● **Algorithm & Data Structures & Design:**

For the execution of all the varied size dataset, we choose to go with HashMap since we don’t know how many distinct words we need to store as key. Also used LinkedHashMap for reverse sorting the HashMap. Our algorithm works in O(nlogn) time and O(n) space complexity.

Reason and Advantage for choosing HashMap:

1) Constant time complexity for searching and insertion.
2) Mutable since we have small as well as a huge dataset.
3) Capable to handle huge data after configuring little bit the JVM

Design:

Step1: Iterating the whole dataset using buffer readers and check if the word already exists in the map, if exists increment the value. If a word does not exist put the key as word and value as 1.

Step 2: Values in HashMap are not in sorted order and we want most 100 repeating words so reverse sorting the HashMap according to values using comparators.

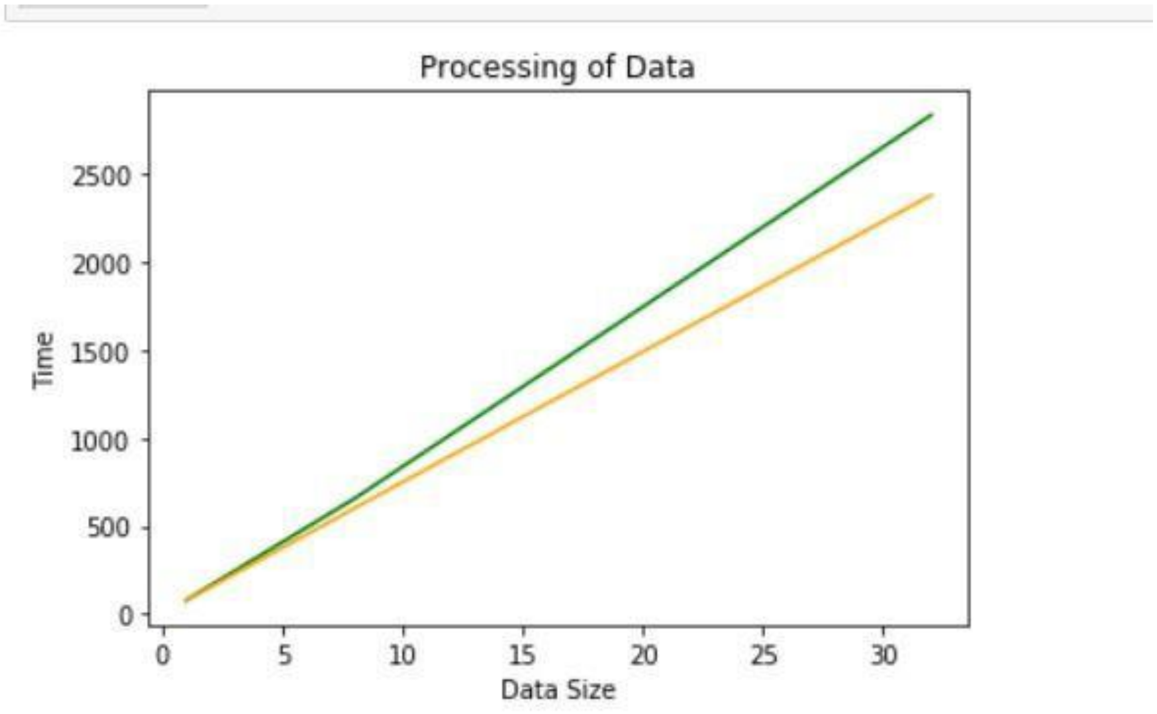
Step 3: Iterate the reverse sorted LinkedHashMap for 100 first entries and write it into output file simultaneously.

PS: Initially, we executed the 32 GB dataset with default heap size and the same algorithm. After 2 hours, we received GC out of memory error. After that, we changed the heap size (default JVM parameter which is set to 512 MB) to 4096 MB. Then 5190 MB for optimal performance.

● **Presenting Performance Data:**

Plotting it onto graph we have linear relationship between execution time to the size of data. There is slight increase in the slope as the size increases i.e. 32 GB ideally should have taken 2528s (1GB time * 32 = 79*32 = 2528 seconds) but it is actually taking 2831 seconds. It is due to garbage collectors overhead as the size of the data increases.

Code for plotting data:



Increasing the heap size using VM variable -Xmx from default 512MB to 4196MB and then to 5120 MB. Above green line is for 4196 MB heap size and orange for 5120 MB.

Observation:

For -Xmx4196M

1 GB file executed in 79 seconds

8 GB file executed in 655 seconds

32 GB file executed in 2831 seconds

<u>For -Xmx5120M</u>

1 GB file executed in 77 seconds

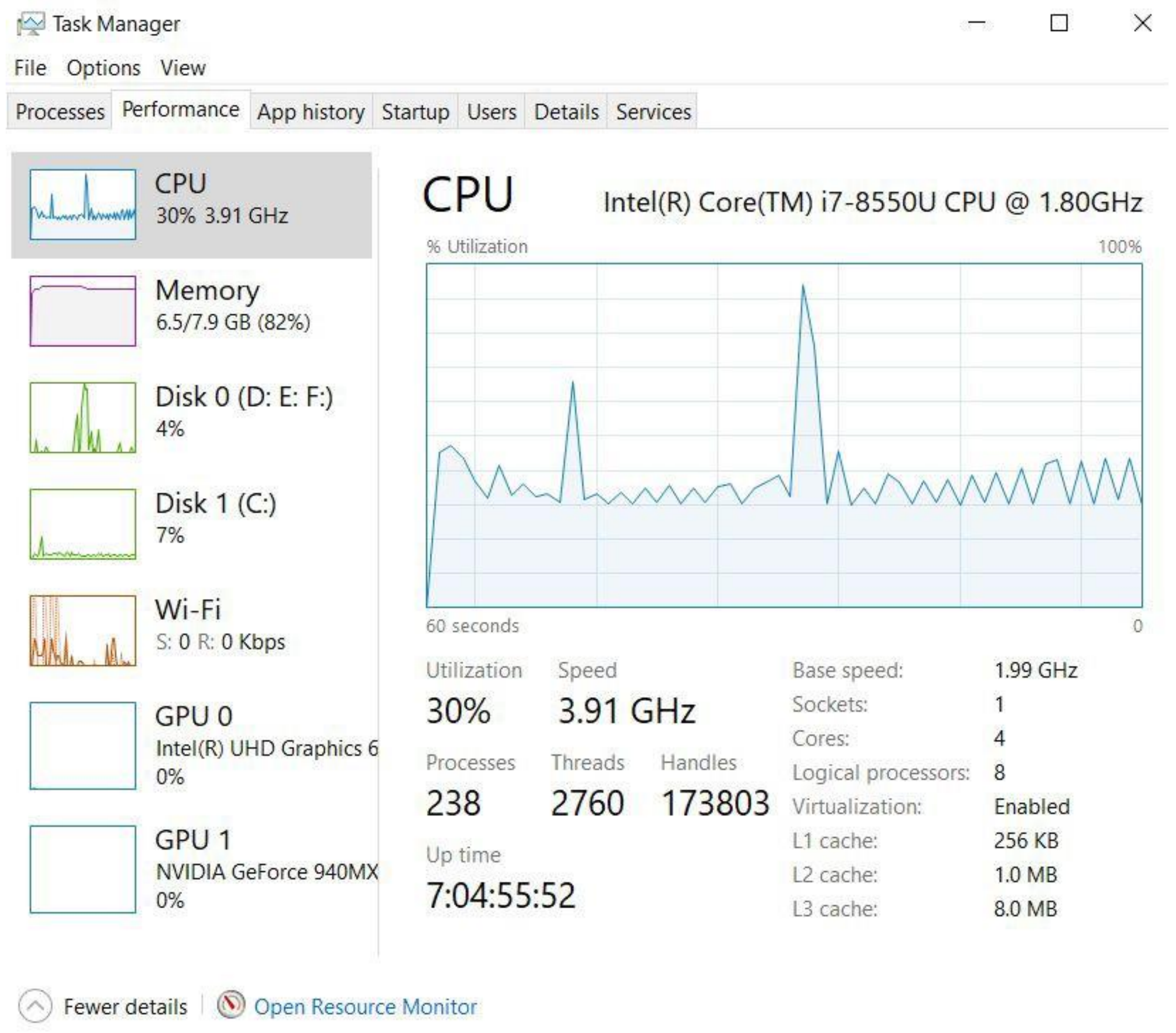
8 GB file executed in 601 seconds

32 GB file executed in 2376 seconds

Increasing further value of heap i.e Xmx variable did not make much difference so we got 5GB heap size as optimum one for processing all the three dataset.

- **Good Coding Practices:**

System Specification:



CPU Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

Utilization 30%
Speed 3.91 GHz
Processes 238
Threads 2760
Handles 173803
Up time 7:04:55:52

Base speed:	1.99 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB

CPU: Intel Core ™ i7-8550U 1.80GHz

Cores: 4

Memory: 8GB

Operating System: Windows 10

IDE: Eclipse

Out of box thinking:

We tried to implement the program using Bucketlist concept which ran for the 1GB dataset and 8GB dataset but failed for 32GB since it threw out of memory error initially. Even after increasing the heap size to the maximum to our laptop’s RAM it failed.
But for 1GB and 8GB dataset it was pretty fast, 25s and 188s respectively. Due to drawback of it not able to handle huge dataset and excessive usage of extra space our implementation is trade-off between execution time and memory usage.