Submission Instructions Please a single single text file (.py) with a single python file that contains the class MinHeap. If additional classes are necessary, they can be included as well (i.e. HeapNode).

`tcss501-hw03.py`

Implement a MinHeap(20 pts)

A MinHeap is a common data structure used for for efficient storing of ordered information in such a way that the lowest value in the data structure can be accessed in constant $O(1)$ time, removed in logarithmic $O(\log n)$ time, and insertion of values can be done in logarithmic $O(\log n)$ time. The Heap should support numeric data types (integers, floats) and string values. It needn't support mixed data types (i.e. some elements being strings and others being integers).

To meet the requirements of this assignment, you may implement any internal / helper methods as desired, but must include the following method signatures as an interface. While there are methods of implementing a Heap using a dynamic array or list, implement your Heap using a Binary Tree with parent and child nodes.

```
def pop(self):
""" Returns and removes the minimum value from the heap.  If the heap is empty, return None.
    Must be implemented with no worse than O(log n) time complexity. """

def peek(self):
""" Return but do not remove the minimum value of the heap.  If the heap is empty, return None.
    Must be implemented with no worse than O(1) time complexity. """

def insert(self, data):
""" Insert the provided data onto the heap.  Should support integers and strings at least.
    Must be implemented with no worse than O(log n) time complexity. """

def clear(self):
""" Removes all elements from the heap.  Must be implemented in no worse than O(1) time. """
```

The following explains how pop, peek and insert should function.

.peek() Return the Root Node of the Data Structure

.pop() Store the value of the Root Node. Replace Root Node with the value of the "last element" and push the element down as far as necessary in the heap.

.push(data) Create a new node in the appropriate location in the tree (last node of last level). Bubble the element up as far as necessary in the heap.

Note: Remember that a Binary Heap is a complete tree, any new nodes are added from left to right on the last level of the tree. By maintaining the tree's count, you will be able to tell what level and what branch of the tree the node is in, or maintaining a link to the "last element" will give you direct access to the node.