COSC 320 – 001
Analysis of Algorithms
2020 Winter Term 2

Project Topic Number: #4

String Matching Problem - Milestone 3

Group Lead:

Matthew Borle - 42615435

Group Members:

Barry Feng - 12981156

Guy Kaminsky - 65234775

Ross Morrison - 36963411

Abstract.

In this milestone, we implemented one of our three algorithms, KMP. We first began by looking over our pseudocode to refamiliarize how this algorithm works. Next, we went to use an implementation from GeeksForGeeks of the KMP algorithm. Before testing the algorithm, we wrote out a couple of datasets to test out. The first one was a simple dataset with ten text files, each containing a single word, to compare against a master text file that contains one word. The next one was ten text files with sentences to compare to a master document of sentences. Once the dataset was complete, we had to expand on our code for KMP. We wrote the main function that stores the words/sentences in an array. Also, we had an array that would hold the number of documents that are plagiarized. Moreover, we printed the number of documents which are considered plagiarized and how many sentences were plagiarized. Finally, we calculated the time it took for the algorithm to run several times, each time increasing the number of documents that will be plagiarizing.

Dataset.

The dataset is ten unique comparator documents of varying length of randomly generated words and sentences, and three subject documents with varying amounts of plagiarised content taken from the ten comparators. The subjects are [*unique.txt, sentences.txt, paragraph.txt*] where *unique* has no blatant plagiarism, *sentences* has plagiarised sentences from three of the comparator documents, and *paragraph* has an entire paragraph that includes many plagiarised sentences.

Comparators	**Subjects**
0.txt	unique.txt
1.txt	sentences.txt
...	paragraph.txt
8.txt	
9.txt	

Implementation.

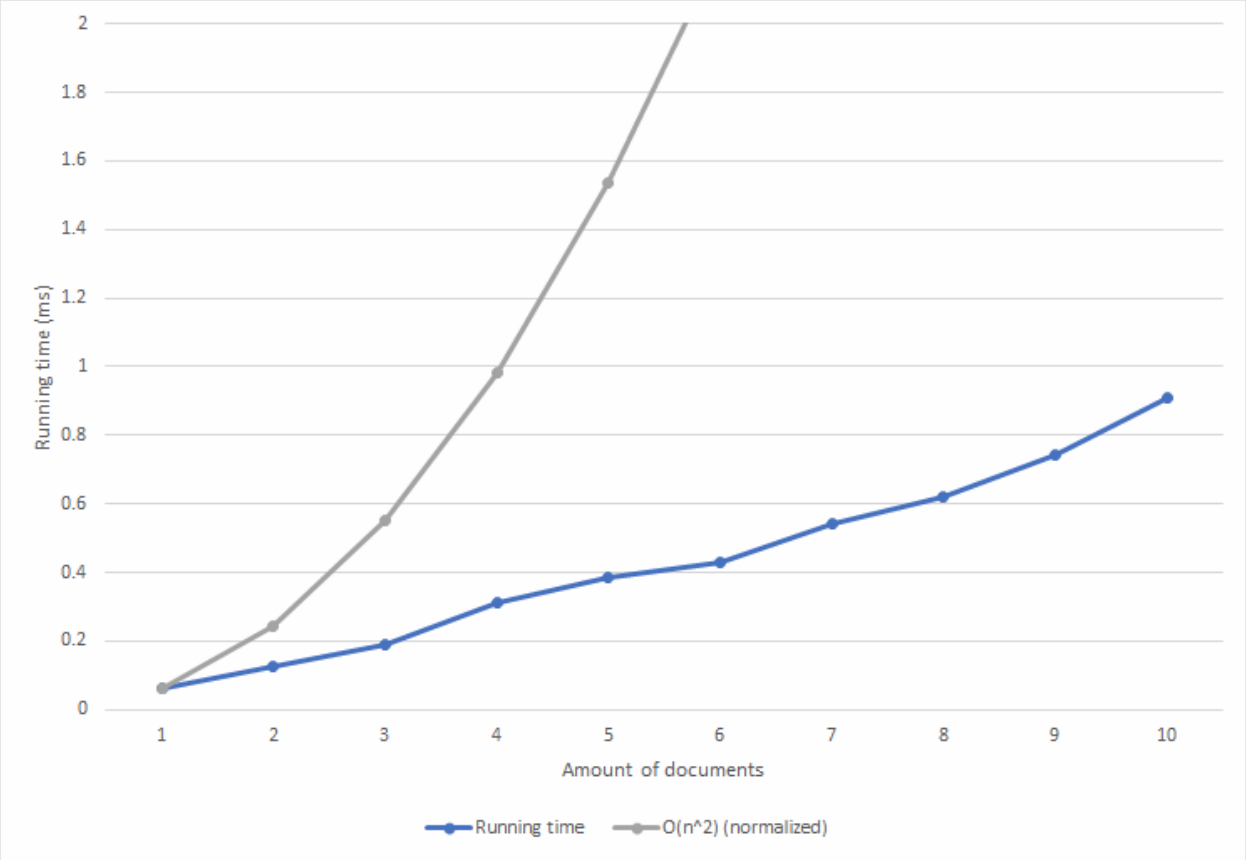
We began by implementing the KMP algorithm from GeeksForGeeks. Then the dataset was loaded into an array called dataset. Each dataset file is numbered from 0 to 9, so loading was as simple as iterating and loading the file from the iterator value. Then the input data was loaded into a string called sentenceData.

The sentenceData was split into sentences by checking each character, which was then added to the temporary sentence string, if it was a sentence ending character, the temporary string was added to the sentence array. Finally, each sentence was checked using KMP for each dataset file. If a dataset contained more than 1 sentence it was added to the foundDocuments list and the algorithm moved on to the next dataset document.

Reference: https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

Results.

Documents	1	2	3	4	5	6	7	8	9	10
Running time	0.061491	0.127834	0.188594	0.309973	0.384265	0.428581	0.542876	0.621387	0.744389	0.911736
O(n^2)	1	4	9	16	25	36	49	64	81	100
O(n^2) (normalized)	0.061491	0.245966	0.553423	0.983864	1.537288	2.213694	3.013084	3.935456	4.980812	6.14915



We implemented a timer in our algorithm, and ran 1 to 10 documents into our algorithm, and logged the time to find out what the running time of KMP is. Since the graph grew exponentially, the asymptotic notation would be $O(n^2)$. We normalized the first point to the same as our one document and then allowed it to grow as usual. We did this by dividing all the values by ~16.26.

The running time of our algorithm was not what we expected as we had a while loop followed by a for loop and followed finally by a nested for loop. Since our KMP uses the list data structure we did expect it initially to match the normalized $O(n^2)$ curve. We do not believe that the constant values of our algorithm nor did our choice of data structure affect the result. However, we believe that because the running time is in milliseconds the accuracy and fast running time of modern CPUs may have caused it to run at a much better speed than the $O(n^2)$curve. If we run hundreds to thousands of documents it may have followed the $O(n^2)$ curve more precisely.

Unexpected Cases/Difficulties.

The first challenge faced in this milestone started with gathering a dataset. In the beginning, we used random string generation software, but the results were not as expected. Every time we ran the program, every document we tested was considered plagiarized. This issue proved to show how likely it is to find “false positive matches”. The solution we came up with is to create ten documents with each document containing a single word. Then, we matched each file against a master document. With this solution working, we expanded by creating ten more documents. This time, each document contained multiple sentences rather than words. We compared each file to a master document.

An issue that quickly arose during the testing phases was that our current implementation of the algorithm would not work for most documents. The KMP algorithm would not match the documents properly because of the white spaces between words and at the end of the sentences. To fix this issue, we trimmed the whitespace at the end of each sentence in our main function.

The last difficulty during this milestone was the collection of plot data. We were not sure what sample size of data was large enough. Most importantly, we were not sure how many times we had to change the number of plagiarization that occur. We attempted to increase the number of document matches and plot the time as it grows. The time to complete seemed to grow linearly which was unexpected.

Task Separation and Responsibilities.

In terms of task separation, we continue to implement peer programming techniques. Specifically, Guy Kaminsky Wrote the abstract. Guy Kaminsky and Ross Morrison wrote on the implementation of KMP with some input from Matt Borle and Barry Feng. Next, Matt Borle was in charge of the data collection. He created several datasets for us to test out the KMP algorithm to ensure it works properly. Lastly, Matt Borle and Barry Feng were in charge of working on the results section. They used the data collected and created the plots using excel. Finally, everyone contributed to the unexpected cases and difficulties.