PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2010 EXAMINATIONS

CSC 148 H1S
Instructor(s): J. Clarke, P. Gries, A. Tafliovich

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 8 questions on 16 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 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:	______/ 8
# 2:	______/10
# 3:	______/10
# 4:	______/ 6
# 5:	______/10
# 6:	______/12
# 7:	______/ 5
# 8:	______/10
TOTAL:	______/71

Good Luck!

Question 1. [8 MARKS]

These questions have you draw trees. You don't need to draw the full picture of memory; instead, use circles with values inside them.

Part (a) [4 MARKS]

This question is about binary search trees. Draw the tree resulting from the following operations, where when we remove a value we replace it with the smallest value in the right subtree.

1. Insert 5
2. Insert 8
3. Insert 10
4. Insert 2
5. Remove 8
6. Insert 7
7. Insert 1
8. Insert 9
9. Remove 5

Part (b) [4 MARKS]

This question is about min-heaps. Draw the heap resulting from the following operations.

1. Insert 5
2. Insert 8
3. Insert 10
4. Insert 2
5. Remove the value at the root
6. Insert 7
7. Insert 1
8. Insert 9
9. Remove the value at the root

Question 2. [10 MARKS]

You've started a game-writing club and have decided to implement a game named "BB". Read the following description of the game and perform an object-oriented analysis to determine which classes, methods, and instance variables you will need. Each row describes the information about a single class. For each class, write the class name **and its parent's name** in the left column, the method names in the middle column, and the instance variables **and their types** in the right column. You may not need to use every row, and you do not need to include inherited variables or methods when describing a child class.

> In "BB", the player explores a 2-D underwater world. The game itself consists of a sequence of increasingly challenging levels filled with creatures. Each level has a large background image and a collection of creatures for the player to meet.
>
> The player navigates through a level by swimming in a specified direction and with a specified speed. As the player swims, her position in the level changes, so she will meet the creatures that live in the level.
>
> Each creature in the game has a "sprite" – an image to represent it, a starting position in the level, and an interaction function that determines how it behaves when the player is nearby. "Fish" are a special type of creature which have a movement function that determines how their position in the level changes.

Classes	**Methods**	**Instance Variables**

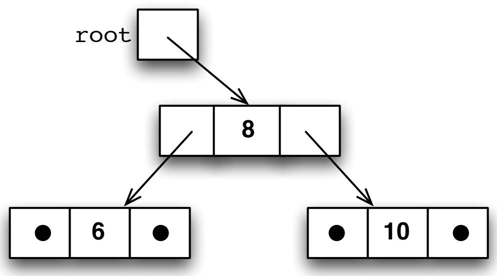
Question 3. [10 marks]

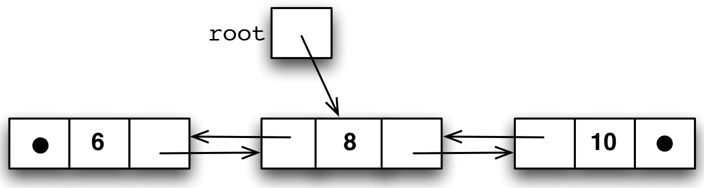
```
class Node(object):
    def __init__(self, v):
        self.value = v
        self.left = None
        self.right = None
```

The `Node` class to the right can be used either for a binary tree or for a doubly-linked list, where `left` points to the previous node and `right` points to the next one. In fact, you can turn a binary search tree into a sorted doubly-linked list *without* creating any new `Node` objects. Cool, huh?

Part (a) [2 marks]

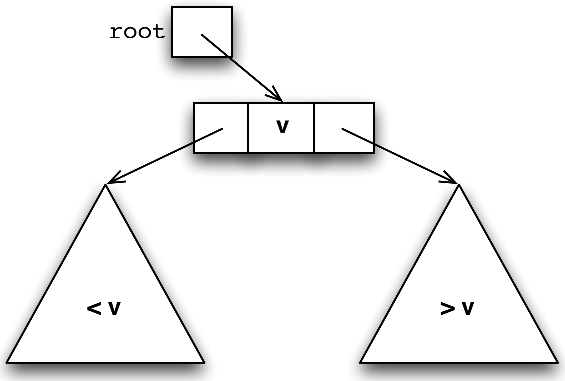
Here is a binary search tree and the corresponding sorted doubly-linked list.





Write two assignment statements that will turn the tree on the left into the doubly-linked list on the right:

Now consider a generic binary search tree such as the one shown to the right. If you can turn the left subtree into a doubly-linked list, and you can turn the right subtree into a doubly-linked list, and you have pointers to the left head and left tail and pointers to the right head and right tail, then you can turn the whole tree into a doubly-linked list by stitching it all together doing something like you did in your answer above.



Read the following function header. Notice that it does not handle empty trees.

```
def to_list(r):
    '''Convert the binary search tree rooted at Node r to a doubly-linked list
    where the elements are in order, and return a tuple containing the first
    and last Nodes in the resulting list.  Precondition: r is not None.'''
```

Part (b) [4 MARKS]

Draw four trees that would be good test cases for `to_list`. Write a *brief* sentence for each test case describing what it tests.

Part (c) [4 MARKS] Write the body of `to_list`. Helper functions are **not** allowed.

```
def to_list(r):
```

Question 4. [6 MARKS]

Here is working selection sort code. It has inefficiencies. It also is profiled; see the last call.

```
import cProfile

def smallest(L, start):
    '''Return the smallest item in L[start:]'''

    smallest = L[start]
    for i in range(start + 1, len(L)):
        if L[i] < smallest:
            smallest = L[i]
    return smallest

def get_index(L, start, v):
    '''Return the index of v in L[start:]. Precondition: v is in L[start:]'''

    for i in range(start, len(L)):
        if L[i] == v:
            return i

def choose_smallest(L, start):
    '''Return the index of the smallest item in L[start:].'''

    smallest_item = smallest(L, start)
    smallest_loc = get_index(L, start, smallest_item)
    return smallest_loc

def selection_sort(L):
    '''Sort the items in list L in non-decreasing order.'''

    for i in range(len(L) - 1):
        # print L
        index = choose_smallest(L, i)
        L[i], L[index] = L[index], L[i]

if __name__ == '__main__':
    L = range(5000)
    L.reverse()
    cProfile.run("selection_sort(L)")
```

Here is the profiling output. `tottime` is the time a function is on the top of the call stack and `cumtime` is the total time a function is anywhere in the stack.

```
         34998 function calls in 2.136 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    2.136    2.136 <string>:1(<module>)
     4999    0.561    0.000    0.663    0.000 selection.py:16(get_index)
     4999    0.006    0.000    2.132    0.000 selection.py:23(choose_smallest)
        1    0.004    0.004    2.136    2.136 selection.py:30(selection_sort)
     4999    1.361    0.000    1.464    0.000 selection.py:7(smallest)
     9999    0.001    0.000    0.001    0.000 {len}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     9999    0.203    0.000    0.203    0.000 {range}
```

Part (a) [1 MARK] Which two functions are on the top of the call stack the most?

Part (b) [5 MARKS] Why is this version of selection sort inefficient, and how would you improve it?

Question 5. [10 MARKS]

In each part of this question you are provided with a data structure but not told its implementation. You are required to answer the question using only that data structure. Python's built-in structures such as lists and dictionaries are not allowed, and you also may not implement your own versions of other data structures. (Variables that refer to simple objects such as integers are allowed.)

Part (a) [5 MARKS]

Assume you have a `Stack` class (where the constructor creates an empty Stack) with `push`, `pop`, and `is_empty` methods, and you don't know how it's implemented. Write a function called `flip` that takes a `Stack` as a parameter and reverses the content of the stack. The original `Stack` must contain the flipped values. For example, if the original stack contained the values 1 and 2 with 2 on the top of the stack, then your function would make 1 the top with 2 underneath.

The only data structure that you are allowed to use is `Stack` (although you can use as many `Stack`s as you want). No lists, dictionaries, queues, or other data structures are allowed.

Part (b) [5 marks]

Assume you have a `Queue` class (where the constructor creates an empty Queue) with `enqueue`, `dequeue`, and `is_empty` methods, and you don't know how it's implemented. Write a function `remove_dups` that takes a `Queue` as a parameter and removes consecutive duplicate values from the `Queue`, leaving the order of the remaining contents unchanged. Use `==` to compare for equality. You must leave the contents in the original Queue, not in a new one. For example, if the function is called with a `Queue` that contains 1, 2, 2, 3, 4, 4, 4, 5, 4, then when the function ends the `Queue` would contain 1, 2, 3, 4, 5, 4.

The only data structure that you are allowed to use is `Queue` (although you can use as many `Queue`s if you want). No lists, dictionaries, stacks, or other data structures are allowed.

Question 6. [12 marks]

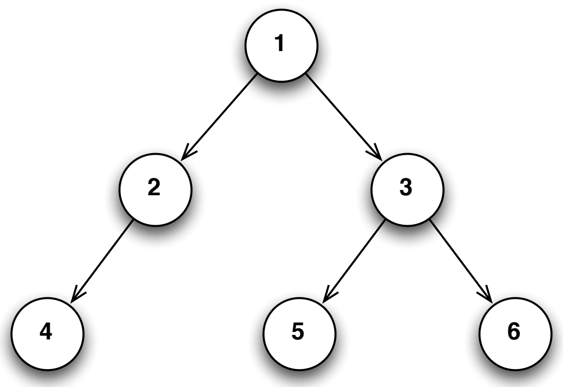
Consider the following implementation of a binary tree node and a function that makes use of it.

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

```
def func(node):
    if node:
        node.left = func(node.left)
        node.right = func(node.right)
        node.left = node.right
        node.right = node.left
    return node
```

Part (a) [3 marks]

Draw the tree that results when `func` is called on this tree:



Answer:

Part (b) [3 marks] Write a docstring for `func`:

Part (c) [6 MARKS]

Implement a function `fringe(r)` that returns a list of all values stored in the leaves of the binary tree rooted at `Node r`, in order from left to right. For example, when called on the tree from Part (a), `fringe` should return the list `[4, 5, 6]`.

Question 7. [5 MARKS]

Recall our implementation of the Heap ADT, in which we store the elements of the Heap in a list as follows:

- The root of the heap is stored at index `0`.
- For any node `n` stored at index `i`,
  - the left child of `n` is stored at index `2i+1`, and
  - the right child of `n` is stored at index `2i+2`.

Write a function `is_min_heap` that takes a list and returns `True` if the list represents a valid min-heap and `False` otherwise. Duplicate values are not allowed.

Question 8. [10 marks]

Write a function that merges two linked lists according to the docstring given below.

If you use any loops or helper methods, you can earn at most 5 marks on this question.

The linked lists' members are all `Node` objects from this `Node` class:

```
class Node(object):

    def __init__(self, value, link=None):
        self.data = value
        self.next = link

    def __cmp__(self, other):
        if other:
            return self.data.__cmp__(other.data)
        else:
            return -1

def merge_ll(list1, list2):
    '''Return a new linked list whose members are all the elements from list1
    and list2, in non-decreasing order. list1 and list2 are each sorted in
    non-decreasing order. The Nodes from list1 and list2 are not re-used;
    instead, while building the new list, new Nodes are created to store the
    elements from list1 and list2.'''
```

If you need the space, you can continue your Question 8 answer here.

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 = 71