CPE 202

Lab 8: Implement a MaxHeap class to contain Integers

a. **Implement a class MaxHeap** (a maximum binary heap) that contains integers that represent both the priority and name of the object.

- Implement class MaxHeap of integers:
  - **def __init__ (self, capacity=50)** Constructor creating an empty heap with default capacity = 50 but allows heaps of other capacities to be created.
  - **def enqueue (self, item)** inserts "item" into the heap, returns true if successful, false if there is no room in the heap
  - **def peek (self)** returns max without changing the heap
  - **def dequeue (self)** returns max and removes it from the heap and restores the heap property
  - **def contents (self)** returns a list of contents of the heap in the order it is stored internal to the heap. (This may be useful for in testing your implementation.)
  - **def build_heap (self, alist)** Method build_heap that has a single explicit argument "list of int" and builds a heap using the **bottom up method** discussed in class. If the capacity of the heap is not sufficient to hold the elements of alist, increase the capacity of the heap as necessary.
  - **def is_empty (self)** returns True if the heap is empty, false otherwise
  - **def is_full (self)** returns True if the heap is full, false otherwise
  - **def get_capacity (self)** this is the maximum number of a entries the heap can hold - 1 less than the number of entries that the array allocated to hold the heap can hold.
  - **def get_size (self)** the actual number of elements in the heap, not the capacity
  - **def perc_down (self, i)** where the parameter i is an index in the heap and perc_down moves the element stored at that location to its proper place in the heap, rearranging elements as it goes.
  - **def perc_up (self, i)** where the parameter i is an index in the heap and perc_up moves the element stored at that location to its proper place in the heap, rearranging elements as it goes.
    Since perc_down/perc_up are internal methods, we will assume that the element is either in the correct position or the correct position is below/above the current position.
    Note that in order to comprehensively test these functions you will need to either directly access the heap list or provide a set_heap(self, alist) function that will set the heap contents (and size) to the list that is passed in.

b. **Add a function,** heap_sort_asecending (self, alist) to perform heap sort given a list of positive integers.

- **heap_sort_ascending (self, alist)** takes a list of integers and returns a list containing the integers in ascending (lowest to highest) order using the Heap Sort algorithm as described in class. As each max item is dequeued, the value can be placed in the "vacated" list spot, which enables reuse of the list space. The sorted values will occupy the array locations that are beyond the active portion of the heap array. Since your MaxHeap class is a max heap, using the list internal to the heap to store the sorted elements will result in them being sorted in ascending order. After all items have been dequeued (can actually stop when one item remains, since it is in the correct location), you can just return the appropriate part of the internal list since you will not be using the heap anymore.

Submit a file **heap_lab.py** containing the above and a file **heap_file_tests.py** containing your set of test cases. Your test cases should test all the functions. Some are quite simple to test but some will require multiple test cases. Again, developing test cases as you develop each function will save you time overall.