ECS529U Algorithms and Data Structures **Week 9 Lab sheet**

This lab gets you to work with binary trees and binary search trees in particular.

Marks (max 5): Questions 1-2: 1.5 each | Questions 3-4: 0.5 each | Question 5: 1

Question 1

This question is about understanding binary search trees (BSTs).

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

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

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

c) Let `t` point to the root node of the BST you constructed in part a. Draw the BST that results by applying each of the following sequences of operations:

   i. `t.left = t.left.right`

   ii. `t.left.right.right = t.right.right`

   In each of these cases, is the resulting structure a binary tree? Is it a binary search tree?

d) Using `t`, write code that (each time starting from the tree constructed in part a):

   i. removes the node containing 3
   ii. removes the node containing 40
   iii. removes the node containing 56

   In each case, `t` should remain a BST.

Question 2

Add in `BST` the following functions, assuming that we work with BSTs that store integers:

a) `def min(self)`

   that returns the smallest element of the tree.

b) `def _sumAllRec(self, ptr)`

   that uses recursion and returns the sum of all the elements of the subtree starting from the node `ptr`.

   Hint: you can simply use depth-first search, and ignoring the fact that this is a BST rather than a simple binary tree.

c) `def sumAll(self)`

   that sums all the elements of the tree (use the function from part b).

d) `def sumAllBFS(self)`

   that sums all the elements of the tree using breadth-first search.

   Hint: you can adapt the code for breadth-first search that we saw in the lecture (week 9). You will need to use a queue (see lecture of week 8).

Question 3

Write a function

```
def inOrderPrint(t)
```

that prints the elements of the tree `t` in the following manner. At each node of the tree, it first prints all nodes of the subtree under the node's left child, then it prints (the data of) the node itself, and then the nodes of the right subtree. For example, applied on the tree of slide 20 (week 8 lecture), it would print its element in this order:

15, 5, 51, 2, 21, 42, 8, 18

Hint: you can implement this by first recursively calling `inOrderPrint` on the left subtree of `t`, then printing the contents of `t`, and then calling `inOrderPrint` on the right subtree.

Note: inorder is essentially depth-first search, only that we do not look at a node as soon as we get to it, but only after we have looked at all the nodes in the subtree that is under its left child. In fact DFS as we saw it is called **preorder**, while there is also **postorder** (guess how that works!).

Question 4

Add in BST a function

```
def toSortedArray(self)
```

that returns an array containing the elements of the tree in ascending order.

Hint: Use a helper function to do an inorder traversal of the BST.

Question 5

You are asked to write a class `BSTQueue` which implements a BST and a Queue at the same time. That is, a BSTQueue is an object containing a BST whose elements are also stored in a queue. Here is the constructor you can use:

```
class BSTQueue:
    def __init__(self):
        self.root = None
        self.queue = Queue()
        self.size = 0
```

Adding and removing elements to a BSTQueue should be done by enqueuing and dequeuing respectively, while searching and counting should be done by using the BST. In particular, you are asked to implement the following functions:

- `search(self,d)` for searching the value `d` in the BSTQueue. This should use BST search and return True if the value is found, otherwise false.
- `count(self,d)` for counting the times the value `d` appears in the BSTQueue. This should use BST search and return the number of times that the value appears in the BSTQueue.
- `enq(self,d)` for adding an element `d` in the BSTQueue. The element should be added at the tail of the queue and at its right position in the BST.
- `deq(self)` for returning the element at the head of the queue. The element should be removed both from the queue and the BST.