COSC 320 – 001
Analysis of Algorithms
2022/2023 Winter Term 2

Project Topic Number: #2

Milestone 4

Group Lead: Sumer Mann-65751364

Group Members:

Jay Bhullar -21474457

Dhairya Bhatia-27754175

DataSet and Laguange used:

We used - Dataset, which is a dataset containing 2,688,878 news articles and essays from 27 American publications. It contains .txt files. We trimmed the data because of the lack of computation power in our system. We had 500 articles in our dataset which has over 5 million characters. We initially decided in the project proposal to implement our code in Java but during Milestone 3 we realized that visualizing data in Python is much easier and effective to implement, hence we shifted to Python.

GitHub Repository:

https://github.com/Sumer26/project-cosc-320.git

Summary (Complete Project):

- **Milestone 1**

For Milestone 1, our group worked on the analysis of the Knuth-Morris-Pratt(KMP) algorithm. We explained the pseudo-code, the proof of correctness. In order to detect plagiarism, Due to the fact that the algorithm was not discussed in class, our group had to conduct initial research. We divided the tasks equally among the team members. The process involved looking for a pattern in a master array and determining where the pattern occurs. The algorithm's complexity was examined, and its runtime was found to be O(n+m).

- **Milestone 2**

For Milestone 2, our group worked collaboratively to analyze and document the LCSS algorithm. We included pseudo-code, algorithm analysis, proof of correctness, unexpected cases/difficulties, and citations. We divided tasks equally among three members, with each member responsible for a specific aspect of the project. We faced some difficulties in understanding the algorithm, but we did thorough research to overcome those difficulties. We found that The algorithm's runtime was O(nm).

- **Milestone 3**

For Milestone 3 we did the implementation of KMP. We trimmed the data because of the lack of computation power in our system. We used KMP to detect what part of the algorithm is plagiarised by making an input string and searching it in all the articles to detect from where the input string was taken. Various test strings were tested and searched across the articles in our dataset. We also tested empty strings to see the effect of how the KMP algorithm works. To compare the performance of the algorithm in various cases, we have plotted the graph of the number of iterations the KMP algorithm needs to search a pattern to the varying lengths of the patterns. We initially decided in the project proposal to implement our code in Java, during Milestone 3, we shifted to Python. We found that the implementation and visualization of data was easier in python. Moreover running large data was much easier in Python.

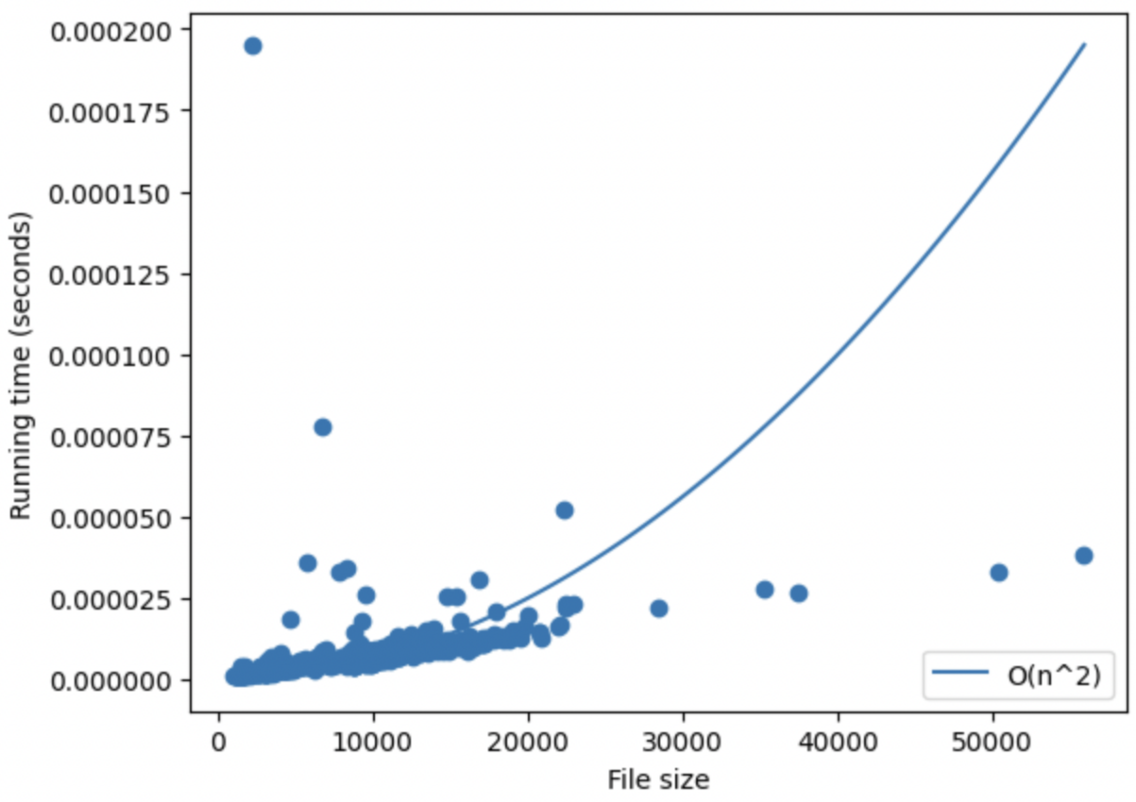
- **Milestone 4**

For milestone 4, we have completed all three algorithms with implementation and graphs. The LCSS algorithm is the longest subsequence finding algorithm that is one of the many algorithms needed for plagiarism detection. Then we also implemented the Rabin-Karp Fingerprints Algorithm. This is one of the best string matching algorithms which is used when any kind of plagiarism is to be detected. This is based on the hash mapping technique. This technique is chosen because of its efficiency and can detect matches in a very constant time. Furthermore, taking the help of graphs, we tried to answer some of the questions that were put for this project.

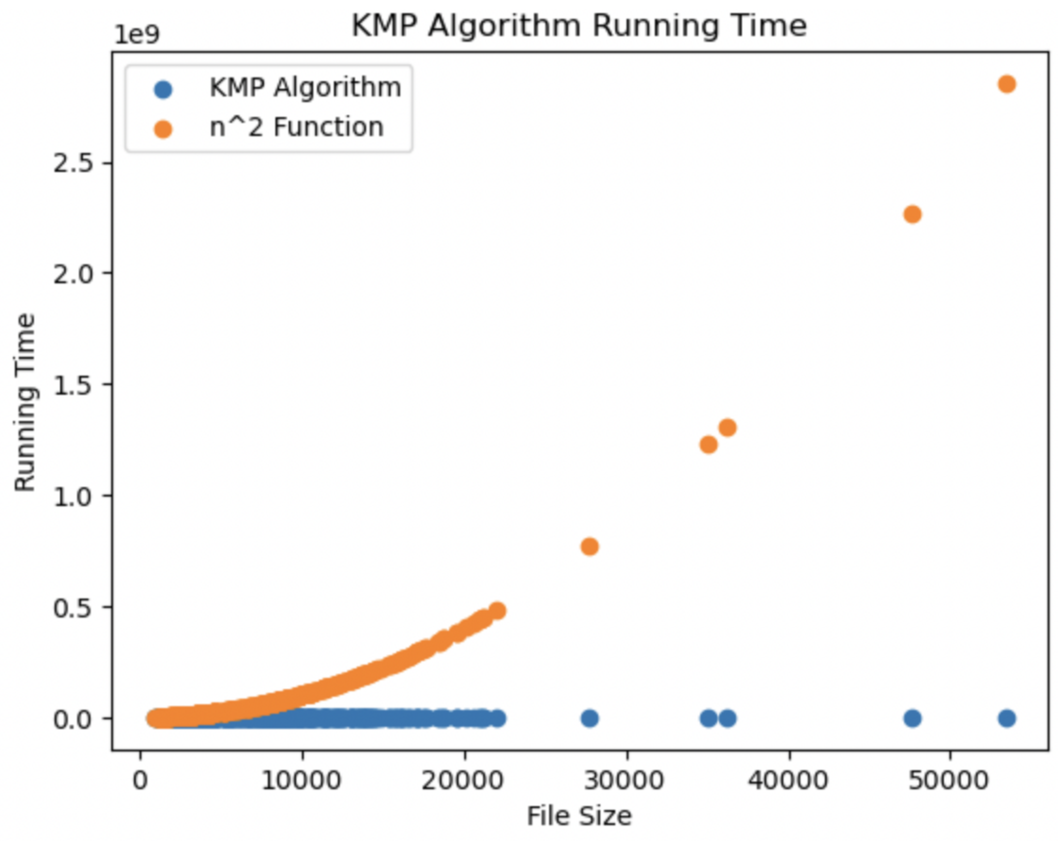
Results (Milestone 4):

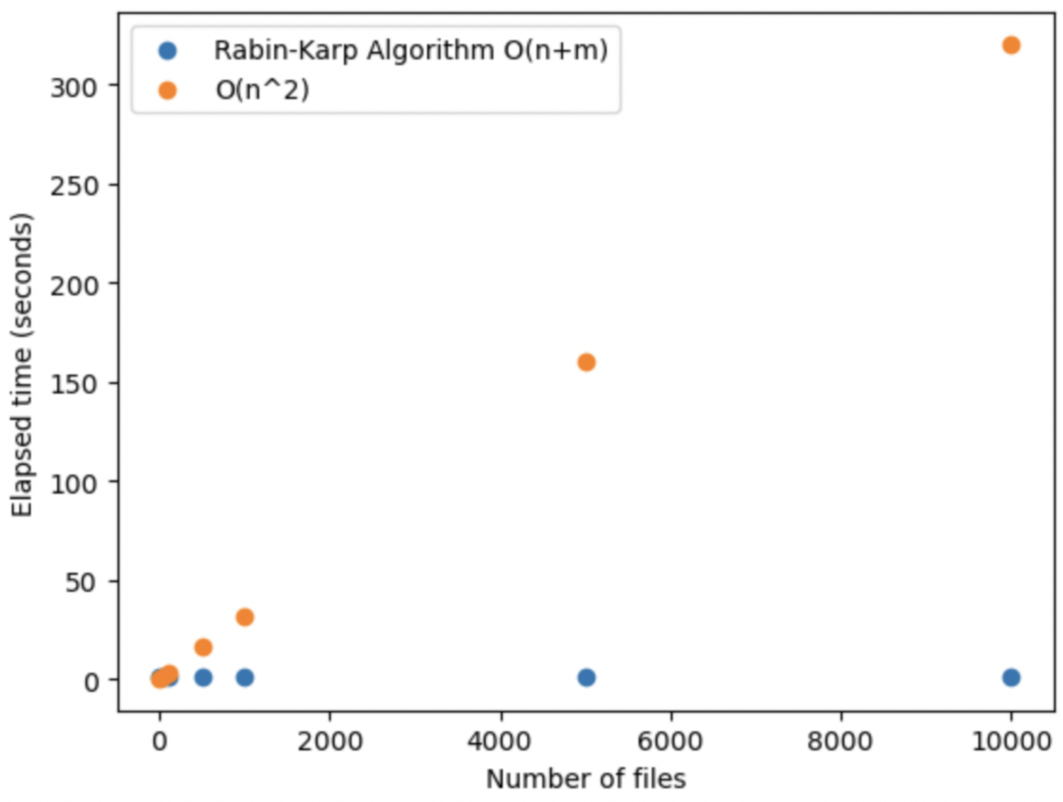
All three algorithms are very good string-matching algorithms and all of them prove to be very useful when detecting plagiarism.

1. KMP Algorithm: Knuth-Morris-Pratt Algorithm is a string-matching algorithm that allows efficient matching of the text with the pattern we are trying to find. Its time complexity is O(m+n). This algorithm is very useful when we have to find occurrences of the pattern in long texts.
2. LCSS Algorithm: The longest common subsequence algorithm is used to find the longest subsequence that is common in the pattern and the text provided. This algorithm is proved to be very successful when the two strings are very long and are similar but not identical. The time complexity is O(m*n).
3. Rabin-Karp Algorithm: This algorithm is also used for pattern matching in a string



When we compare the above LCSS scatter plot when compared against the O(n^2) graph , we find that the lcss matches that of the O(n^2), proving o


KMP Algorithm Running Time
1e9
KMP Algorithm
n^2 Function
Running Time
File Size
0.0
0.5
1.0
1.5
2.0
2.5
0
10000
20000
30000
40000
50000


Rabin-Karp Algorithm O(n+m)
O(n^2)
Elapsed time (seconds)
Number of files
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000

Unexpected Cases/Difficulties:

One unexpected difficulty we encountered during the implementation and testing of the KMP algorithm was our group member got ill and it caused us some delays during the milestone but we figured and managed to complete the project collectively as a group. The graph showed a linear relationship between the length of the pattern and the number of iterations required for the algorithm to search for it.

Task Separation and Responsibilities:

We divided tasks equally into 3 members as:

- Sumer Mann: Implementation and graphs
- Jay Bhullar: Implementation and Dataset collection.
- Dhairya Bhatia: Documentation and Implementation.

Citation(s):

All the News 2.0 — 2.7 million news articles and essays from 27 American publications. (n.d.). Components. https://components.one/datasets/all-the-news-2-news-articles-dataset