CSCI 241 Data Structures
Project 1: A Battle of Sorts

The programming content is not extensive; we simply ask you to implement two sorting functions and compare their performance in different examples.

Create a file called Sort.py (note the capitalization). Within that file, write two different Python functions. Each function will take an array of integers as a parameter and sort those integers in increasing order. One will use insertion sort, and the other will use selection sort (described below). Simple functions will suffice here; do not create any classes. Your insertion sort function should be called `insertion_sort(arr)`. Your selection sort function should be called `selection_sort(arr)`.

Insertion sort works by considering each value in the array starting at position 1 (not position 0). It then determines the correct location for that value by comparing it to the values to its left, shifting right any greater values it encounters. You know you have found the correct location for the value as soon as the value to your left is not greater than what you are trying to place. This is the approach that we worked through in class, and there is a video example available on the course website.

Selection sort is slightly different. Instead of looking at a value and determining where to place it in the array, we look at a cell and determine which value should go in it. Starting at cell 0, locate the minimum value in the cells indexed from 0 to n-1. Remember the index of the cell containing the minimum value. Once you know which cell has the minimum value, swap the values in that cell and cell 0. Repeat this for cell 1, locating the minimum value in the cells indexed 1 to n-1. Repeat until you have placed a value in every cell. Like insertion sort, this has the effect of building a sorted subarray on the left hand side of the array. Unlike insertion sort, the sorted portion is never changed, because having considered every candidate for those cells, we know they have been placed correctly.

For this project to demonstrate what we want you to see, it is imperative that you structure both functions the same way. **Your `selection_sort` implementation should use a while loop nested inside of a for loop.**

In addition to your two functions, create a main section like the one at the bottom of the Caesar Cipher example from section 1.2:

```
if __name__ == '__main__':
  # main section here,
  # within this big conditional
```

In that main section, test your functions with a variety of input arrays to ensure that they are sorted correctly. Once you are confident that the functions are correct, analyze and compare their performance for three types of inputs:

- How do these sorts perform when given an array that is already sorted in increasing order?
- How do these sorts perform when given an array that is already sorted in decreasing order?
- How do these sorts perform when given an array that contains random values?

To obtain the timing for a function call, surround the call with time markers as follows:

```
start = time.clock()
ins_sort(my_array)
end = time.clock()
```

The difference between end and start is the runtime of your function in seconds. By default, Python will likely display that difference in scientific notation when you print it to the screen. That's annoying. Use the string formatter to correct this:

```
print('Ten Thousand Increasing Insertion: ' + '{:.20f}'.format(end-start))
```

This will print the difference between end and start with 20 decimal places of precision and make the values more easily comparable.

Populating a test array with increasing or decreasing values is easily accomplished with a loop. To generate a random value for a cell, assign it the return value of `random.randint(a, b)`, where `a` is the minimum value you would like and `b` is the maximum value you would like. **It is important to populate two arrays with exactly the same random values in the same order.** One will be sent to insertion sort, and the other will be sent to selection sort. Because we want to compare their performance, the contents must be the same.

Note that to use time and random, you must import them, so the first two lines of your program should be

```
import time
import random
```

Once you are satisfied that your functions and timings work correctly, generate three graphs:

1. Plot the runtime of insertion sort and selection sort for an array of increasing values.
2. Plot the runtime of insertion sort and selection sort for an array of decreasing values.
3. Plot the runtime of insertion sort and selection sort for an array of random values.

The x-axis for each graph should represent the number of cells in the array being sorted. Use 1000, 2500, 5000, 7500, and 10000 cells, so your graph should have five points for each plot. Each point should be the average of 5 runs. **Do not use a**

loop to repeat the timings five times. A single run of your program should generate thirty timings: the insertion sort and selection sort timings for each of the three distributions, at each of the five lengths. Run your program five times to obtain and average the timings.

Notice the differences between insertion sort and selection sort for the various input patterns. Create a document that contains your graphs and why they are different.

Be sure to run `P1_Submit_Check.py` in the same folder as your implementation. This checks for likely submission issues such as name mismatches, etc. You should see a message describing whether or not we see any logistical issues with your submission.

Submission Expectations

1. **`Sort.py`**: A file containing your two sorting functions and the main section that generates 30 timings. Be certain that your functions are named exactly as described above, including capitalization and underscores. Please include only your team name and no personally identifying marks in the file.
2. **`Writeup.pdf`**: A document containing three graphs, each with two plots, and a prose explanation of why the plots in each graph differ or are similar. There should be one graph for each of increasing, decreasing, and random. Each graph should contain two plots, one for insertion sort and one for selection sort. Each point on the plots should be the average of five runs. Please include no personally identifying marks in the file.