Lab 2 - Description

(Max-Heap Priority Queue)

Lab Overview:

For this lab, we will be taking what we've learned so far up a notch and learning how to implement a max-heap priority queue. Priority queues are another commonly used Abstract Data Type (ADT) due to the fact that they are self-sorting (they sort themselves) and that their time complexity ranges from constant O(1) to linear O(n) depending on the operation and how they are implemented. Our central task for this lab is to implement the max-heap variety of priority queues.

Core Tasks:

1. Write the *PriorityQueue* class.

Program Requirements:

Task 1: Write the priority queue class.

For task 1, you will need to write a *PriorityQueue* class. The class is detailed as follows:

- $\_\_init\_\_(\text{self}, \text{capacity}) \rightarrow \boldsymbol{None}$
  - **Complexity**: O(1)
  - **Valid Input:** An integer in (0, ∞]. Use a default, valid capacity on invalid input.
  - **Description:** This is the constructor for the class. You will need to create the following instance variables here:
    - A variable *_maxSize*
    - A variable *_currentSize*
    - A variable *_heap* (contains an empty list)
    - Any other instance variables you want.
- $\_\_str\_\_(\text{self}) \rightarrow \boldsymbol{string}$
  - **Complexity:** O(n)
  - **Description:** The string method returns a string representation of the queue. **Use the function provided in the main.**
- $\text{enqueue}(\text{self}, ticket) \rightarrow \boldsymbol{Boolean}$
  - **Complexity:** O(lg(n))
  - **Valid Input:** An object of type: MealTicket. Return False on invalid input.
  - **Description:** This method will add a ticket to the queue based on its priority.
    - **Note 1:** The priority of a ticket is determined by its ID.
    - **Note 2:** When adding a node to your heap, remember that for every node *n*, the value in *n* must be greater than or equal to the values of its children, but your heap must also maintain the correct shape. (i.e., for any node there can be at most two children, and the parent has a greater priority than all subsequent nodes.)

- dequeueMax(self) → ***Boolean|MealTicket***
  - **Complexity:** O(n lg(n)) or less
  - **Description:** This this method will remove and return the ticket with the highest priority. If the queue is empty, then it will return False.
- peekMax(self) → ***False|MealTicket***
  - **Complexity:** O(1)
  - **Description:** This method lets the user peak at the ticket with the highest priority without removing it from the queue. Returns false if the queue is empty.
- $isEmpty$(self) → ***Boolean***
  - **Complexity:** O(1)
  - This method will return True or False depending on if the queue is empty or not.
- $isFull$(self) → ***Boolean***
  - **Complexity:** O(1)
  - This method will return True or False depending on if the queue is full or not.

Note 1: The methods listed above are the only methods that will be tested. However, you may add additional methods as you please. I would recommend the following:

- _heapify – to restructure your heap.
- _swap – to swap parent and child nodes.
- _getParent – to get the parent nodes index.

Note 2: See Figure 1 at the bottom of this section for a sample test run of the Queue class. The provided main can be used to achieve this exact output which may be helpful for testing.

Note 3: Pay attention to the complexity bounds mentioned in the method descriptions. Your implementation **must** run within these bounds.

Note 4: For this assignment you **must** use a list to implement your heap. The use any other built-ins data structures are explicitly forbidden. Using them will result in a 0 for the assignment. The same goes for method annotations (e.g. def isEmpty(self) -> int:).

Extra Credit: If you are looking for general bonus points (10%): Implement heap-sort (you will already have a max-heapify). The details of this method are given below:

- heap_sort(self, array) → ***Boolean|List***
  - **Complexity:** O(n lg(n))
  - **Valid Input:** A list of MealTickets. Return False on invalid input.
  - **Description:** This method will sort the given list of MealTickets.
    - **Note 1:** Extra credit is all or nothing.

I/O Specifications and Program Robustness:

For this assignment 2 files are provided: lab2-main.py and mealticket.py. You are free to use the main provided or to make your own. Inside the main you will find the method for the __str__ class. Copy-paste this method. Your string output must be exact, so use this method.

Warning: I will be testing your code with a more robust main that will check all of the corner cases and uses a wider variety of tickets. Keep this in mind while developing your data

structures. The provided main tests some of the corner cases but not all of them. You will need to use the methodology discussed in lab to figure out all corner cases and account for them. Your programs **must** be robust (i.e. crash on bad input or calls).

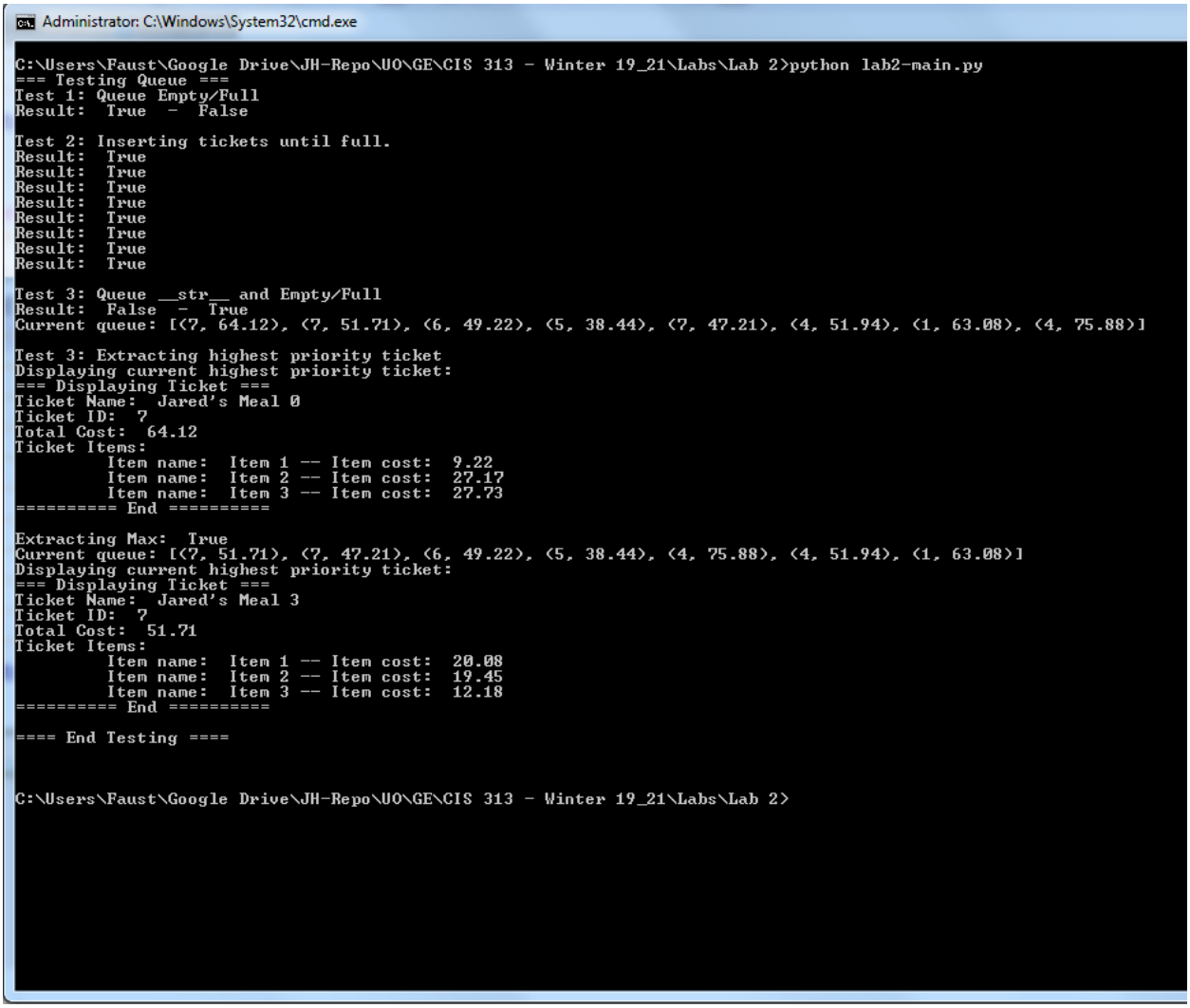

Figure 1: Queue testing output

Remarks:

All programs written in this class will be done using the newest version of Python available in the lab (Python 3.7/3.8). This is because Python 3.5+ is platform independent (i.e. you can code on a PC and run it on a Mac). The computers in KLA 26 all have python on them so make sure your code runs on the computers in the lab as it will be tested on an identical system. Feel free to install python on your own computer and bring it to the lab.

If you are new to python, here are some super helpful resources to get you up to speed:

1. Cheat sheets: https://www.pythoncheatsheet.org/ ← Most recommended! Many thumbs up!
2. Python documentation: https://docs.python.org/3/

All programming assignments are to be done **individually**. Your code will be looked at with professional software for cheating. ***Warning:*** This includes using online sources. (e.g. Do not go online and copy code from stack overflow. People have tried this before. You will fail.) Be extra careful with your code. Do not ever show your work to anyone other than the TA (me) or the professor. They will most likely copy your work and your will both fail.

Submission Requirements:

In order to receive any credit for the assignment the student ***must*** do the following:

1. Name your program **"<Duck-ID>_lab2.py"**. (i.e. my duck ID is jhall10 so my submission would be named **jhall10_lab2.py**. **Note:** your duck-ID is the same as your email id and the username to log on to CIS computers ***not*** your 951… number that is your UO PID.
2. Submit your python file onto Canvas.
3. That's it! Make sure that you test your code to make sure it works.

Grading:

Your work will be graded along three primary metrics: Correctness, Completeness, and Elegance.

Correctness: (60% of total grade)

- You wrote the class methods as specified.
- Your class methods meet the complexity requirements.
- You utilize a list to build your heap.
- You implement the correct algorithms.
- You implemented the queue with a max-heap.
- Your classes are robust, fault tolerant and follow the specified behavior on invalid input.

Completeness (25% of total grade)

- Program contains a class named: *PriorityQueue*
- The class contains methods as defined above.
- The method signatures were implemented as specified.

Elegance: (11% of total grade)

- See grading guide posted on canvas for information on elegance.

Late Policy:

The late policy for this class is a bit unique. Make sure to read the info below carefully:
Your homework is always due on a Tuesday at 11:59 pm. On Wednesday, I will take 10% off of the total points available. On Thursday, I will take 20% off. On Friday 30%, no homework will be accepted after Friday. I do not start grading the homework until the following Monday. Keep

this in mind when submitting your assignment: It may be better for you to seek help and submit the homework a day late than to submit on time and fail.

If you encounter an unfortunate event or are working with a disability: Please email or speak to me. I am super flexible and am always on your side. I will give extensions as needed and am willing to work with you to make sure you get the most out of this course.