22C:16 Homework 4

Due via ICON on Thursday, March 1st, 4:59 pm

What to submit: Your submission for this homework will consist of four files. One of these will be a pdf file called `homework4.pdf`. This will contain the answers to Problems 3 and 5. This pdf file should start with your name, section number and student ID. The remaining files should be called `fastTwoDRandomWalk1.py` (solution to Problem 1), `manyFastRandomWalks.py` (solution to Problem 2), and `fastTwoDRandomWalk2.py` (solution to Problem 4). Each of these Python files should start also with your name, section number and student ID appearing at the top of the file as Python comments. Also make sure that your Python code is appropriately commented. You will get no credit for this homework if your files are named differently, have a different format (e.g., docx), or if your files are missing your information.

1. Take a look at Problem 3 on Homework 5 in my spring 2011 offering of this course. This is the problem in which students are asked to write a function called `twoDRandomWalk`. The solution to this problem is also available on my 22C:16, Spring 2011 course website.

   For the current problem, define a function called `fastTwoDRandomWalk1` that is a "faster" version of the 2-dimensional random walk being simulated by `twoDRandomWalk`. In particular, `fastTwoDRandomWalk1` takes an additional parameter called `jump` and at every step of the random walk, the "robot" moves a distance that is *at most* `jump` in any one of the 4 directions. For example, if `jump` is 2, then the "robot" could, in one step, move a distance of 1 or 2 in any one of the 4 directions. Note that whether the "robot" moves one or two steps is also to be determined randomly. More generally, in order to figure out where the "robot" should go next, your function should first pick a direction at random and then pick at random an integer distance between 1 and `jump` (inclusive of 1 and `jump`). The "robot" should then move the chosen distance in the chosen direction. The function should have the following header:

   ```
   def fastTwoDRandomWalk1(n = 100, printOn = False, jump = 1)
   ```

   The first two parameters have exactly the same meaning as they did in Problem 3, Homework 5, Spring 2011. Note that since `jump` can be larger than 1, your "robot" might end up moving outside the "barrier" in a certain step of the random walk. The random walk ends when this happens.

   Save your function in a file called `fastTwoDRandomWalk1.py` for submission.

2. Now define a function called `manyFastRandomWalks` with the following header:

   ```
   def manyFastRandomWalks(n = 100, jump = 1, numRepititions = 1000)
   ```

   This function simulates as many "fast" random walks as specified by `numRepititions` and returns the *average* length of the random walk, where the average is taken over the all of the simulations. Note that the function also takes `n` and `jump` as arguments and simulates random walks with these parameters.

   Save this function in a file called `manyFastRandomWalks.py`.

3. Use the function `manyFastRandomWalks` defined above to find out the average length of a random walk (averaged over 1000 simulations) for (i) $n = 100, jump = 2$, (ii) $n = 200, jump = 5$, and (iii) $n = 500, jump = 10$.

4. Now let us modify our notion of when a random walk ends. We say that a random walk ends when it reaches the barrier in the positive quadrant (i.e., quadrant I). More precisely, the random walk ends as soon as it reaches a point (x, y) with $x \geq n$ and $0 \leq y \leq n$ or $y \geq n$ and $0 \leq x \leq n$.

There is one situation that might arise in this random walk that is worth clarifying. Suppose the "robot" is at a location (x, y) and then in one step moves outside the barrier in quadrant II, III, or IV. For example, consider the situation where $n = 100$ and $(x, y) = (-99, 90)$ and the next step requires the "robot" to move a distance of 3 units further to the left. In this case, if we do move the "robot" will it end up going beyond the barrier. But note that the random walk should not be terminated at this point. The easiest way to deal with such a situation is to not move the "robot" in the current step and just leave it at $(x, y) = (-99, 90)$.

Define a new function called `fastTwoDRandomWalk2.py`. This function does the same thing as `fastTwoDRandomWalk1`, except that it uses the new definition of when a random walk terminates. Save this function in a file called `fastTwoDRandomWalk2.py`.

5. Use the function `manyFastRandomWalks` defined above to find out the average length (averaged over 1000 simulations) of the new type of random walk (defined in the above problem) for (i) $n = 100, jump = 2$, (ii) $n = 200, jump = 5$, and (iii) $n = 500, jump = 10$.
