PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL–MAY 2008 EXAMINATIONS

CSC 148 H1S
Instructor(s): P. Gries

Duration — 3 hours

Examination Aids: None.

Student Number: _ _ _ _ _ _ _ _ _ _

Last (Family) Name(s): _____

First (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*.

Instructions:

- Check to make sure that you have all 16 pages.
- Read the entire exam before you start.
- Not all questions are of equal value, so budget your time accordingly.
- You do not need to add `import` lines or do error checking unless explicitly required to do so.
- You do not need to write docstrings or comments unless explicitly required to do so, although it may help get you part marks if your answer is otherwise incorrect.
- If you use any space for rough work, indicate clearly what you want marked.

MARKING GUIDE

# 1:	_____/10
# 2:	_____/10
# 3:	_____/10
# 4:	_____/14
# 5:	_____/18
# 6:	_____/15
# 7:	_____/10
# 8:	_____/ 5
TOTAL:	_____/92

Good Luck!

CONT'D...

Question 1. [10 MARKS]

Part (a) [2 MARKS] Circle the truth value of each statement.

$f(n) = log_2(n) * n^2$ is $O(n^3)$. True False

$f(n) = n(n-1)(n-2)$ is $O(n^2)$. True False

Part (b) [4 MARKS]

If `b` has a boolean value, what effect does the following statement have?

```
b = (b == False)
```

Circle the correct answer:

A: It causes an exception.

B: It causes `b` to have the value `False` regardless of its value just before the statement was executed.

C: It always changes the value of `b`.

D: It changes the value of `b` if and only if `b` had value `True` just before the statement was executed.

E: None of the above.

Part (c) [4 MARKS]

```
def huh(n):
    if n > 1:
        huh(n / 2)
    print n,
```

What sequence of numbers will the function call `huh(16)` print? (Circle the correct answer.)

A: `10 8 6 4 2`

B: `16 8 4 2 1`

C: `1 2 4 8 16`

D: `32 16 8 4 2`

E: `2 4 8 16 32`

Question 2. [10 MARKS]

A complete binary tree is a tree where all leaves are at the same depth, and the tree is full. (That is, it looks like a triangle.)

Let `t` be a reference to a complete **binary search tree** with $n > 0$ nodes where the datum in each node is an integer, and there are no duplicate values. Consider the following function, which computes the smallest number in a tree. Note that `smallest(t.root)` returns the value of the smallest node in the entire tree.

```
def smallest(root, s=None):
  if root == None:
      return s
  else:
      d = root.data
      return min(d, smallest(root.left, d), smallest(root.right, d))
```

Part (a) [3 MARKS] How many times is `smallest` called on a tree with n nodes? _____

Part (b) [5 MARKS] Write a new version of `smallest` with a faster running time:

Part (c) [2 MARKS] How many times is your new function called on a tree with n nodes? _____

Question 3. [10 MARKS]

Here is the implementation for `Stack.peek` from early in the course; as you know, i doesn't work if the stack is empty; it raises an `IndexError`.

```
def peek(self):
    '''Return the top item.'''
    return self.stack[-1]
```

Now consider this new docstring:

```
def peek(self):
    '''Return the top item.  Raise an EmptyStackError if the stack has no elements.'''
```

Some of the following methods satisfy the docstring, and some of them don't. The incorrect ones may not compile, or they just may never raise an `EmptyStackError`.

A

```
def peek(self):
    return self.stack[-1]
    except EmptyStackError, \
        "Can't peek at an empty stack."
```

B

```
def peek(self):
    try:
        return self.stack[-1]
    except:
        raise EmptyStackError, \
            "Can't peek at an empty stack."
```

C

```
def peek(self):
    try:
        return self.stack[-1]
    except IndexError:
        raise EmptyStackError, \
            "Can't peek at an empty stack."
```

D

```
def peek(self):
    if len(self.stack) == 0:
        raise EmptyStackError()

    return self.stack[-1]
```

E

```
def peek(self):
    try:
        raise EmptyStackError
            "Can't peek at an empty stack."
    except:
        return self.stack[-1]
```

F

```
def peek(self):
    if self.stack[-1] == None:
        raise EmptyStackError, \
            "Can't peek at an empty stack."
    return self.stack[-1]
```

Part (a) [2 MARKS]
Briefly explain the difference between B and C.

Part (b) [2 MARKS]
Write the letters from the boxes that have code that **does not** run—Python produces a `SyntaxError`:

Part (c) [2 MARKS]
Write the letters from the boxes that have code that runs, but **never** deals with `EmptyStackException` properly:

Part (d) [2 MARKS]
Write the letters from the boxes that contain code that (almost always) does what the docstring says:

Part (e) [2 MARKS]
Write the letter of the box that contains the best code stylistically, and provide a **brief** explanation as to why you think so:

Question 4. [14 MARKS]

Consider the class `BTNode` that could be used in a binary search tree.

```
class BTNode(object):
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.size = 1
```

Part (a) [5 MARKS]

Let `root` refer to a `BTNode` that is the root of a binary search tree. Write a function named `set_size` such that, after executing `set_size(root)`, for every node `v` in the tree, the value of `v.size` is set to be equal to the number of nodes that are in the subtree rooted at `v`. For example, `root.size` would be equal to the total number of nodes in the tree, and all leaves should have a `size` of 1.

Note that `set_size` is a function, and not a method of class `BTNode`.

Part (b) [5 MARKS]

Suppose that `set_size(root)` has been executed and all the nodes have the correct `size` value. Write a function `insert` such that, if `insert(k)` is called for for a key `k`, `k` is inserted, and after the insertion all the `size` values are correct for all nodes in the tree. For full marks, `insert` should only examine the nodes that it needs to.

Part (c) [2 MARKS]

Assuming that a tree has n nodes, in the *worst* case, how many nodes will `insert` visit? _____

Part (d) [2 MARKS]

Assuming that a tree has n nodes, in the *best* case, how many nodes will `insert` visit? _____

Question 5. [18 MARKS]

Consider the following implementation of a linked list.

```
class Node(object):

    def __init__(self, data):
        '''Create an Node with data whose next Node is None.'''
        self.data = data
        self.next = None

class LinkedList(object):
    '''A Linked List.'''

    def __init__(self):
        '''Create a new LinkedList that is empty.'''
        self.first = None

    def insert(self, data):
          '''Insert data as the last element in this linked list.'''
          self.first = _insert_helper(self.first, data)
```

Part (a) [5 MARKS] Complete recursive function `insert_helper`:

```
def _insert_helper(n, data):
    '''Insert object data at the end of the linked list pointed to by Node n,
    and return the first Node in the new list.'''
```

Part (b) [5 MARKS] Complete the following **non-recursive** method.

```
def remove_after(self, v):
    '''Find the first node containing v and remove the Node AFTER that one.
    If v is at the end of the list, do nothing. If v does not occur in the
    list, raise a ValueError.

    For example, if the list contains 1, 2, 3, and 4, then remove_after(2)
    would remove the 3, leaving 1, 2, and 4.'''
```

Part (c) [3 MARKS] Write a nose test that checks whether the `ValueError` is raised when appropriate.

Part (d) [5 MARKS]

In the method docstring for `remove_after`, one test case is described; that is included in the table below. Fill in the table with a good set of test cases. (You may not need every row of the table; if that's the case, leave them blank.)

Linked list contents	**v**	**Resulting list**	**Brief explanation of why this is interesting**
1, 2, 3, 4	2	1, 2, 3	General case

Question 6. [15 MARKS]

Complete each of the following functions by filling in the blanks.

Part (a) [5 MARKS]

```
def my_pow(x, n):
    '''Return x to the power n.
    Precondition: x and n are integers and n >= 0.'''

    if n == _________________:
        return 1
    else:
        return x * my_pow(_________________, _________________)
```

Part (b) [4 MARKS]

```
def rev_string(s):
    '''Return the reverse of str s.'''

    if ___________________:
        return s
    else:
        return rev_string(_________________) + s[0]
```

Part (c) [6 MARKS]

```
def sumN(n):
    '''Return the sum of the integers from 1 to n.  Precondition: n >= 0.'''
    if n < 2:
        return n
    else:
        return sumNtoN(1,n)

def sumNtoN(n1, n2):
    '''Return the sum of the integers from n1 to n2.  Precondition: n2 >= n1.'''

    # This is calculated as the sum from n1 to (n1 + (n2 - n1) / 2),
    # plus the sum of (n1 + ((n2 - n1) / 2 + 1)) to n2.

    if  _________________:
        return n1
    else:
        diff = (n2 - n1) / 2
        return sumNtoN(_________________) + sumNtoN(_________________)
```

Question 7. [10 MARKS]

Part (a) [5 MARKS]

Assume that we insert the following items into a min heap, in order: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20. Draw the tree representation:

Fill in the Python list that stores this heap:

```
[    ,    ,    ,    ,    ,    ,    ,    ,    ,    ]
```

Part (b) [5 MARKS]

Assume that we insert the following items into a min heap, in order: 20, 18, 16, 14, 12, 10, 8, 6, 4, 2. Draw the tree representation (you may wish to use one of the rough-work pages at the end of the exam):

Fill in the Python list that stores this heap:

```
[    ,    ,    ,    ,    ,    ,    ,    ,    ,    ]
```

Question 8. [5 MARKS]

Consider the following code:

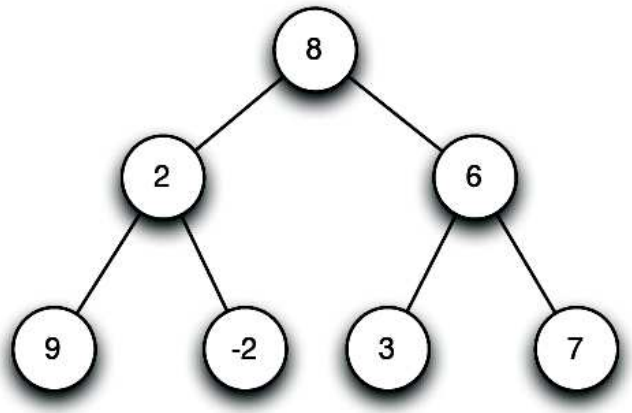
```
class Node(object):
    def __init__(self, v):
        self.data = v
        self.left = None
        self.right = None

def do_something(root):
    s = Stack()

    curr = root
    while curr is not None:
        s.push(curr)
        curr = curr.left

    while not s.is_empty():
        curr = s.pop()
        print curr.data,
        curr = curr.right
        while curr is not None:
            s.push(curr)
            curr = curr.left
```

What does `do_something` do when given this binary tree?



Answer (for full marks, give a general explanation; for part marks, write the output):

*[Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

*[Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

*[Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

Total Marks = 92