|  |  |
| --- | --- |
| **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 min 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)
* parent(i)
  + Finds the parent of a node at index i
  + Returns the index of the parent node
  + Time complexity: O(1)
  + Space complexity: O(1)
* left(i)
  + Finds the left child of a node at index i
  + Returns the index of the left child node
  + Time complexity: O(1)
  + Space complexity: O(1)
* right(i)
  + Finds the right child of a node at index i
  + Returns the index of the parent node
  + Time complexity: O(1)
  + Space complexity: O(1)
* has\_left(i)
  + Indicates if a node at index i has a left child
  + Returns boolean
  + Time complexity: O(1)
  + Space complexity: O(1)
* has\_right(i)
  + Indicates if a node at index i 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(logn) \*1
  + Space complexity: O(1) \*2
* remove(value)
  + Removes the node with a given value, does nothing if value not present
  + Time complexity: O(logn) \*1
  + Space complexity: O(1) \*2
* swap(i, j)
  + Swaps the elements at indices i and j of a list
  + Time complexity: O(1)
  + Space complexity: O(1)
* upHeap(i)
  + Moves node at index i up the tree via swaps until it is in the proper position
  + Time complexity: O(logn) \*1
  + Space complexity: O(1) \*2
* downHeap(i)
  + Moves node at index i down the tree via swaps until it is in the proper position
  + Time complexity: O(logn)
  + Space complexity: O(1) \*2
* remove\_min()
  + Removes and returns the root node, then updates list so it remains a MinHeap
  + Returns: the node with the minimum value
  + Time complexity: O(logn) \*1
  + Space complexity: O(1) \*2

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: O(nlog(n)) \*1
  + Space Complexity: O(1)
* get\_stats(unsorted)
  + Given an unsorted list, returns a list of information in the following format: [minimum val, maximum val, mean, median, mode]
  + Elements in the unsorted list will be either all ints or all strings. If strings, return the character that represents the average ASCII value of the inputs (truncated down) for the median
  + mean should be of type float, all others type int
  + Time complexity: O(nlog(n)) \*1
  + Space complexity: O(1)

\*1 refers to amortized time, or average case performance when the operation is done many times. On some occasions, inserting or removing an element from a heap will take a maximum number of swaps can equal the height of the tree, while other times a best case of 0 swaps will be needed.

\*2 indicates that the function should be in place. This means you may only edit the pre-existing list object and are NOT allowed to create a new list object(s).

**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.
* 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