|  |  |
| --- | --- |
| **CSE 331** | **Semester** |

*Project 3*

Authors of this project are Yash Vesikar and Danny Marshall

**Assignment Overview**

The purpose of this project is to gain a basic understanding of the uses of priority queues and max heaps. Your job is to implement a max heap using the given function signatures. The heap ADT in this project is supposed to mimic the job of an air traffic controller, where the priority of landing a plane is determined by the key and the time spent in the air.

**Assignment Deliverables**

Be sure to use the specified file name(s) and to submit your files for grading **via D2L Dropbox** before the project deadline.

* heap.py

**Assignment Specifications**

Describe what needs to be completed

For example :Your task will be to complete the method listed below.

* Class Plane
  + def \_\_gt\_\_(self, other)
    - This function defines which plane is greater than another. If the keys of 2 planes are the same then compare the amount of time the planes have been in the air. Otherwise only compare the key values.
    - Time complexity: O(1)
* Class Heap
  + def \_parent(self, j)
    - Return the position of the parent given j - position of current Plane
    - Time complexity: O(1)
  + def \_left(self, j)
    - Return the position of the left child given j - position of current Plane
    - Time complexity: O(1)
  + def \_right(self, j)
    - Return the position of the right child given j - position of current Plane
    - Time complexity: O(1)
  + def \_has\_left(self, j)
    - Return True if the current Plane has a left child given j - position of current Plane
    - Time complexity: O(1)
  + def \_has\_right(self, j)
    - Return True if the current Plane has a left child given j - position of current Plane
    - Time complexity: O(1)
  + def \_swap(self, i, j)
    - Swap the positions of 2 Planes in a python list
    - Time complexity: O(1)
  + def \_upheap(self, j)
    - Percolate up an Plane to the correct position in the heap given the position of the Plane - j.
    - Time complexity: O(logn)
  + def \_downheap(self)
    - Percolate down an Plane to the correct position in the heap ADT given the position of the Plane - j.
    - Time complexity: O(logn)
  + def is\_empty(self)
    - Return True of the heap is empty, False otherwise
    - Time complexity: O(1)
  + def add(self)
    - Add an Plane to the correct location in the heap ADT.
    - Time complexity: O(logn)
  + def max(self)
    - Return the Plane with the highest priority in the heap. DO NOT remove the Plane, simply return the Plane itself.
    - Time complexity: O(1)
  + def remove\_max(self)
    - Remove the Plane with the highest priority from the heap and return the Plane.
    - If the queue is empty, return None
    - Time complexity: O(logn) \*
  + def increase\_time(self)
    - The function increases the time each Plane has been waiting by 1. The time a plane has been waiting is incremented every time a plane is landed (removed) from the heap.
    - Time complexity: O(n)
  + def increase\_priority(self)
    - This function increases the priority (key) of every plane that has been in the heap ADT for longer than the average time currently in the heap ADT. The value that each incremented key increases by is 1/2 the value of the max key.
    - Time complexity: O(n)

\* Indicates that this function runs in amortized time

You can make additional helper functions, if useful.

**Assignment Notes**

Points will be deducted if your solution has any warnings of type:

* The newest distribution python 3.6 intrepreter will be used to execute your solution.
* The time of all planes is incremented every time a plane is removed from the heap.
* Multiple Planes can have the same key value, but may have the same or different time values. It is important to recognize that when writing the \_\_gt\_\_ method the object should always be compared to the other object.
* The \_\_init\_\_ methods and \_\_repr\_\_ method should not be edited in any way.
* You are required to complete the docstrings for any unmade and created function signatures.
* To test your classes, main.py is provided. Compare your results to the output below.
* Errors when using your solution that cause the grading script to fail will result in a 25% deduction.

