PLEASE HAND IN

PLEASE HAND IN

**UNIVERSITY OF TORONTO**
**Faculty of Arts and Science**

**DECEMBER 2012 EXAMINATIONS**

**CSC 148 H1F**
**Instructor: V. Pandeliev**

**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*.

---

MARKING GUIDE

# 1: ______/11

# 2: ______/12

# 3: ______/ 7

# 4: ______/10

# 5: ______/ 5

# 6: ______/ 7

# 7: ______/ 6

# 8: ______/12

TOTAL: ______/70

This final examination paper consists of 8 questions on 17 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.*

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.

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).

*Good Luck!*

## Question 1. [11 MARKS]

### Part (a) [5 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 of the linked list an $O(1)$ operation.

TRUE FALSE

If a queue is implemented using a Python list, it doesn't matter which end items are enqueued to in terms of efficiency.

TRUE FALSE

If a stack is implemented using a Python list, it doesn't matter which end items are pushed to in terms of efficiency.

TRUE FALSE

The in-order traversal of a binary tree always yields the values in the tree in ascending order.

TRUE FALSE

Exception class definitions can consist only of a `pass` statement as long as they inherit from `Exception`.

TRUE FALSE

### Part (b) [6 MARKS]

Circle the *worst case* time complexity of each operation below.

Inserting a node at the end of a linked list with `head` and `tail` variables.

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

Inserting a node into a binary search tree.

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

Deleting a node from a min-heap.

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

Retrieving the last element in a Python list.

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

Heapsort.

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

Binary search in a sorted Python list.

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

## Question 2. [12 MARKS]

In this question you will be writing code about binary trees. You are given the following classes to use in your code (you do not need to import them):

```
class BTNode:
    '''A generic binary tree node.'''

    def __init__(self, v):
        '''(BTNode, int) -> NoneType
        Initialize a new BTNode with value v.'''

        self.value = v
        self.left = None
        self.right = None


class LLNode:
    '''A doubly linked list node.'''

    def __init__(self, v):
        '''(BTNode, int) -> NoneType

        Initialize a new LLNode with value v.'''
        self.value = v
        self.prev = None
        self.next = None
```

The two parts of this question begin on the next page.

## Part (a) [5 MARKS]

Complete the function `is_max_heap` that checks whether a binary tree whose values are *positive integers* is a max-heap. You may use helper functions. If you do, *write a docstring* with a signature line and an explanation for every function you create. You may assume the binary tree is complete - you do not need to check for that. *Hint:* Use the recursive definition of max-heaps.

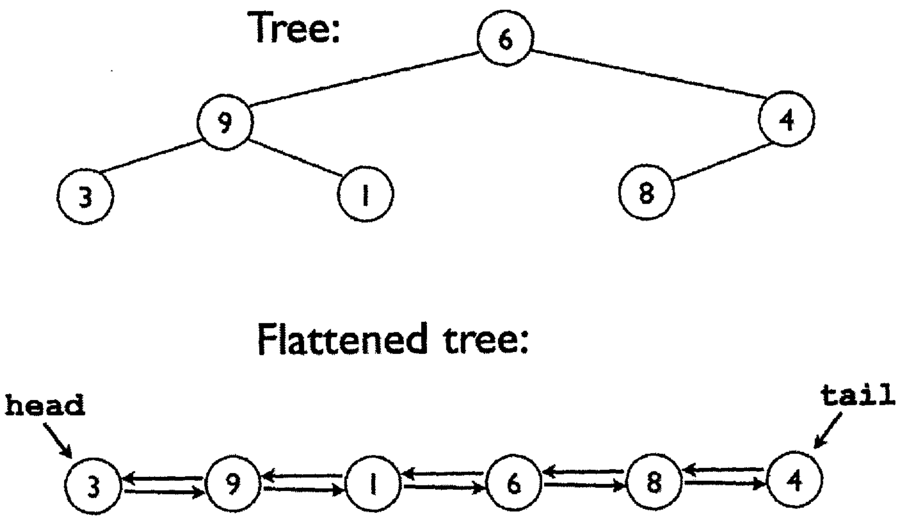
```
def is_max_heap(node):
    '''(BTNode) -> bool
    Return True iff the binary tree rooted at node is a max-heap.
    Precondition: the tree is complete.'''
```

### Part (b) [7 MARKS]

We call *flattening a binary tree* the process of converting the tree into a doubly-linked list representing its in-order traversal. The following rules apply:

- The leftmost node in the tree is the head of the linked list.
- The rightmost node in the tree is the tail of the linked list.
- For every node `n`, its `prev` is the tail of its flattened left subtree and its `next` is the head of its flattened right subtree.

Example:



On the next page, write a recursive function called `flatten` that takes a `BTNode` as the root of a binary tree and returns a pair of `LLNodes` representing the head and tail of the resulting doubly linked list. You are *not* allowed to use helper functions for this.

```
def flatten(node):
    '''(BTNode) -> (LLNode, LLNode)
    Return a pair consisting of the head and tail LLNodes of the doubly
    linked list representing the flattened binary tree rooted at node.
    Return (None, None) if the tree is empty.'''
```

## Question 3. [7 MARKS]

This question is about binary search trees.

### Part (a) [4 MARKS]

Draw the binary search tree generated by inserting the following values in order:

```
5, 7, 4, 3, 6, 9, 1, 2, 8
```

### Part (b) [3 MARKS]

Draw the binary search tree above after the deletion of the node with value 7. If the node has two subtrees, you may decide which neighbour to replace it with.

## Question 4. [10 MARKS]

This question is about min-heaps.

### Part (a) [6 MARKS]

Draw the min-heap generated by inserting the following values in order:

```
5, 7, 4, 2, 1, 8
```

### Part (b) [4 MARKS]

Assume the min-heap from the previous part was being represented as a list. Fill in the necessary values in the list at the correct index. If any items in the list should be left blank, leave them blank.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

## Question 5. [5 MARKS]

You are given the following code. Assume that AttributeError is the error Python raises if it can't find an attribute (instance variable) name inside a given class. In the box on the right, write down the sequence of lines that this code will produce when it runs.

```
class A:
    def __init__(self):
        self.x = "Ringo"

    def print_or_error(self):
        print(self.y)
        print(self.x)

class B(A):
    def __init__(self, q):
        self.y = q
        A.__init__(self)

    def print_or_error(self):
        print(q)
        print(self.x)
        print(self.y)

q = "Paul"
x = B("George")
y = A()
try:
    x.print_or_error()
    print("Printed x!")
except AttributeError as ae:
    print("Penny Lane!")
try:
    y.print_or_error()
    print("Printed y!")
except Exception as e:
    print("Exception!")
except AttributeError as ae:
    print("Michelle!")
```

**Program output:**

## Question 6. [7 MARKS]

Answer the following short-answer questions about sorting. You do *not* need to write code for these.

### Part (a) [2 MARKS]

Selection sort and insertion sort are both $O(n^2)$ in the worst case. However, one of them performs significantly fewer comparisons in the average case. Which one and why?

### Part (b) [2 MARKS]

Assume you have a version of quicksort that uses the last element in a list as the pivot. Order the integers from 1 to 10 in such a way as to make quicksort run as slowly as possible. (If there are multiple ways to do this, just provide one of them.)

### Part (c) [3 MARKS]

Which sorting algorithm would you choose to sort $n$ 9-digit student numbers and why? What would its time complexity be?

## Question 7. [6 MARKS]

Implement mergesort. You may use helper functions.

```
def mergesort(L):
    '''(list) -> list
    Sort list L by recursively sorting its halves and
    recombining.'''
```

## Question 8. [12 MARKS]

This question is about Object-Oriented Design.
You will design a set of classes and methods to satisfy the following specifications:

You have to maintain a simple social network that consists of individual profiles. Each profile holds a person's name, date of birth (including day, month and year) and a data structure that stores that person's friends and makes it possible to retrieve a friend's profile given his or her name. Each profile also holds a list of posts that person has created.
Each post has a date of publication and a list of names of people who are tagged in it, and it supports tagging additional friends after it has been created. Each post is either a note, containing the text of the note, or a picture, containing a string-based path to the image file.

Each profile supports:

- adding or removing a friend given their profile
- checking whether it is that person's birthday
  (assume you can retrieve today's date without passing it in)
- posting a note given its content and a list of friend names to tag in it
- posting a picture given its path and a list of friend names to tag in it

For every class you decide to implement:

- Write the class definition line.
- Write a full `__init__` method and docstring (including signature line).
- Write the method definition line and docstring (including signature line) for the rest of the methods in the class.

Things you do *not* need to write:

- the class docstring
- any code inside methods that are not the `__init__` method of each class

*You may continue your answer to the question on the following page.*

*Continue your answer to the question on this page.*

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.

## 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.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[start:]
    Create a new list containing the items in
    to the end of the list
  L[:end]
    Create a new list containing the items in
    of the list through end-1
  L[:]
    Create a new list containing all the item

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

CONT'D...

Total Marks = 70