- **What is the largest number of workers M that your implementation supports?**

At least on the computer that I have been testing my code the number of M's that I can use is around 10000. If I push higher than 10000 and try to run it the code will end up freezing at the start. Here is the example of the input line that was used, I used all the defaults except for M.

```
>>621Project1 "C:\Users\chris\621Project1\datafile4.dat" 10000
```

I left the rest the same for this test as the default values would be the best to try and figure out what the best M would be.

- **What is the least (expected) elapsed (wall) time for a random datafile of 1GB in your implementation?**

  With the max of the M that we have above, if we change no other values we know that it would take around 4m23seconds. Even when I start to increase the size of N and C the file still takes roughly around 4 minutes and 20 seconds. I think this is as fast as my code can do this data for a 1GB size file. Below are a few of my different runs I did all giving around 4m20seconds.

  ```
  >>621Project1 "C:\Users\chris\621Project1\datafile4.dat" 10000 256000
  >>621Project1 "C:\Users\chris\621Project1\datafile4.dat" 10000 256000
  >>621Project1 "C:\Users\chris\621Project1\datafile4.dat" 5000 256000
  >>621Project1 "C:\Users\chris\621Project1\datafile4.dat" 1000 256000
  ```

- **What is the largest (random) datafile you can process within 3 mins (wall) elapsed time?**

  260sec/1GB = 180sec/X => 260x = 180 => .692GB

  You can see in the math above we took the assumption made in the previous question that a 1GB size file runs for about 4m20sec or 260 seconds. This would mean if we ran our code with the exact choices for M N and C that we can run at .692. I don't believe changing them around will effect too much as the more we added and changed even above for the 1GB test it always was around 4m20sec

- **How does the elapsed time change as M ranges from 1 to the maximum value in Q1, for the datafile and N, C parameter values in Q3?**

  We know that the larger we make M the faster time will get. If we run the large file against our code with only 1 worker it will run through all of the data with only 1 worker doing everything giving us a time of T lets say. The more Ms we add we start to split that work up amongst different workers making the time less with each M we add, giving us a time of T/M meaning that if we know that our time for the 1GB file is 260 seconds for

10000 Ms that for only 1 M it would be around 2600000 seconds long which is a huge difference. This is why threading is important as it helps our data stay split and quicker.

**For each question Q1-Q3 above, provide the setting (command-line arguments) for all M, N, and C parameters, and the basic statistics (min, max, average, median) of the number of jobs completed by the workers, and the (wall) elapsed time (in msecs) for your main function.**
**Assume that N and C take values in some small predefined sets of values: eg. N in {1KB, 32KB, 64KB, 256KB, 1MB, 64MB}; C in {64B, 1KB, 4KB, 8KB}.**