REPORT – QUESTION 4b

This question entailed us to compare the performance of the command “sort” with the merge sort script for files that we create.
The merge sort script is written for Bash and splits files according to MAX_LINES_PER_CHUNK value which can be set so that the split file size is 100MB/200MB/300MB, etc as desired.
The script takes two arguments. The first argument is the large 1GB file to be sorted. The second argument is the file to which the output is to be stored.
The script then performs merge sort on the input file as required by the question.

The size of the input file used was 1.1 GB and it contained random 3 digit numbers.

The “sort” command's performance and the merge sort script's performance for size of split file as specified is as follows:

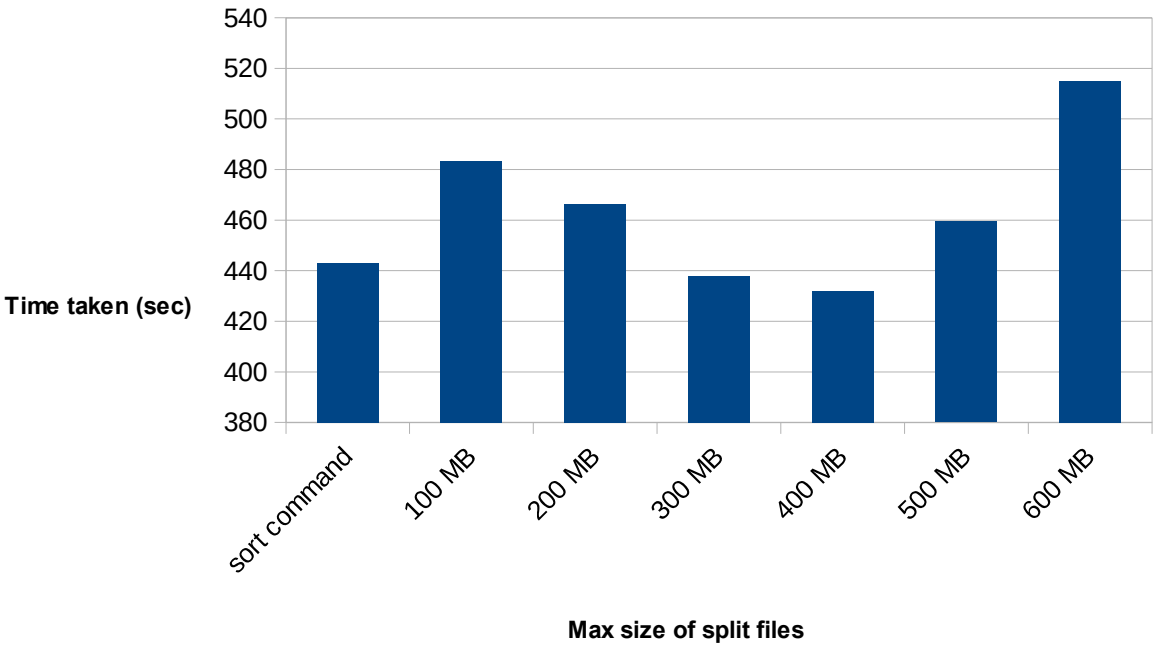
Size of split files	**“sort” command**	**100 MB**	**200 MB**	**300 MB**	**400 MB**	**500 MB**	**600 MB**
Time taken (sec)	443.24	483.26	466.34	438.02	432.15	459.47	515.20
Task clock (msec)	1,145,542.09	1,035,820.30	995,855.53	993,081.86	1,027,099.29	1045134.27	1,097,596.35
CPU's utilized	2.584	2.143	2.135	2.267	2.377	2.275	2.130
CPU clock (msec)	1,145,580.89	1,035,845.39	995,878.43	993,106.56	1,027,129.95	1045176.77	1,097,625.69
Context switches	67,110	204,117	202,510	218,544	182,700	229,695	213,172
CPU migrations	801	7,405	5,536	5,731	5,621	6,501	8,793
Page faults	28,953	884,311	459,072	228,281	240,039	242,086	124,801

Note that 100 MB split size corresponds to approximately 11-12 files to be sorted individually, 200 MB split size corresponds to approximately 5-6 files, 300 MB split size corresponds to approximately 4-5 files, 400 MB split size corresponds to approximately 3-4 files, 500 MB split size corresponds to approximately 2-3 files and 600 MB split size corresponds to 2 files.

These split files are sorted individually and then merged.

The “sort” command uses an External R-Way merge sorting algorithm. In essence it divides the input up into smaller portions (that fit into memory) and then merges each portion together at the end. It stores working data in temporary disk files (usually in /tmp).
Two things can greatly affect the performance of this command. They are
1) The location of the temporary files which can be specified with the -T option and should be stored on the fasted available device.
2) Amount of memory to use which can be specified through the option -S. A large value should be given but care should be taken that over-subscription doesn't occur which would result in swapping to disk.

The script that we have written splits the file into two parts recursively, using the split command, until the size of the partition is as desired. Then each of the smaller files is sorted using the sort command and finally the smaller sorted files are merged together.
The splitting is done based on maximum number of lines each split file can contain because splitting by size is dangerous as it may lead to integers being split in the middle.
The number of lines corresponding to the file size is calculated beforehand by trial and error.



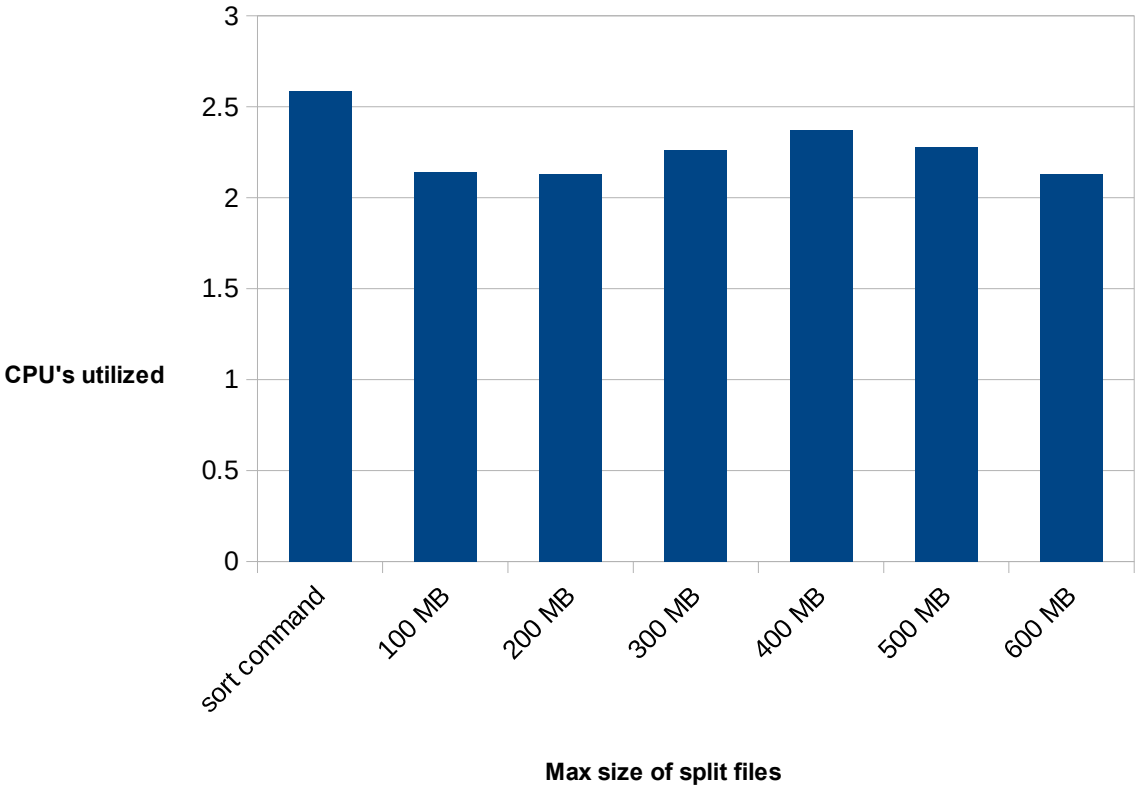
The best result for algorithm is achieved when the size of the split is around 400 MB which would result in approximately 4 evenly sized files to be sorted individually which is done in parallel on the 4 available CPUs. Each of the individual sorting is done in parallel and as a result the overall sort takes less time.

The rest of the split sizes correspond to either less than or more than 4 files to be sorted individually, which would mean that though the individual files are sorted in parallel, there are more number of smaller files to be sorted or there are less number of large files to be sorted than the number of CPUs and this leads to the overall sort taking more time.

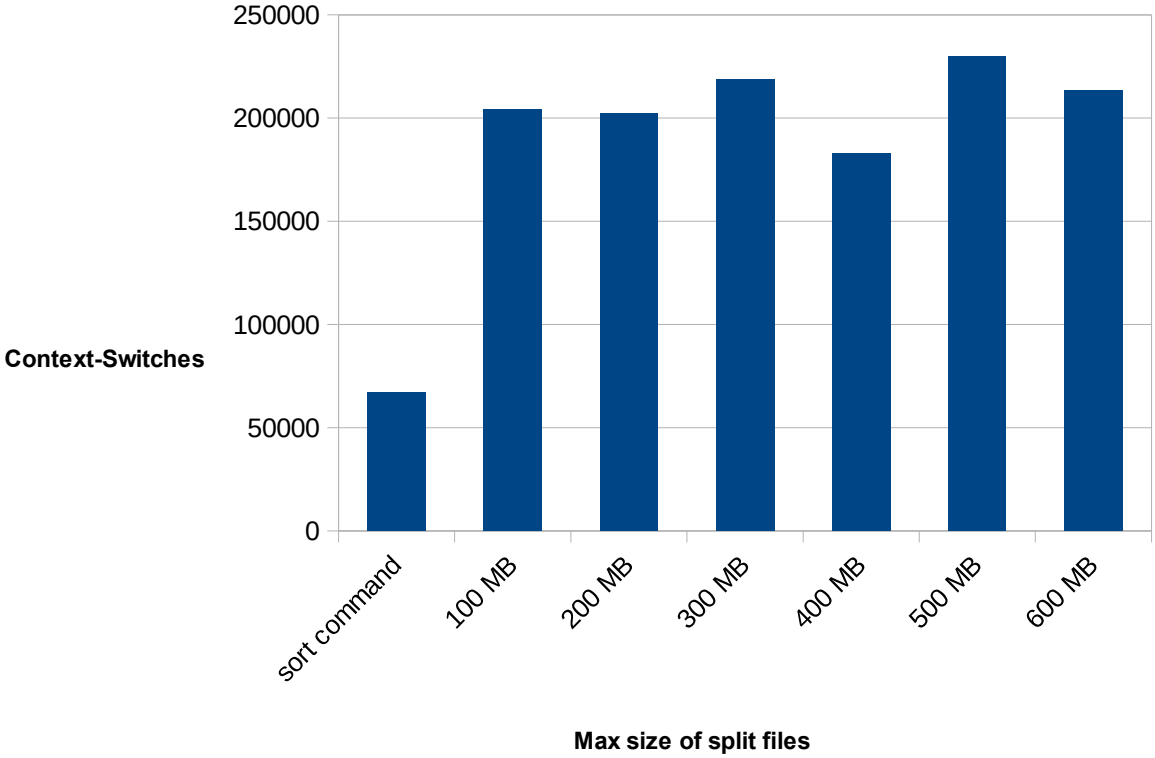
One more factor which comes into play is the number of disk accesses that take place.
Since the temporary split files are stored on the disk, we should strive to minimize the number of accesses to those files. This also implies that the files which we sort individually should be able to fit into the memory as a whole, otherwise the sort command would split that file and store it on the disk.

The most optimal sort can be achieved when we split the input file to exactly the number of CPUs available, which would result in optimum utilization of the CPU. This is provided that each file which is supposed to be sorted individually fits into the memory as a whole.

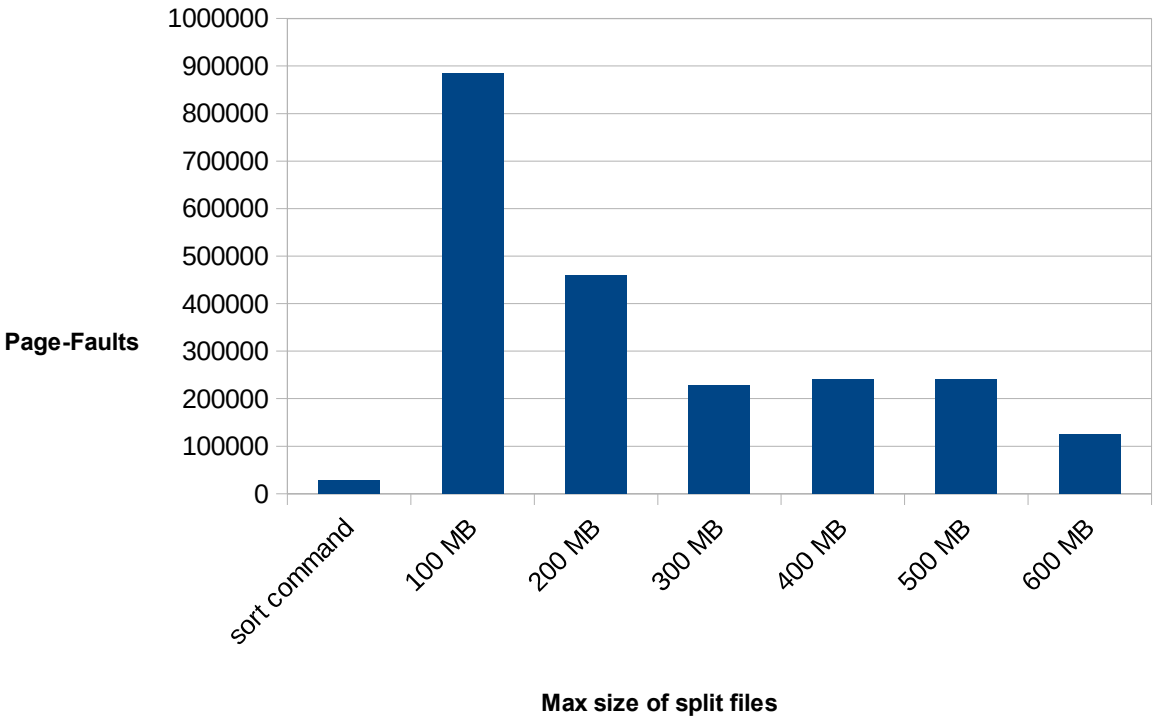
There should be optimal utilization of the CPU and the memory while minimizing the disk accesses.



The CPU Utilization of the merge sort version and the sort command is more or less the same.



The number of context switches is as shown in the graph.



The number of page faults is maximum when the size of the split is minimum as the small files have to be repeatedly fetched from the disk. Only the file which is currently being sorted individually is stored in the memory and then another has to be fetched from the disk on a context switch.
The number of page faults is minimum when the size of the split is maximum as the whole of the large split file resides on the memory and disk accesses are therefore minimized.