**CSCI 4041, Fall 2018, Programming Assignment 3**

Due Tuesday, 9/25/18, 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.

You have been hired as a manager for Epicperson, the local superhero. Epicperson's greatest weakness is that they are extremely distractible: several villains have escaped justice because Epicperson received a text reminder to pick up laundry mid-battle and simply left. You have decided to implement a priority queue to handle all of Epicperson's tasks, in which you assign an importance value to each task, and ensure that Epicperson only hears about the item that is currently of highest priority.

Download the template PA3.py from the class website. The template includes a `Task` class, which consists of a description of a Task and a number representing its priority (higher values are more important). It also includes a `Heap` class, which is an extension of the normal Python list that includes a `heap_size` instance variable, and some slight alterations to a few built-in methods to differentiate between the list of Tasks, and the heap that is represented by only the first `heap_size` of those Tasks. The file also includes some test cases representing sets of tasks that Epicperson received, and the order that he should have completed them based on their relative priorities. You'll need to implement several methods which essentially mirror those in Chapter 6 of the textbook (max_heapify corresponds to Max-Heapify in the textbook, etc) in order to create a functional Priority Queue for Epicperson's Tasks.

Remember, the city is depending on you. Good luck.

Requirements:

- You must download the template file PA3.py and edit the `max_heapify`, `heap_maximum`, `heap_extract_max`, `heap_increase_key`, and `max_heap_insert` functions. 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.

- You must implement the Heap-based Priority Queue methods based on the methods of roughly the same name described in Chapter 6 of the textbook. Any other priority queue algorithms will receive very little credit, even if you pass every test case.
- However, note that while the textbook algorithms describe how to implement a priority queue for a list of numbers, this problem requires you to implement a priority queue of Task objects, so you will need to adjust the algorithm slightly.
- In particular, be careful on the difference between `heap_increase_key` and `max_heap_insert` functions: in the textbook both of these just took a number, key, as input, but here `heap_increase_key` still takes a key (representing the new priority of a given task), but `max_heap_insert` function takes a Task object as input.
- 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.
- Unlike in previous homeworks, the test cases in this one build off of each other as you create a Priority Queue and add and extract tasks from it. Therefore, if a given test doesn't work, no subsequent tests will run. This should also make it easier to debug, since the last test to run will always be the broken one.
- 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 PA3.py file to the Programming Assignment 3 link on Canvas before 10:30 AM on 9/25/18. No credit will be given for late submissions.