PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2013 EXAMINATIONS

CSC 148 H1Y
Instructor: V. Pandeliev

Duration — 3 hours

Examination Aids: None

PLEASE HAND IN

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 22 pages (including this one). *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.*

If you do not obtain a mark of 40% on this examination, you will be unable to pass the course.

You do **not** need to write docstrings or comments except where we ask for them.

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

If you are unable to answer a question (or part), you will get 20% of the marks for that question (or part) if you write "I don't know" and nothing else. You will **not** get those marks if your answer is completely blank, or if it contains contradictory statements (such as "I don't know" followed or preceded by parts of a solution that have not been crossed off).

MARKING GUIDE

# 1:	______/ 15
# 2:	______/ 6
# 3:	______/ 6
# 4:	______/ 10
# 5:	______/ 14
# 6:	______/ 11
# 7:	______/ 22
# 8:	______/ 6
# 9:	______/ 10
TOTAL:	______/100

Good Luck!

Question 1. [15 MARKS]

Part (a) [7 MARKS]

Answer the following True/False questions by circling the correct answer.

Keeping a `tail` variable for a singly linked list makes deleting the last node an *O(1)* operation.

TRUE FALSE

Each function has access to its own namespace, its caller's namespace, and the global namespace.

TRUE FALSE

Even if a function called `foo` has been defined in the code, the statement `a = foo` will generate an error.

TRUE FALSE

In a linked node representation of a binary search tree, the `Node` class can inherit from the `BSTree` class.

TRUE FALSE

Building a BST and performing an in-order traversal to obtain a sorted list of items has a worse time complexity than performing heapsort.

TRUE FALSE

The expression `an_object.some_method()` indicates that the method's definition line is `def some_method():`

TRUE FALSE

The largest possible BST that is also a min-heap consists of two nodes.

TRUE FALSE

Part (b) [2 MARKS]

Describe the purpose of an activation record and state what it stores.

Part (c) [4 MARKS]

Draw the box representation and indicate the type of the local namespace of function f at the point of execution marked by ***. Draw the box representation and indicate the type of any object it is aware of. You do not need to draw the call stack, activation records, or any other namespaces. To link objects, you may use arrows or memory addresses.

```
def f(a, b):
    q = str(a + b)
    # *** Draw namespace here ***
    return q

f(4, 6)
```

Part (d) [2 MARKS]

Where will Python find the name f? What is it bound to?

Question 2. [6 MARKS]

This question is about stacks and queues. Assume the following class specifications:

```
class Queue:
    __init__(): create an empty queue
    is_empty(): return True iff the
                queue is empty
    enqueue(n): enqueue element n
    dequeue():  remove and return the
                front element of the queue.
                Return None if queue is empty.
```

```
class Stack:
    __init__(): create an empty stack
    is_empty(): return True iff the
                stack is empty
    push(n): push element n to the stack
    pop(): remove and return the
           top element of the stack.
           Return None if stack is empty.
```

It is possible to implement a queue using only two Stacks for storage. Any values enqueued to the queue are actually pushed to the top of the first stack. If that is the case, think about how dequeueing the correct element might be accomplished. Draw some stacks to understand the problem (this is not worth marks but will definitely help).

Now, assuming the Stack class is fully implemented, consider the partially implemented Queue class below. It uses two stacks as its instance variables. **On the next page**, implement the enqueue and dequeue methods of this class. You are allowed to use temporary variables, but you may **not** use any additional instance variables or data structures, including dictionaries and lists. Using any other *structure or method* besides the ones provided will result in a 0 for this part of the question.

```
class Queue:
    def __init__(self):
        '''(Queue) -> NoneType
        Create a new Queue. This method is complete.'''

        self.stack1 = Stack()
        self.stack2 = Stack()

    def is_empty(self):
        '''(Queue) -> bool
        Return True iff self is empty. This method is complete.'''

        return self.stack1.is_empty()
```

```
def enqueue(self, e):
    '''(Queue, object) -> NoneType
    Add e to self.'''
```

```
def dequeue(self):
    '''(Queue) -> object
    Remove and return the front element of self. Return None if self is empty.'''
```

Question 3. [6 MARKS]

This question is about singly linked lists represented as linked nodes. You may assume the implementations of the `LinkedList` and `LLNode` classes begin as follows:

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def size(self):
        return self.head.size()
```

```
class LLNode:
    def __init__(self, v, next=None):
        self.data = v
        self.next = next
```

Part (a) [3 MARKS]

In the `LLNode` class, implement a method called `size` that returns the size (the number of elements) of the part of the linked list that begins at the `LLNode` represented by `self`. If `size` is called on the head of a linked list (as it is in the `LinkedList` class), it should return the size of the entire linked list. This method may be recursive or iterative.

```
    def size(self):
        '''(LLNode) -> int
        Return the size of the portion of the linked list that begins at self.'''
```

Part (b) [3 MARKS]

The way the `size` method of the `LinkedList` class is written, it does not account for the possibility that the linked list could be empty. If it is run on an empty linked list, it will raise an `AttributeError`. Write a block of code that causes this behaviour, catches the error, and prints the message `"Empty list!"` to the screen.

Question 4. [10 MARKS]

This question is about generic binary trees. For this question, you may assume that binary trees are implemented using the following Node class and that the data attribute of every Node contains an integer.

```
class Node:
    def __init__(self, v):
        '''(Node, int) -> NoneType
        Create a new binary tree node with data v.'''

        self.data = v
        self.left = None
        self.right = None
```

Part (a) [2 MARKS]

The largest possible height of a binary tree with 12 nodes is ____________ and the smallest is ____________.

Part (b) [3 MARKS]

Draw the binary tree that has the following traversals:

```
Pre-order: 4, 8, 5, 6, 1, 3, 7, 2
 In-order: 8, 6, 5, 4, 3, 1, 7, 2
```

Part (c) [5 MARKS]

Write a **recursive** function called `multiply_leaves` that takes the root of a binary tree and returns the product of all leaves in the tree. You may **not** use helper functions for this part. Recall that a leaf node has no children.

```
def multiply_leaves(node):
    '''(Node) -> int
    Return the product of all leaves in the tree rooted at node.
    An empty tree has a leaf product of 1.'''
```

Question 5. [14 MARKS]

This question is about binary search trees. Assume that the binary search trees are implemented using the `Node` class from the previous question and that duplicate values are not allowed.

Part (a) [2 MARKS]

State the binary search tree property or draw the recursive definition of binary search trees.

Part (b) [2 MARKS]

Draw the BST resulting from the consecutive insertion of the following numbers into an empty tree:
3, 6, 4, 9, 2, 7, 1

Part (c) [4 MARKS]

Write a **recursive** function called `remove_smallest` that takes the root of a binary search tree and removes the smallest element in the tree. Assume the tree has at least one node before the function runs. The function should return the root of the tree.

```
def remove_smallest(node):
    '''(Node) -> Node
    Remove the smallest node from the tree rooted at node. Return the root the tree.'''
```

Part (d) [6 MARKS]

Assume that there is a function called `is_bst` that takes the root of a generic binary tree and returns `True` if and only if that tree is a binary search tree that contains unique values. Draw enough trees (at least five) to thoroughly test `is_bst`, specify the outcome expected of running `is_bst` on each tree, and explain in a few words why these test cases are enough to cover all possibilities. You do not need to write code for this part.

Question 6. [11 MARKS]

This question is about heaps.

Part (a) [2 MARKS]

Draw the tree representation of the heap stored in this list:

2	3	5	9	7	6

Part (b) [2 MARKS]

Insert 1 into the heap from **Part (a)** and draw the resulting heap (as a tree or as a list).

Part (c) [2 MARKS]

Insert 4 into the heap from **Part (b)** and draw the resulting heap (as a tree or as a list).

Part (d) [2 MARKS]

Perform a `delete()` operation on the heap from **Part (c)** and draw the resulting heap (as a tree or as a list).

Part (e) [3 MARKS]

Describe the `percolate_down` operation for a min-heap. When is it necessary? What does it do? When does it finish?

Question 7. [22 MARKS]

This question is about sorting algorithms

Part (a) [7 MARKS]

Complete the table, filling in the names or the descriptions of the sorting algorithms below.

Algorithm	**Description**
Radix sort	
	Split the list into smaller and smaller lists until lists of size one element are reached. Recombine them with elements in the right order.
	Find the smallest element in the unsorted section and put it at the end of the sorted section. Eventually, the sorted section will expand to encompass the entire list.
	Iterate through the list several times, repeatedly swapping elements with their neighbour if appropriate. Eventually, elements will move into their correct position.
	Take an element from the unsorted section and put it in its correct place in the sorted section. Eventually, the sorted section will expand to encompass the entire list.

Part (b) [7 MARKS]

The quicksort algorithm we studied in class is vulnerable to the choice of pivot. Computer scientists have discovered that using two pivots instead of one dramatically decreases the chance of picking bad pivots. Implement two-pivot quicksort using the first and the last elements in the list as pivots and three partitions. Make sure that the first and last element are distinct elements (i.e. the list has at least two elements) before you attempt to partition, and that the first pivot is smaller than the second pivot. If that is not the case, swap them. This version of quicksort should deal with duplicate values correctly and it should **return a new list without changing the original list**. Recall that you can concatenate two or more lists with the + operator. The new list should look like this:

x <pivot1	**pivot1**	pivot1 <= y <= pivot2	**pivot2**	z >pivot2

```
def quicksort(L):
    '''(list) -> list
    Sort L by partitioning into three sublists with two pivots.
    Return new, sorted list. Original list L remains unchanged.'''
```

Part (c) [2 MARKS]

In the two-pivot quicksort on the previous page, what is a worst-case input and why?

Part (d) [6 MARKS]

Perform heapsort on the following list. Note that the list is **not** a heap to start. You should fill in a new row every time an element changes position in the list or two elements are swapped. You may not need all 12 lines to complete this part.

00.	6	9	1	7	3

01.						07.					
02.						08.					
03.						09.					
04.						10.					
05.						11.					
06.						12.					

Question 8. [6 MARKS]

You are given the following partial implementation of three classes. Complete the implementation so that the code in the main block produces no assertion errors.

```
class A:
    # THIS CLASS IS COMPLETE. DO NOT ADD ANYTHING TO ITS DEFINITION.
    def __init__(self, name):
        self.p = "I'm " + name

    def __str__(self):
        return "It is very nice to meet you, " + self.p

class B(A):
    # *** IMPLEMENT CLASS B HERE ***
```

```
class C(B):
    # THIS CLASS IS COMPLETE. DO NOT ADD ANYTHING TO ITS DEFINITION.
    def __str__(self):
        return "Name: " + self.p + self.get_age()

if __name__ == '__main__':
    b = B("Paul")
    c = C("John", 40)
    assert str(b) == "My name is Paul(71)"
    assert str(c) == "Name: John(40)"
```

Question 9. [10 MARKS]

This question is about time complexity.

Part (a) [3 MARKS]

What is the worst-case time complexity of radix sort? When does it approach $O(n)$?

Part (b) [1 MARK]

What is the worst case input for insertion sort?

Part (c) [6 MARKS]

Circle the *worst case* time complexity of each operation below (for n >1000).

Delete the element at index `-4` of a Python list of size **n**.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

Create a BST by consecutively inserting **n** elements.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

Create a heap by consecutively inserting **n** elements.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

Enqueue and dequeue **n** items from a `Queue` that enqueues to the beginning of a Python list.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

Search for a node in a BST of size **n**.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

Search for a node in a generic binary tree of size **n**.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

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.

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

Student #:

YOU CAN TEAR THIS PAGE OFF IF YOU LIKE.

Short Python function/method descriptions:

```
__builtins__:
  len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
  max(L) -> value
    Return the largest value in L.
  min(L) -> value
    Return the smallest value in L.
  range([start], stop, [step]) -> range of integers
    Return a range containing the integers starting with start and
    ending with stop - 1 with step specifying the amount to
    increment (or decrement). If start is not specified,
    the range starts at 0. If step is not specified, the values
    are incremented by 1.
  sum(L) -> number
    Returns the sum of the numbers in L.

dict:
  D[k] -> value
    Return the value associated with the key k in D.
  k in d -> boolean
    Return True if k is a key in D and False otherwise.
  D.get(k) -> value
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
  D.items() -> list of (key, value) pairs
    Return the (key, value) pairs of D, as 2-tuples.

list:
  x in L -> boolean
    Return True if x is in L and False otherwise.
  L.append(x)
    Append x to the end of list L.
  L1.extend(L2)
    Append the items in list L2 to the end of list L1.
  L.index(value) -> integer
    Return the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.pop()
    Remove and return the last item from L.
```

```
  L.remove(value)
    Remove the first occurrence of value from L
  L.reverse()
    Reverse *IN PLACE*
  L.sort()
    Sort the list in ascending order.
  L[-1]
    Retrieve the last item in the array
  L[start:end]
    Create a new list containing the items in L
  L[start:]
    Create a new list containing the items in L
    to the end of the list
  L[:end]
    Create a new list containing the items in L
    of the list through end-1
  L[:]
    Create a new list containing all the items

str:
  x in s -> boolean
    Return True if x is in s and False otherwis
  str(x) -> string
    Convert an object into its string represent
  S.count(sub[, start[, end]]) -> int
    Return the number of non-overlapping occurr
    in string S[start:end]. Optional arguments
    interpreted as in slice notation.
  S.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S
    where the string sub is found or -1 if sub
  S.isdigit() -> boolean
    Return True if all characters in S are digi
  S.lower() -> string
    Return a copy of the string S converted to
  S.replace(old, new) -> string
    Return a copy of string S with all occurren
    replaced with the string new.
  S.split([sep]) -> list of strings
    Return a list of the words in S, using stri
    and any whitespace string if sep is not spe
  S.strip() -> string
    Return a copy of S with leading and trailin
  S.upper() -> string
    Return a copy of the string S converted to
```

CONT'D...

Total Marks = 100