UNIVERSITY OF TORONTO

Faculty of Arts and Science

Midterm 2

CSC148H1F

November 12, 2014 (**50 min.**)

Examination Aids: Cheat sheet on back, detachable!

Name: Rui Qiu

Student Number: 999290509

Please read the following guidelines carefully!

- The last page just has an aid sheet; detach this for your convenience during the exam.
- Please write your name on the front **and back** of the exam. The latter is to help us return the exams.
- This examination has **4** questions. There are a total of **8 pages, DOUBLE-SIDED**.
- Any question you leave blank or clearly cross out your work and write "I don't know" is worth **10% of the marks**.

Take a deep breath.

This is your chance to show us

How much you've learned.

We **WANT** to give you the credit

That you've earned.

A number does not define you.

Good luck!

1. [5] Write a function **filter_pos_rec**, which takes a *recursive* linked list, and outputs a new recursive linked list containing the same items as the original, except with the items less than or equal to 0 removed.

You may not use a loop, nor any LinkedListRec methods other than the constructor and is_empty in your solution. Instead, use recursion and access attributes directly. (Remember, recursive code is pretty short: you can do this in under 6 lines!)

```
def filter_pos_rec(lst):
    """ (LinkedListRec of int) -> LinkedListRec of int

    Return a new LinkedListRec whose items are
    the ones in lst that have value > 0.
    The items must appear in the *same order*
    they do in lst.

    >>> lst = LinkedListRec([3, -10 , 4, 0])    # [3 -> -10 -> 4 -> 0]
    >>> pos = filter_pos_rec(lst)               # pos is [3 -> 4]
    """
```

```
~~if lst.is_empty():~~
    ~~return LinkedListRec([])~~
~~else:~~
    ~~result = LinkedListRec([])~~ []
if lst.is_empty():
    return ~~lst~~      -1
else:
    if lst.first>0:
        filter_pos_rec(lst.rest)
    else:  # lst.first <=0
        lst.first = lst.rest.first
        lst.rest = lst.rest.rest          -1
        filter_pos_rec(lst)
return lst
```

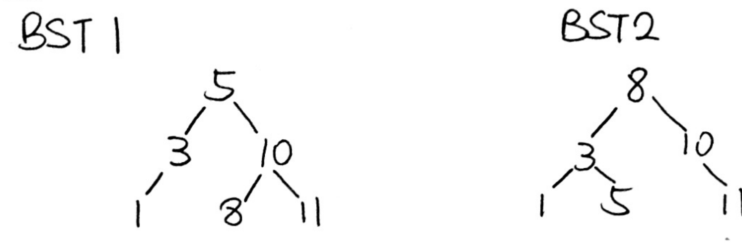
nothing done with the return value *

* lst should not be modified

3

```
if lst.is_empty():
    return LinkedListRec([])
elif lst.first > 0 :
    new_lst = LinkedListRec([lst.first])
    new_lst.rest= filter_pos_rec(lst.rest)
    return new_lst
else:
    return filter_pos_rec(lst.rest)
```

2. (a) [2] Draw two **different** binary search trees which both contain the items `1,3,5,8,10,11`. Label them **BST1** and **BST2**.



2

(b) [2] Write the list returned from the **preorder traversal of BST1**, and then the list returned from the **postorder traversal of BST2**. Clearly state which one is which!

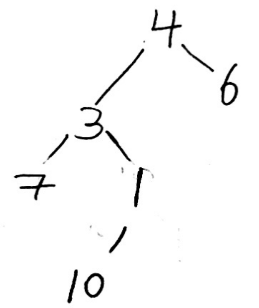
Note that you will receive a *zero* for this question if you haven't labelled your trees in part (a)!

preorder BST1 : 5 , 3, 1, 10, 8, 11 ✓

postorder BST2: 1, 5, 3, 11, 10, 8 ✓

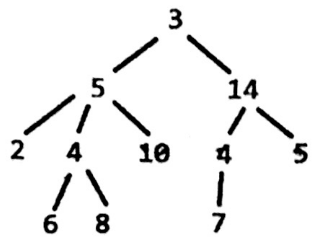
2

(c) [3] A mystery binary tree (*NOT* a binary search tree) has preorder traversal **[4,3,7,1,10,6]** and inorder traversal **[7,3,10,1,4,6]**. Draw this binary tree. (*Warning*: this question is a little tricky; if you get stuck, move on!)



3

3. Recall that the **depth** of a node in a tree is its distance from the root (where the root has depth 1). For example, the tree below has five nodes at depth 3:



In this question, you will write a function to determine the number of nodes in a tree of a certain depth.

(a) [1] State, *in English*, the relationship between the number of nodes of a tree at depth *d* with the nodes at depth *d* − 1 in its subtrees.

# of nodes at depth d-1 is actually depth 2 towards its corresponding parent at depth d. ?

The number of nodes in tree at depth d is equal to the total number of nodes at depth d-1 in each of the subtrees.

0

(b) [4] Implement the function below using recursion. You may use a loop to process all of the trees in the subtrees attribute, but you may not use any Tree methods other than is_empty.

```
def count_depth(tree, d):
    """ (Tree, int) -> int
    Return the number of nodes in tree at depth d.
    You may assume that d >= 1.
    """
    if d=1 :
        ~~return~~
        if tree.is_empty():
            raise EmptyTreeError # to be defined
        else:
            return 1
    ~~elif d=2~~:
        ~~return len(tree.subtrees)~~:
sum = 0
    else:
        for subtree in ~~su~~ tree.subtrees :
            sum += ~~return count subtree~~.count_depth(d-1)
        return sum
```

3

wrong signature -1

4. Here is a question regarding the **BinarySearchTree** class. Assume that we have implemented a method **size** for this class, which returns the number of items contained in a BST.

   (a) [1] Suppose we are searching for the 10th smallest element in a BST, and suppose we know that the left subtree has 3 nodes. Should we next search in the left or right subtree, and what item should we be searching for in that subtree? (**Hint:** "look for the 5th smallest item in the left subtree" is incorrect, but has the right structure for the answer.)

   Right subtree.
   Should search for the 6th smallest item in the right subtree.

   ✓

   (b) [5] Your task is to implement the following function. You must use recursion, and may not use loops, or any other BinarySearchTree methods other than **size** and **is_empty**.

```
def size(self):
    """ (BinarySearchTree) -> int
    Return the number of items contained in this BST.
    """
    ... # You are NOT required to implement this!

def kth_smallest(self, k):
    """ (BinarySearchTree, int) -> object
    Precondition: 1 <= k <= size of this BST

    Return the kth_smallest value in this BST. So if a BST b contains
    the items {1,3,5,8,10,11}, then b.kth_smallest(2) returns 3, and
    b.kth_smallest(5) returns 10.

    Note: the return value depends only on the items in the tree, and
    not the structure. So if two trees contain the same items, but have
    different roots, calling kth_smallest still returns the same value.
    """
    # YOUR CODE GOES HERE
    llst = self.lst_bst
    llst.sort()
    return llst[k]
```

0

WRL

```
num_left = self.left.size()
 if num_left == k-1
 return self.root
 elif num_left <k-1
   return self.right.kth_
        smallest(k-num_left-
 else:
   return self.left.kth_smalle
```

```
# helper function below
def  list_bst(self):
  " (BinarySearchTree) -> list
  '''
   if self.is_empty:
      return []
   else:
       lst = []
       lst.append (self.root)
       lst +=   self.left. lst_bst()+self.right.lst_bst
```

Bonus Question [2]

Warning: this is a difficult question, and will be marked harshly. Only attempt it if you have finished all of the other questions!

Write a function pre_loop that takes a BST, and returns a preorder traversal of that BST, without using recursion.