SUMMARY DOCUMENT

- The goal of this project is to be able to convert huge JSON based log files into columnar files.
- The language used here is JS and the engine that runs the JS code is Node.js.
- As we start through the project we have to read through a huge JSON log file and to do that reading it normally will need as RAM equal to the file size which is not efficient.
- So we use Streams, they are objects that let you read data from a source or write data to a destination in continuous fashion.
- Starting with a read only stream we read through the JSON file.
- The given JSON file is in ndjson format, also called Newline delimited JSON, where basically you're going to write a JSON object per LINE.
- To loop through the data one line at a time we use a command called split, which is used to split the whole data into multiple single lines.
- While looping through a line which is a JSON object we loop through each object, property, nested object inside the JSON object.
- Making a write stream to store object according to their name into columnar files while looping through all objects was implemented, but the time consumption was too much to open and close a write-able stream every time.
- So to reduce the time consumption or to make it efficient a Map was created to store the write able stream every time a new object is looped through, and if a similar object is looped through again then the same mapped write stream is used to write data to a columnar file.
- It achieves the required column file format mentioned in the Problem Statement.

- The Bottlenecks in this tool is the usage of a large JSON file with more than 1024 different column files, as the Windows OS is limited to 1024 by default and keeping more than 1024 write streams streams open throws an error. Though using the first implementation where every time we loop through an object a write stream is opened and after the data is written the stream is closed, there will be no bottleneck as mentioned above but the time consumption was simply way too much to be used as a tool.

- To improve the performance a logical implementation can be used where mapping is still used but the open file limit can be maintained by the logic such as random stream close or the last 10 streams to be closed as we reach the limit.

ACTUAL TIME & SPACE COMPLEXITY

- Looping through every object in every line of the JSON file will give us a time complexity of O(n*m)
  where n = number of lines in the JSON file and
  m = maximum number of objects compared to all lines in a single line.

- Whereas the mapping we used is responsible for the space complexity which is given as O(n*k) because the mapping stores all the objects in every line of the JSON file,
  Where n = number of lines in the JSON file and
  k = maximum of number of column objects compared to all lines in a single line.

Table demonstrating the actual time & space complexity, time taken for conversion & max. RAM usage for various file sizes

Actual Time Complexity	Actual Space Complexity
O(n*m)	O(n*k)

Files	time taken for conversion	max. RAM usage
128MB.log	13.2s	48MB
256MB.log	26s	48MB
512MB.log	50s	48MB
dummy.log	28.323ms	1MB