PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2011 EXAMINATIONS

CSC 148 H1F
Instructor(s): P. Gries

Duration — 3 hours

Examination Aids: None

Student Number:	__________
Family Name(s):	__________
Given Name(s):	__________

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This final examination paper consists of 9 questions on 14 pages (including this one), printed on one side of each sheet. *When you receive the signal to start, please make sure that your copy of the final examination is complete and fill in the identification section above.*

You don't have to write docstrings or comments except where we ask for them.

Unless stated otherwise, you are allowed to define helper methods and functions.

MARKING GUIDE

# 1:	_____/ 20
# 2:	_____/ 15
# 3:	_____/ 15
# 4:	_____/ 25
# 5:	_____/ 15
# 6:	_____/ 10
# 7:	_____/ 15
# 8:	_____/ 10
# 9:	_____/ 20
TOTAL:	_____/145

Good Luck!

Question 1. [20 MARKS]

Part (a) [10 MARKS]

Draw the Binary Search Tree resulting from the following sequence of insertions and removals.

A. Insert 6

B. Insert 3

C. Insert 1

D. Insert 9

E. Remove 3

F. Insert 4

G. Insert 10

H. Insert 2

I. Remove 6

Part (b) [10 MARKS]

Draw the binary tree and Python list representations of the Min-Heap resulting from the following sequence of insertions and removals.

A. Insert 7

B. Insert 14

C. Insert 10

D. Insert 5

E. Remove root value

F. Insert 9

G. Insert 3

H. Insert 12

I. Remove root value

Question 2. [15 MARKS]

Draw the trees referred to by `b`, immediately before, and immediately after, calling `f`.

```
class BTNode(object):
    def __init__(self, left, value, right):
        self.left = left
        self.value = value
        self.right = right

def f(b):

    if b != None:
        b.left = f(b.right)
        b.right = f(b.left)

    return b

b = BTNode(BTNode(BTNode(None, "A", None),
                  "B",
                  BTNode(None, "C", None)),
           "D",
           BTNode(BTNode(None, "E", None),
                  "F",
                  BTNode(None, "G", None)))

f(b)
```

Question 3. [15 MARKS]

Implement prune.

```
class BSTNode(object):
    def __init__(self, v, left=None, right=None):
        self.value = v
        self.left = left
        self.right = right

def prune(b, k):
    '''Return the root of Binary Search Tree b modified to contain only values <= k.'''
```

Question 4. [25 MARKS]

Implement `index_leaves`.

DO NOT use any global variables, nor any 'counter' variables, except parameters.
DO NOT use any other container objects: lists, vectors, stacks, queues, heaps, new trees, etc.

Hint: write a recursive helper function taking a tree and an index and returning an index.

Run time efficiency counts.

```
class BinaryTree(object):
    '''The left and right instance variables refer to a BinaryTree or a Leaf.'''
    def __init__(self, left, right):
        self.left = left
        self.right = right

class Leaf(object):
    def __init__(self, data):
        self.data = data

def index_leaves(t):
    '''Number the Leafs of BinaryTree t in order from left to right, replacing their
        data with integers 0, 1, ..., number-of-Leafs - 1.'''
```

Question 5. [15 MARKS]

Part (a) [3 MARKS]

In a call to mergesort on a list of length n, approximately how many calls are on the call stack at its deepest? Circle the correct answer.

3	$log_2 n$	$log_{10} n$	n	$10 * n$
$n * log_2 n$	$10 * n * log_2 n$	$n\sqrt{n}$	n^2	$2 * n^2$

Part (b) [3 MARKS]

In a call to quicksort on a list of length n, in the worst case approximately how many calls are on the call stack at its deepest?

3	$log_2 n$	$log_{10} n$	n	$10 * n$
$n * log_2 n$	$10 * n * log_2 n$	$n\sqrt{n}$	n^2	$2 * n^2$

Part (c) [9 MARKS]

Recall from lecture that we discussed merging 2 non-decreasing lists.
Consider the more general case of merging $m \geq 2$ non-decreasing lists of k elements, by continually picking and removing the smallest element from the front of the m lists.
Give the O running time (use the smallest, simplest formula) for this algorithm.
It will involve both m and k.
Briefly justify your answer.

Question 6. [10 MARKS]

Part (a) [5 MARKS] Draw the unique tree that has these traversals:

```
preorder: 1 2 3 5 6 4
inorder:  5 3 6 2 4 1
```

Part (b) [5 MARKS]

You know that, given preorder and inorder traversals, you can reconstruct the tree. Describe why you can't do this if you are only given preorder and postorder traversals. (Give an explanation or a counterexample.)

Question 7. [15 MARKS]

This code works:

```
def is_sorted(L, i):
    '''Return whether L from index i onward is sorted, where 0 <= i < len(L).'''

    if i == len(L) - 1:
        return True
    else:
        return L[i] < L[i + 1] and is_sorted(L, i + 1)

def sum(L):
    '''Return the sum of the numbers in L.'''

    if len(L) == 0:
        return 0
    elif len(L) == 1:
        return L[0]
    else:
        mid = len(L) / 2
        return sum(L[:mid]) + sum(L[mid:])
```

Consider this attempt to implement `is_sorted` using a similar approach as for `sum`:

```
def is_sorted(L):
    if len(L) <= 1:
        return True
    elif len(L) == 2:
        return L[0] <= L[1]
    else:
        mid = len(L) / 2
        return is_sorted(L[:mid]) and is_sorted(L[mid:])
```

Part (a) [5 MARKS] Why doesn't this attempt work?

Part (b) [5 MARKS] Give an example list that reveals the problem.

Part (c) [5 MARKS] The implementation can be fixed by either adding or subtracting 1 from one of the expressions in the code. Precisely describe such a fix.

Question 8. [10 MARKS]

Part (a) [5 MARKS]

Define a function `rev(q)` that reverses the items in queue `q`, using only standard queue functions `enqueue`, `dequeue`, `is_empty` and `size`.
DO NOT use any other container objects: lists, vectors, stacks, queues (other than `q`, of course), heaps, trees, etc. Hint: use recursion.

Part (b) [5 MARKS]

Discuss whether the equivalent function `rev(s)` can be defined for a stack `s`, with similar restrictions.

Question 9. [20 MARKS]

Write a Queue class with enqueue, dequeue, is_empty, and size that uses Nodes as storage.
Make all methods run in $O(1)$ time: use no loops nor recursion.
Think carefully about which end should be which.

```
class Node(object):
    def __init__(self, v)
        self.value = v
        self.next = None
```

Continue your `Queue` code here if necessary:

Student #: └┴┴┴┴┴┴┴┴┘ CONT'D...

Use this page for rough work and for answers that didn't fit. Indicate clearly what you want us to mark.

Use this page for rough work and for answers that didn't fit. Indicate clearly what you want us to mark.

Total Marks = 145