Homework 2

Due:

Part A Thursday, February 20, 2020, 11:55pm
Part B Thursday, February 27, 2020, 11:55pm

Guidelines:
Each homework is composed of two parts - a part A, which is theory-based to be done with “pen and paper”, and a part B, which is one or more python coding assignments.

A homework assigned on a given Tuesday will have its part A due the following Thursday, and its part B due the Thursday after that. So in practice, you have 1 week to complete each theory assignment, and 2 weeks to complete the programming assignment.

All homeworks will be submitted via Courseworks. For a given assignment, all files should be put in a folder called “uni-hw#-part”, all lowercase, which is then zipped and submitted on courseworks. The folder should contain a single readme.txt which has your uni, the homework number, and a brief summary of your work for each part (if necessary).

For example, for homework 2, I will create a folder called tkp2108-hw2-a. Inside of this folder I will have a single plain text file called readme.txt (you can make this file using Textedit on Mac, Notepad on Windows, your preferred editor on Linux). I will put my work in a txt, PDF, doc, etc, file unless otherwise directed by the problem. Then to submit, I turn the folder into a zipfile (right click and compress on mac, 7zip on windows, etc), and upload that for the assignment.

Part A

1: Recall selection sort from class. Given the following input list, provide the state of the list at every iteration. The first iteration is given (note, the table is just a placeholder, you will need to add more rows).

Iteration 0	3	0	4	6	2	5	1	7
Iteration 1	0	3	4	6	2	5	1	7
Iteration 2								
Iteration 3								

2: Imagine if a phone book was not in order and had one name per page. If there are only a few names, it doesn't make sense to sort it. If we use it many times, or there are lots of names, it makes sense to sort it once, and then search, since our search will be much faster.

If we have 10,000 names, how many times do we need to search before Sort+Binary search makes sense over Sequential Searchs? If we are going to be searching our phone 1,000,000 times, how big does it need to be for sorting to be worth it? Answer the questions below.

As explained in class, sequential search is where we simply iterate through all records until we find what we're looking for. If we sort our records first, we can do binary search, pseudocode for which is supplied in the notes.

a. Write the formula for the number of steps to perform **k** sequential searches on a list of size **n**

$$f(n,k) =$$

b. Write the formula for the number of steps to perform **k** binary searches on a list of size **n**

$$g(n,k) =$$

c. If we have 10,000 names, how many times do we need to search before Sort+Binary search makes sense (e.g. is less than) over sequential search?

I.e. solve $f(10{,}000,k) < g(10{,}000,k)$ (I recommend wolfram alpha!)

c. If we are searching 1,000,000 times, how big does the list need to be to make sorting worth it?

I.e. solve $g(n,1{,}000{,}000) < f(n,1{,}000{,}000)$

3: Consider the following pseudocode:

- Given a list, iterate through the elements starting at the second
  - Note the element at this position, call it element "X"
  - Walk back through the previous elements in the list
    - For each element, move it forward in the list by 1 position
    - if you find a smaller element or hit the beginning of the list, put X in that spot

What is the time complexity of this algorithm? Justify your answer.

Part B
1. Simulations

Physics 101: When a projectile is fired straight up into the air with initial velocity v_0, it's position may be calculated as a function of time by the following equation:

(1) $\boldsymbol{s(t) = -0.5\, g\, t^2 + v_0\, t}$
where $g = 9.81\ m/sec^2$ is the acceleration due to gravity.

What is sometimes missed in an introductory course are the many assumptions that go into making this equation so simple. Here are just a few:

- The mass of the earth is uniformly distributed
- The earth is a perfect sphere
- There is no air resistance
- The earth does not rotate
- The acceleration due to gravity does not change with altitude

So why do we make assumptions like these? Well, including even some of these effects in our model makes calculating the projectile's position much more complicated. So complicated, in fact, that you can't do the calculation with a simple pencil and paper. That is, you need a computer. In this problem we will take into account one of these effects: The variation of gravitational acceleration with altitude. To do this we will replace the constant g with the function:

$\boldsymbol{g(s) = G \cdot M_E / (R_E + s)^2}$
where
$G = 6.6742 \times 10^{-11}$ is the gravitational constant
$M_E = 5.9736 \times 10^{24}\ kg$ is the mass of the earth
$R_E = 6{,}371{,}000\ m$ is the mean radius of the earth

So now the original equation becomes:

(2) $\boldsymbol{s(t) = -0.5\, g(s)\, t^2 + v_0\, t}$

To determine position using this equation you will write a program in Python that simulates the projectile's motion in the following way. Make the assumption (I know, they're hard to get away from when you're building models.) that in a very short period of time, Δt, the velocity is constant. This means that you can calculate position in the following way:

$\boldsymbol{s(t+\Delta t) = s(t) + v(t) \cdot \Delta t}$

where **v(t)** may be calculated at each time step using:
$\boldsymbol{v(t+\Delta t) = v(t) - g(s(t)) \cdot \Delta t}$

Write a module in Python that contains the following functions:

1) g(h) - where h is altitude and the function returns the value of acceleration due to gravity at altitude h.

2) s_next (s_current, v_current, delta_t) - which returns ***s(t+Δt)*** using the equation above where s_current is ***s(t)*** , v_current is ***v(t),*** and delta_t is ***Δt.***

3) v_next(s_current, v_current, delta_t) - which returns ***v(t+Δt)*** using the equation above.

4) s_sim(t, v_init, s_init, delta_t) - which returns the projectile's position at time ***t*** from now ***.*** This function uses functions 1-3 above to return the simulation's estimate of the projectile's position at time ***t*** given the initial values of position and velocity and the constant ***delta_t***.

5) s_standard(t,v_init) - which returns the position using equation (1) and and initial position = 0.

6) plot_trajectories(v_init, s_init, delta_t) which plots the entire trajectories of the two methods and the difference in value between them.

Your Python module should be named projectile.py . Attached you will find a Python file called projectile_tester.py containing a main function. *Your module must work with the tester provided here without modification.* ***You must NOT alter the projectile_tester.py file.***

Extra: You may notice that our s_sim function is extremely inefficient - we are rerunning the entire simulation every time we want to calculate the next point! Modify the code (including the tester) to make it more efficient, ideally by tracking the simulation values in a list.

2. Machine Learning
For the rest of the semester, we will work to build a complete application for analyzing a real world dataset. We won't be building it all for one homework, that would be way too difficult. Instead, we will build a library of functions up that we will reuse to build our final application.

The dataset we will be analyzing is a real one, measured from the cells in human breast cancer tissue. By the end of the semester, we will have written a program to predict whether a given tumor is benign or malignant with 90%+ accuracy, as well as a collection of utilities for parsing and visualizing our dataset.

To start, let's implement a handful of utility functions for our dataset to practice the python skills we have learned.

```
def askConfig():
    '''
    Ask the user for the following config variables:
        filename: the name of the csv file

    Returns the information as a dictionary
    '''
```

This function will ask for values from the user using the **input** function, and return them in a dictionary

```
def parseCSV(filename):
    '''
    Read the file called "filename", expected to be a csv file.

    Return the data as a list-of-lists
    '''
```

This function will open up a csv file, and split the lines by commas returning a list-of-lists.

```
def datasetInfo(dataset):
    '''
    Takes the dataset as an N x M list of lists.

    Returns the following statistics as a dictionary:
        rows: N from above, as an integer
        columns: M from above, as an integer
    '''
```

This function will return a dictionary measuring the number of rows and columns in the list-of-lists.