Problem 3: Below is a table showing the performance of the parallel program versus the serial program.

File Size	Linear	Parallel (Memory Footprint)		
		100KB	1MB	10MB
1 KB	0.000075 sec	0.033613 sec	0.032818 sec	0.059609 sec
10 KB	0.000268 sec	0.037570 sec	0.040653 sec	0.030199 sec
100 KB	0.001718 sec	0.040167 sec	0.038415 sec	0.047596 sec
1 MB	**0.017737 sec**	**0.042930 sec**	**0.021341 sec**	**0.031960 sec**
10 MB	0.165096 sec	0.095811 sec	0.057167 sec	0.081258 sec
100 MB	1.635020 sec	0.745876 sec	0.445818 sec	0.456290 sec
1 GB	16.354252 sec	7.443185 sec	4.259830 sec	4.177663 sec
5 GB	84.646819 sec	37.468847 sec	21.499828 sec	20.393480 sec

As demonstrated by the table, there is a clear trend showing that the parallel program is faster than the serial as the size of the file increases in size. As a secondary measurement I also varied the amount of memory used by the program. When going from 100KB to 1MB the parallel program seemed to run faster. However, when taking the next step to 10MB the program slows down. This is probably caused by the program having to much memory to hold and having to traverse further into the caches. Whereas smaller memory footprints means that I can have all the information in caches closer to the CPU. That being said, the more of the file the program can load at once the faster it can count all the words in it. It therefore appears that 1MB is the best compromise between the two.

Going back to the performance of parallel versus linear. The linear version appears to be faster in smaller files and degrades as the file size increases. This is likely due to the smaller files being able to fit in single cache blocks. The linear program is able to quickly access the entire file without going to disk several times. As the file size increases, the file will not fit in single cache lines. This is where the parallel program comes in and takes advantage of multiple file pointers to different parts of the file. Resulting in a faster execution time.