CSCI 4041, Fall 2018, Programming Assignment 4

Due Tuesday, 10/2/18, 10:30 AM (submission link on Canvas)

This is not a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or discuss the solution with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Obtaining or sharing solutions to any programming assignment for this class is considered academic misconduct. If you are not sure what this means, consult the class syllabus or discuss it with the course instructor.

A certain boarding school has hired you to manage their data. The first thing you notice is that their student roster is sorted by name, and you think that there are likely scenarios where a list sorted by house and year in school is far more useful. Each student is in one of four houses: Eagletalon, Lannister, Pufflehuff, and SNAKES. Each student's year in school is an integer between 1 and 8, inclusive (it used to be only 7, but administration found that the last year had too much content and had to be split in half). You have a CSV file containing the name, year, and house of each student, sorted by name, and you would like to produce a version of it which is sorted in the following priority: house, then year, then name (so all Eagletalon 1 students would appear at the top of the list, in alphabetical order, then Eagletalon 2, and so on).

Since the CSV is already sorted by name, you realize that all you have to do is use a stable sort to sort the CSV by year, and then use a stable sort again to sort it by house, and you'll have the format you want.

However, there's a complication. A powerful sorcerer has placed a curse on your computer that will turn you into a newt if you execute a Python file containing the characters ">" or "<", even if they're part of a comment or a string. So, you've decided to use Counting Sort, since it doesn't require any comparisons and runs in $\Theta(n)$ time.

Download the PA4.py template from the course website, the four test rosters roster0.csv, roster1.csv, roster2.csv, and roster3.csv, and their correctly sorted versions roster0sorted.csv, roster1sorted.csv, roster2sorted.csv, and roster3sorted.csv from **Canvas** (not shared publicly for student privacy reasons). The tests are all contained in a single ZIP file, PA4tests.zip, which is available in the Files tab on Canvas The template contains a Student class, which holds information about Students, and includes a function getIndex which maps the year and house attributes onto indexes from 0 to k for Counting Sort. In particular, Eagletalon maps to 0, Lannister to 1, Pufflehuff to 2, and SNAKES to 3 when sorting by house, and when sorting by year, this simply maps years 1-8 onto indexes 0-7 by subtracting one. The file also includes some code for turning the CSV file data into lists of Student objects, and test cases based on those CSV test files (make sure they're in the same folder as your PA4.py script). You'll need to implement Counting Sort, similar to the pseudocode in Chapter 8, to get the program to work.

Requirements:

- You must download the template file PA4.py and edit the `CountingSort` function. Do not edit any other part of the file.
- Your program must run without errors on the version of Python installed on the CSELabs machines, Python 3.5.2. (if you're testing this on CSELabs, you need to type python3 or idle3 instead of python or idle to start the correct version from the terminal)
- You are not permitted to use any built-in Python sorting routines like the `sorted()` function or the `.sort()` list method. You are also not allowed to use any Python function that asks for user input, such as `input()`, as this will break the grading script. Finally, you are not permitted to use the characters '>' or '<', since we don't want you to be turned into a newt.
- You must implement the Counting Sort algorithm found in Chapter 8 of the textbook. Any other sorting algorithm will receive very little credit.
- However, note that while the textbook algorithm describes how to sort a list of numbers, this problem requires you to sort a list of Student objects, so the algorithm may need some adjustments.
- In particular, note that while the textbook Counting Sort algorithm took in an input array A, an output array B, and a cap on the number of possible values k, your Counting Sort will take in an input list studentList, and a character houseOrYear, which is 'h' if we need to sort by house, or 'y' if we need to sort by year. You'll notice in the SortStudents function already written for you, CountingSort is called with 'y' first, and then with 'h', to yield a final list sorted in the correct way, and whether you're sorting by year or by house will determine what k needs to be. You will be returning the output list rather than passing it in as an argument.
- This assignment will be graded automatically based on the number of test cases your program passes. There will be several secret test cases in addition to the ones included in the template to ensure you're not hard-coding in the solutions.
- Similar to the previous homework, this program will only run test cases until you fail one, avoiding the problem of having to scroll through test output to find the one broken test case.
- The grading breakdown for this assignment is as follows:
    - 30%: File runs without syntax errors
    - 70%: Passing test cases without breaking any requirements.
- The unedited template file already runs without syntax errors (it does encounter some runtime errors on a few test cases though). This means that if your program causes syntax errors, you will get a better score by just submitting the original template unedited.
- Submit your edited PA4.py file to the Programming Assignment 4 link on Canvas before 10:30 AM on 10/2/18. No credit will be given for late submissions. Do not submit the test CSV files.

Disclaimer: The roster creation process is random, so please do not be offended if you end up in house SNAKES in all three test cases: statistically, this is going to happen to about 5 people in the class. We're not out to get you, the random number generator is.