# CMP 201 – Data Structures And Algorithms 1

# String Searching (Boyer Moore & Rabin Karp)

BY CAMERON – 1901441

# Problem

The problem is a production company would like to adapt four scripts into plays. Two scripts are from movies and the other two are from books.

But the production company have asked that you create a program to find out how many times the main characters name appears. This is to give information to the company in order to create a program of the character.

The agency has asked to consider implementing a feature to change the target pattern/or file this is because they would like to find out how many times a minor character might appear or a specific word.

They have asked that you create the program using two standard string searching algorithms to complete this task.

# File Sizes

movie1.txt – 247KB – Batman Script

movie2-.txt – 155KB Shawshank Redemption Script

book1.txt – 421KB  - Frankenstein

book2.txt – 762 KB – Pride and Prejudice By Jane Austen

# Solution

The algorithms will need to search for a given pattern provided by the user and give the option to change the pattern to something else.

The algorithms will need to search through large amounts of data to find the pattern. The algorithm should be quick and efficient in achieving this.

# Algorithms

The two algorithms that were selected were the Boyer-Moore algorithm and the Rabin-Karp algorithm.

They are both string searching algorithms but use different methods of finding the pattern within large body's of text.

# Big o notation – Rabin-Karp

The average and best running time for Rabin Karp is O(n+m)

The worst-case running time is O(nm).

Occurs when characters in the pattern and the text are the same hash values

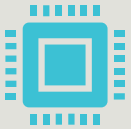
# Big o notation – Boyer Moore

The best-case running time is O(n/m).

The worst-case running time is O(mn).

# Abstract data types used - macros

Macros are used instead of a constant integer as this would improve the program to run more efficiently.



Macros are part of code that is declared and assigned when the program compiles.

It reduces the time to access memory when the variable is needed.

With a constant, it requires a storage location in memory while a macro does not.

Another reason is a constant variable may require it to be initialized during every function call.

```
//Prime Number & ASCII Characters
#define PrimeNum 13
#define NoChars 256
```

Rabin-Karp.h – lines 8-9

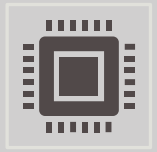
# Other Abstract Data Types Used

I decided to use arrays and vectors within the program.

The vector functions that are used, such as, push are O(1), while the use of Erase is 0(N).

During the programming phase, I used another ADT called positions. However, I found that the program runs more efficiently without it.

# ABSTRACT data Types – Vector & Arrays



**The skip table variable is an array, used in my Boyer Moore algorithm and is a suitable ADT for this algorithm. The program is able choose the position it needs to enter or retrieve data from the array. Which occurs when the algorithm needs to skip.**

**Since the array always is retrieved in the Boyer Moore function during each skip, the time complexity of this is operation is O(1).**

**I declared two vectors that are in my Rabin Karp and Boyer Moore classes. The first one is for collecting the number of locations where the pattern is in the text. The second vector is for a collection of multiple time results after each algorithm.**

# Debugging and Testing

Rabin Karp – I implemented two different rolling hash algorithms to see if it would boost the performance/speed of Rabin Karp.

The hash functions were Adam Sampson's 'terrible hash function' from the week 7 lecture slides and finally a compute hash function.

During testing, I did conduct a test on each hash function, hashing the same the word and having Rabin Karp looking for the pattern multiple times.

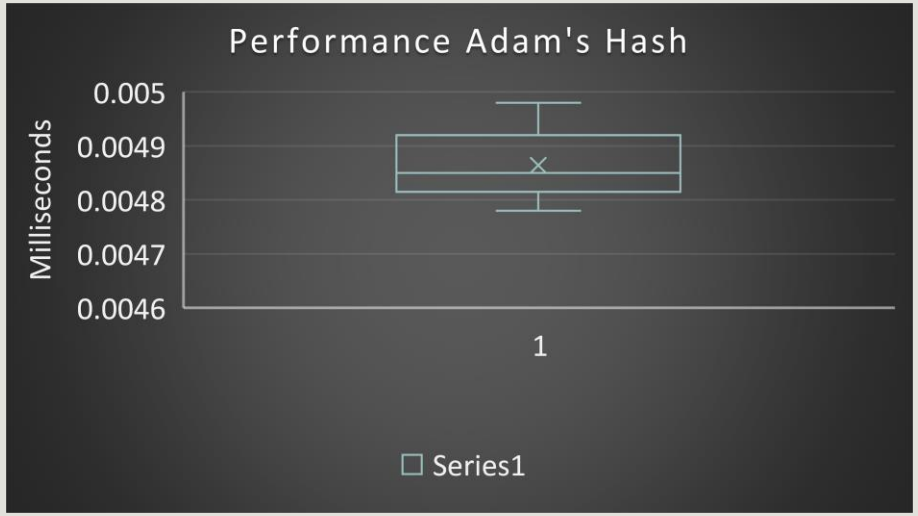
From this, I could work out the mean average of times taken for Rabin Karp to finish using the two different hashing functions. I hashed the word Batman from movie1.txt

# Debug results

## ADAM'S TERRIBLE HASH

By comparing the boxplots, we see that the average mean is 0.004864 milliseconds.

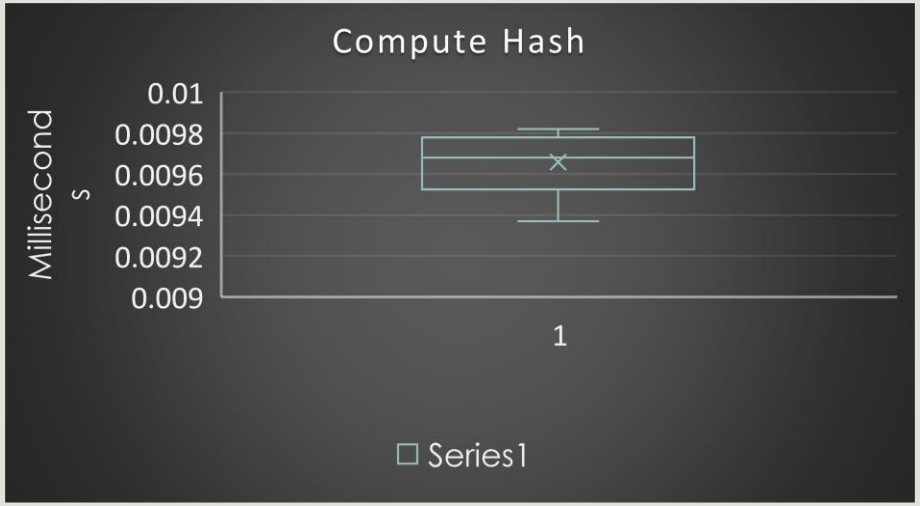
Median = 0.00485ms Interquartile range = 0.000105ms




## COMPUTE HASH

Rabin-Karp with the compute hash takes slightly longer compared to Adams with an average mean of 0.009658 milliseconds for finding the word Batman in the movie script

Median = 0.00968ms, Interquartile range = 0.000255ms



# DEBUG RESULTS CONT’

•Overall, by comparing how long Rabin Karp took with the two rolling hash functions for better performance of Rabin Karp, it is more suitable to use Adams 'terrible hash' function. With a lower average mean compared to the Compute hash.

Another reason is Adam's hash function having a lower interquartile range which means the resulting times will be less varied.

Adam’s Hash Function IN range: 0.000105ms

Compute Hash Function IN range: 0.000255ms

# Expected Results

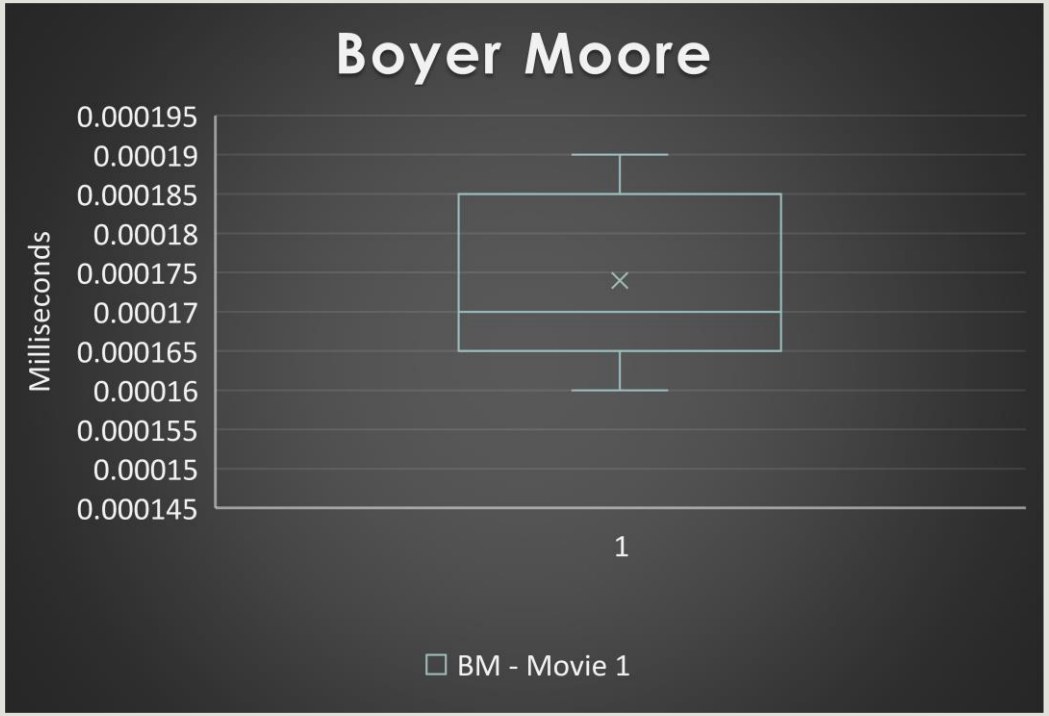
Before comparing the performance of these two algorithms, we should look at the time complexities for both and how they'll manage the task.

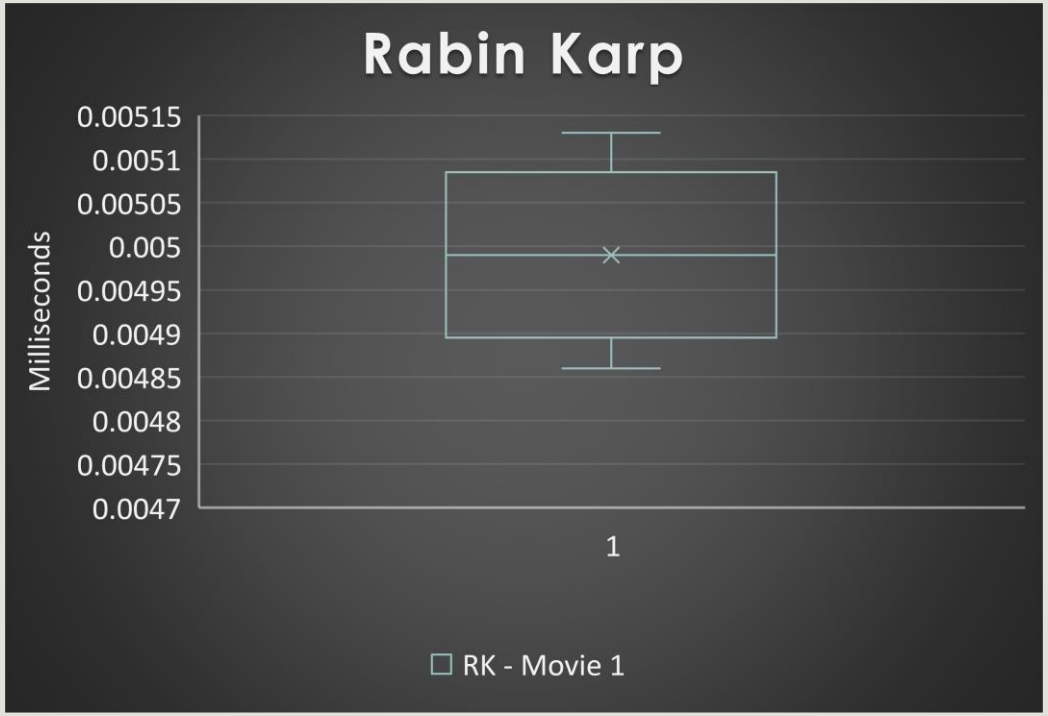
Boyer Moore has the best case running time of O(n/m) and worst-case running time of O(mn), which occurs when characters in the text and pattern are the same.

•Rabin Karp has an average and best running time of O(n+m) and a worst-case running time of O(nm), which occurs if the pattern and the text are the same hash values, an example would be if characters in the script have similar names.

Boyer Moore is likely to be the faster of the two and that it will handle a large amount of data and deal with matching characters more efficiently than Rabin Karp's matching hashes.

# Results: Movie1.txt
# Pattern: Batman



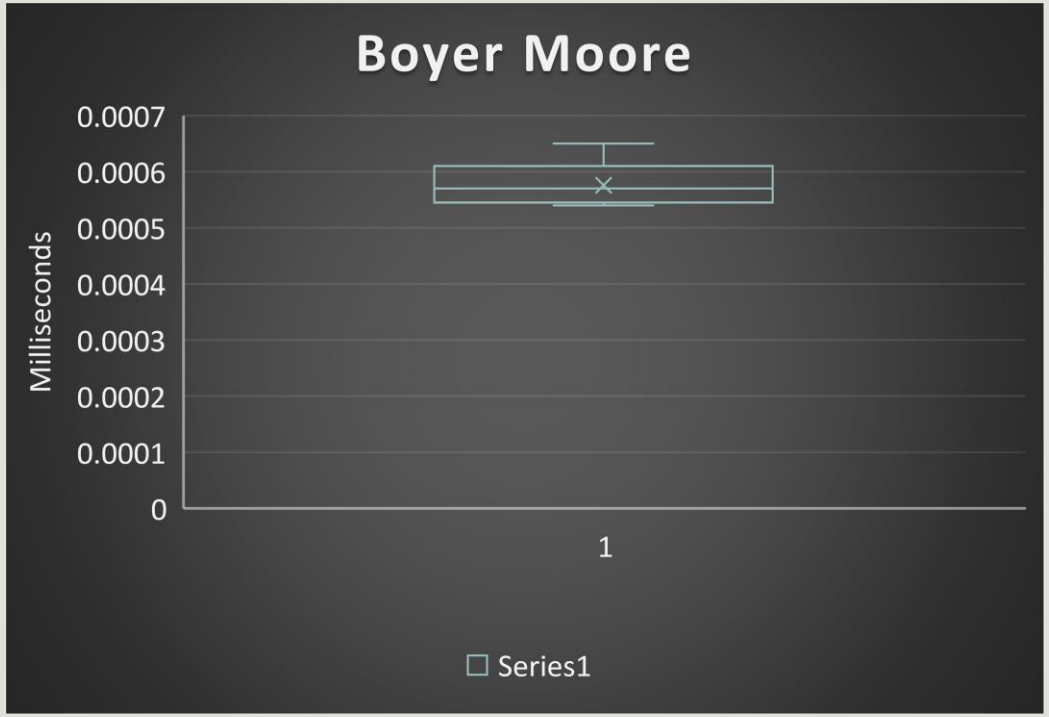

MEDIAN: 0.00017MS

MEAN: 0.000174MS

MEDIAN: 0.00499 MS

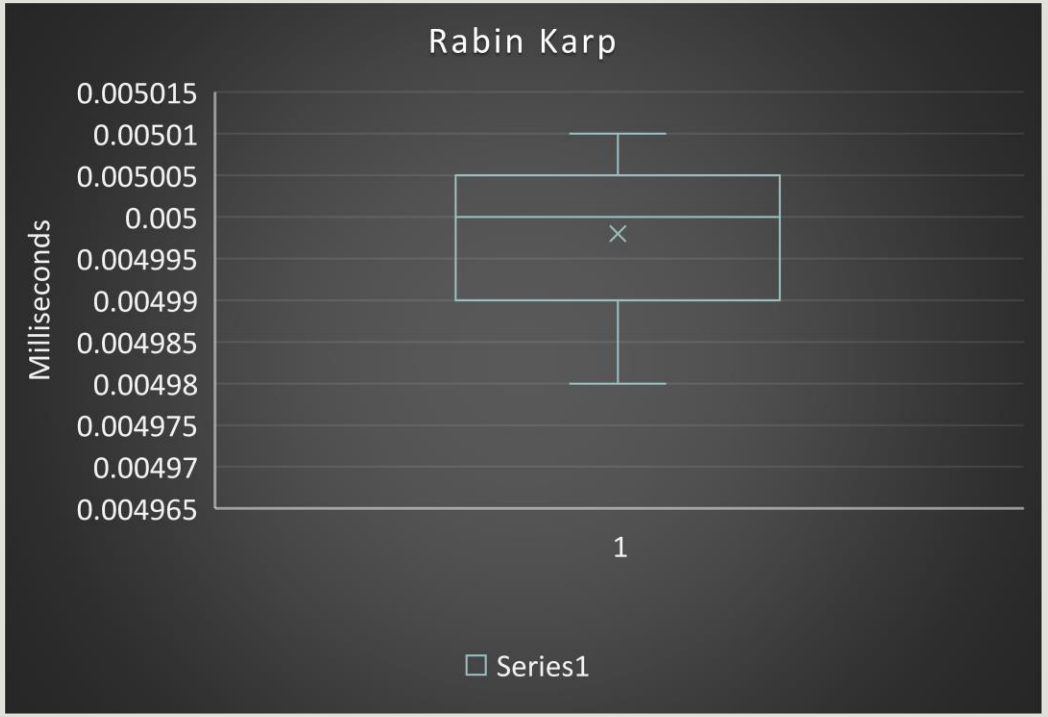
MEAN:0.00499MS

# RESULTS: movie1.txt
# Pattern: he




MEAN: 0.000576MS

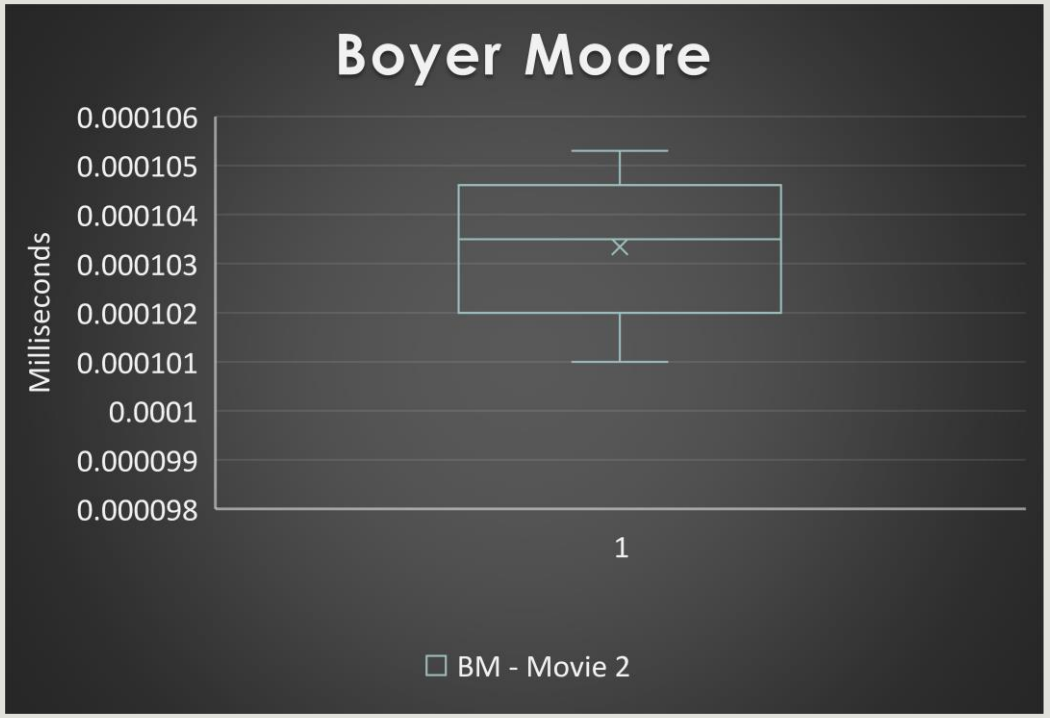
MEDIAN: 0.00057MS




MEAN:0.004998MS

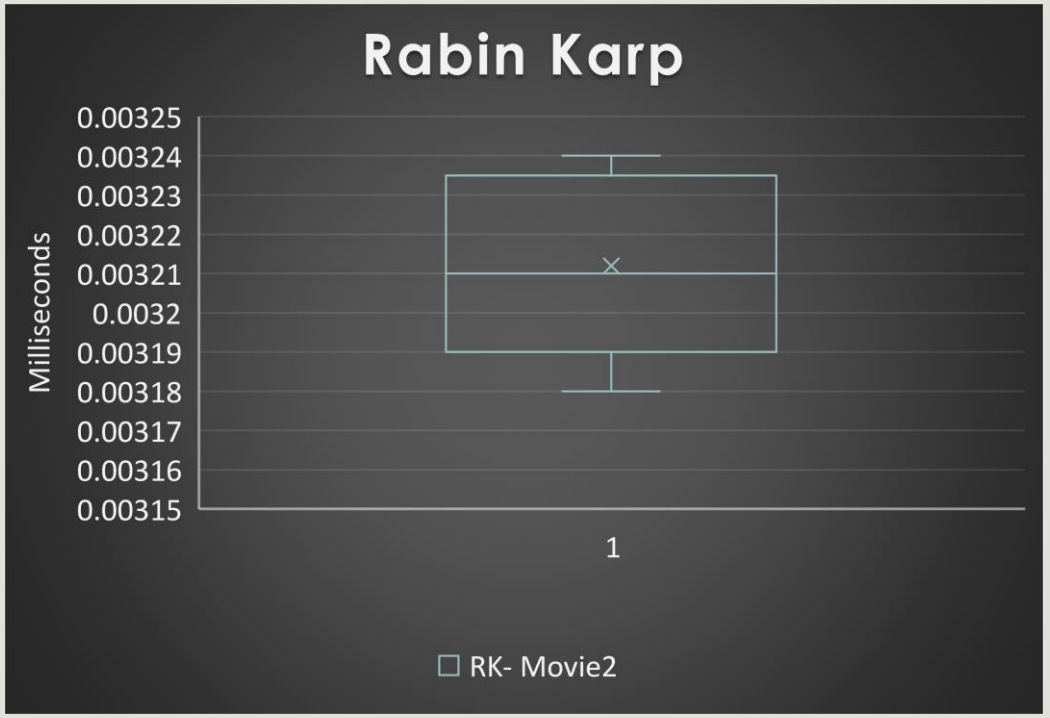
MEDIAN:0.0005MS

# Results: Movie2.txt
# Pattern: Dufresne




MEDIAN:0.000104MS

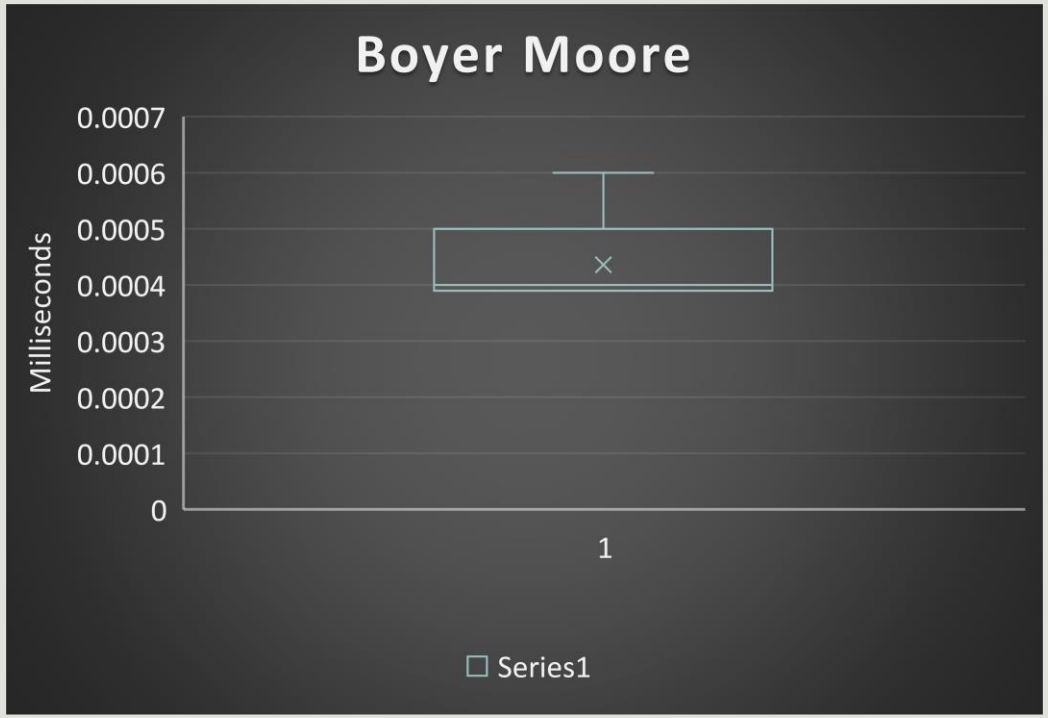
MEAN:0.00010334MS




MEDIAN: 0.00321MS

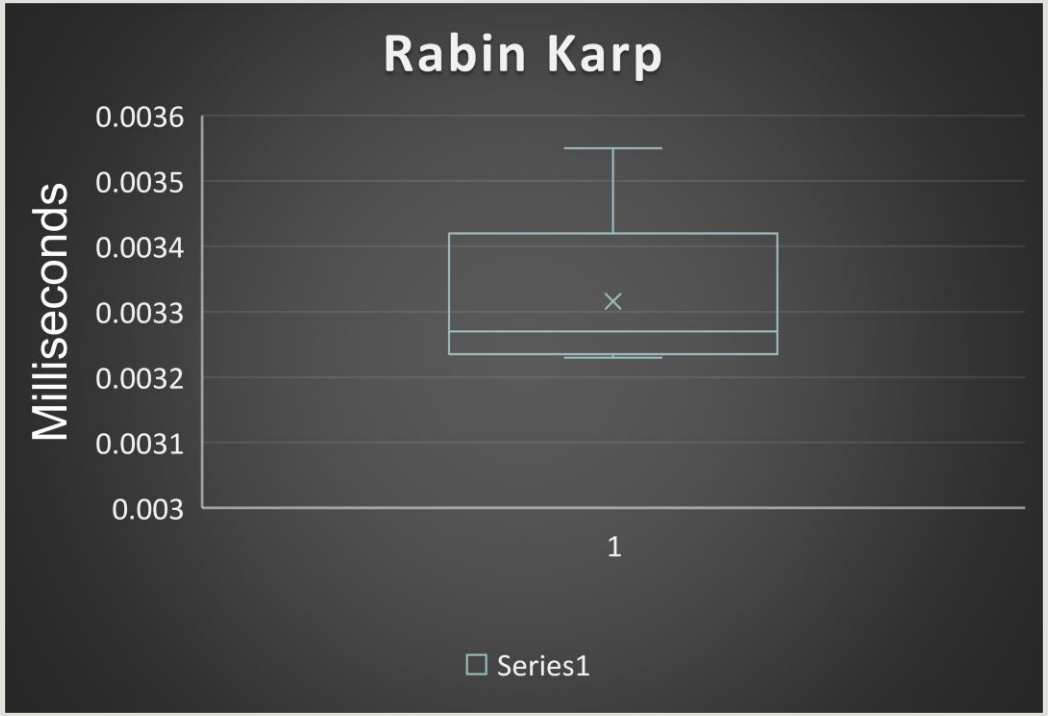
MEAN: 0.003212MS

# Results: Movie2.txt Pattern: he




MEDIAN:0.0004

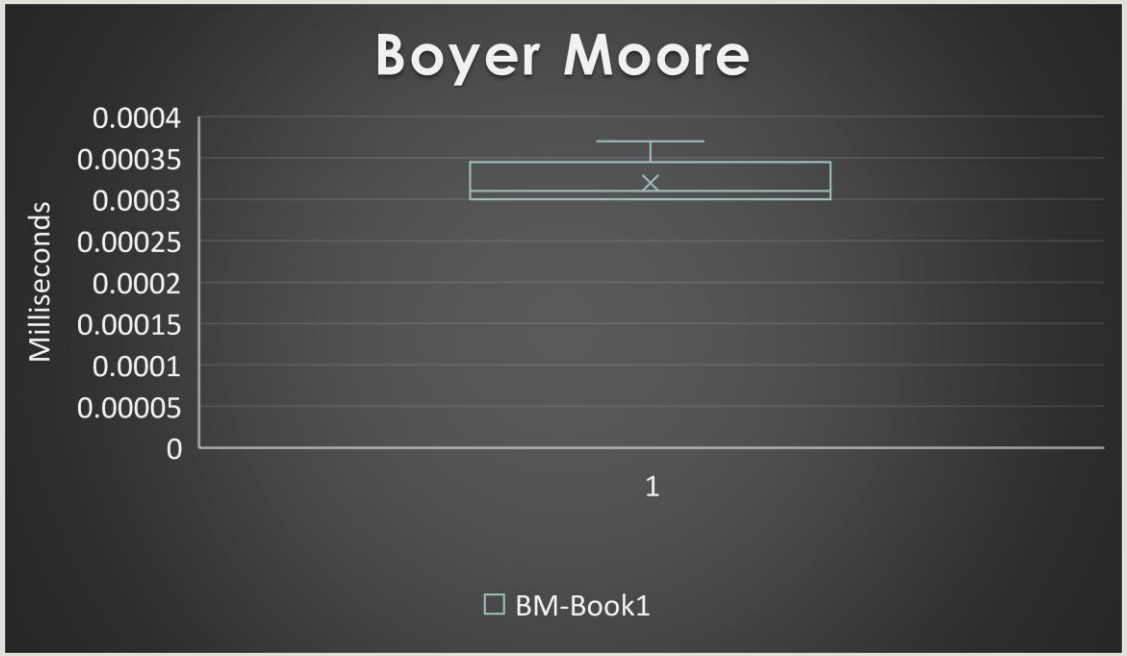
MEAN:0.000436MS




MEDIAN:0.00327MS

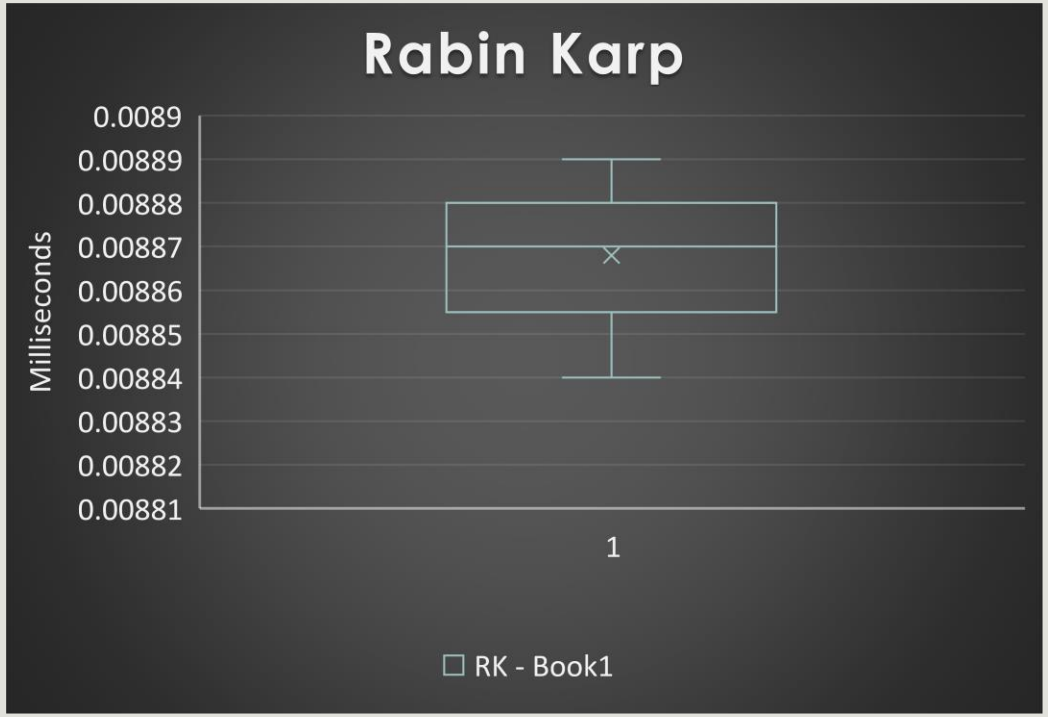
MEAN:0.003316MS

# Results: book1.txt pattern: Saville




MEAN:0.00032MS

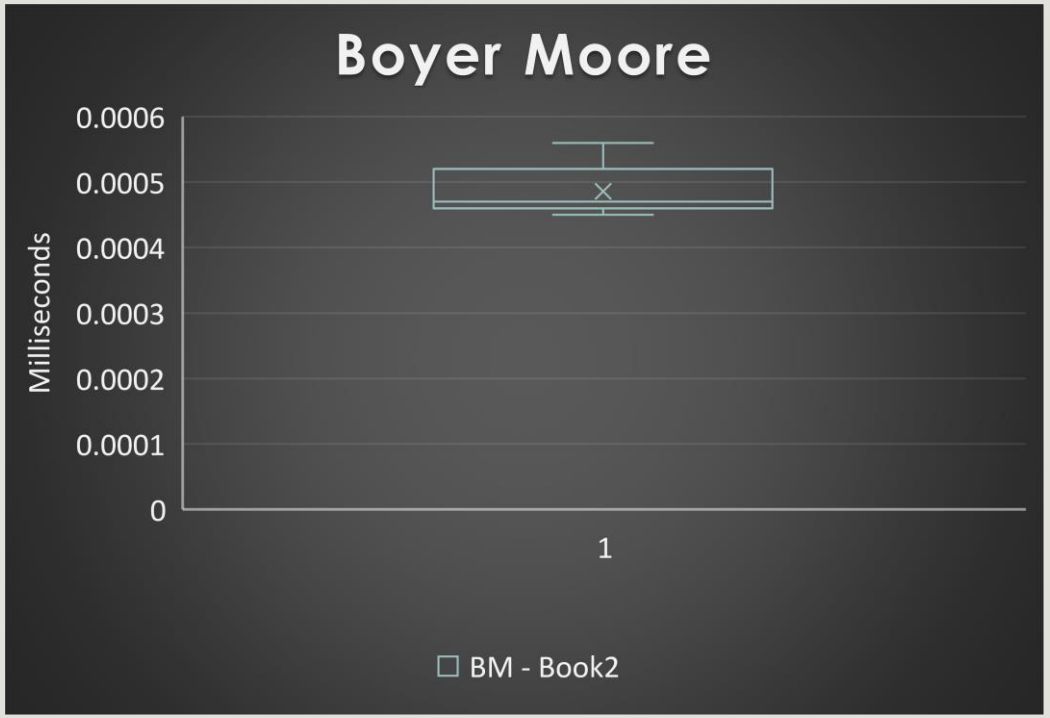
MEDIAN:0.00031MS




MEAN:0.008868MS

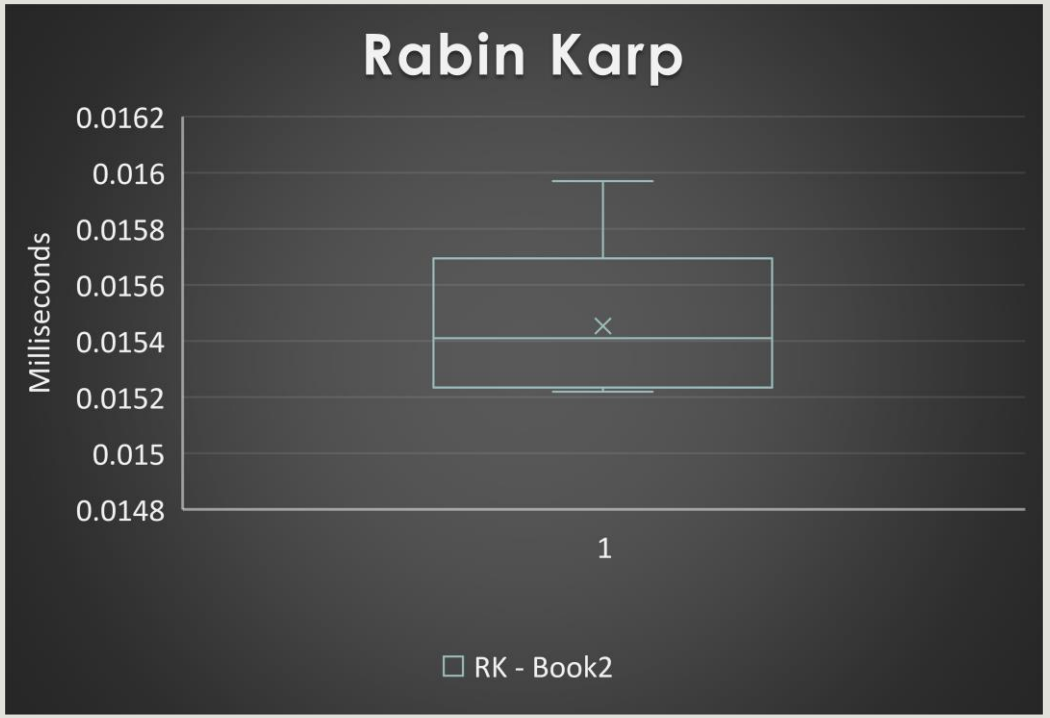
MEDIAN:0.00887MS

# Results: Book2.txt
# Pattern: Elizabeth




MEDIAN:0.00047MS

MEAN:0.000488MS




MEDIAN:0.01541MS

MEAN:0.015454MS

# Conclusion

In conclusion, from the collected data, it is clear that the Boyer Moore algorithm is faster compared to Rabin Karp. As previously mentioned the Interquartile range of the word Batman for Rabin Karp was 0.000105ms.

If we compare that with the Boyer Moore algorithm, we can see its Interquartile range is 0.000020 milliseconds. With this information, it is clear that Boyer Moore is a faster string searching algorithm than Rabin Karp.

Boyer Moore is the more suitable algorithm to solve the problem.

# References

Movie Scripts

https://www.imsdb.com/

Books

https://www.gutenberg.org/

Adam Sampson

The Hash Function from week 7

# Questions?