FOC Project#1 Submission **Name: Muhammad Moeez Khan**

			List Size			
	All times in seconds	**5,000**	**10,000**	**25,000**	**50,000**	**100,000**
Slow Sort Algorithim	**Normal**	0.356789	1.657647	15.189087	64.235277	245.577822
	2 Parallel	0.166184	0.69795	5.577961	28.685291	121.734333
	4 Parallel	0.075599	0.338494	2.544984	11.935069	61.923858
	8 Parallel	0.027911	0.221693	1.239409	5.87218	22.976821
			List Size			
	All times in seconds	**100,000**	**250,000**	**500,000**	**1,000,000**	**5,000,000**
Fast Sort Algorithim	**Normal**	0.117024	0.330806	0.789022	1.649718	1.649718
	2 Parallel	0.136612	0.388895	0.86246	1.668371	1.668371
	4 Parallel	0.146568	0.378893	0.857517	1.750235	1.750235
	8 Parallel	0.120899	0.357884	0.774378	1.76538	1.76538
			List Size			
	All times in seconds	**100,000**	**250,000**	**500,000**	**1,000,000**	**5,000,000**
Built In Sorting Algorithim		0.034838	0.183381	0.309812	0.709302	0.709302

You must include a discussion of the data that you have recorded. Does it make sense? Is it confusing? Did you get what you expected? And so on. I expect about one to two pages total for this document.

Pertaining to understanding the data, all makes complete sense apart from the normal and the parallel processes implementation of the fast sort algorithm (merge sort). I completely expected the parallel processes for the slow sort algorithm (selection sort), since processes increase the speed of a function because they allow for concurrent execution of code while being lightweight, meaning that they are very efficient in terms of memory usage. Thus all the processes for the selection sort algorithm make sense. However, I expected the parallel processes for the merge sort algorithm to be faster than the straightforward implementation due to the advantages of the processes as discussed earlier. Further, regarding the in-built function - since I knew it has a back-end implemented of merge sort or quick sort, thus I am curious as to why it is that fast - compared to other algorithms. I believe this could be due to complex parallelism and tail recursion implementation.