Parallel programming lab 1

*Work on the problem assigned to your group until 11:30. If you finish early, move on to the next problem. Your group will be asked to spend 5 minutes at the end of class discussing the problem, your approach/solution, any challenges you ran into, and anything that you learned today.*

# Group 1.

Download a merge sort implementation from

<https://github.com/ramcdougal/cbb750/blob/master/merge_sort.py>

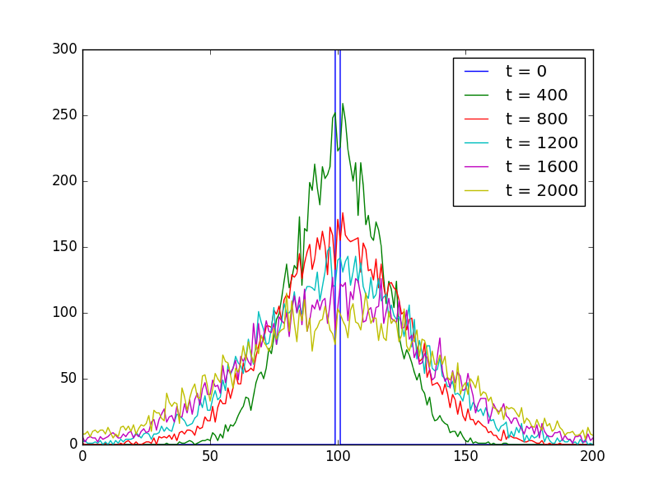
This code generates a pseudo-randomly shuffled list of one million distinct integers, sorts them, displays the result, and displays the sort time.

Parallelize this code using Python’s multiprocessing module. Provide timings for running on 1, 2, 3, 4, and 8 processes.

*Now move on to the task for group 2.*

# Group 2.

Download a stochastic diffusion simulation implementation from

<https://github.com/ramcdougal/cbb750/blob/master/stochastic_diffusion.py>

On my computer, this code takes about 45s to run. It generates a figure and displays the simulation time.

Parallelize this code using Python’s multiprocessing module. Your parallel implementation need not produce identical stochastic values as my serial version, but it must be self-consistent (i.e. it must always produce exactly identical results regardless of how many processors are used). Provide timings for running on 1, 2, 3, 4, and 8 processes.

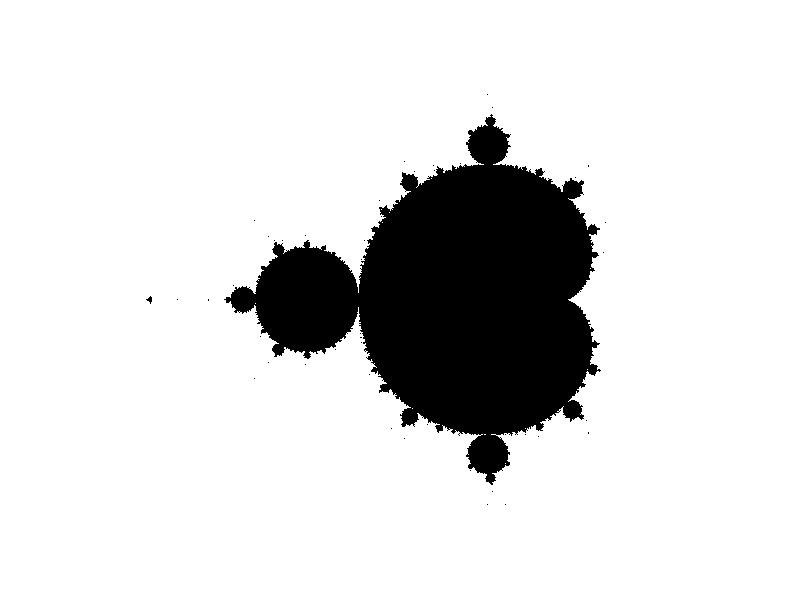
Hint: think about how much advancing can be done by each process before there is a need to synchronize. This is different than the situation in the homework.

*Now move on to the task for group 3.*

# Group 3.

The **Mandelbrot set**, named for former Yale mathematician Benoit Mandelbrot, is for many people their first introduction to fractals. It contains the set of points on the complex plane such that the iteration with stays bounded as . The Mandelbrot set, like all fractals, is self-similar: zooming in reveals a similar structure at every spatial scale.

Download serial code to compute and plot the Mandelbrot set from

<https://github.com/ramcdougal/cbb750/blob/master/mandelbrot_set.py>

It generates a figure and displays the calculation time.

A key challenge to parallelizing the Mandelbrot set is that the amount of work required to test if a point belongs to the set is not constant and cannot be predicted. This same property is true for many problems in biomedical informatics as well. Due to this uneven and unpredictable workload, it would be inefficient to simply partition space across the processors.

Parallelize the Mandelbrot set code using two instances of multiprocessing.JoinableQueue: one for points to process and one for points belonging in the set. Each compute process should get a point from the Queue, decide if it belongs to the set, if so put it on the results Queue, declare task\_done and repeat. Time your parallel code with 1, 2, 4, and 8 compute processes.

Additional question: if for some reason, the choice of which work will be done in which process had to be made before any calculations could proceed, how would you distribute the work in a way that was likely to be approximately the same across all processes?

*Now move on to the task for group 1.*