CMP201 ASSIGNMENT – STRING SEARCHING

CAMERON THUSTAIN 1900594

THE PROBLEM

- The problem I am looking into is how games that have been released in the West (that were originally released in Asia) have poor translation and/or many differences in scripts etc. one main example of a very poorly translated game is Final Fantasy 7. For my assignment I compare two ‘identical’ game scripts from Final Fantasy 7, the original English script and the Japanese script (which has been put through google translate). From there I look for specific patterns (strings) to see if there is differences in the amount of times this pattern appears in the two and if so how this affects the time, CPU usage etc.

THE ALGORITHMS

- The two string searching algorithms I use are the Boyer Moore algorithm and The Rabin Karp algorithm. I chose these algorithms because firstly, the Rabin Karp algorithm is good when searching through large pieces of text like a game script where Boyer Moore although it isn't as effective as searching through a large piece of text its good for searching through text with a large amount of vocabulary which most game scripts have. I chose to use the Rabin Karp algorithm on the Japanese script, because its been put through google translate extra words may have been added or some sentences were maybe shortened and I wanted to see how that would effect the time complexity. Where as the Boyer Moore is used on the English script its time complexity shouldn't be affected as much.

THE DATA STRUCTURES

- Within my program I chose to use sets to store how many times the pattern was found using the algorithms. I chose sets because each value stored within the sets has to be unique and I wanted these sets to store the amount of times the pattern was found/counted. So because of this I could tell my algorithms were working correctly. If there was anything wrong with my sets I would know there is problems somewhere. Secondly I only wanted a key (the amount of times the pattern was found) which is why I never used something like a map, each element would have a key value and a mapped value.

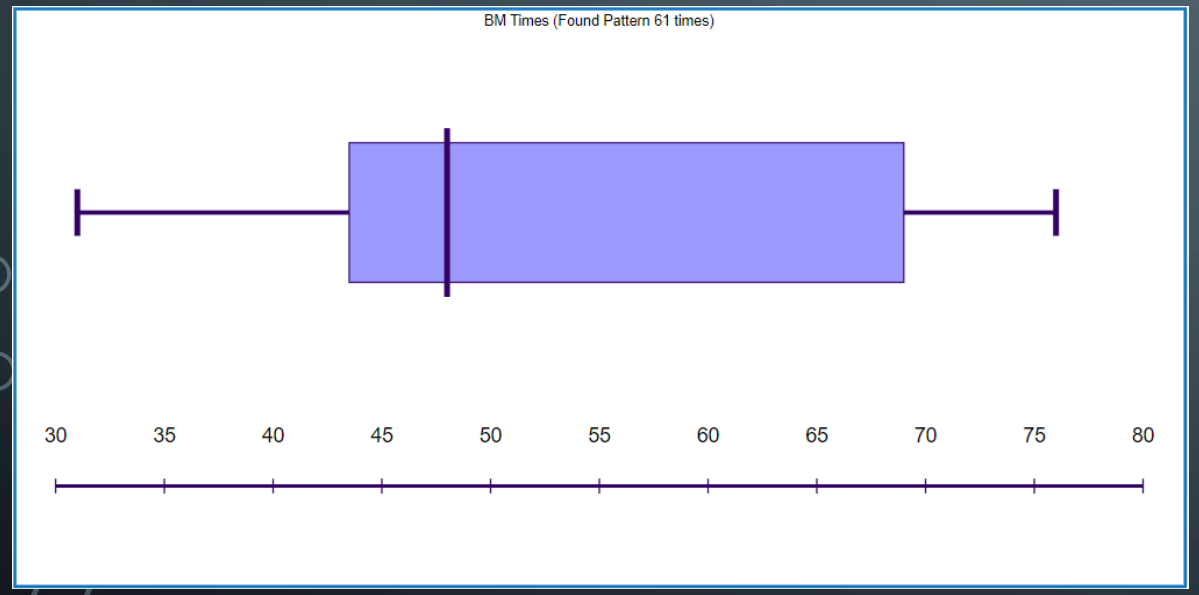
THE DATA STRUCTURES

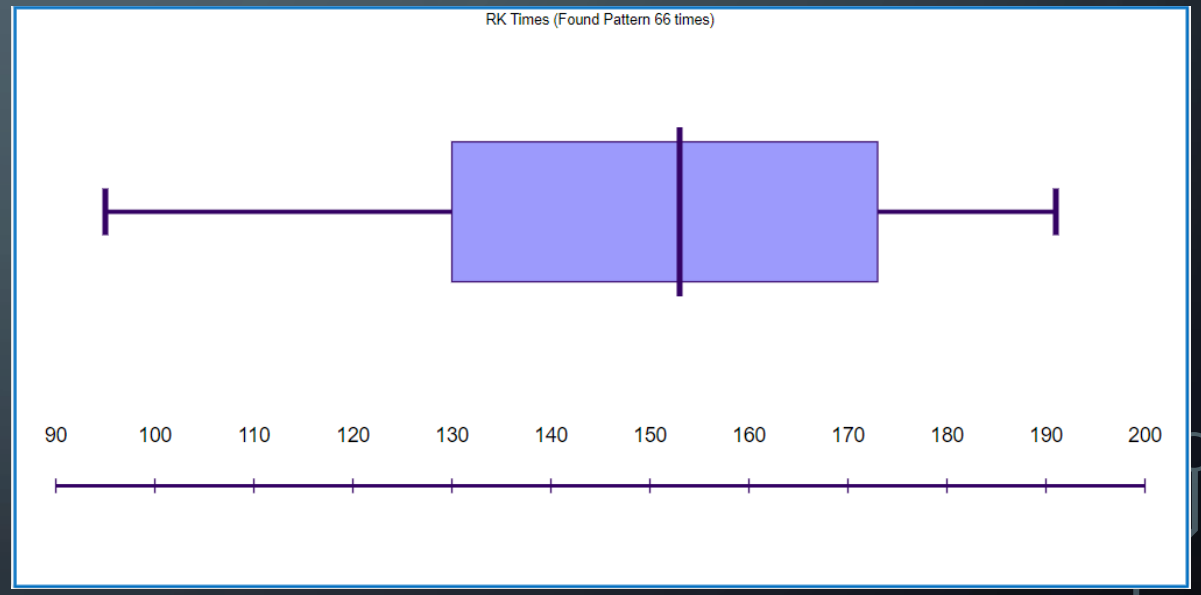
Why I chose Sets over Unordered Sets?

- Although unordered has an average time complexity of O(1) I chose sets because I wanted my data/values to be ordered, since the set is ordered its better for searching. A set has a time complexity of O(log n) which although slower than unordered, I mainly wanted the output of my set to be in order.
- Furthermore using the Boyer-Moore algorithm with a set would increase the time complexity as with Rabin Karp as well.

FINDINGS

Pattern – “ are ” (notice the spaces this is because without them the algorithms would pick up words like “care” or “careful” for example and that would affect our results





Average (MEAN) – 53.5ms

Average (MEAN) – 149.75ms

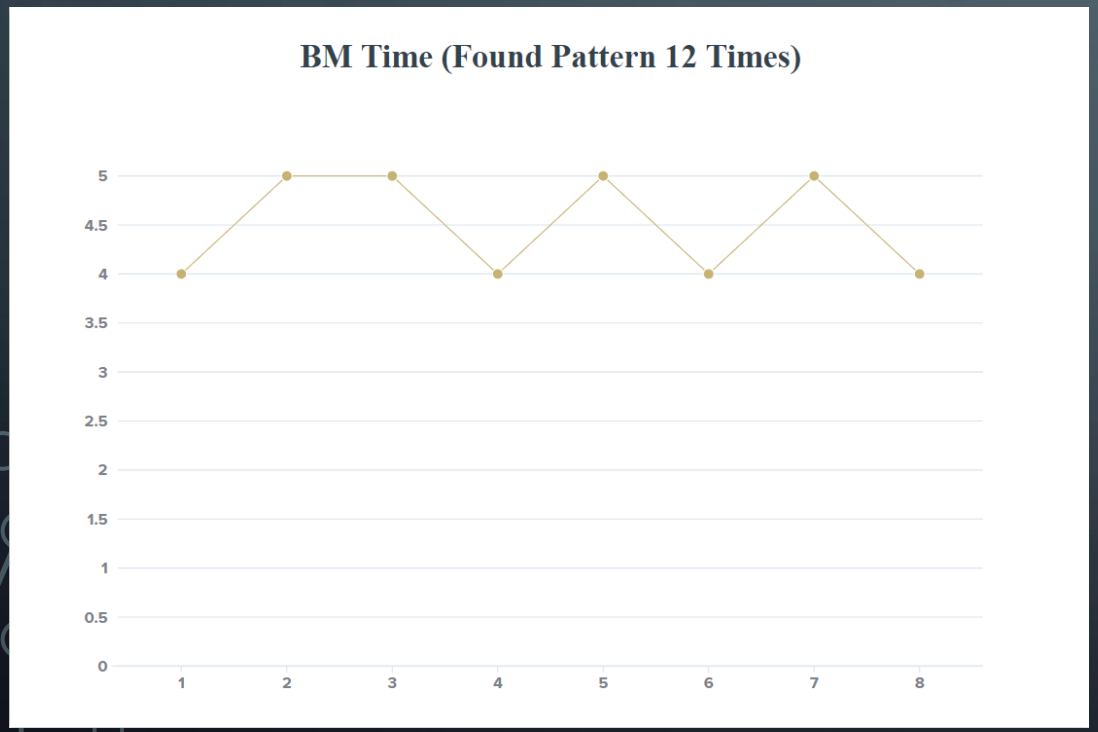
PERFORMANCE PROFILER

- When using the performance profiler to find CPU usage in Debug mode I discovered that the Boyer-Moore algorithm used 10ms of CPU time where the Rabin Karp used 57ms of CPU time.
- My thoughts on why the Rabin Karp algorithm took longer is because the pattern was found significantly more times in that text rather than the Boyer-Moore algorithm but also due to the Boyer-Moore algorithm being quicker at searching through bigger pieces of text.

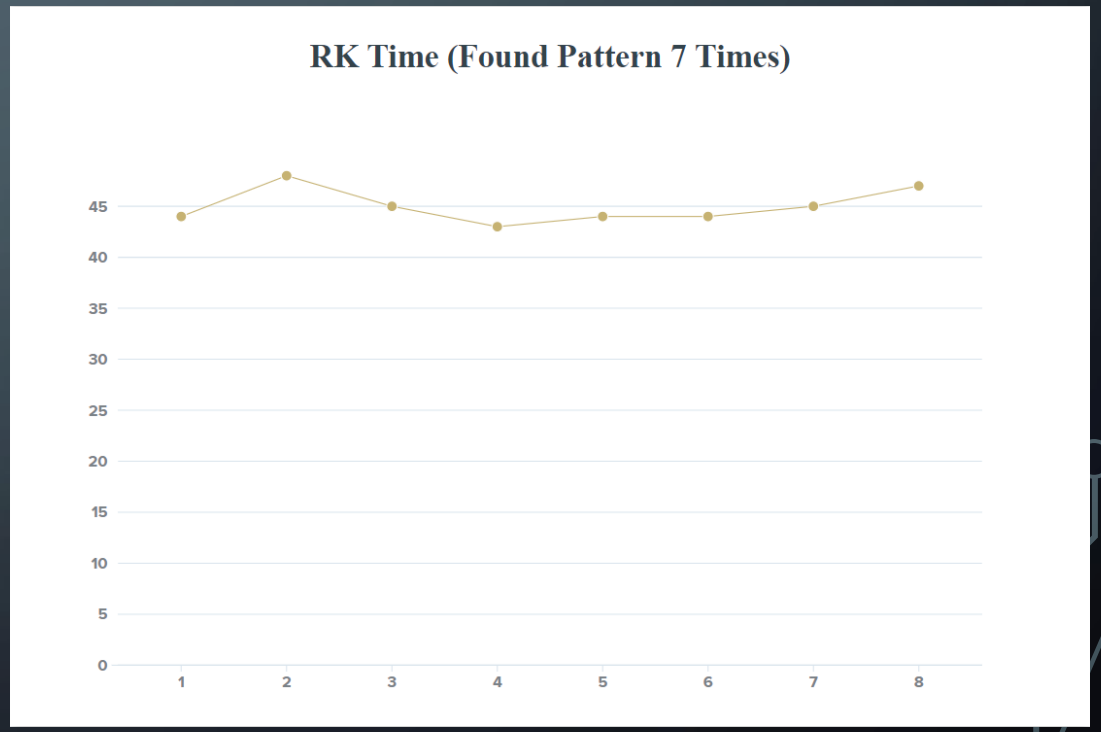
FINDINGS

- I checked another pattern which is longer but this time I tested its times in both debug and release mode this is what I found out…

Pattern – “wrong” (no spaces) DEBUG MODE



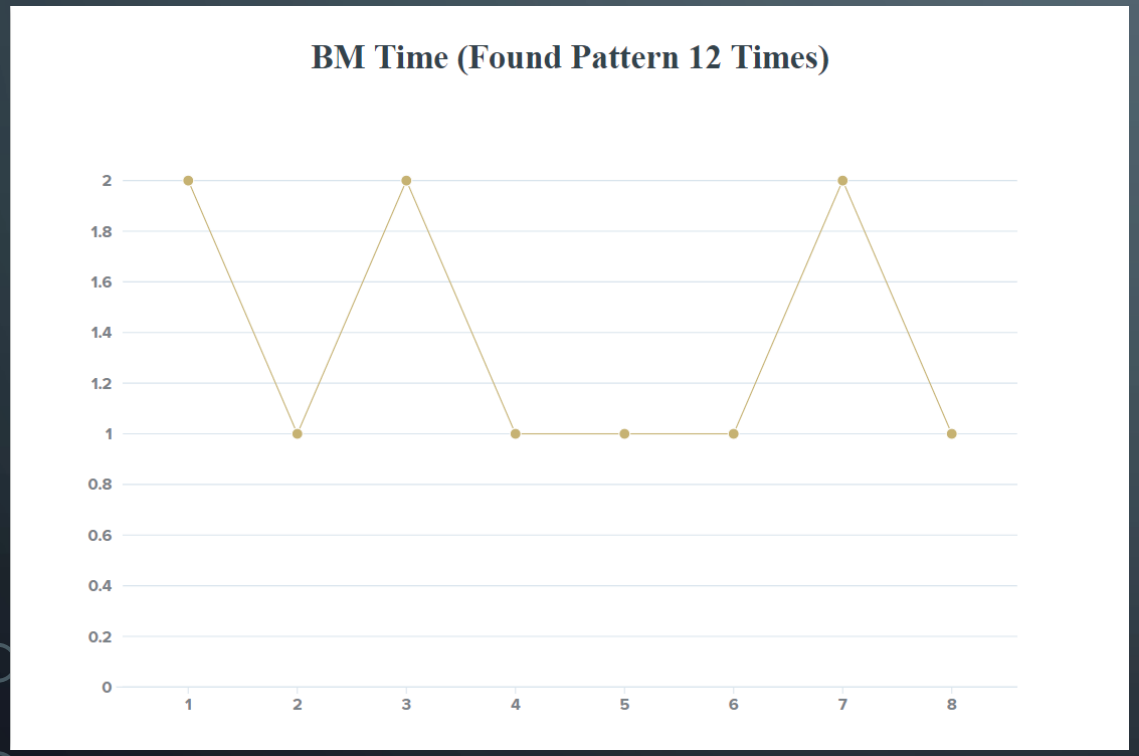
Average (MEAN) – 4.5ms



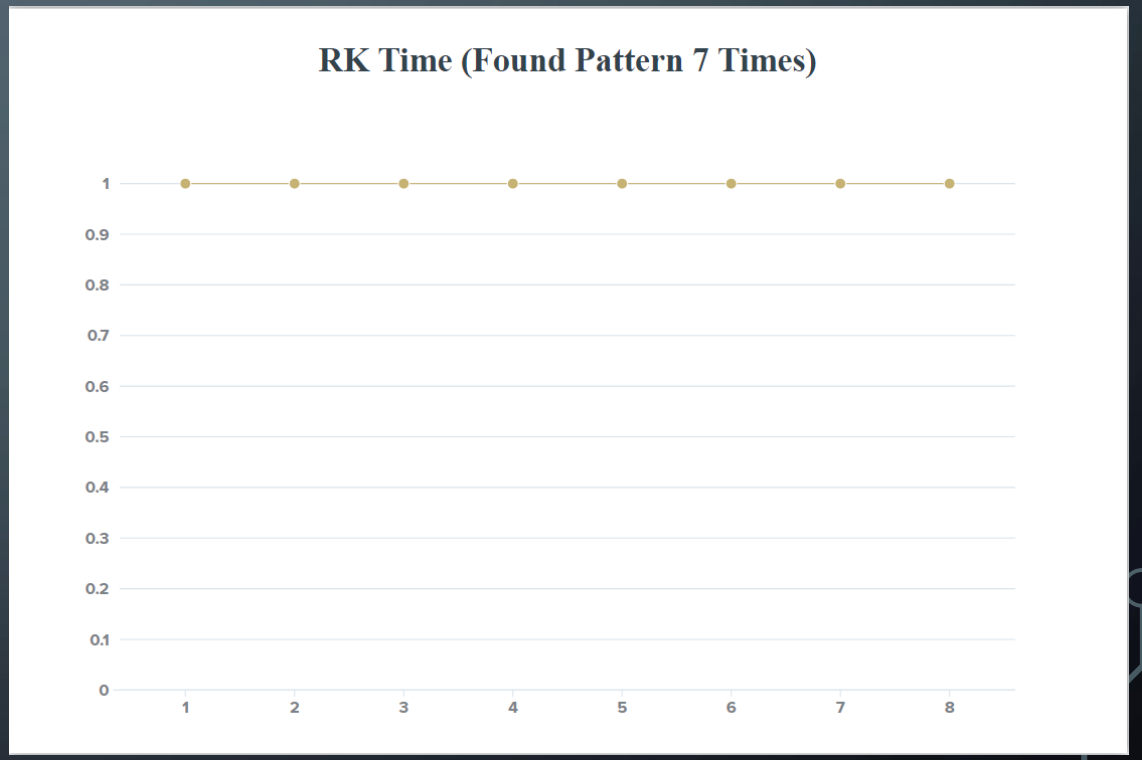
Average (MEAN) – 45ms

FINDINGS

Same Pattern – "wrong" RELEASE MODE



Average (MEAN) – 1.375ms



Average (MEAN) – 1ms

FINDINGS

- This time there is some major differences a few things I noticed were the pattern appeared less in the text the Rabin Karp was searching through but it took longer than the other which had the pattern appear more using the Boyer-Moore.
- Another thing I noticed was the average times. The average time of the Boyer-Moore algorithm was 10 times faster than that of the Rabin Karp algorithm (4.5ms & 45ms).
- However when testing In release mode the Rabin Karp algorithm was quicker (although not significantly) but it had a much more consistent time (an average of 1ms).
- When using the performance profiler it was tested in release mode both algorithms used 1ms of CPU usage.

DOES THE COUNT CHANGE ANYTHING?

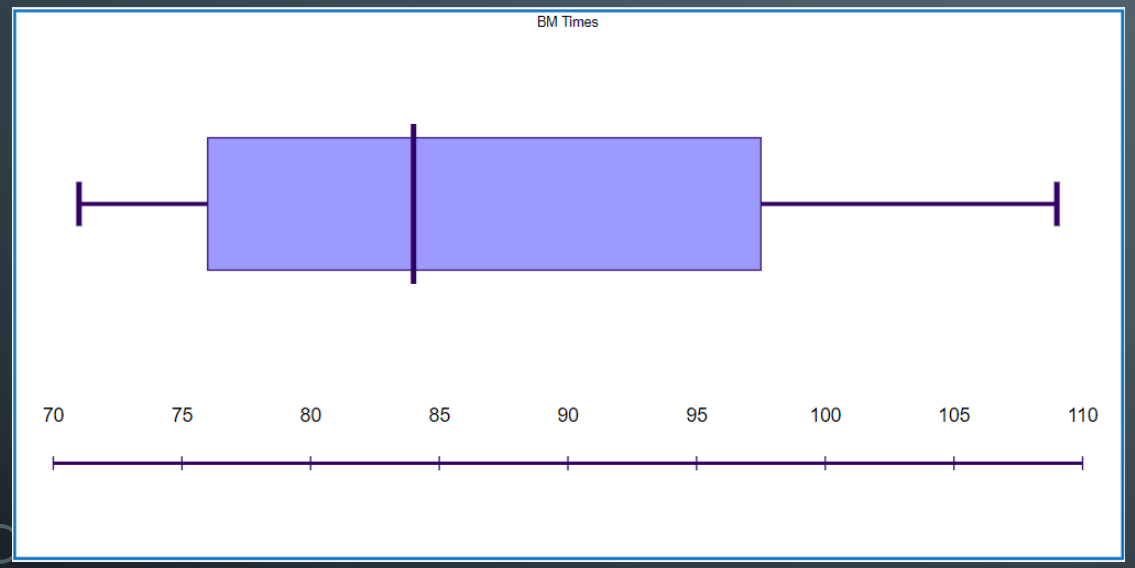
- I Wanted to test if it was the length of the text that was affecting the Rabin Karp (because one is slightly longer than the other). So I tested both algorithms find the same pattern however this time they searched the same text.

Pattern – '!'

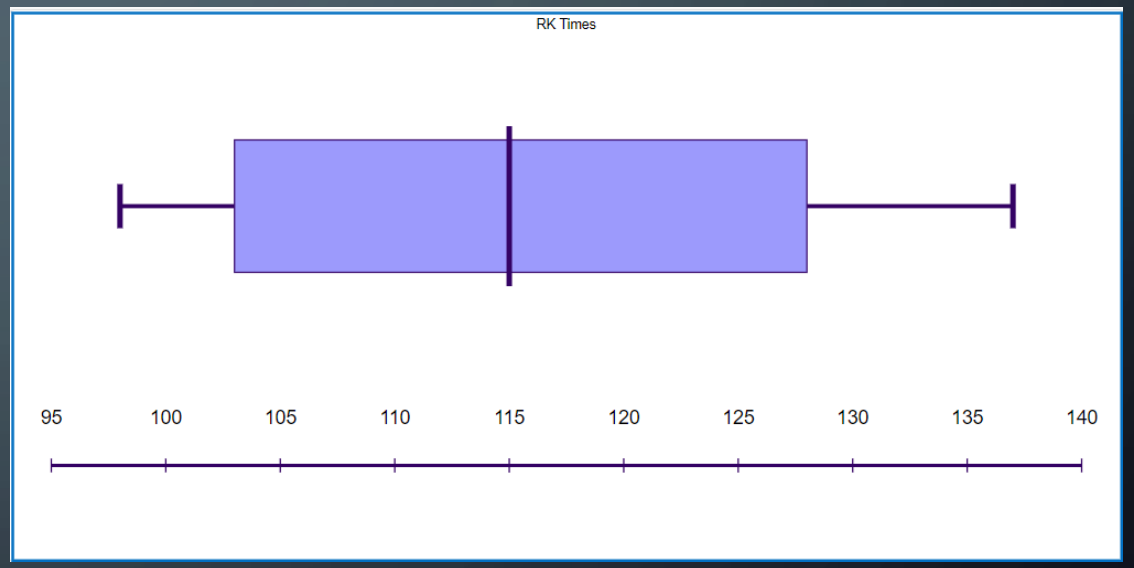
Text – text, The English text which is slightly shorter than the Japanese script as some words in the translated Japanese script are translated into two words. For example C'mon in the English script translates to Let's go in the Japanese Translated script

FINDINGS

- Both algorithms found the pattern 91 times



Average (MEAN) – 86.875ms



Average (MEAN) – 114.5ms

FINDINGS

- In this test the Boyer-Moore again is significantly quicker than the Rabin Karp. However, in one test the Rabin Karp was quicker than the Boyer-Moore by quite a moderate amount (7th test 109ms for BM compared to 98ms for RK).
- Rabin Karp should have an expected time complexity of O(n). However due to hashing, in this case its hashed by length of the pattern its got more of a time complexity of O(n + m), where as Boyer-Moore has a case of O(nm). Which in theory the Rabin Karp algorithm has a quicker time complexity, same amount of operations however the value of O(n+m) will never be greater than the value of O(nm).

CONCLUSION

- To conclude in many of the tests the Boyer-Moore algorithm is significantly faster than the Rabin Karp algorithm. The algorithms were tested to find a pattern when: it appeared more in the text the Boyer-Moore algorithm was searching, it appeared more in the text the Rabin Karp algorithm was searching and using one of the texts some the pattern would appear the same amount of times in both algorithms. Every time but one the Boyer-Moore was significantly faster, although the Rabin Karp algorithm theoretically has a faster time complexity than Boyer-Moor the Boyer-Moore is built better for game scripts, the Rabin Karp algorithm is good for large pieces of text however it would outmatch the Boyer-Moore algorithm in something like detecting plagiarism for example.

CONCLUSION

- All in all this assignment was a very interesting experience. Changing the load_file function to ignore special characters, testing, recording, analysing the results of two algorithms and using appropriate data structures to store said results shows an understanding of the modules content which fulfils the learning outcome within the specification. Having more time with this project I would have liked to have added rules within the program for example only searching for names of the characters within the text or searching for the beginning of a sentence.

REFERENCES

- Sampson, A., 2020. *Utils (Header And Cpp)*. String Searching Lab 7.
- Stack Overflow. 2020. *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Available at: <https://stackoverflow.com/> [Accessed 30 December 2020].
- Tutorialride.com. 2020. *Append To A File - C++ Program*. [online] Available at: <https://www.tutorialride.com/cpp-file-management/append-to-a-file-c-program.htm> [Accessed 30 December 2020].
- Yinza.com. 2021. *Yinza.Com - Final Fantasy VII Complete Script*. [online] Available at: <http://www.yinza.com/Fandom/Script.html> [Accessed 1 January 2021].