Priority Queues

In this lab, you will tweak the priority queue code in a way that the priority of items are defined via a key function. Also, you implement methods to merge two priority queues and verfiy if your data structure satisfies the definition of priority queues.

The Priority Queue ADT

A `PriorityQueue` contains a list of objects with priorities and maintains this list in a heap order. You can think of a heap as a tree that is arranged so that objects with smaller priorities are above objects of larger priorities. So, the object with the minimum priority is at the root or equivalently at index 0 of the list.

The priority of items in a `PriorityQueue` is determined through a key function. A key function takes an object and returns a comparable object. For example, in the following code, `k` is a key function.

```
L = [(3, 2, 10), (4, 6, 1), (7, 1, 10), (5, 5, 5)]
k = lambda x: 1/x[1]
for item in L:
  if k(item) < 0.5:
    print(item)
```

```
(4, 6, 1)
(5, 5, 5)
```

A `PriorityQueue` has the following ADT.

- `__init__(self, entries=None, key=lambda x: x)` - Takes a list of objects and a key function and stores them internally. It also creates a priority queue on the entries using the key function.
- `insert(self, item)` - Inserts an item into the priority queue.
- `_parent(self, i)` - Takes the index of an item and returns the index of its parent.
- `_children(self, i)` - Takes the index of an item and returns the indices of its children.
- `findtop(self)` - Returns the root of the priority queue.
- `removetop(self)` - Removes the root of the priority queue and returns it.
- `_swap(self, a, b)` - Takes two indices and swaps their corresponding items.
- `_upheap(self, i)` - Takes an index and shifts the item at that index upward until it finds the right place for that item.

- `_downheap(self, i)` - Takes the index of an item and shifts that item downward until it finds the right place for that item.
- `__len__(self)` - Returns the length of the priority queue.
- `_heapify(self)` - Rearranges the items in `PriorityQueue` so that they become in a heap order.
- `update(self, other)` - Receives `other` as another `PriorityQueue` and updates the current `PriorityQueue` with the items in `other` .
- `_isheap(self)` - Returns `True` if the items in `PriorityQueue` are in a heap order and `False` otherwise.

What to do

Implement `_upheap(self, i)` , `_downheap(self, i)` , `update(self, other)` , and `_isheap(self)` methods of `PriorityQueue` class in `priorityqueue.py` file. Here the goal is implementing `_upheap(self, i)` and `_downheap(self, i)` in $O(\log n)$ time and `update(self, other)` and `_isheap(self)` in $O(n)$ time.