ALGORITHMS

Searching Algorithms

Agenda

1. Introduction
2. Demo
3. Algorithms used
    3.1. Brute Force
    3.2. Knuth–Morris–Pratt
    3.3. Boyer–Moore
4. Results
5. Conclusion

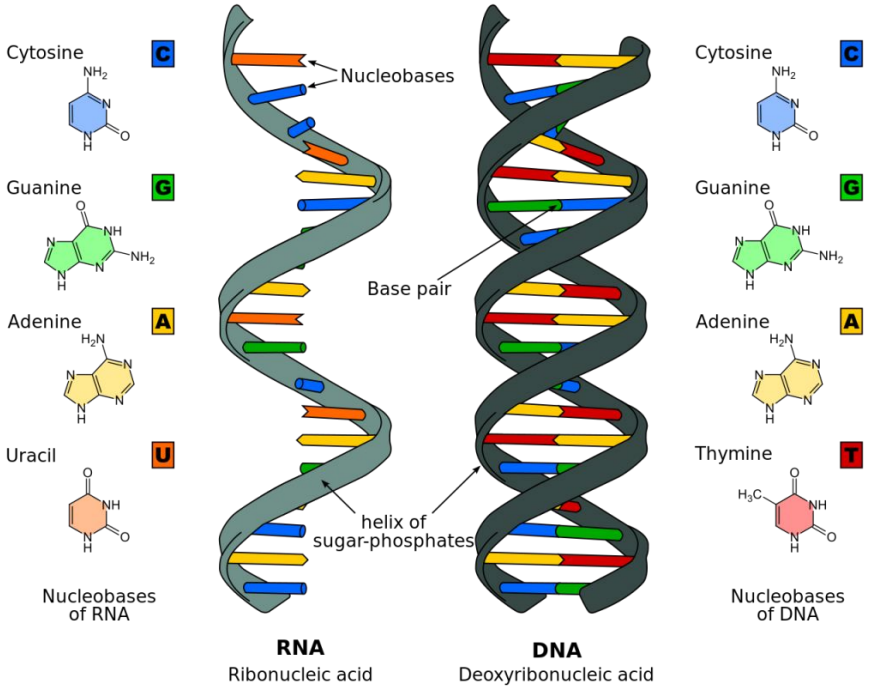
Problem Statement

Performing dna sequence searching

Consist of 4 alphabets

Genome sequence files can range from kilobytes to gigabytes

Time consuming process with bruteforce methods



Aim

Look into how we can **speed up** the searching process through **implementing smarter and better algorithms**, instead of just using brute force methods. As a result researchers would **spend less time waiting** for their search to complete, **greatly improving their efficiency**.

Data Resources Used

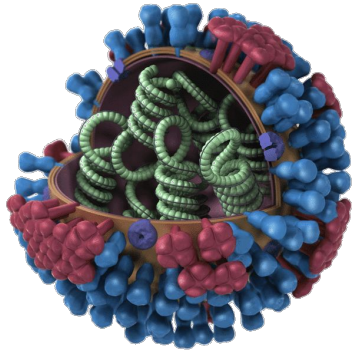
Genome sequences were downloaded from:

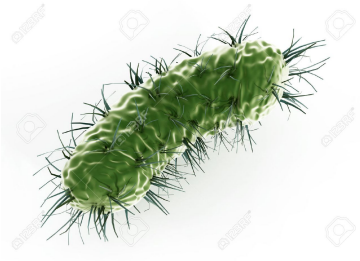
The National Center for Biotechnology Information (NCBI)

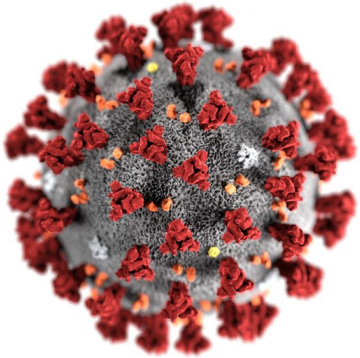
30 KB → COVID.FNA

5.0 MB → SALMONELLA.FNA

1.4 GB → INFLUENZA.FNA





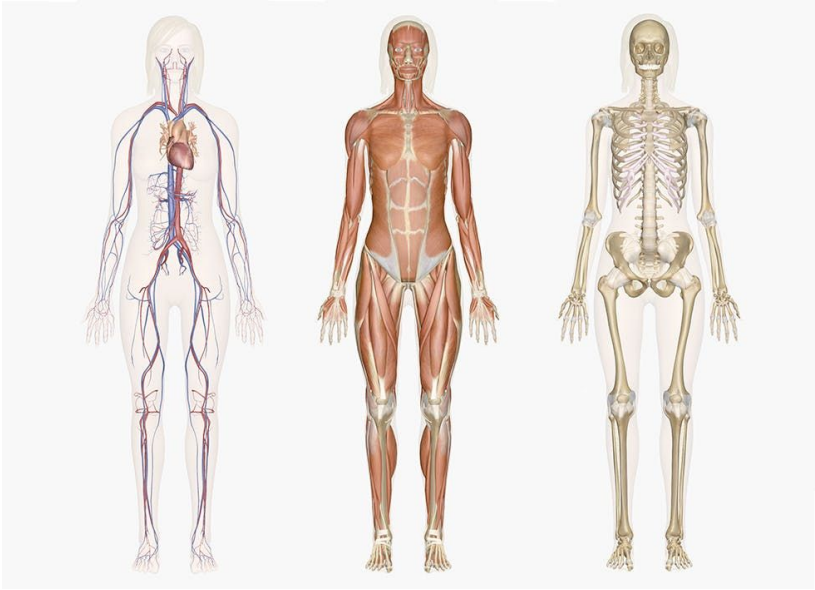


Possible Limitations

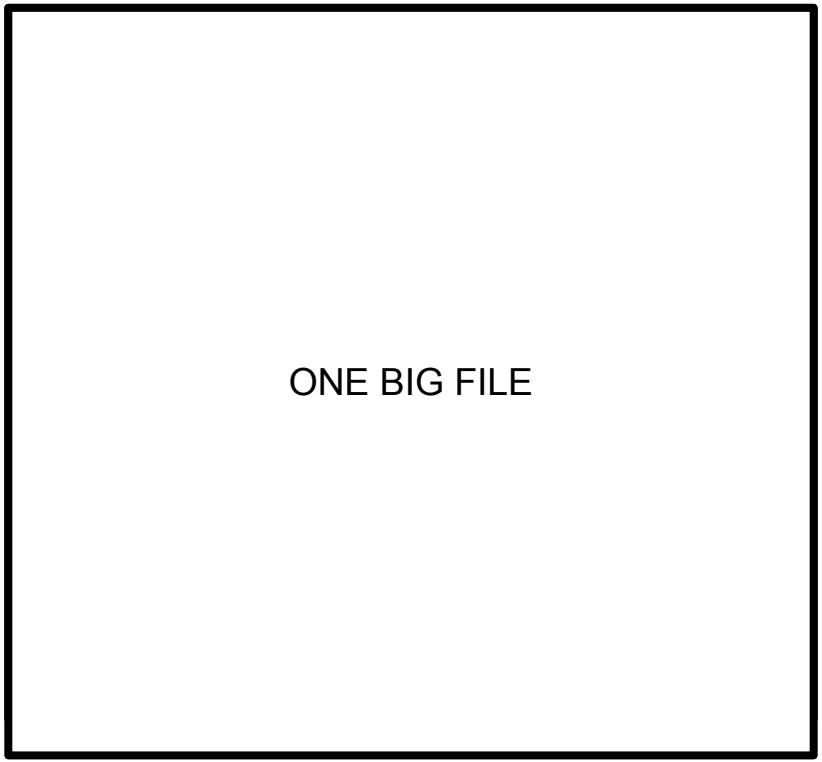
The **human genome** is over 6 billion in length equating to **6 GB**

If it were to be **directly loaded into a memory** to process

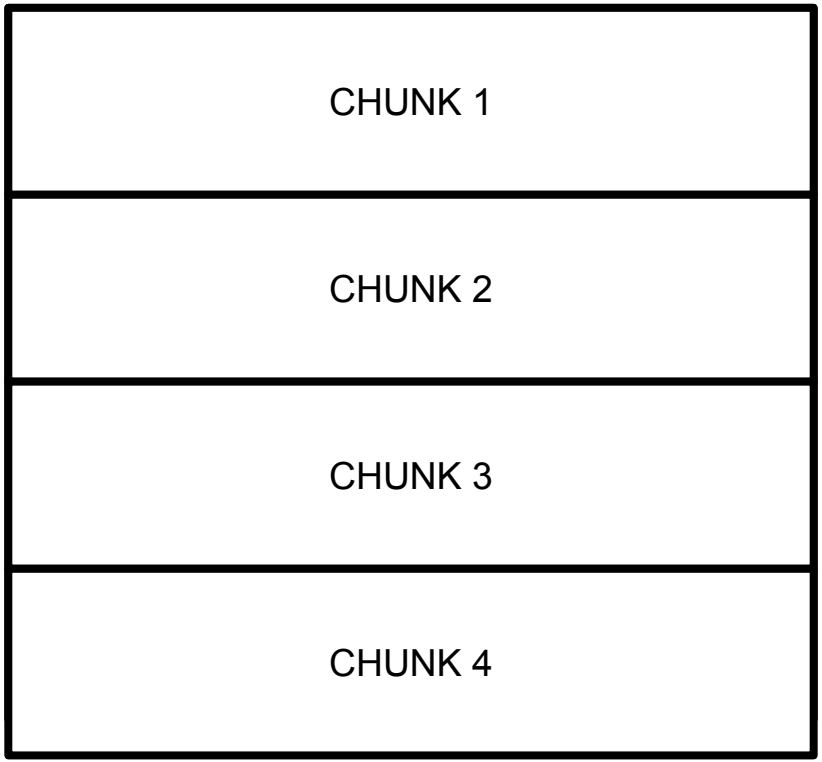
We would possibly face **memory limitations**



Significance of Our Code



Significance of Our Code



Significance of Our Code

Significance of our code

Splitting up the genome that we are searching through

Influenza genome (1.4GB) → 14 Influenza chunks **(0.1GB)**

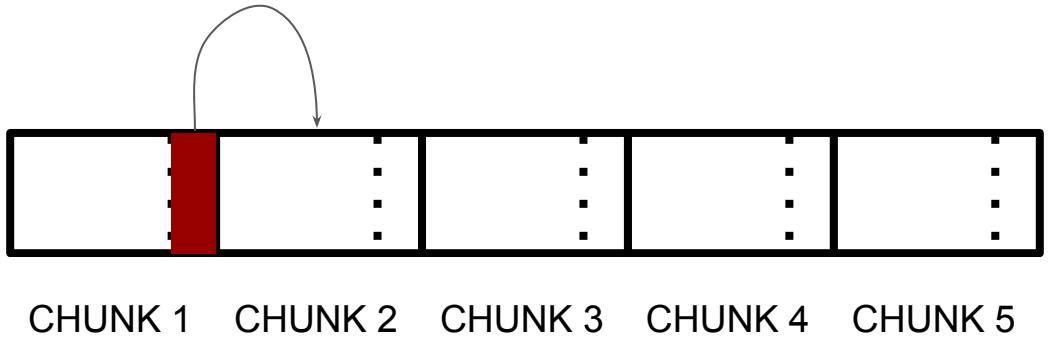
Significance of Our Code

Significance of our code

As some query sequences could be cut apart by splitting process

Ensure the there is a buffer between chunks

Makes sure that dna sequences caught in between chunks are accounted for



Plan of Action

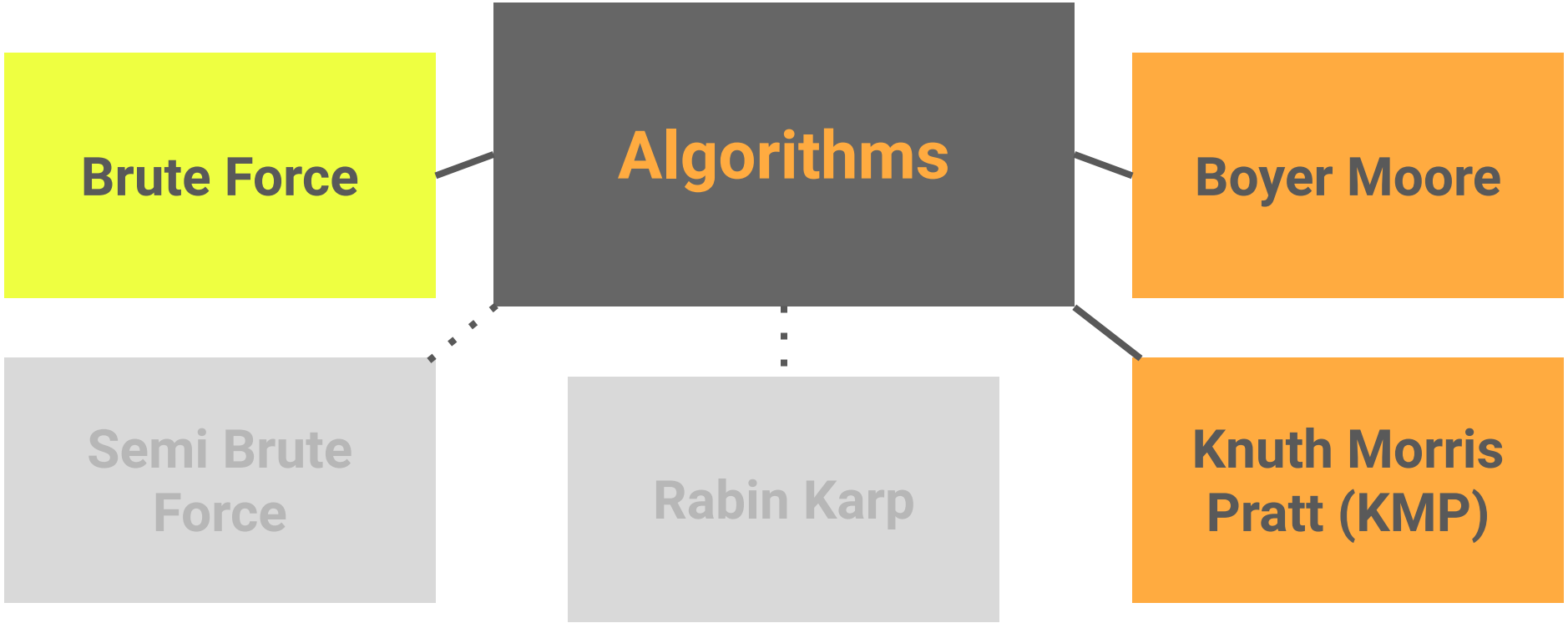
Implement Algorithms

Perform Asymptotic Analysis

Conduct Tests & Visualizing with Graphs

Leading to Conclusion

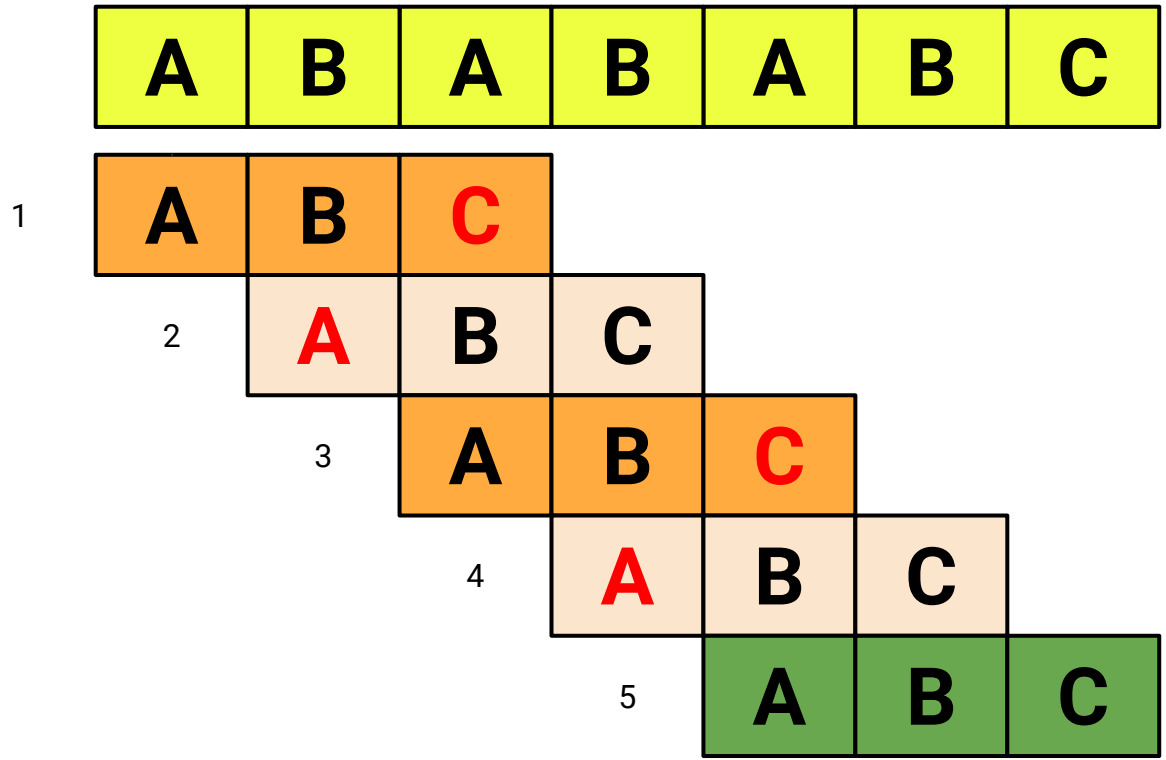
Implementing Algorithms



Implemented Algorithms - Demo with COVID.FNA

...ATCACGTAGTCGCAACAGTTCAAGAAATTCAACTCCAGGCAGCAGTAGGGGAACTTCTCCTGCTA
GAATGGCTGGCAATGGCGGTGATGCTGCTCTTGCTTTGCTGCTGCTTGACAGATTGAACCAGCTTGA
GAGCAAAATGTCTGGTAAAGGCCAACAACAACAAGGCCAAACTGTCACTAAGAAATCTGCTGCTGAG
GCTTCTAAGAAGCCTCGGCAAAAACGTACTGCCACTAAAGCATACAATGTAACACAAGCTTTCGGCA
GACGTGGTCCAGAACAAACCCAAGGAAATTTTGGGGACCAGGAACTAATCAGACAAGGAACTGATTA
CAAACATTGGCCGCAAATTGCACAATTTGCCCCCAGCGCTTCAGCGTTCTTCGGAATGTCGCGCATTG
GCATGGAAGTCACACCTTCGGGAACGTGGTTGACCTACACAGGTGCCATCAAATTGGATGACAAAGA
TCCAAATTTCAAAGATCAAGTCATTTTGCTGAATAAGCATATTGACGCATACAAAACATTCCCACCAA
CAGAGCCTAAAAAGGACAAAAAGAAGAAGGCTGATGAAACTCAAGCCTTACCGCAGAGACAGAAGAA
ACAGCAAACTGTGACTCTTCTTCCTGCTGCAGATTTGGATGATTTCTCCAAACAATTGCAACAATCCA
TGAGCAGTGCTGACTCAACTCAGGCCTAAACTCATGCAGACCACACAAGGCAGATGGGCTATATAAA
CGTTTTCGCTTTTCCGTTTACGATATATAGTCTACTCTTGTGCAGAATGAATTCTCGTAACTACATAGC
ACAAGTAGATGTAGTTAACTTTAATCTCACATAGCAATCTTTAATCAGTGTGTAACATTAGGGAGGAC
TTGAAAGAGCCACCACATTTTCACCGAGGCCACGCGGAGTACGATCGAGTGTACAGTGAACAATGCT
AGGGAGAGCTGCCTATATGGAAGAGCCCTAATGTGTAAAATTAATTTTAGTAGTGCTATCCCCATGTG
ATTTTAATAGCTTCTTAGGAGAATGAC**AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA**

Brute Force - Linear Searching



Boyer Moore

Me: “It’s like doing grep in unix!”

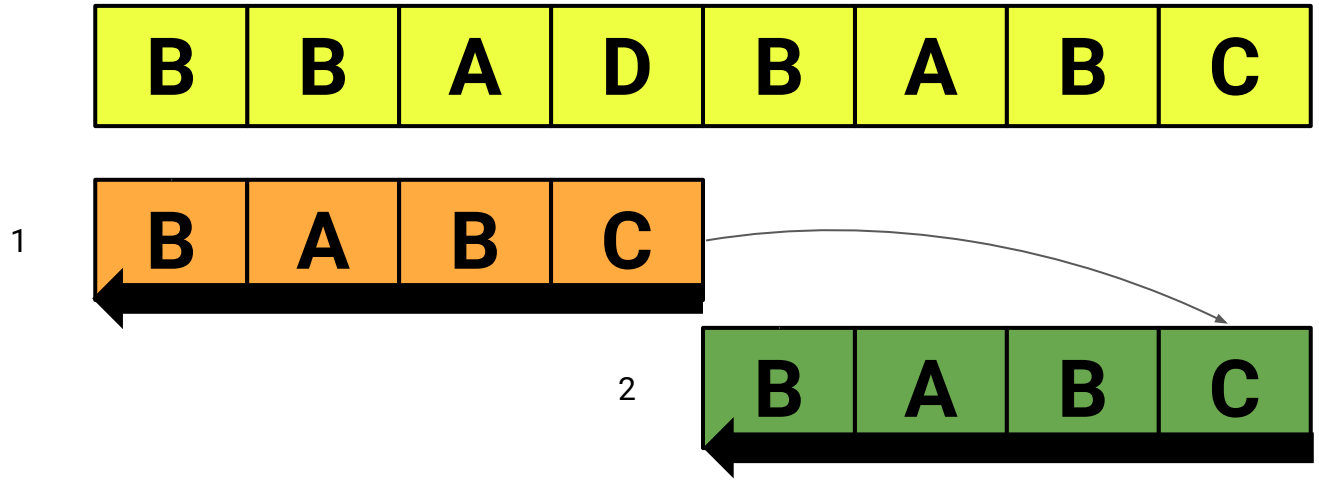
Google: “GNU grep uses the well-known Boyer-Moore algorithm.”

Boyer Moore: “Making full use of what you know to do as less as possible.”

Features

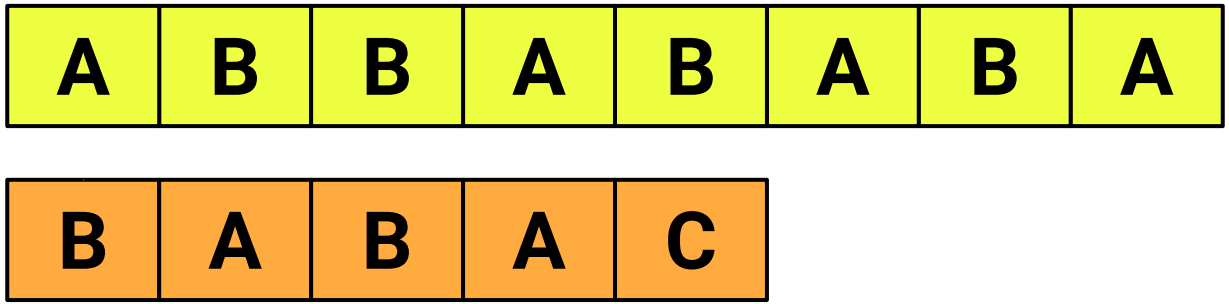
1. Right to left scan
2. Bad character rule
3. Good suffix rule + partial good suffix rule

Boyer Moore - Right to Left Scan

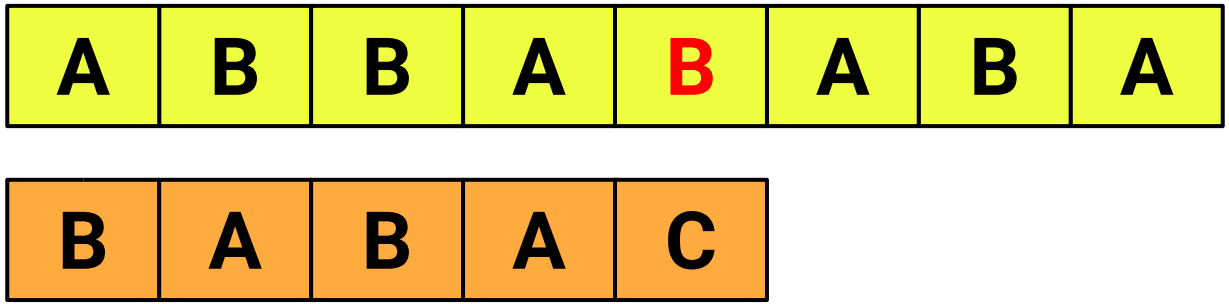


Bad Character Rule

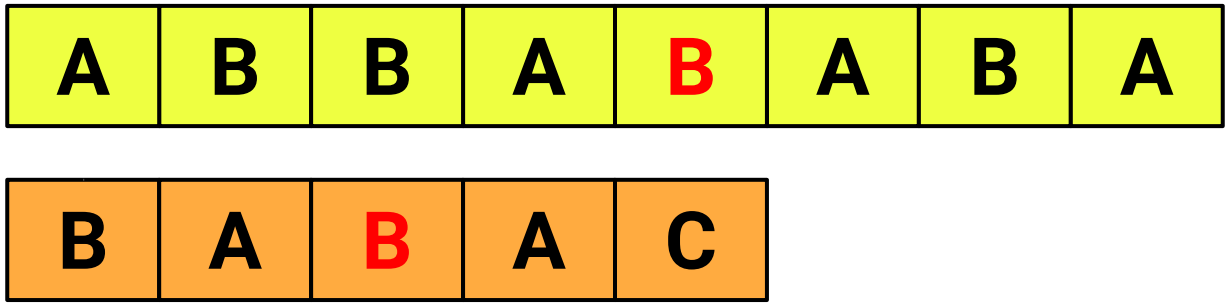
Boyer Moore - Bad Character Rule



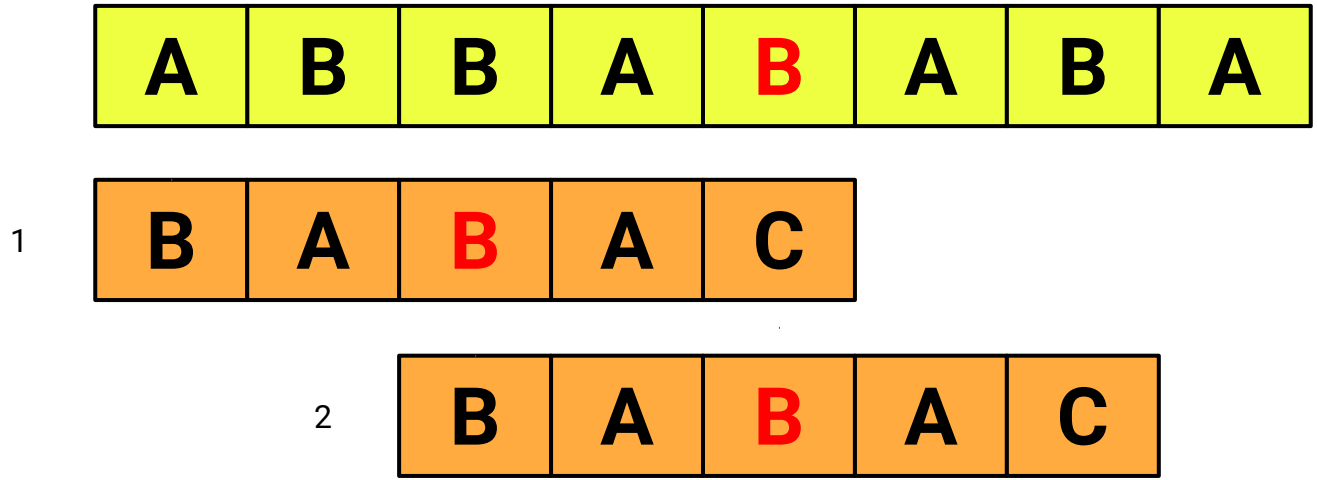
Boyer Moore - Bad Character Rule



Boyer Moore - Bad Character Rule

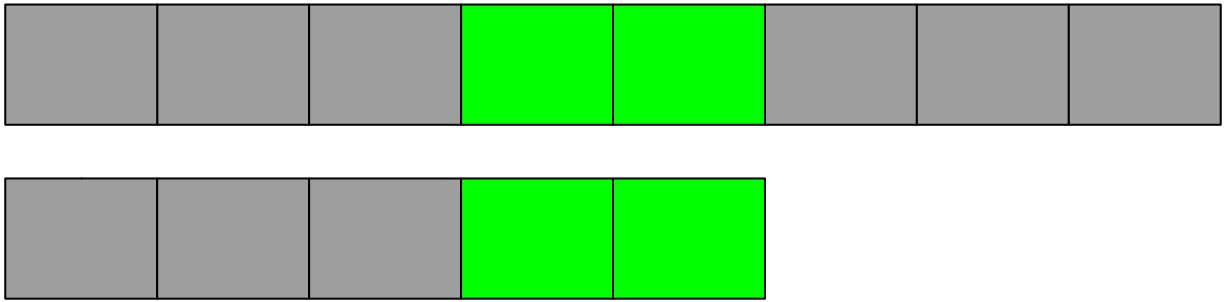


Boyer Moore - Bad Character Rule

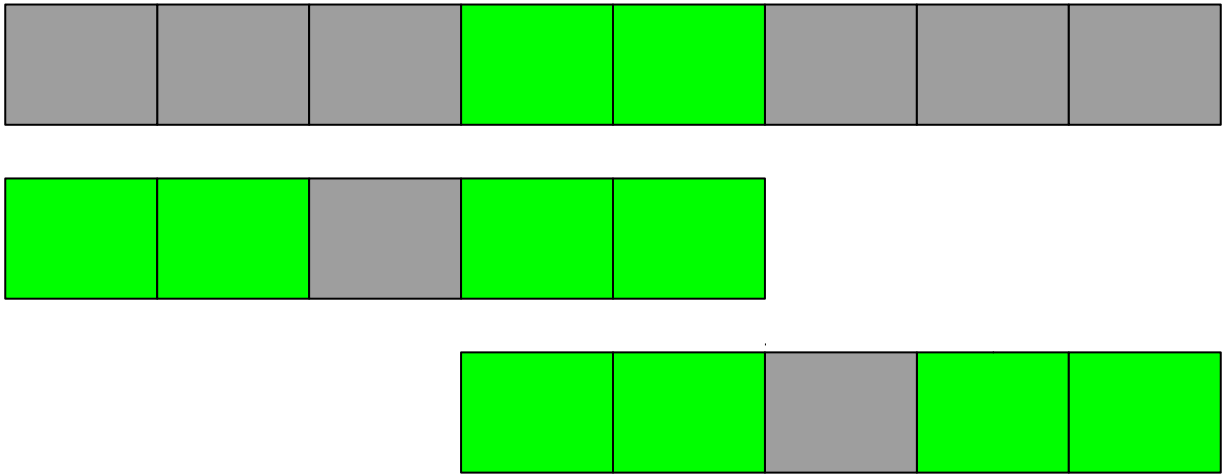


Good Suffix Rule

Boyer Moore - Good Suffix Rule

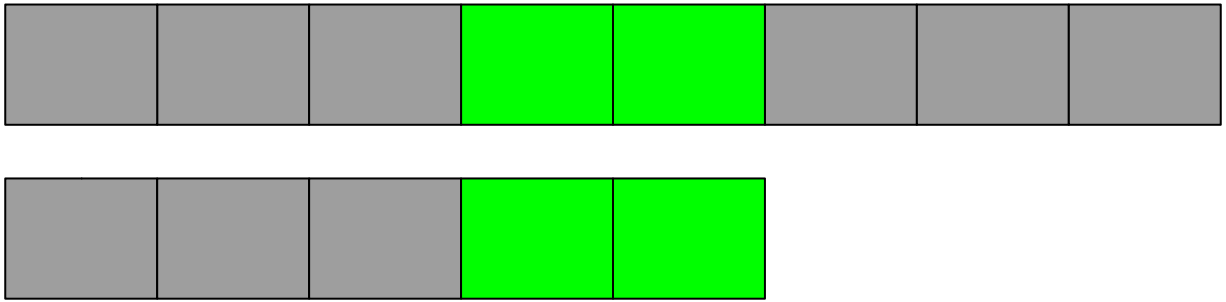


Boyer Moore - Good Suffix Rule

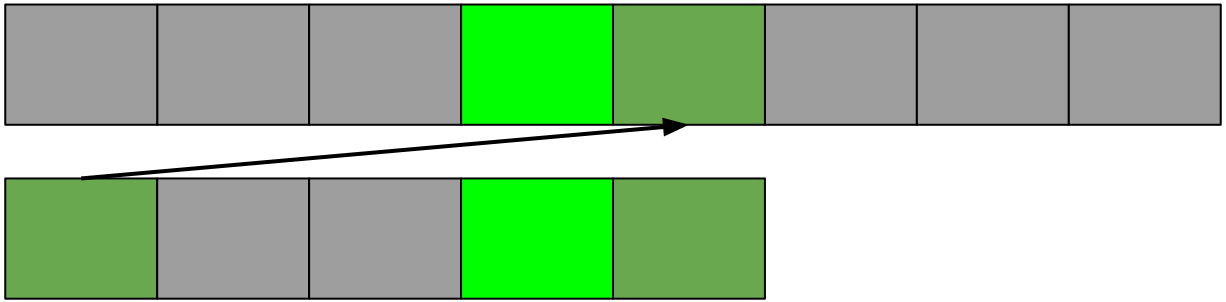


Partial Good Suffix Rule

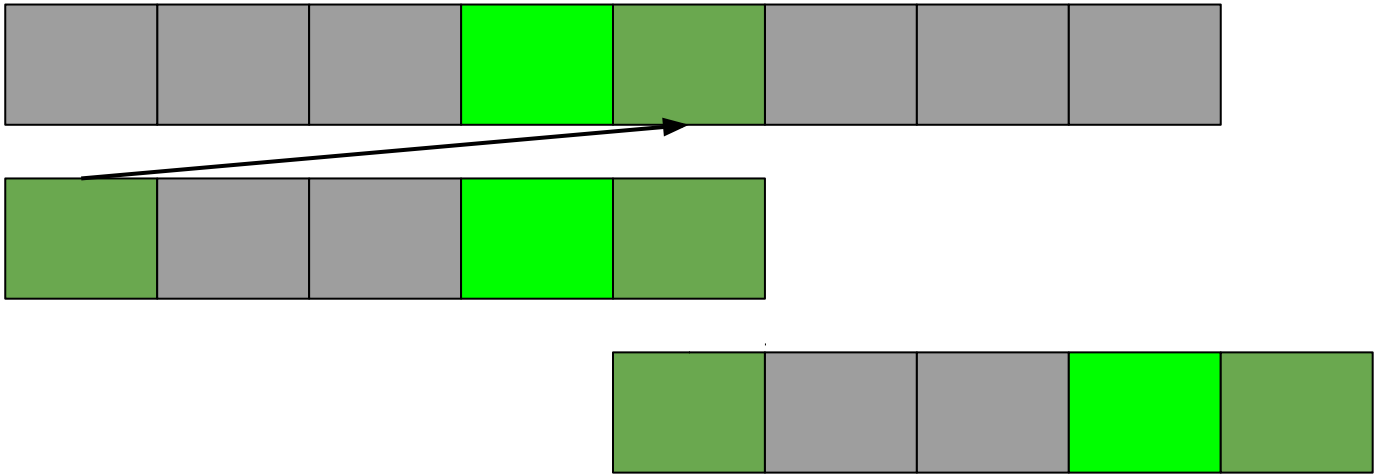
Boyer Moore - Good Suffix Rule + Partial



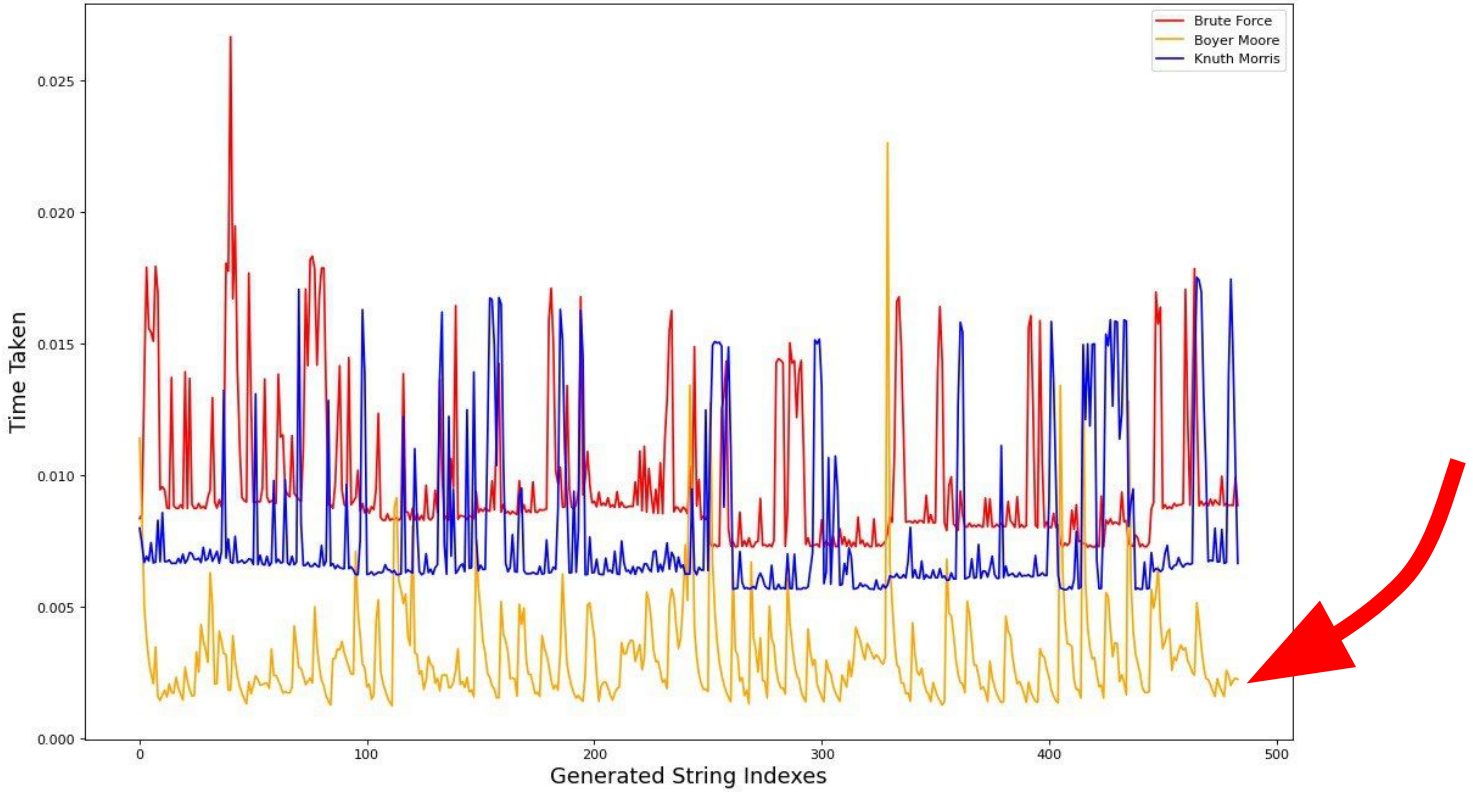
Boyer Moore - Good Suffix Rule + Partial



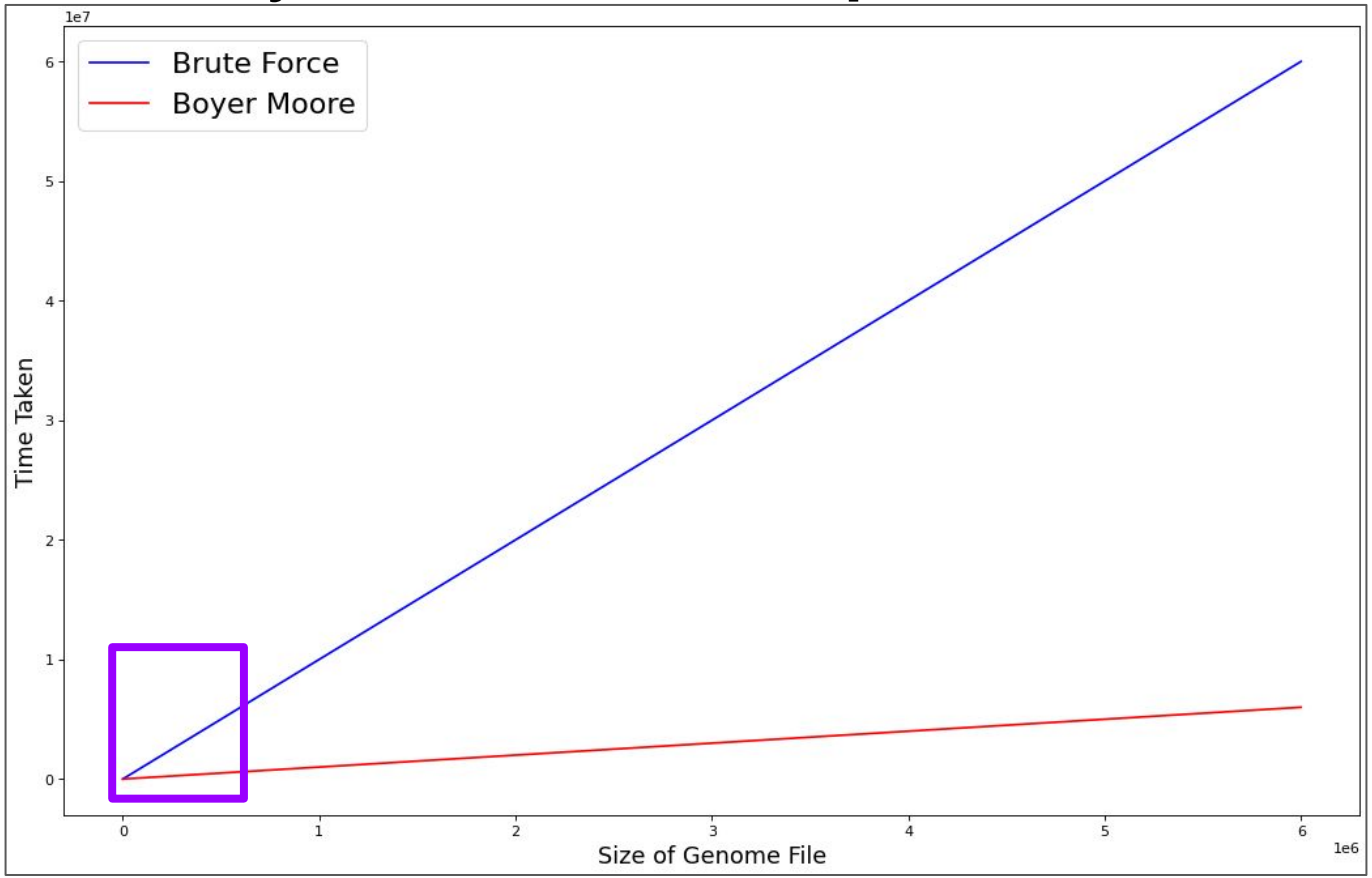
Boyer Moore - Good Suffix Rule + Partial



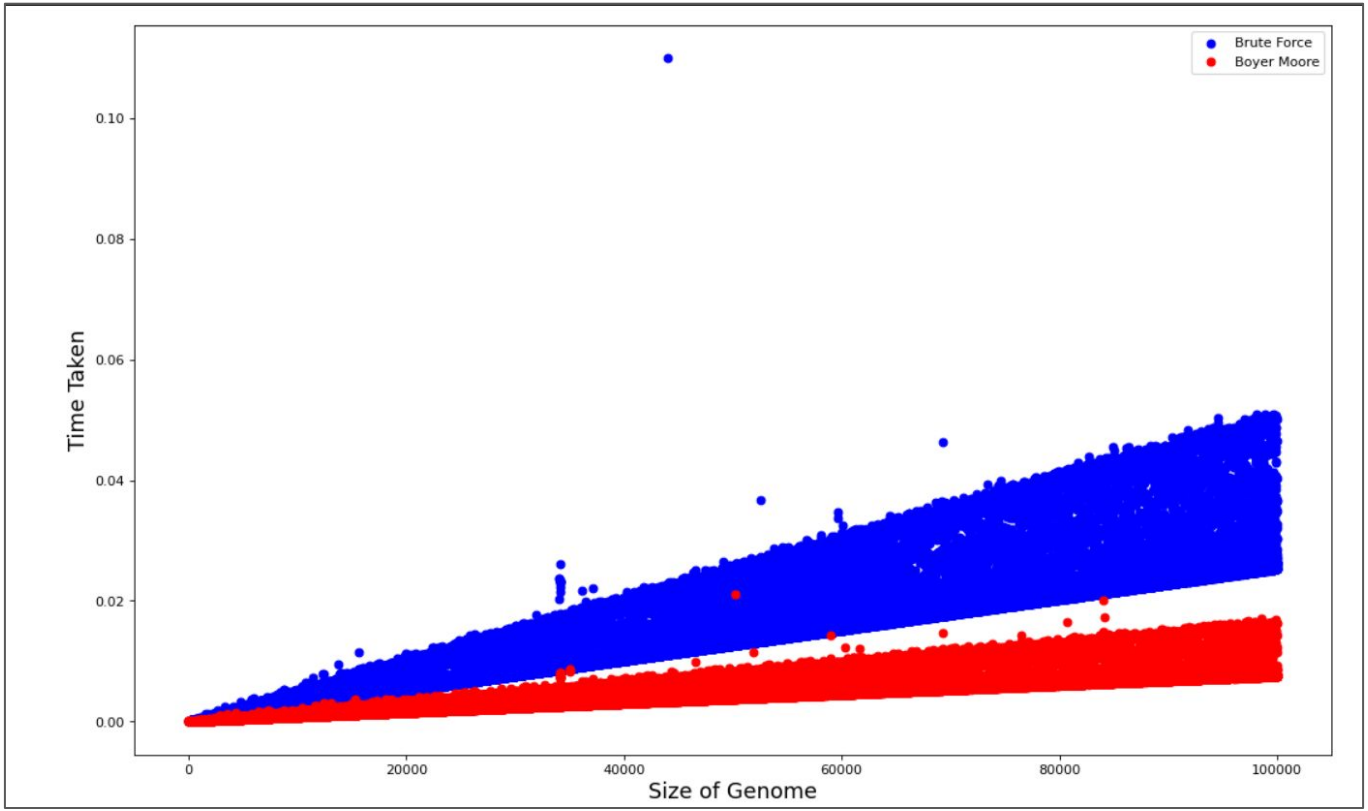
Results of Analysis - As A Whole



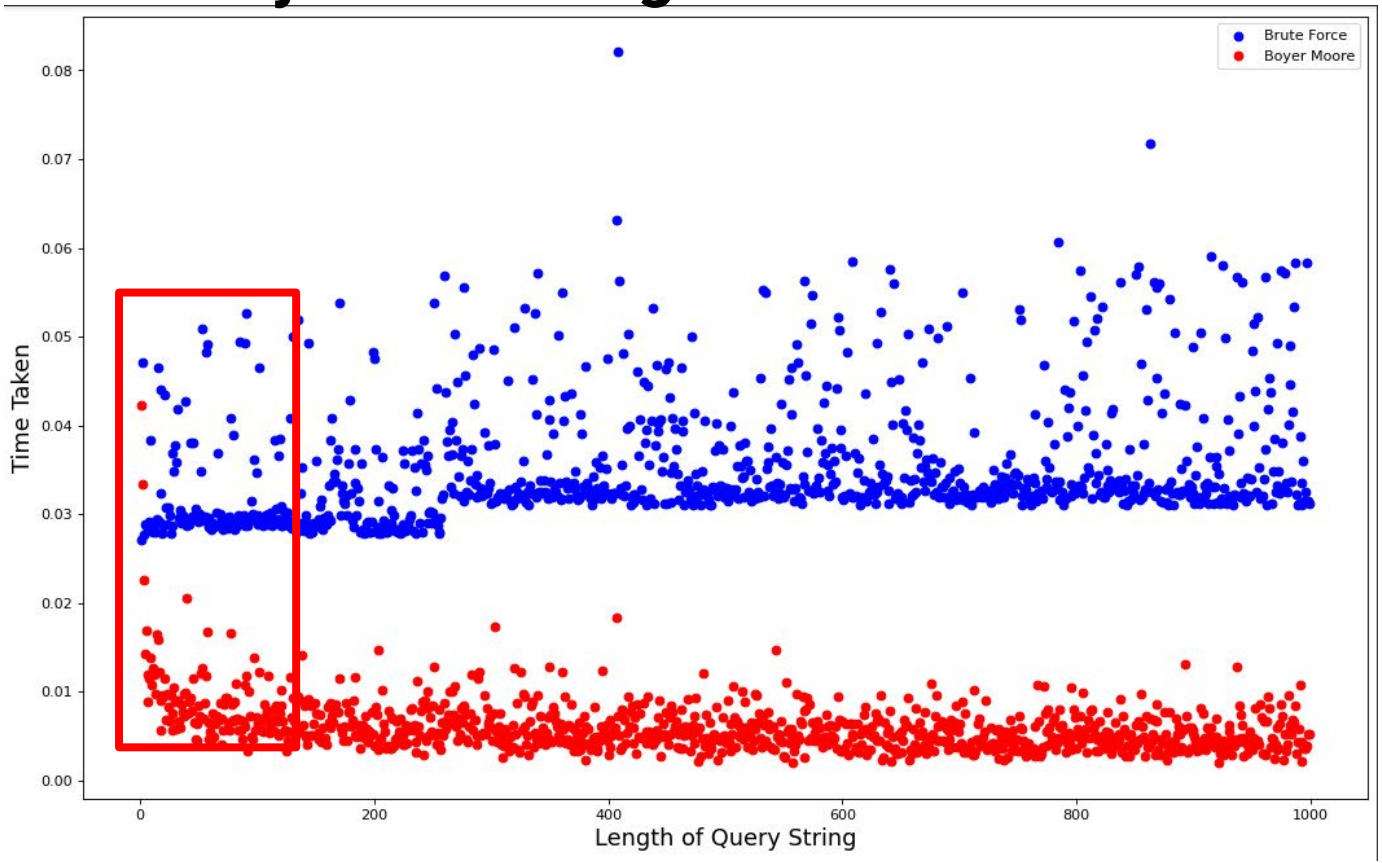
Results of Analysis - Theoretical Speed



Results of Analysis - How Time Taken Affected as Size



Results of Analysis - Testing



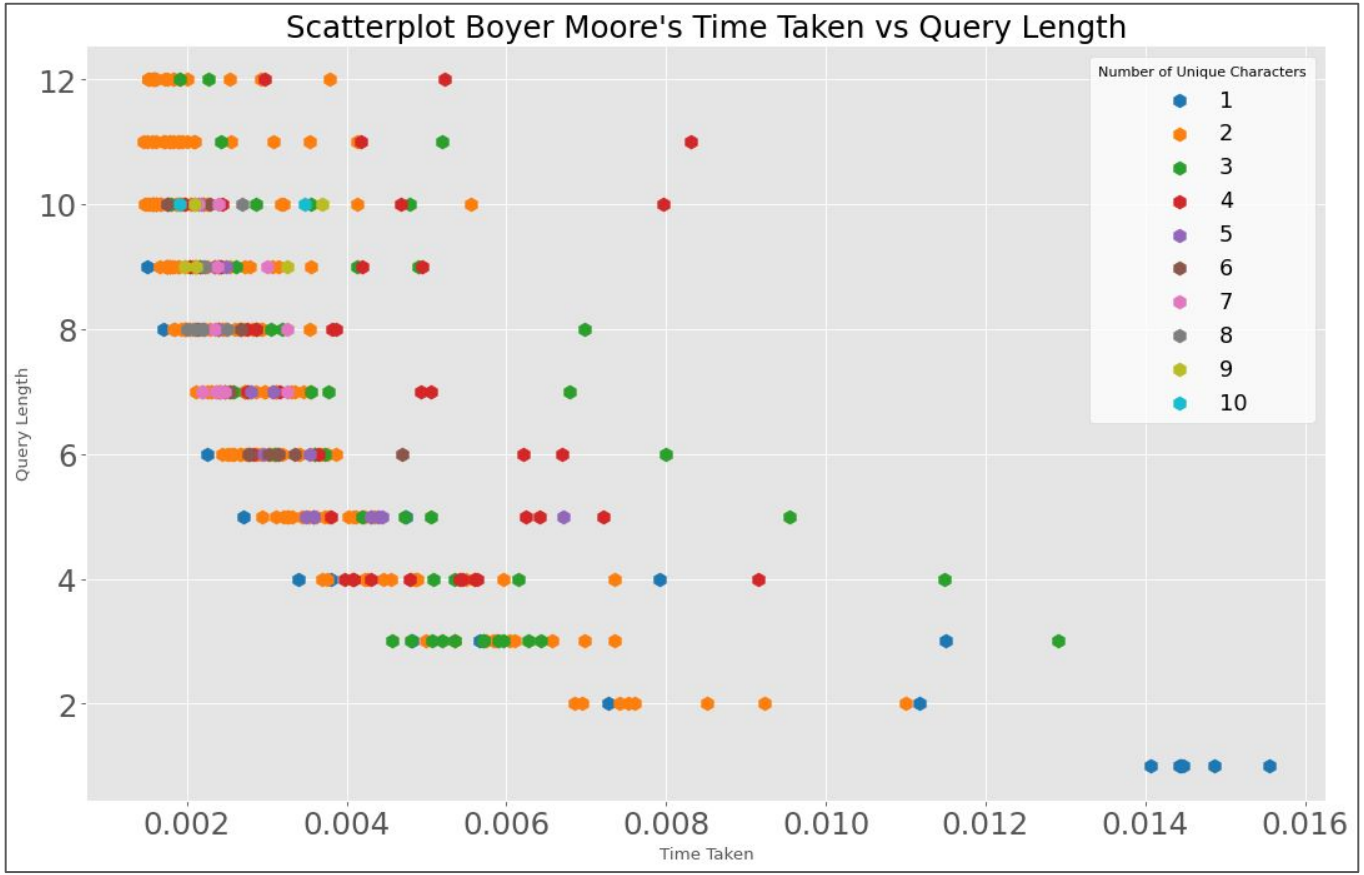
Results of Analysis - Testing

Query Size	BF Time	BM Time
1	0.02707	0.04225

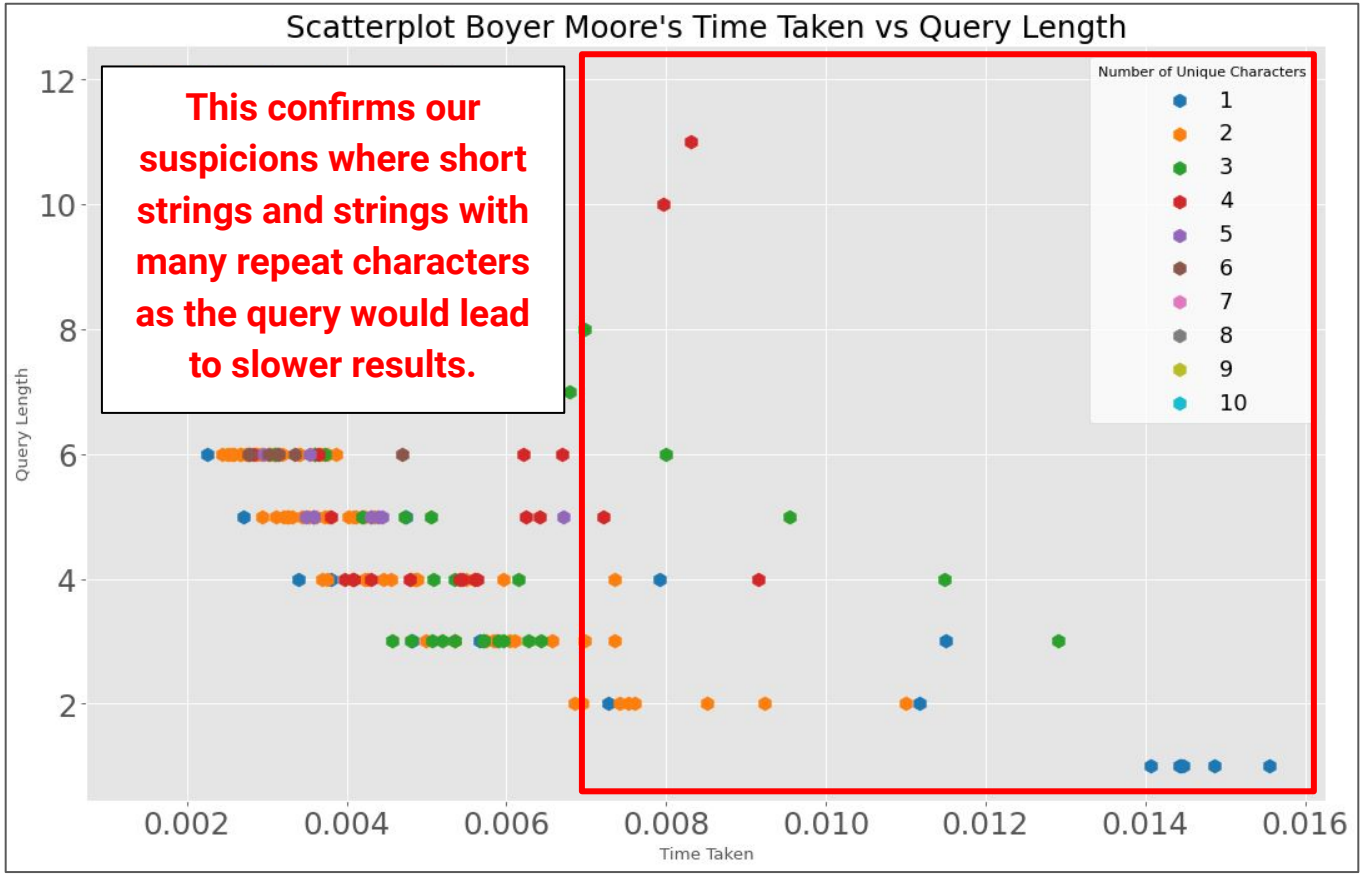
Which is intuitive as longer query length means bigger skips!

As lookup tables are only generated for the query string.

Results of Analysis - Deeper Dive into Boyer Moore



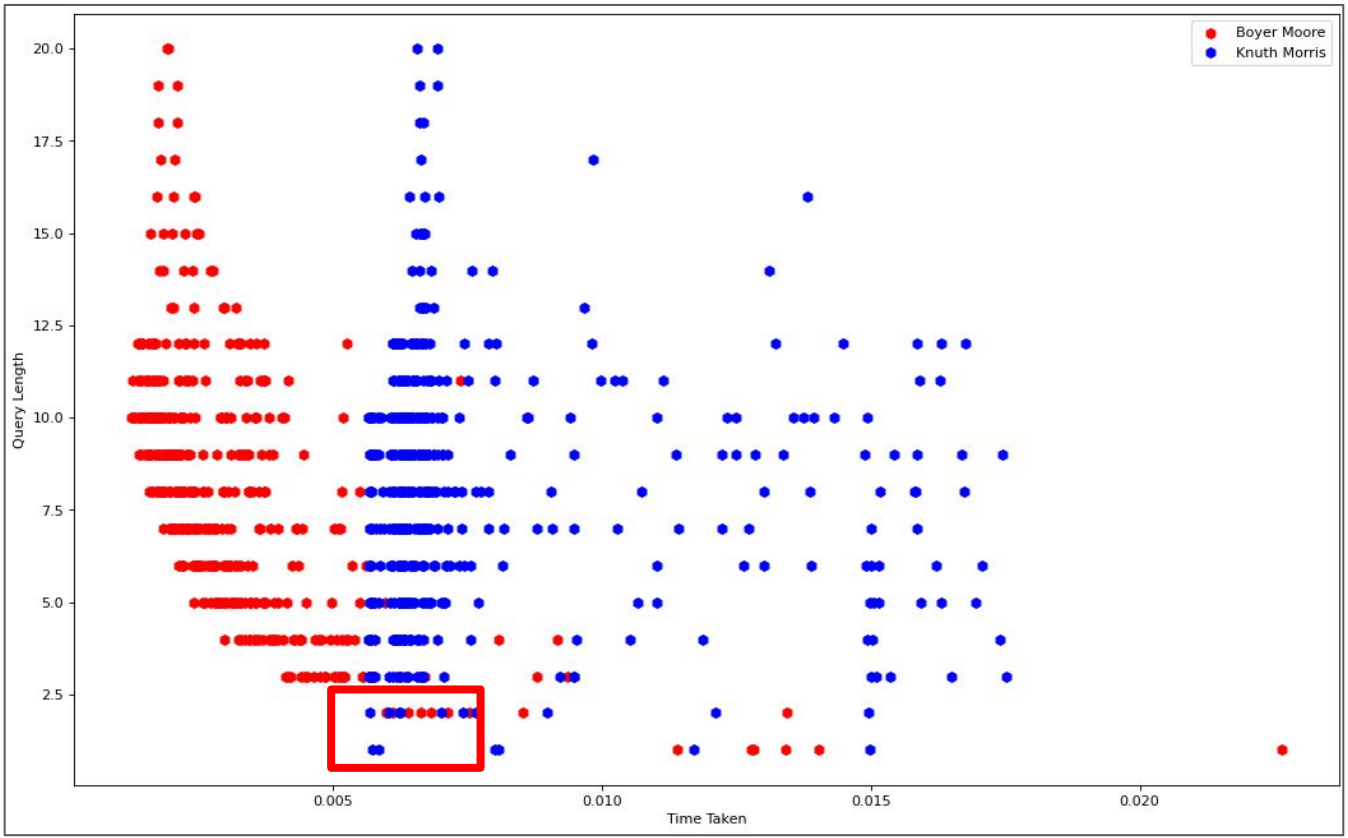
Results of Analysis - Deeper Dive into Boyer Moore



Results of Analysis - Deep Dive into Boyer Moore

Query	Query Length	Time Taken	Search Successful	Position Placed	Unique
E	1	0.02260	True	272	1
C	1	0.01996	True	1053	1
B	1	0.01849	True	4913	1
A	1	0.01844	True	642	1
BBBAAAA	7	0.01770	True	2997	2
J	1	0.01497	True	426	1
D	1	0.01430	True	3845	1
CB	2	0.01368	True	1720	2
TTT	3	0.01147	True	4628	1

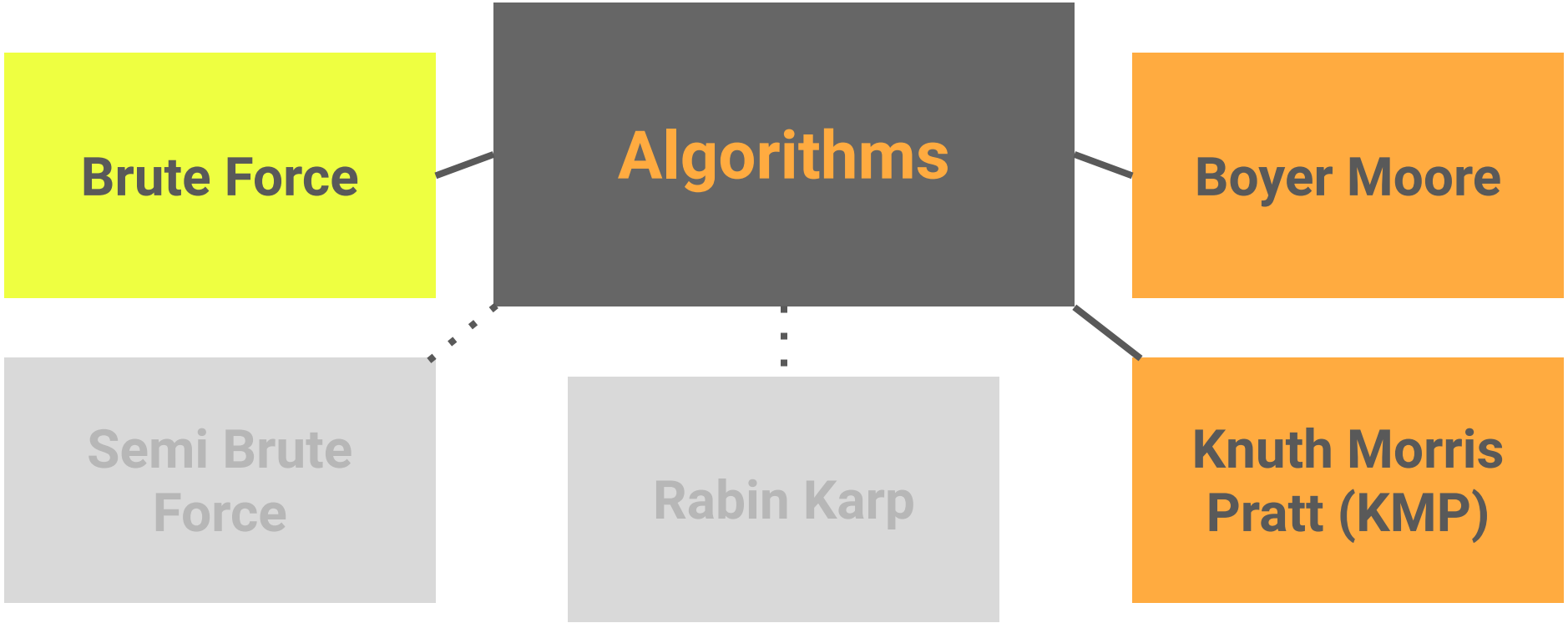
Conclusion - KMP Complementing BM



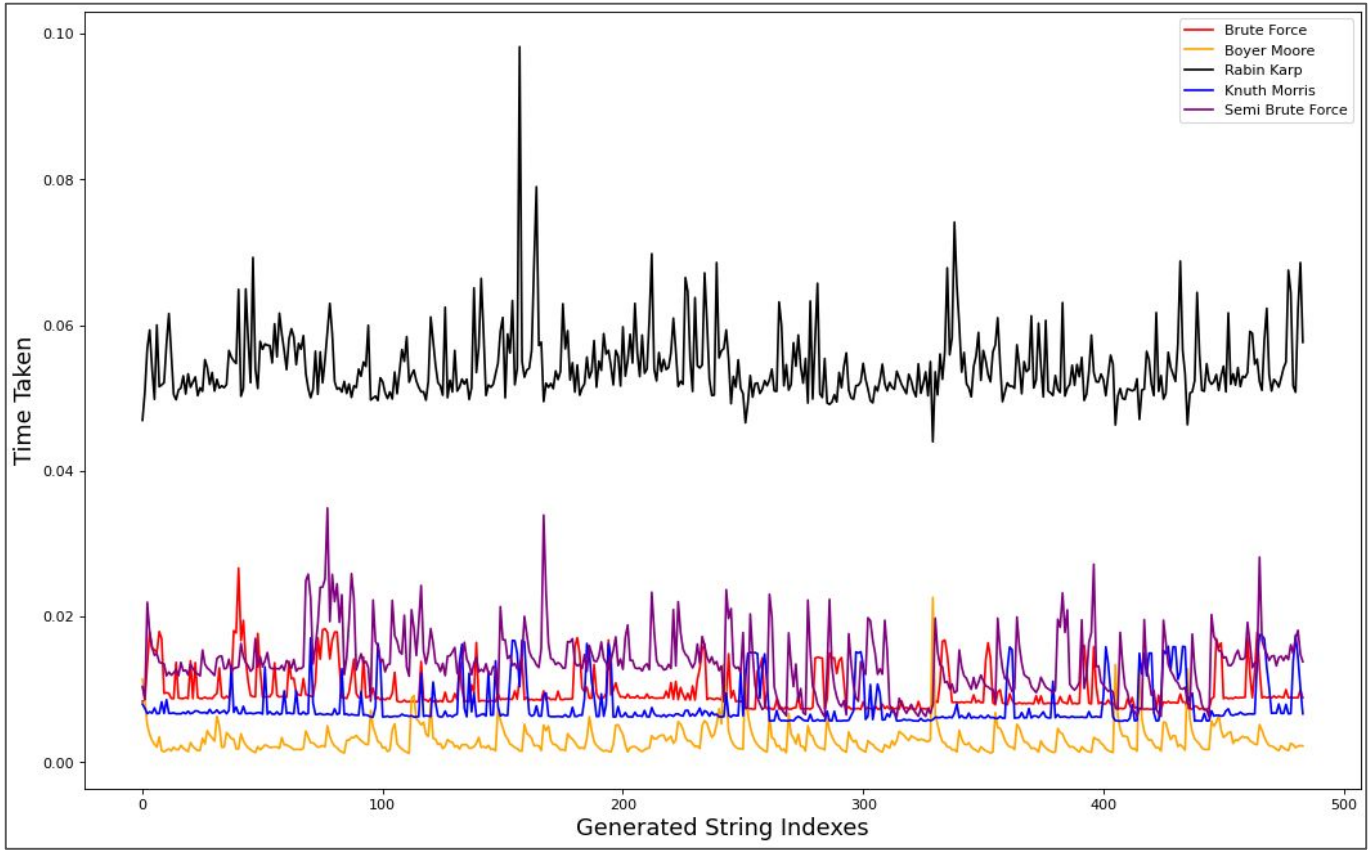
Conclusion - KMP Complementing BM

- Generally, BM performs the best out of all 3
- When string lengths get long, BM performs better, while KMP’s performance is similar to brute force.
- When string lengths gets shorter, especially when searching for one char, KMP performs better than brute force and BM, since the inner loop is not activated.

Implementing Algorithms



Results of Analysis - As A Whole



Modern Solutions to Solving Problems

Just like the search for our query string in a genome sequence,

Looking for a needle in a haystack, can be a nearby impossible task…

Why not just burn it all down?