Intro to Computer Systems

Project 1

Group Members: Mayank Daswani, Husen Valikrimwala, Jimmy Jorge

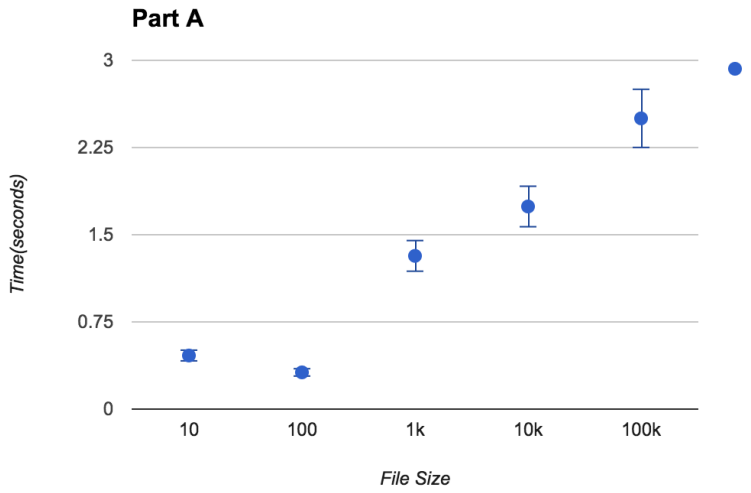
HOW TO COMPILE CODE

An input file is required in all the code.
To compile, write the following command in terminal:
>>gcc <filenamehere>.c –o s
>>./s
The following table shows the run time for all parts with different file sizes.

File Size	Part A Time(sec)	Part B Time(sec)	Part C Time(sec)	Part D Time(sec)
10	0.461	0.87	0.002	0.001
100	0.316	1.229	0.052	0.005
1k	1.318	1.604	0.452	0.019
10k	1.743	1.904	0.808	N/A
100k	2.501	3.243	1.973	N/A

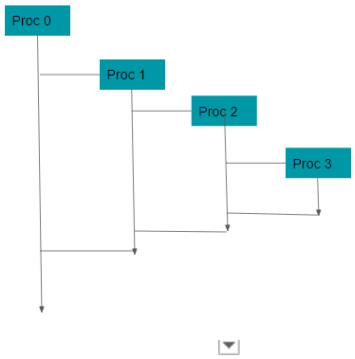
Part A

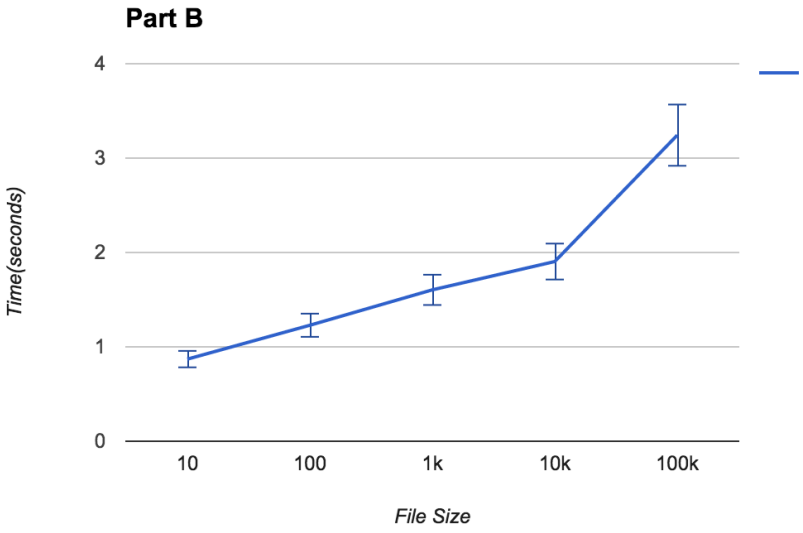
For Part A, we used a straight forward method. We iterated through each value in the file and replaced the minimum/ maximum value if an even lower value or higher value is encountered. For sum, every value is added to sum. The following graph shows the time analysis for Part A.



Part B

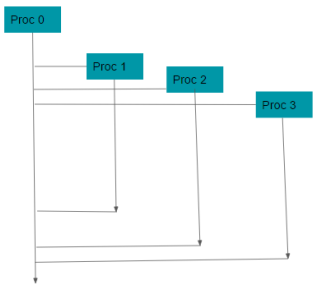
For Part B, we used a method like DFS. Each child spawned no more than one child i.e each child has one parent process. Each child reads the value from the file and compares it to the parent value. If bigger/smaller, it replaces the value in the max/min variable. The following graph shows the time analysis for Part B.

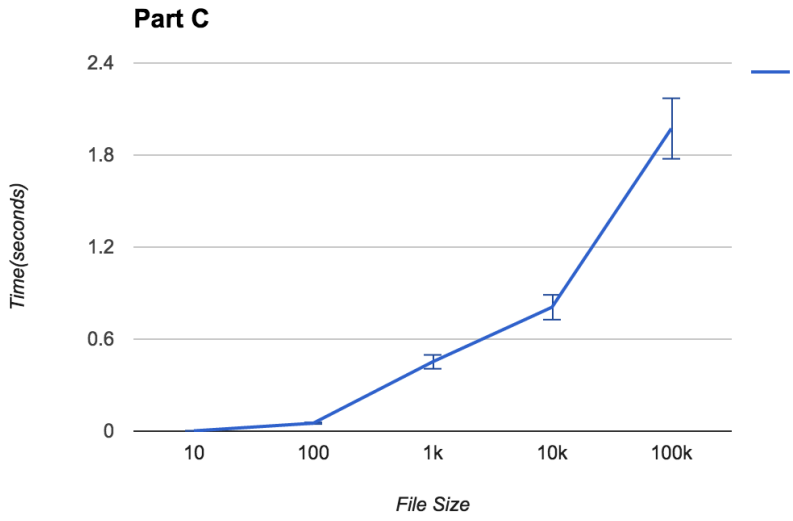




Part C

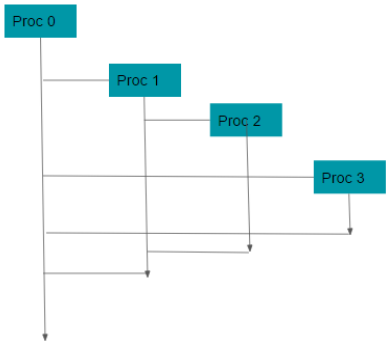
For Part C, we used a method like BFS. Each child was spawned from one parent i.e each child has one parent. Each child reads the value from the file and sends it to the parent process. The parent process compares the value from the min/max variable values. If bigger/smaller, it replaces the value in the max/min variable. For sum, each value is added to the sum variable as it is being compared. The following graph shows the time analysis for Part C.

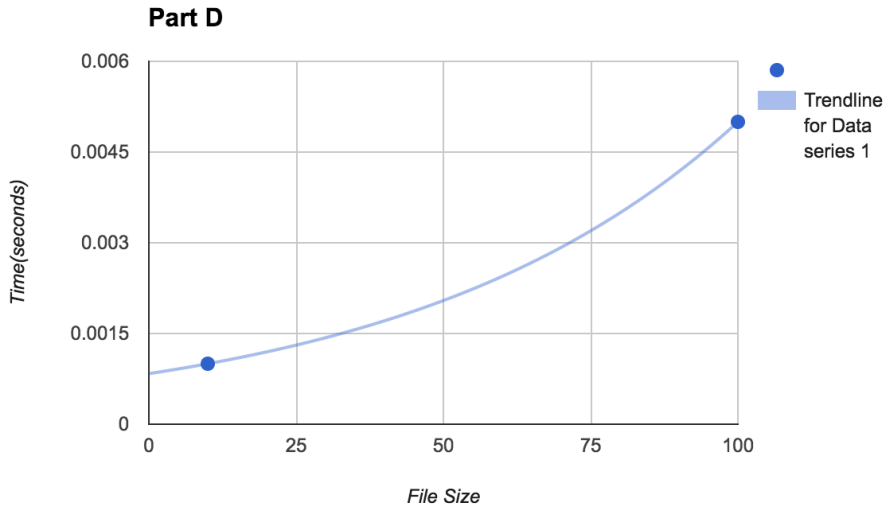




Part D

For Part D, we used a both BFS and DFS. Each child reads the value and creates more processes if there are more than 2 numbers to compare. The execution of this algorithm is very similar to merge sort. Until two elements remain, parent recursively spawns children and then compares to find max. min values. The following graph shows the time analysis for Part C.





Extra Credit

While observing all the methods from above, we notice:

Part A has the most runtime as it uses just one process. Part B creates child repeatedly and splits the work. But it is less than part C as the it takes longer to compare at each process.

Part C is more efficient as each child is directly connected to the parent though pipe and the parent just compares the values.

Part D, Splits the work among all processes like merge sort. It creates processes until it there are just two values to compare.

The following graph shows the time analysis of each approach in the above code. We observe that Part A> Part B> Part C> Part D.