This lab gets you to work with heaps.

Marks (max 5): Questions 1-2: 1.25 each | Questions 3-5: 0.5 each | Question 6: 1

Question 1

This question is about understanding heaps.

a) Draw the heap we obtain if we start from the empty tree and add consecutively the numbers:

42, 21, 40, 3, 56, 99, 58, 21, 46, 49

b) Write down the numbers of the heap you constructed, starting from the root using depth-first search, and using breadth-first search

c) Draw the heap we obtain if we remove its root, using the technique followed by the removeRoot function that we saw in the lecture (week 10).

Question 2

Add in `Heap` the following functions, assuming that we work with heaps that store integers:

a) `def max(self)`

that returns the largest element of the heap.

b) `def getRelative(self, pos, flag)`

that returns the "relative" of the element in position `pos` of the heap (in its array list representation) according to the value of `flag`:

- if `flag` is 0 then it should return the parent of the element
- if `flag` is 1 then it should return the left child of the element
- if `flag` is 2 then it should return the right child of the element

c) `def sumAll(self)`

that returns the sum of all the elements of the heap. The heap remains unchanged.

Question 3

Add in `Heap` a function

```
def min(self)
```

that returns the smallest element of the heap. Your function should search between all the leaves of the tree and find the smallest element – it should not look at internal nodes (i.e. nodes that are not leaves).

Hint: you can do a simple linear search in the array list storing your heap, making sure you only search the part of the array list that is between the last element and its parent.

Question 4

Add in `Heap` a function

```
def removeVal(self, d)
```

that removes (one occurrence) of `d` from the heap.

Hint: you can simply do linear search to find `d` in the heap. What is more difficult is to make sure that, after removal, your tree remains a heap. Using the technique of removeRoot alone will not be enough!

Question 5

Write a Python function

```
def heapsort3(A)
```

that sorts the array `A` using the heapsort technique *in-place*, i.e. by not creating an external heap but, rather, by forming the heap inside `A` (i.e. by transforming `A` itself into a heap) and then reading out in order the elements from `A` back to `A`.

For the next question, we look again at priority queues. A priority queue is a queue in which each element has a priority, and where dequeueing always returns the item with the greatest priority in the queue.

We start by defining a class of priority queue elements (PQ-elements for short):

```
class PQElement:
      def __init__(self, v, p):
            self.val = v
            self.priority = p
```

So, a PQ-element is a pair consisting of a value (which can be anything, e.g. an integer, a string, an array, etc.) and a priority (which is an integer).
In `lab7.ipynb` we also implemented the `__str__` function to be able to print PQ-elements.

Question 6

Write a Python class `PQueue` that implements a priority queue using a heap of `PQElement`'s. In particular, you need to implement 5 functions:

- one for creating an empty priority queue
- one for returning the size of the priority queue
- one for enqueueing a new PQ-element in the priority queue
- one for dequeueing from the priority queue the PQ-element with the greatest priority
- one that prints the elements of the priority queue into a string (call this one `__str__`)

Test each of the functions on examples of your own making. For example, running:

```
pq = PQueue()
for i in range(15):
    pq.enq(PQElement(i,10-i))
print(pq)
print(pq.deq(),pq)
```

should give this printout:

```
[(14,-4),(13,-3),(12,-2),(11,-1),(10,0),(9,1),(8,2),(7,3),(6,4),(5,5),(4,6),(3,7),(2,8),(1,9),(0,10)]
(0,10) [(14,-4),(13,-3),(12,-2),(11,-1),(10,0),(9,1),(8,2),(7,3),(6,4),(5,5),(4,6),(3,7),(2,8),(1,9)]
```

Note: the print function should print the queue elements in order, without changing the queue. One idea is to create a function `toSortedArray` in `Heap` to help you in that.