**Speed Test Report**

**Introduction**

This report aims to evaluate and compare the speeds of three different algorithms intended to be used in a TCP server to search for a string value inputted by a client server amongst a list of acceptable string values in a text file.

**Algorithm A**

The approach used in this algorithm was to search for the string value in the file line-by-line and return the response STRING EXSISTS if found, and STRING DOES NOT EXSISTS if otherwise.

**Code Snippet:**

def search\_in\_file(file\_name, search\_value):

with open(file\_name, 'r') as file:

for line in file:

if search\_value in line:

return line

return False

**Algorithm B**

The approach used in this algorithm was to put all the accepted string values into chunks (groups) and then search for the string value from the client-server in these chunks.

**Code Snippet:**

def search\_in\_large\_file(file\_name, search\_value):

with open(file\_name, "r") as f:

with mmap.mmap(f.fileno(), 0, access=mmap.ACCESS\_READ) as mm:

if mm.find(search\_value.encode()) != -1:

return True

return False

**Algorithm C**

The approach used in this algorithm was to load parts of the file that were needed for the search. The algorithm used a python module: mmap, that makes it possible to access the contents of a file without reading the entire content into memory.

**Code Snippet:**

def search\_in\_large\_file(file\_name, search\_value):

with open(file\_name, "r") as f:

while True:

chunk = f.read(1024) # Read 1024 bytes at a time

if not chunk:

break

if search\_value in chunk:

return True

return False

**Methodology**

To test the performance of the algorithms, we used the file provided in the essential task: 200k.txt in our TCP server. The total number of rows in the file was reduced to 250,000, as stated in the specifications, to test how efficiently it will work when deployed into full production. For each algorithm test, we executed ten search queries each and determined the overall time used in execution by adding a timer in our codes and had the total time logged out into our terminal. The average speed measured in milliseconds was calculated for each algorithm in milliseconds.

**Results**

The table below shows the average time each algorithm is used to execute a search query task when the REREAD\_ON\_QUERY is TRUE

|  |  |
| --- | --- |
| Algorithm | Execution Time (ms) |
| A | 9.54 |
| B | 5.19 |
| C | 1.67 |

The table below shows the average time each algorithm is used to execute a search query task when the REREAD\_ON\_QUERY is FALSE

|  |  |
| --- | --- |
| Algorithm | Execution Time (ms) |
| A | 5.64 |
| B | 1.19 |
| C | 0.39 |

**Conclusion**

In conclusion, the results of our testing indicate that Algorithm C is the most efficient of the three algorithms tested, with Algorithm B coming in second place and Algorithm A being the slowest. While Algorithm B may have some advantages in certain situations, such as ease of implementation, Algorithm C was chosen as the preferred choice when speed is the primary concern. Also, during testing, I concluded that my program efficiently works with files having a maximum of 500,000 words.