Lab 7

In this lab, you will explore one implementation of a priority queue. Additionally, you will implement a sorting algorithm based on the data structure.

1 Maximum Binary Heap

1.1 Properties

A maximum binary heap is a binary tree structure (*not* a BST) with two additional properties:

- The structure property: a binary heap is a *complete* binary tree. This means that every level of the tree, except possibly the last, is fully filled. The last level of the tree must be filled in from left to right.
- The heap property: the value at each node is *greater than or equal to* the values of its children. (Note that this ordering would be reversed for a minimum binary heap.)

1.2 Our Implementation

Rather than reinvent array-based lists again, in this lab, you'll be using Python lists with Python's list methods to store the underlying tree structure. You can (and should) use `list.append(item)` to add to the end of a list, `list.pop()` to remove (and return) the last item in a list, and `len(list)` to return the length of a list.

You *should not* import *any* Python standard library modules into your code.

As discussed in class, index 0 of the list should be left blank to allow for easier formulas when "traversing" the tree.

1.3 Operations

You will implement the following operations for a maximum binary heap:

- `enqueue` This function takes a maximum binary heap and an item as arguments and adds the item to the heap in the appropriate position. After the function finishes, the heap should still satisfy the above properties. This function will not return anything.

  This function should have $O(\log n)$ worst case and $O(1)$ average case time complexity (as discussed in class).
- `dequeue` This function takes a maximum binary heap as an argument and removes (and returns) the largest item from the heap. After this function finishes, the heap should still satisfy the above properties.

  If the heap is empty, this function should raise an `IndexError`.

  This function should have $O(\log n)$ worst case time complexity.
- `peek` This function takes a maximum binary heap as an argument and returns (without removing) the largest item in the heap.

  If the heap is empty, this function should raise an `IndexError`.

  This function should have $O(1)$ time complexity.

- `size` This function takes a maximum binary heap as an argument and returns the number of items in the heap. (Note that this will not be the size of the underlying list, but it will be close.)

  This function should have $O(1)$ worst case time complexity.

- `_contents` This function takes a maximum binary heap as an argument and returns a list of the contents of the heap in the order they are stored in the heap's internal list. The return value should only include the "filled" spots in the list.

  You may find this useful for testing.

  Note that the leading underscore is not a typo. It's generally bad practice for a function like this to exist, and so we're using the Python standard practice of a leading underscore to mean that it's not meant to be used directly by "users" of your `MaxHeap` structure. You should only be using this for testing.

While not explicitly specified, you will likely find it helpful to write separate functions for sifting (percolating) up/down.

1.4 Other Functions

You will additionally implement the following functions related to the heap structure:

- `heapify` This function takes a list as an argument and builds and returns a maximum binary heap out of the items in the list. The input list should not be modified in any way.

  You *should not* implement this by repeatedly enqueuing each item in the list. Rather, you should use the heapify algorithm discussed in class, where we repeatedly sift (percolate) items down, from the bottom up.

  This function should have $O(n)$ worst case time complexity.

- `heap_sort` This function takes a list as an argument and mutates the list to be sorted in *ascending* order. It will do this by building a heap with the items in the list and them dequeuing them back into the given list.

  This function should have $O(n \log n)$ worst case time complexity.

2 Testing

As mentioned in the syllabus, your code is periodically graded prior to the deadline and you will receive automated feedback based on the results of my tests. *But*, in order to receive *any* feedback, your tests much provide 100% test coverage of the code you are submitting. If you do not have 100% coverage, this is the only feedback you will receive.

100% test coverage means that every line of your code is run at some point in some test. Imagine if that weren't the case. That means that you have a line of code (or multiple lines of code) that could do anything, and you'd never know. You never tested them!

Your tests will go in files named `heap_tests.py`.

3 GitHub Submission

Push your finished code back to GitHub. Refer to Lab 0, as needed, to remember how to push your local code.