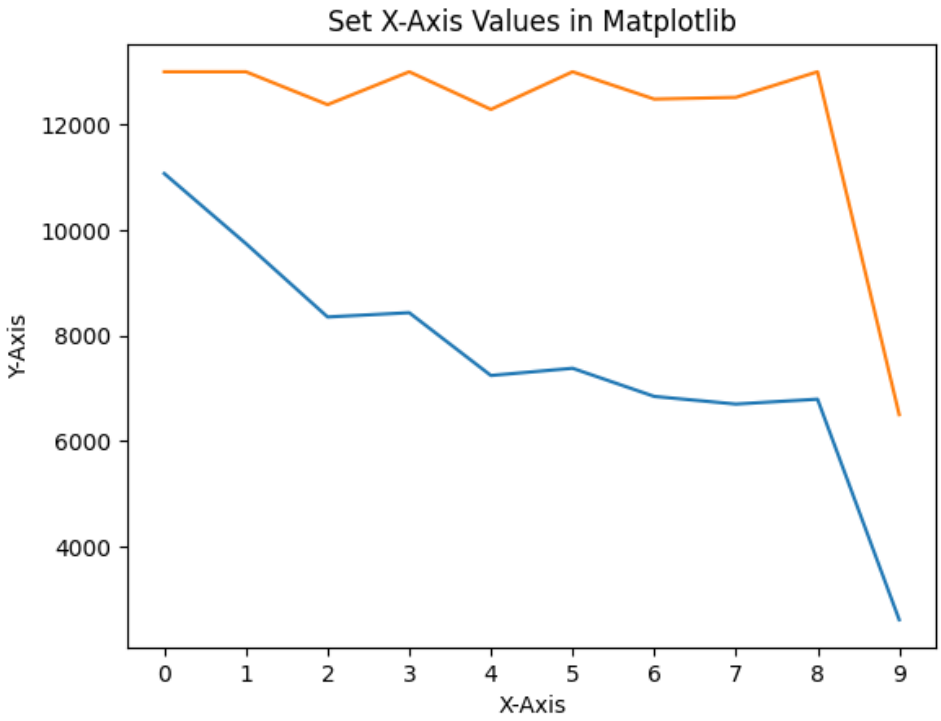
Summary plot:



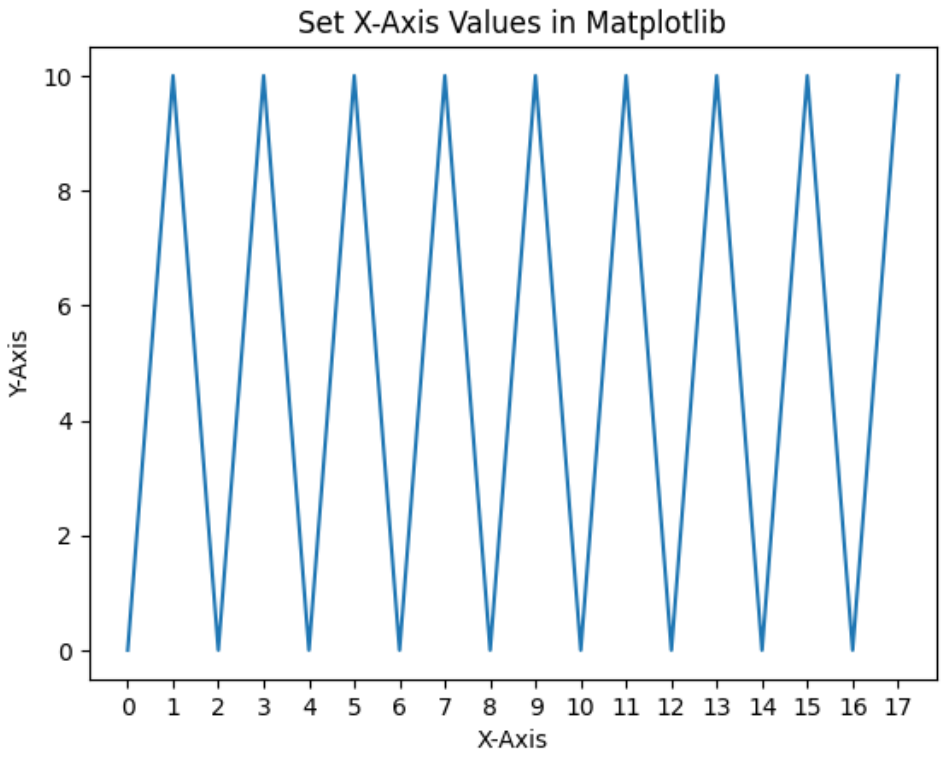
Approach 2 – Per window processing

Driver calls the process and consumer and the consumer waits for every 10 seconds and then processes the data.

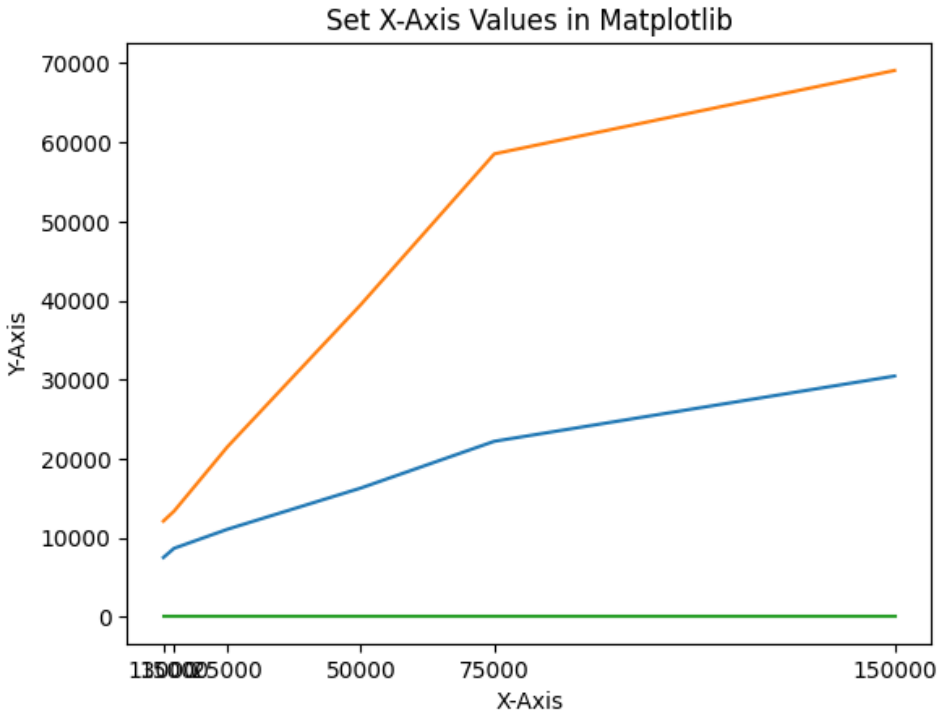
Throughput = 13000



Latency plot



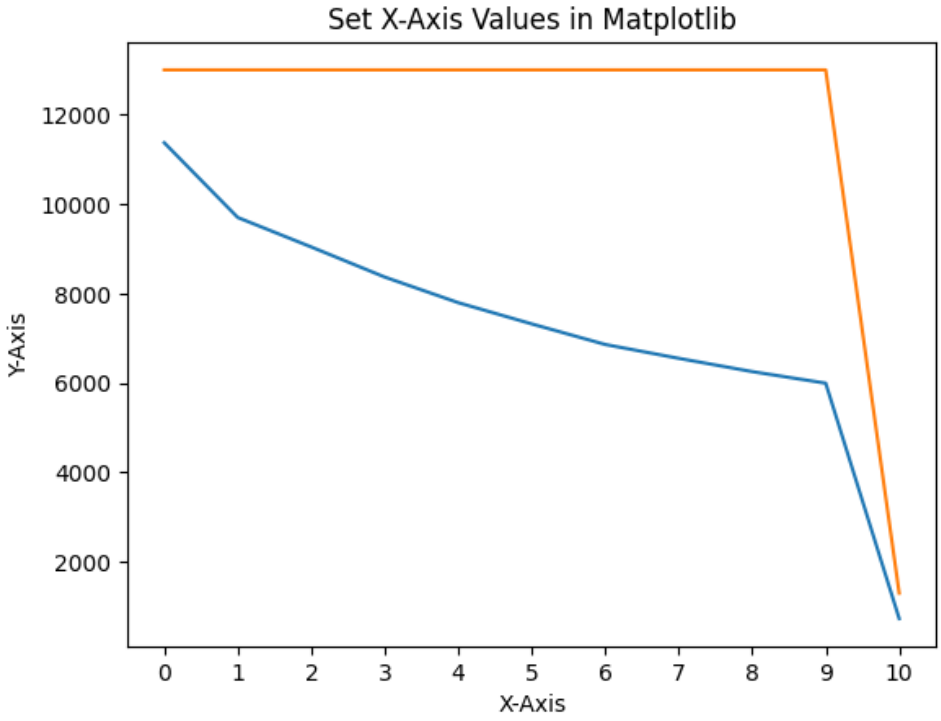
Summary plot:



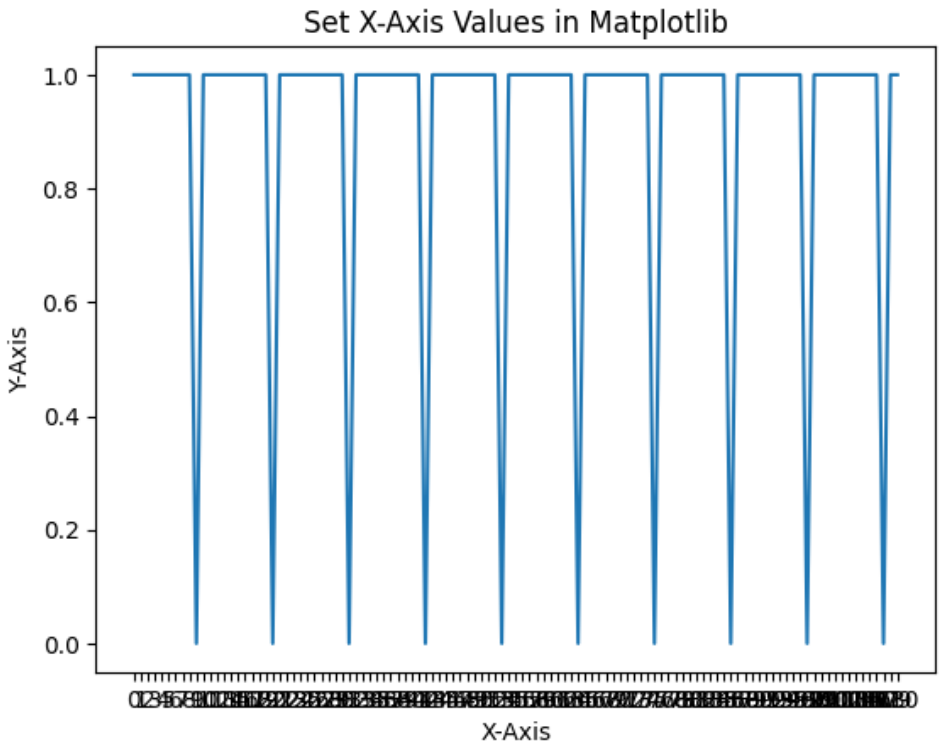
Approach 3: Slicing the window

The driver calls the producer and consumer and the consumer processes the data every second and increments the results and outputs the result at the end of every 10 seconds.

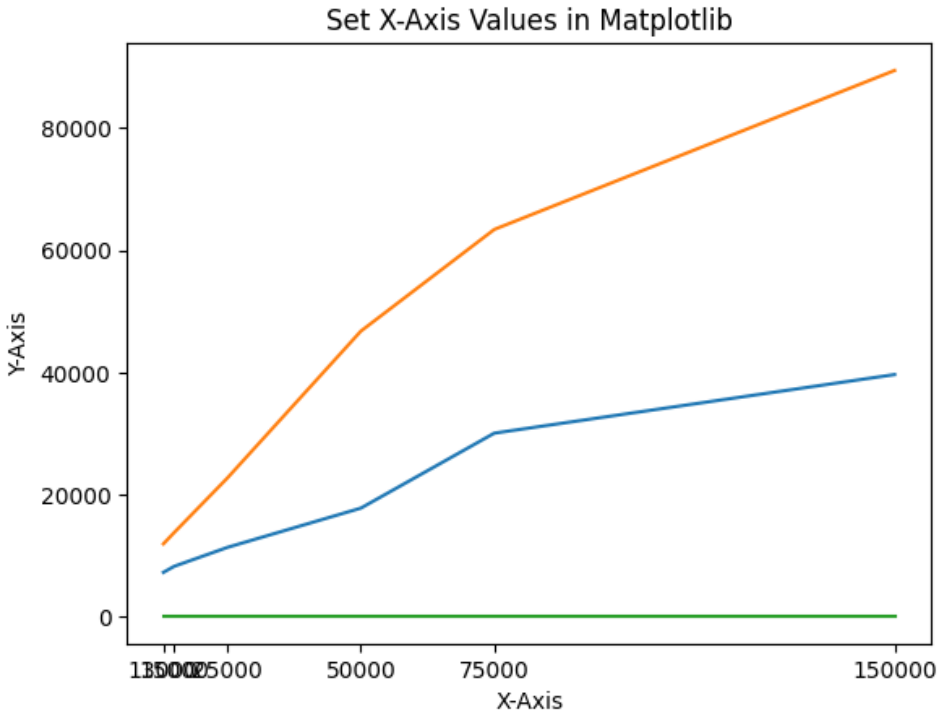
Throughput = 13000



Latency plot:

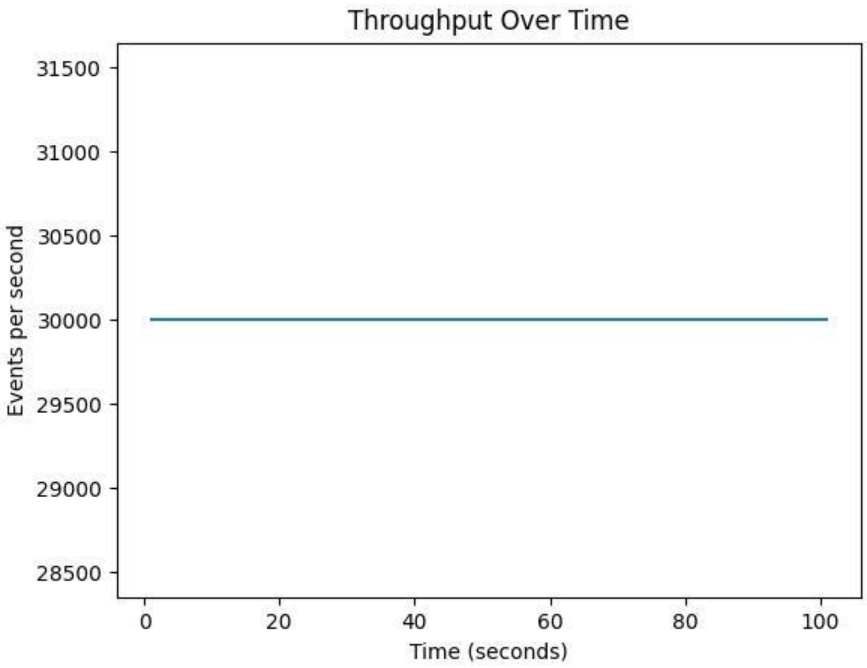


Summary plot:

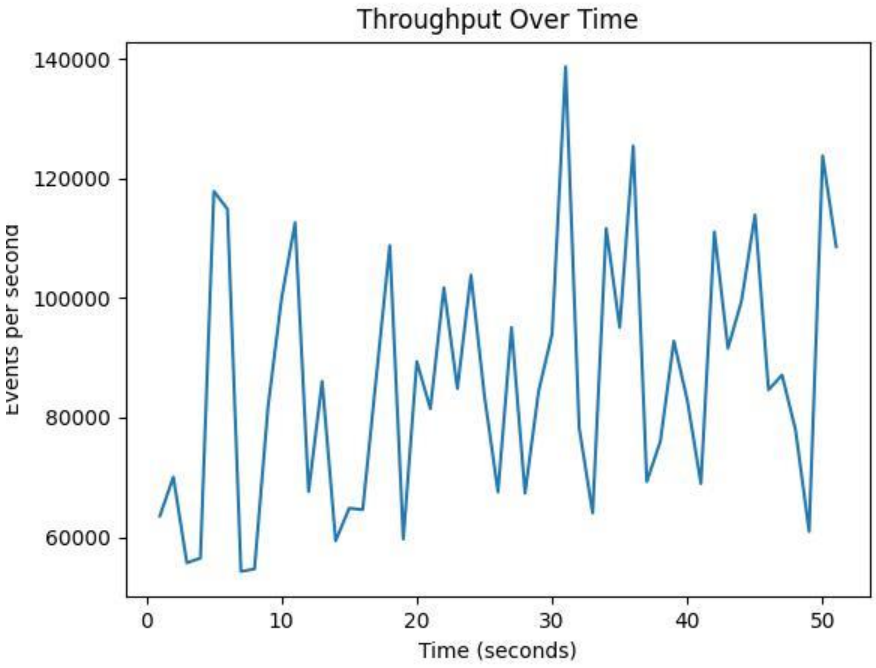


To increase the producer throughput, I ran the producer in two different threads.

Throughput = 15000

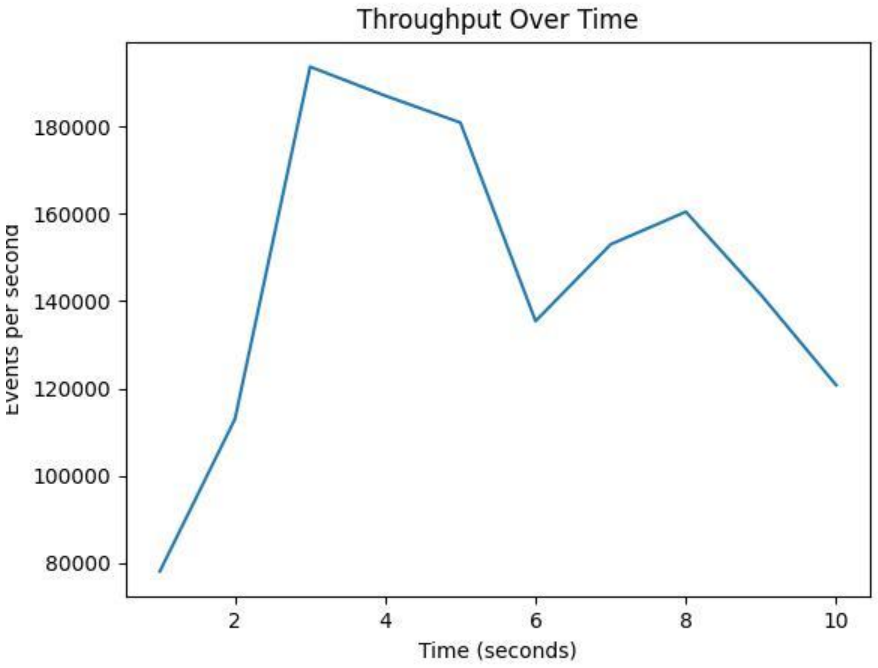


Unbounded



Approach 2:

Instead of writing row by row, I appended the data to one list and then wrote the data into the file all at once. This increased the throughput considerably.



Addition points:

The data being generated in a file might span across two seconds so the program waits for the nearest whole number and then calls the producer and consumer functions.

File locking is done while writing into file so the consumer cannot access the file that is being written

The throughput is calculated using the “Event Time”: The time at which the event was generated because in an ideal scenario, the moment the data was generated, it would have been processed instantly.

In case, the consumer tries to access a file that does not exist or is still being generated, the consumer process is slowed down by making it go to sleep for one second. After one second, it tries to read the file again. After 5 retries, the process terminates.

Made use of regex library: Time complexity = O(N)

Question 2:

Brute Force:

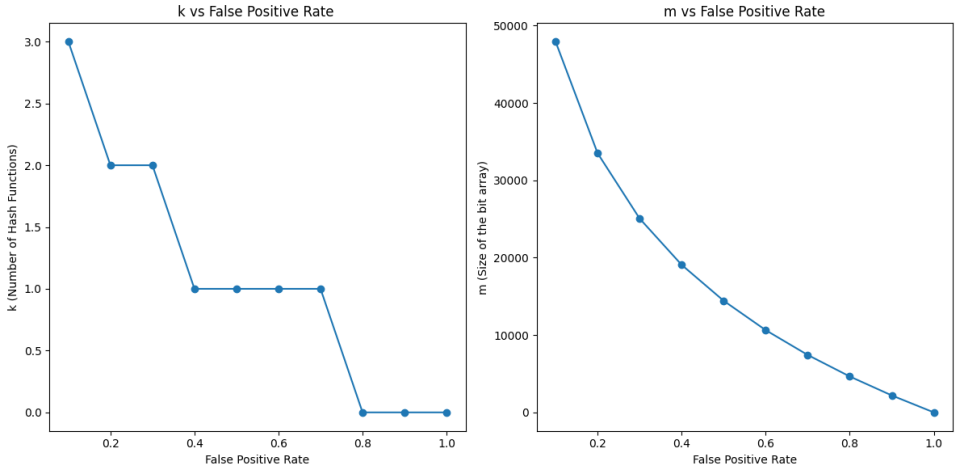
I wrote a function to check and write a character to the temp file if it's not already present

I ran this for data of length =20,000
Execution Time = 16.47seconds

Time complexity = O(n^2)

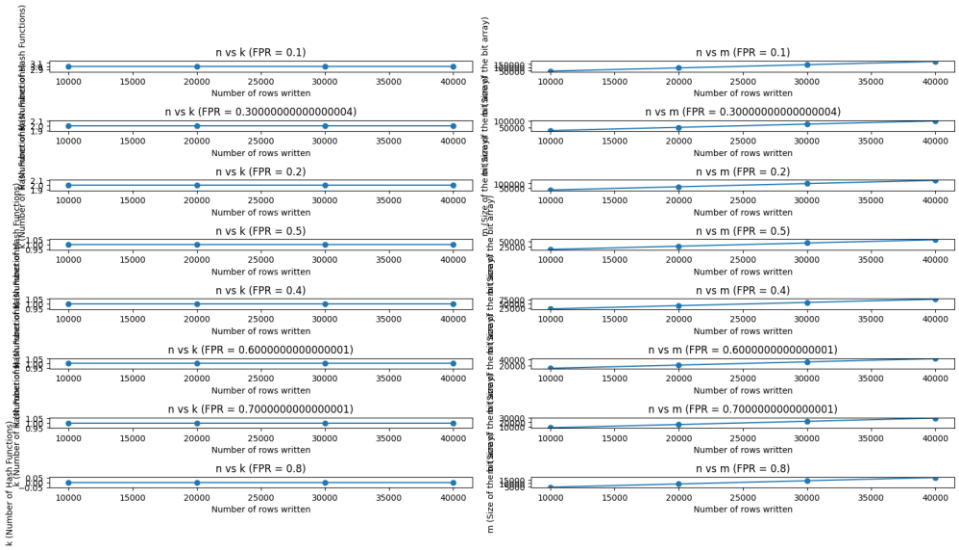
Bloom Filters:

The variation of k and m with FPR is as follows:



As **k** increases, FPR decreases. More hash functions distribute the bits more uniformly, reducing the likelihood of false positives.
As **n** increases, FPR increases. A larger expected number of elements means more bits are set to 1, increasing the chances of collisions and false positives.
As **m** increases, FPR decreases. A larger bit array provides more space for distributing hash values, reducing the likelihood of collisions and false positives.
The relationship between k, n, and m is interdependent. Adjusting one of these parameters may necessitate adjustments to the others to achieve a desired false positive rate.

After this, I ran the bloom filter for different FPR (0,0.8) and n(10000,40000) to get different values of k and m.



Question 3

Palindromic Subsequence:

Assumption: The data generated in one file belongs to the same "second"

I solved the following questions:

Question 1

Given sequence A[2], find the the occurrences of the pattern AA where the difference between the timestamps is less than equal to 2.

For this, I have written a function which takes (data,timestamp,pattern) as input.
To optimize the time complexity, the data passed here will contain characters which are present in the string only, all the other characters are discarded.
The timestamps are the "relative timestamps" corresponding to the data that is sent to this function.
Now for each timestamp, I find the number of occurrences of "A" and maintain a map, where the key is the timestamp, and the value is the number of occurrences of "A" in that timestamp.

After this, I used a sliding window to multiply the value corresponding the beginning of the window and the values present in the window.
This was done because A of timestamp 0 can be mapped with any A of timestamp<=2.

Question 2:

Given sequence A[2]B[2]C[2], find the occurrences of the pattern ABCABC where the difference between the timestamps of the first “ABC” and the next one is less than equal to 2.

This is an extension of the previous problem.
Here instead of mapping the occurrences of “A” to the timestamps, I mapped the occurrences of “ABC” to the timestamps.

Here, the function only gets characters present in the pattern wrt to the original data as input.
This means that the code tries to find the following pattern

A[2]*B[2]*C[2]*A[2]*B[2]*C[2]

Question 3:

Given sequence A[2]B[2]C[2], find the occurrences of the pattern ABCCBA where the difference between the timestamps of the first “ABC” and the next “CBA” is less than equal to 2.

This is very close to the given question in the assignment and this is an extension of the previous question.
Now, I maintain, two list: one for mapping the occurrences of “ABC” and once for mapping the occurrences of “CBA” for each timestamp.
While calculating the occurrences , I multiply the value of “ABC” of the start timestamp of the window with the values of “CBA” of the other timestamps present in the window.

This in turns finds the number of occurrences of the following palindromic subsequence:

A[2]*B[2]*C[2]*C[2]*B[2]*A[2]

Further improvements:

The current algorithm maintains a key for each second and then, uses the values to find the occurrences.
To improve the precision to milli-seconds, we can maintain a set of keys for each milli second but at the cost of a lot of space.