|  |  |
| --- | --- |
| **CSE 331** | **Semester Spring 2018** |

Project XX

Authors of this project are David Osinski and Rahul Yalamanchili

**Assignment Overview**

Your job is to create a max heap. The data set should be able to insert, remove, and get information from the heap. In addition to this, you will use the data structure to implement a heap sort.

**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**

Your task will be to complete the methods listed below.

* get\_size()
  + Returns number of items currently in the Heap
  + Returns boolean
  + Time complexity: O(1)
  + Space complexity: O(1)
* has\_left()
  + Indicates if a node has a left child
  + Returns bool
  + Time complexity: O(1)
  + Space complexity: O(1)
* has\_right()
  + Indicates if a node has a right child
  + Returns boolean
  + Time complexity: O(1)
  + Space complexity: O(1)
* insert(value)
  + Creates a node with the given value and adds it to the Heap
  + Returns nothing
  + Time complexity: O(1)\*
  + Space complexity: O(1)\*
* remove(value)
  + Removes the node with a given value, does nothing if value not present
  + Time complexity: O(1)
  + Space complexity: O(1)
* heapify()
  + transforms self.list so it meets the requirements for a list based representation of a max heap
  + current\_list may be completely unsorted or closely resembling a max heap
  + Time complexity:
  + Space complexity:
* swap(i, j)
  + Swaps the elements at indices i and j of a list
  + For use in heapify function
  + Time complexity: O(1)
  + Space complexity: O(1)

In addition to these functions, you must also implement the following functionality:

* heapSort(unsorted)
  + Given an unsorted list, performs a Heap Sort
  + Returns a sorted list of elements
  + Time Complexity:
  + Space Complexity:
* get\_stats(unsorted)
  + Given an unsorted list of integers, returns a list of information in the following format: [minimum val, maximum val, mean, median, mode]
  + mean should be of type float, all others type int
  + Time complexity: O(1)
  + Space complexity: O(1)

\*refers to amortized time, or average case performance when the operation is done many times. Normally, inserting an item on the HashTable takes constant time, until the case where you have to grow it. Each grow/shrink operation takes O(N) time, but you’ve waited longer to do it (until λ is approaching 1.00), so the cost is “spread out” among the prior insertions/removes.

You will also need to implement the following function:

* string\_verify(key, value)
  + Checks to see if key is already in table
  + If present, compares value within table against value passed into function
  + Returns boolean; True if values are the same and False otherwise
  + Time complexity: O(1)
  + Space complexity: O(1)

**Assignment Notes**

* You can make additional helper functions, if useful
* Use of python list methods is allowed
* All functions that you edit / create must have docstrings with pre and post conditions.

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

* The newest distribution python 3.6 interpreter will be used to execute your solution.
* A HashTable that at any time is **λ** *>* **1.00** will result in points lost.
* 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.
* You may not change any function signatures in anyway, which include class definitions.
* Your solution will be running against 10 test cases checking for various edge cases against your solution.

Testing your work

Run your project on Pycharm see sample run below

Below are the results to testcases 1, 2, and 3

