When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. You can download the **q8helper** project folder (available for Friday, on the **Weekly Schedule** link) in which to write/test/debug your code. Submit your completed **q8solution** modules (**81**, **82**) and **empirical.pdf** online by Thursday, 11:30pm. I will post my solutions to EEE reachable via the **Solutions** link on Friday morning.

1a. (5 pts) Write a script that uses the **Performance** class and generates data that we can use to determine empirically the complexity class of the **closest_2d** recursive function defined in the **nearestneighbor.py** module. This function takes a list of **(x,y)** coordinates and finds the two that are closest together and returns the distance between them. A simple/inefficient algorithm would just find the minimum after computing the distance between all pairs of points, which would be $O(N^2)$. Call the **evaluate** and **analyze** functions on an appropriately constructed **Performance** class object for lists of random coordinates (from sizes 100 to 25,600 doubling the size each time). Do **5** random timings for each size: **each of these 5 timings** should run on a **different** random list of coordinates, and creating the list of random coordinates should not be timed. Use the **random.random** function to generate each coordinate: it returns **float** values in the range **[0,1)**. Hint: Write a script with a **create_random** function that stores a random list of coordinates of the correct size into a **global** name, then time the **closest_2d** function using that **global** name as an argument. If an exception is raised for any size, print an error message for that size but continue collecting data: this might happen for small sizes. I had about **20** lines in my module (including blank lines). See the file **sample8.pdf** (included in the download) for what your output should look like: of course, your times will depend on the speed of your computer (but the complexity class estimation will not). The process can take a few minutes. **Do not time the execution of the create_random function!**

1b. (3 pts) Fill in part 1b of the **empirical.doc** document (included in the download) with the data that you collect (or use the data in **sample8.pdf** if you cannot get your code to produce the correct results) and draw a conclusion about the complexity class of the **closest_2d** function by seeing how much time it takes to run as the size (in number of coordinates) of its input list doubles. Then predict how long this function will take when running on an input list of 1 billion coordinates.

2a. (5 pts) Write a script that uses the **cProfile** module to profile all the functions called when the **closest_2d** function is run on a random list with 25,600 coordinates. Generate the random graph first (do not include its generation in the profile information) and then call **CProfile.run** so that it runs **closest_2d** on that list; also specify a second argument, which is the file to put the results in (and the file on which to call **pstats.Stats**) to print the results.

For the first one, sort the results decreasing by **ncalls** (print at most the top 12); for the second one, sort the results decreasing by **tottime** (print at most the top 12). Hint: The notes show how to instruct the profiler put the profile information into a file and then show how to access that file and format the results it displays to the console; I had about a dozen lines in my module (including blank lines). It should take only a few seconds to profile this code. See the file **sample8.pdf** (included in the download) for what your output should look like: of course, your counts and times will depend on the speed of your computer and the random list of coordinates generated.

2b. (2 pts) Answer the questions in part 2b of the **empirical.doc** document (included in the download) with the data that you collect (or the data in **sample8.pdf** if you cannot get your code to produce the correct results).

After editing empirical.doc for parts 1b and 2b, convert it into a .pdf document and submit the document in that format on checkmate. You can use the lab machines to do the conversion.