Computer Science 112
Problem Set 6, **2%**

Parallel Mergesort and Monte Carlo Simulation

In this assignment, you will implement a parallel mergesort and a parallel Monte Carlo simulation for computing the value of π (pi). As your guide, you may use the example `parallel*.py` programs we have seen and developed in class. Please submit `parallelMergesort.py` and `parallelMontePi.py` to your `turnin/ps6` subfolder. Both programs should show timings of the sequential and parallel versions of the algorithms, as in the example programs.

Mergesort

There are various ways of approaching a parallel implementation of the mergesort algorithm. A complicated implementation could involve a parallelization of the `merge` operation, where two Processes responsible for separate sorted lists swap elements with each other to ensure that the lists don't overlap. Then the lists could simply be concatenated to form the overall sorted list.

You can take a more simple approach, duplicating the kind of parallelism used in the `parallelQuicksort.py` example. Here, we keep the `merge` operation as before and rely on the non-parallel `mergesort`, but simply run more than one instance of `mergesort` at a time.

Monte Carlo Simulation

Monte Carlo methods are the class of algorithms that use random sampling to compute their results. For this part, you will implement both a sequential and parallel implementation of one of the more basic Monte Carlo methods, that for computing the value of the mathematical constant π. Your `main` method will show the sequential versus parallel timings for performing this computation. The `parallelIntegration.py` example can be used as a guide.

The process for randomly approximating π is quite simple. Consider the disk of radius 1 centered at the origin. Considering only the part of the disk within the first quadrant (positive x and y), we see the area within the disk is $\pi/4$ while the area within the unit square (from 0 to 1 in x and y) is obviously 1. If we generate a random point in the unit square (using `random.random()` for the x coordinate and again for the y coordinate), then the probability that the point is within the unit disk is $\pi/4$. We can determine whether the point is within the disk by evaluating $x^2+y^2 <= 1$. If we generate n points, then the proportion of points that were within the disk is an approximation of $\pi/4$. Multiply by 4 and you have an approximation of π.