Analysis Report

Name : Piyush Atri

Roll No : 2020201009

System Configuration:

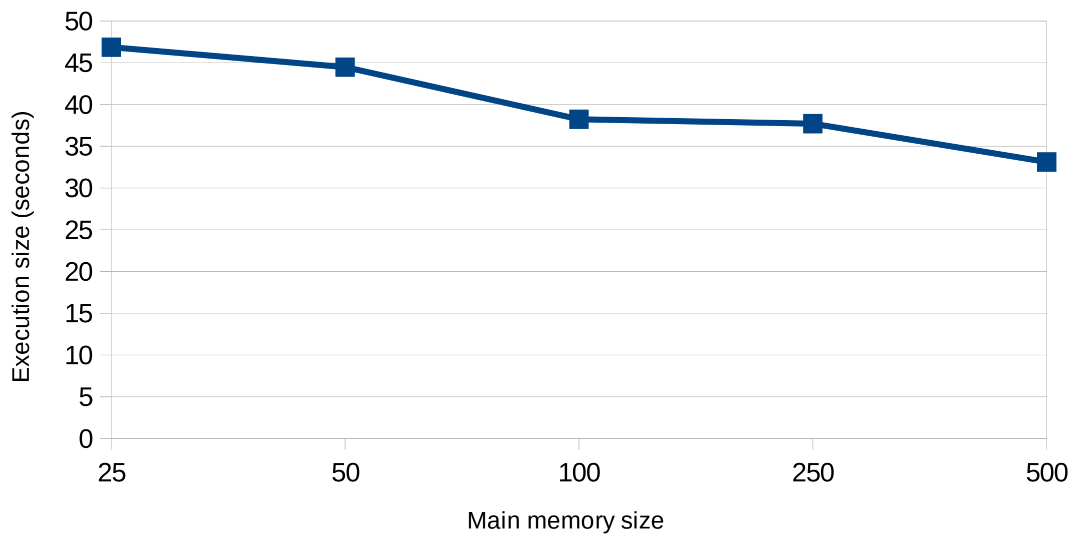
Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	39 bits physical, 48 bits virtual
CPU(s):	8
On-line CPU(s) list:	0-7
Thread(s) per core:	2
Core(s) per socket:	4
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	158
Model name:	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Stepping:	10
CPU MHz:	900.012
CPU max MHz:	4000.0000
CPU min MHz:	800.0000
BogoMIPS:	4599.93
Virtualization:	VT-x
L1d cache:	128 KiB
L1i cache:	128 KiB
L2 cache:	1 MiB
L3 cache:	8 MiB
NUMA node0 CPU(s):	0-7

1) **Varying memory with constant FileSize:** File of size 500MB

a) **Without threading**

Example command: python3 sort.py input.txt output.txt 500 desc C2 C1

Memory size(MB)	Time taken(s)
25	46.851927042007446
50	44.474347829818726
100	38.231404781341553
250	37.691375970840454
500	33.112123012542725

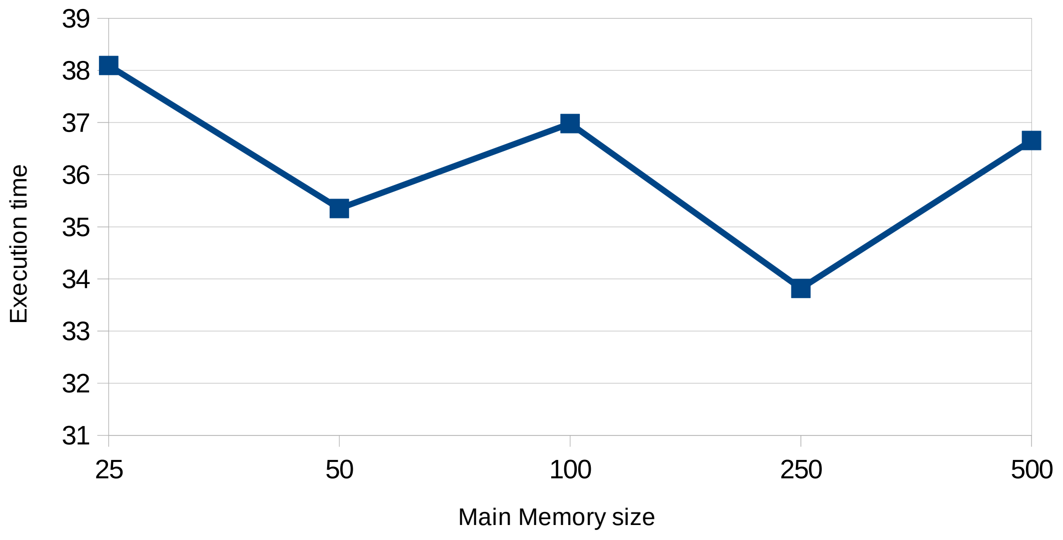


Explanation: Increasing main memory size improves performance as more numbers are sorted in the main memory which is faster

b) **With threading** : Four threads were used. Threading was applied to sorting sublist/chunks and writing it as well.

Sample command: python3 sort_threaded.py input.txt output.txt 500 4 desc C2 C1

Memory size(MB)	Time taken(s)
25	38.09420037269592
50	35.350953102111816
100	36.980873584747314
250	33.817294120788574
500	36.65710663795471



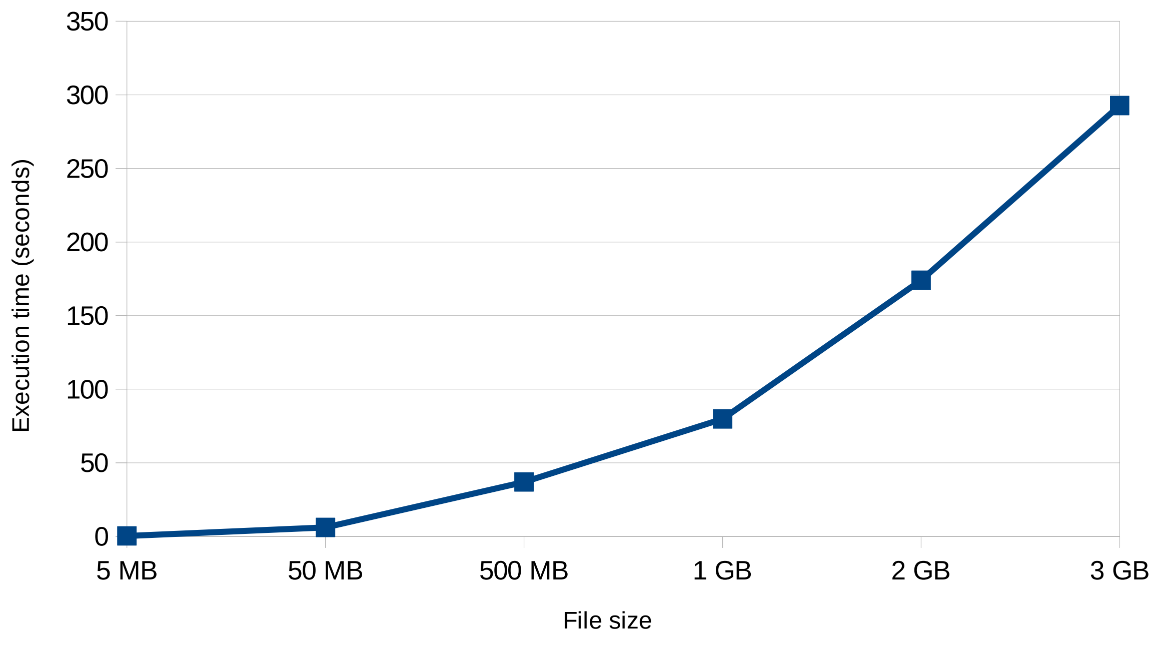
Explanation: Increasing main memory size improves performance as more numbers are sorted in the main memory which is faster. Threading further increases the sorting speed as 4 threads are sorting 4 lists simultaneously. The curve does’t show a particular pattern for varying memory size but all the times are lesser when compared to that of non threaded sort.

2) **Varying FileSize with constant memory:** Main memory size 100MB

a) **Without threading**

Sample command: python3 sort.py input.txt output.txt 100 desc C2 C1

File size	Time taken(s)
5 MB	0.359586238861084
50 MB	6.111742734909058
500 MB	36.980873584747314
1 GB	79.84135580062866
2 GB	174.05430507659912
3 GB	292.75607442855835

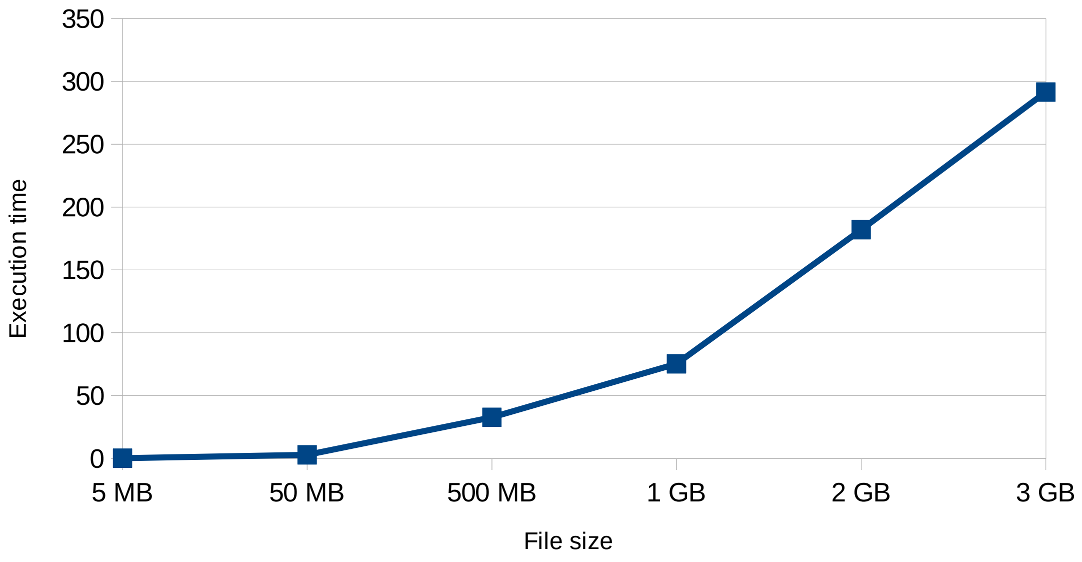


Explanation: Large files require more time to sort so increasing curve seen.

b) **With threading** : Four threads were used. Threading was applied to sorting sublist/chunks and writing it as well.

Sample command: python3 sort_threaded.py input.txt output.txt 100 4 desc C2 C1

File size	Time taken(s)
5 MB	0.20571541786193848
50 MB	2.9057397842407227
500 MB	32.78119683265686
1 GB	75.29243278503418
2 GB	182.06663608551025
3 GB	291.468474149704



Explanation: Large files require more time to sort so increasing curve seen. For smaller files threads are making sort faster because of parallel sorting of list.
However as the size increases the number of files that are open also increases. At that point read write time of disk becomes significant and reduces performance. So nearly same times are seen for larger files.