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

Project Topic Number: #1

Keyword Replacement

Group Lead:

Sanjith Senthil

Group Members:

Cole Van Steinburg, Issa Hashim, Sanjith Senthil

Abstract

In this milestone, we have successfully implemented the second algorithm - an improved approach to the keyword replacement problem. We tested the algorithm across a range of test cases, varying the number of inputs from 1000 records to millions. We generated plots depicting the algorithm's runtime and analyzed how it scales with the input size. In addition, we compared the runtime to its corresponding big O function.

Summary

Previously, we designed and implemented two algorithms for the keyword replacement problem.

The first algorithm, which is the naive approach, works by checking each word against every word in the replacement list and replaces it accordingly. The time complexity is O(n * m) where n is the number of words in the paragraph list and m is the number of words in the replacement list. This is because n words are checked against m entries in the replacement list. The auxiliary space complexity is O(1) as the algorithm does not use any additional space. However, the total space complexity is O(n + m) as the paragraph list and replacement list is stored in the memory.

The second algorithm, which is the improved approach, works by checking each word by looking up a replacement hashmap and replaces it accordingly. The time complexity is O(n) where n is the number of words in the paragraph list and m is the number of words in the replacement list. This is because each lookup in the hashmap runs in constant time. The auxiliary space complexity is O(m) as the algorithm uses additional space for storing all the entries from the replacement list into a replacement hashmap. However, the total space complexity still remains O(n + m) as the paragraph list and replacement list is still stored in the memory.

Dataset Link

The dataset was provided by the professor. The link to the dataset: Dataset Link

Implementation

For each csv file in the dataset, our algorithm reads in all the reviews to a list. Once the list is read in, it loops through each review. For every word in the review, we first strip and save any punctuation to be preserved after doing word replacement. After stripping punctuation, we look up the replacement word hashmap doing a comparison to the current word for each. If a match is found, the word is replaced. If a replacement is made, we add back the punctuation that was stripped earlier. Once the whole file has had words appropriately replaced, we write the results to a new csv file in a new directory.

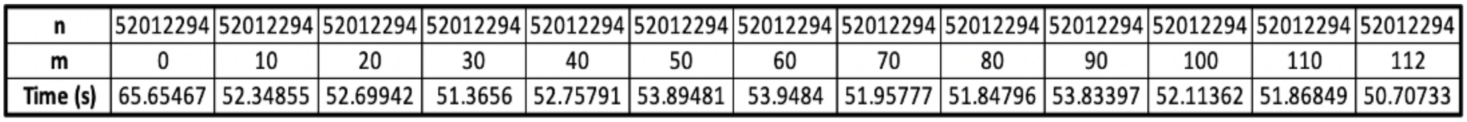
Code Link

The link to the code (GitHub repository): Code Link

Results

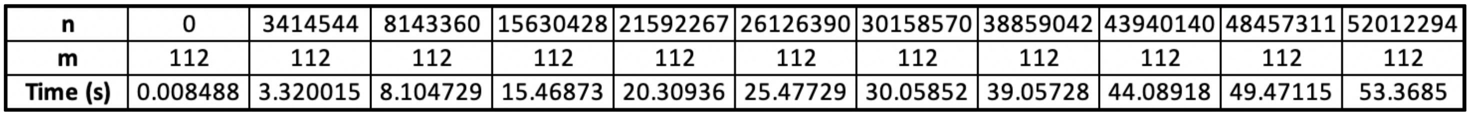
The runtime of the second algorithm is O(n). Since the runtime of the algorithm depends on two variables ‘n’ and ‘m’, we made different graphs to show what happens as ‘n’ increases with ‘m’ held constant and vice versa. Here, ‘n’ denotes the number of words in the paragraph list and ‘m’ denotes the number of words in the replacement list.

The data obtained and presented below maintains a constant value of ‘n’ while increasing the value of ‘m’.



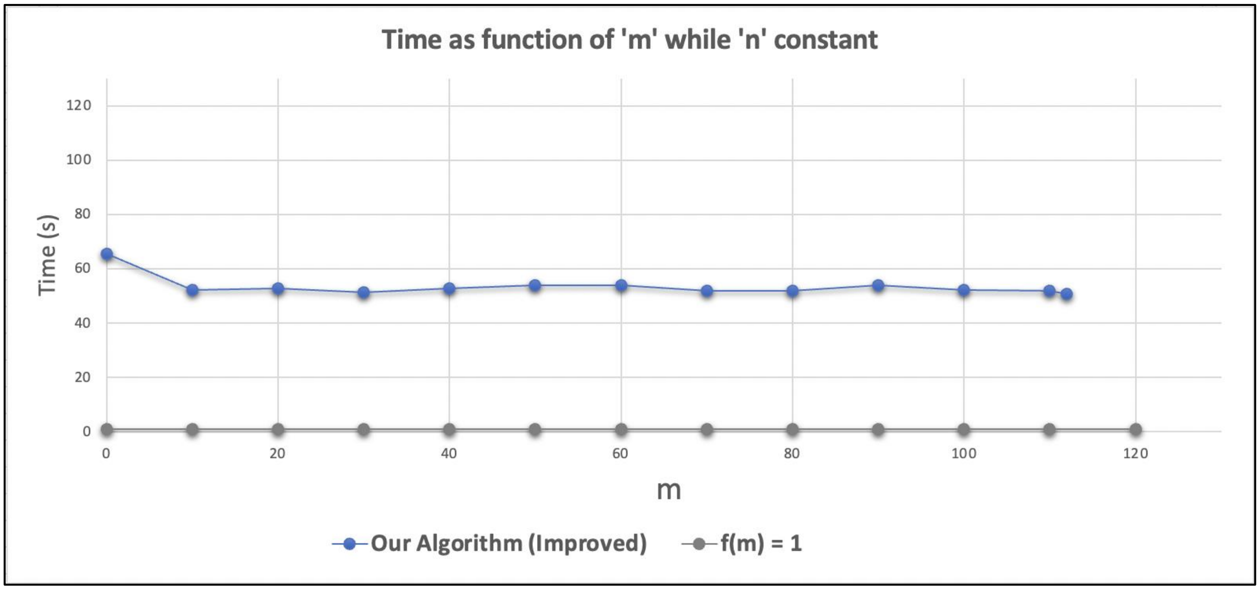
n	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294	52012294
m	0	10	20	30	40	50	60	70	80	90	100	110	112
Time (s)	65.65467	52.34855	52.69942	51.3656	52.75791	53.89481	53.9484	51.95777	51.84796	53.83397	52.11362	51.86849	50.70733

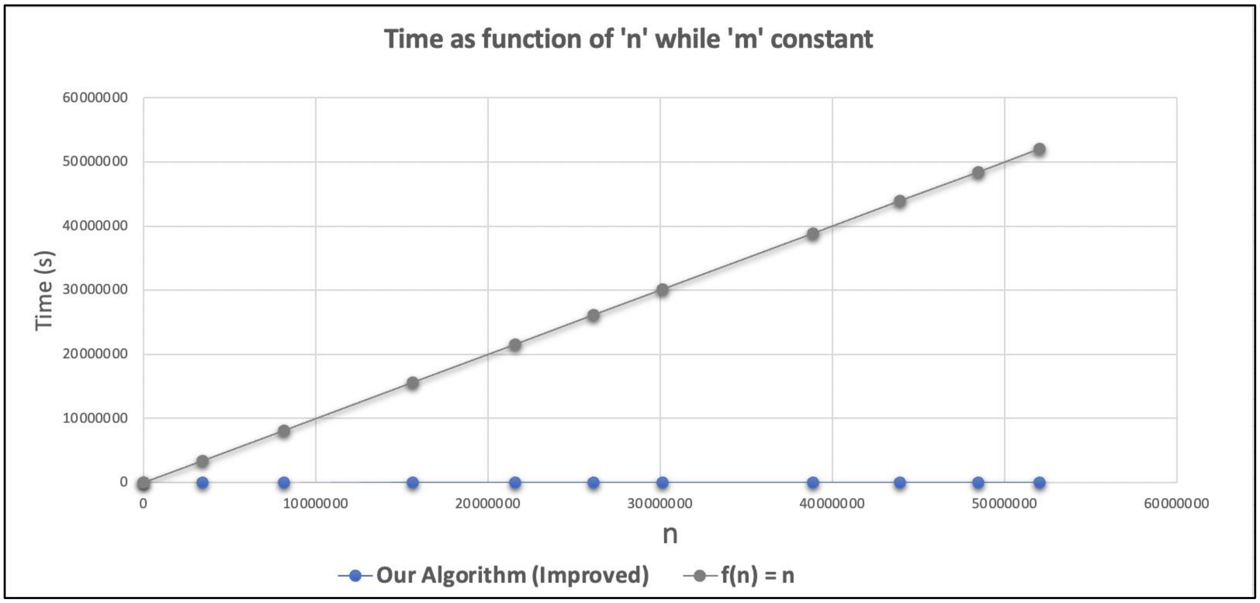
The data obtained and presented below maintains a constant value of ‘m’ while increasing the value of ‘n’.



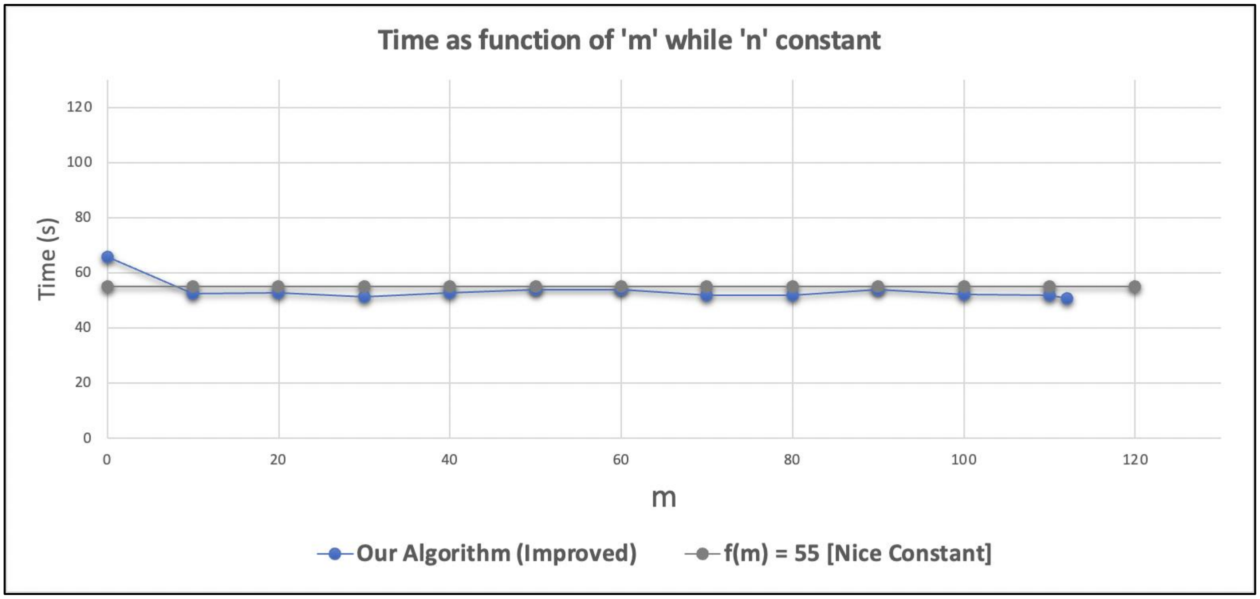
n	0	3414544	8143360	15630428	21592267	26126390	30158570	38859042	43940140	48457311	52012294
m	112	112	112	112	112	112	112	112	112	112	112
Time (s)	0.008488	3.320015	8.104729	15.46873	20.30936	25.47729	30.05852	39.05728	44.08918	49.47115	53.3685

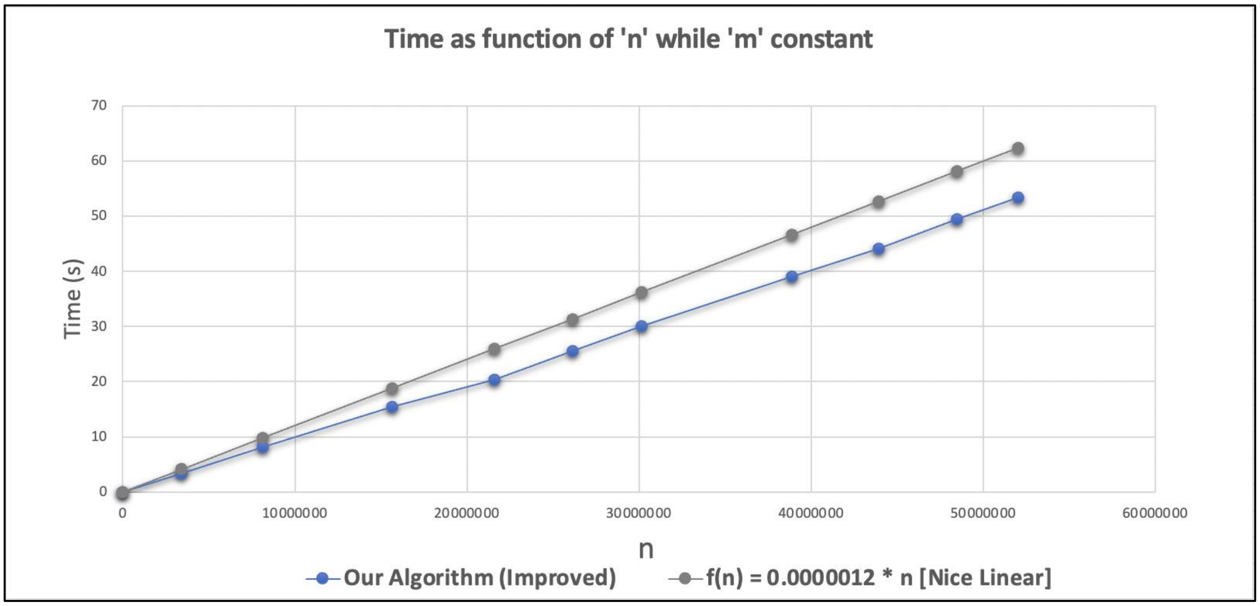
Generating the plots of the algorithm's runtime and comparing it with the big O functions f(x) = 1 and f(x) = x respectively:





Generating the plots of the algorithm's runtime and comparing it with a linear function and a constant function of matching slope respectively:



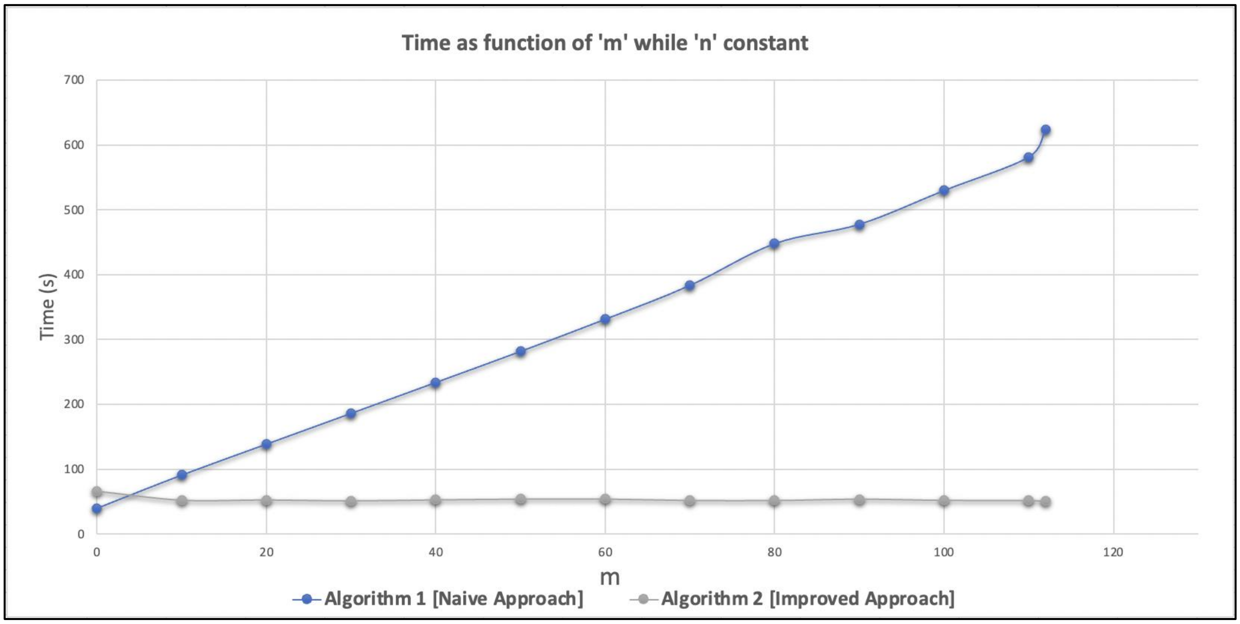


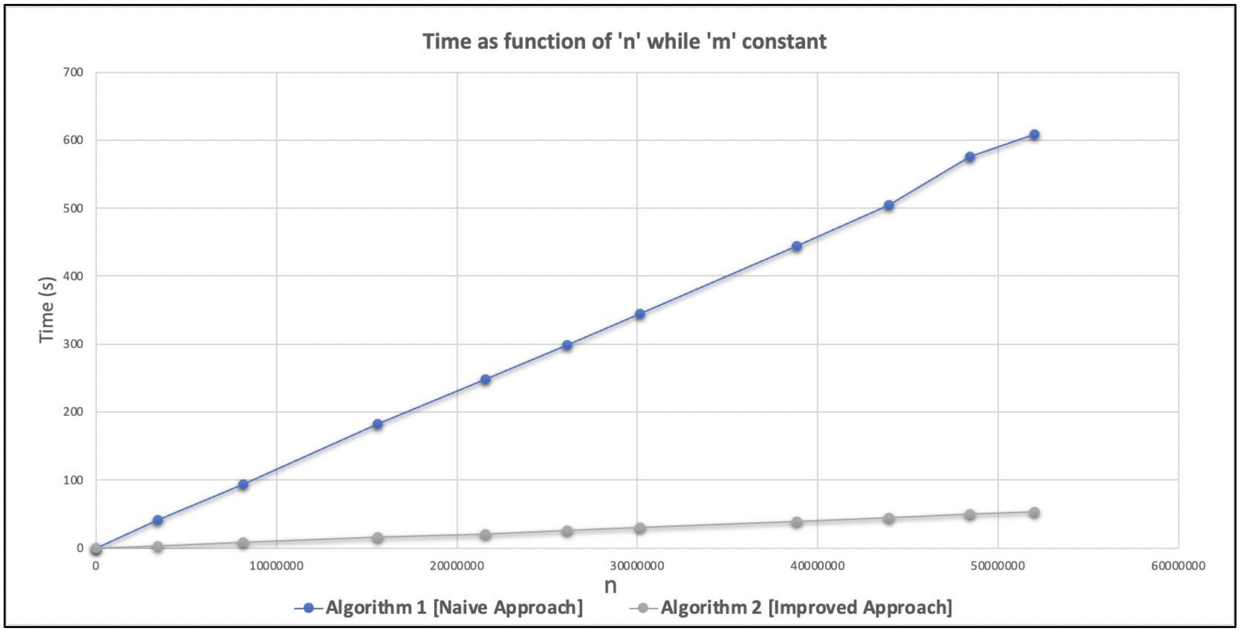
As expected, when 'n' is held constant, the run time is constant with the adjustment of 'm'. Conversely, when 'm' is held constant, the run time varies linearly with the adjustment of 'n'.

The implementation of the algorithm has affected the constant values. This is because there are lower order terms / constants pertaining to 'n' to read the files in, as well as stripping the punctuation from each word. These steps happen even if there are no words in the replacement list.

In the naive approach, a list of tuples is used to store the replacement list, leading to an overall time complexity of O(m * n). However, the improved approach uses a hashmap to store the replacement list, which allows for a O(1) time complexity for lookups. As a result, the overall time complexity of the improved algorithm becomes O(n).

Comparing the two algorithms





We observe that when holding either 'm' or 'n' constant while the other variable is varied, algorithm 2 outperforms algorithm 1. Hence, algorithm 2 is more optimal and faster in terms of time.

However, the tradeoff comes in auxiliary space complexity, where algorithm 1 requires O(1) space, and algorithm 2 requires O(m) additional space to store the replacement hashmap.

Unexpected Cases/Difficulties

The dataset contained partial corruption in certain places, leading to null bytes that caused exceptions during processing. To address this issue, we replaced all null bytes with empty spaces. This allowed us to work with the dataset without encountering errors and ensured that our analyses were based on reliable and error free data.

In addition to partial corruption and null bytes, we also encountered some unexpected characters that turned out to be emojis. These were not able to be read by default. To tackle this issue, we converted the reader to take in UTF-8 format.

Task Separation and Responsibilities

We divided the work among us equally throughout the project, meaning that all members of our group contributed to every task of all milestones together. We collaboratively worked on the algorithm implementation, testing, data collection, plot generation, interpretation, and analysis. In addition, Cole and Issa were responsible for setting up the group meeting, and Sanjith was responsible for communicating with the instructor and TA’s.