PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2008 EXAMINATIONS

CSC 148 H1Y
Instructor(s): R. Danek

Duration — 3 hours

Examination Aids: None.

PLEASE HAND IN

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 22 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:	______/ 14
# 4:	______/ 10
# 5:	______/ 10
# 6:	______/ 10
# 7:	______/ 8
# 8:	______/ 12
# 9:	______/ 16
TOTAL:	______/100

Good Luck!

Question 1. [10 marks]

This question is on both this page and the next.

In this question, assume we have the following Stack implementation, which is identical to the one you saw in lecture:

```
class Stack:
    def __init__(self):
        '''Make a new empty stack.'''
        self.stack = []

    def push(self, o):
        '''Push o onto the top of the stack.'''
        self.stack.append(o)

    def pop(self):
        '''Pop and return the top element of the stack.'''
        return self.stack.pop()

    def top(self):
        '''Return the top element of the stack.'''
        return self.stack[-1]

    def isEmpty(self):
        '''Return True if the stack is empty, and False otherwise.'''
        return self.stack == []

    def size(self):
        '''Return the number of elements in the stack.'''
        return len(self.stack)
```

Below you are given a partial implementation of the Queue ADT. As you can see, it does not use a python list like we did in lecture; instead, it makes use of a Stack instance.

```
class Queue:
    def __init__(self):
        '''Make an empty queue.'''
        self.container = Stack()

    def enqueue(self, o):
        '''Add o to the end of the queue.'''
        self.container.push(o)

    def size(self):
        '''Return the number of elements in the queue.'''
        return self.container.size()
```

Ignore style and efficiency issues in answering the following questions.

Part (a) [4 MARKS]

Is the implementation of `dequeue` below correct, assuming the other methods in the `Queue` class are correct?

Yes [✓ No] (circle one)

If it is not correct, briefly explain in plain english why not.

```
def dequeue(self):
    '''Remove the front element from the queue and return it.'''

    tmpstack = Stack()

    elt = self.container.pop()
    while not self.container.isEmpty():
        tmpstack.push(elt)
        elt = self.container.pop()

    self.container = tmpstack

    return elt
```

SOLUTION: The implementation assumes that the next item to dequeue is at the bottom of the stack. However, when pushing items from self.container to tmpstack, the items on the stack are reversed. This means that after self.container is assigned tmpstack, the assumption that the next item to dequeue is at the bottom of the stack no longer holds.

Part (b) [4 MARKS]

Is the implementation of `dequeue` below correct, assuming the other methods in the `Queue` class are correct?

[✓ Yes] No (circle one)

If it is not correct, briefly explain in plain english why not.

```
def dequeue(self):
    '''Remove the front element from the queue and return it.'''

    return self.container.stack.pop(0)
```

Part (c) [2 MARKS]

Is it possible to implement the Priority Queue ADT using an instance of `Stack`?

[✓ Yes] No (circle one)

Question 2. [10 MARKS]

The *power set* of a set S is the set of all possible subsets of S. For example, the power set of $\{2, 3, 4\}$ is $\{\{\}, \{2\}, \{3\}, \{4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{2, 3, 4\}\}$.

Write a recursive function `powerset(S)` that returns the power set of the set `S`.

Note: Although python has a set data type, assume that we represent sets in this question using python lists. For example, a valid call to your function is `powerset([2,3,4])`, which should return `[[],[2],[3], [4],[2,3],[2,4], [3,4],[2,3,4]]`. The order of elements does not matter.

SOLUTION:

```
def powerset(S):
    '''Return the powerset of S.'''

    # The base case is the empty set.
    if S == []:
        return [[]]

    # To determine the power set of S, first use recursion to determine the
    # power set of S[1:] (i.e., the set S excluding its first element).

    tmpset = powerset(S[1:])
    ret = []

    # The power set of S consists of all the sets of powerset(S[1:]), plus each
    # of those sets with S[0] added to it.

    for subset in tmpset:
        ret.append(subset)
        # make a copy of the current subset and add S[0] to it
        tmp = subset[:]
        tmp.append(S[0])
        ret.append(tmp)

    return ret
```

Question 3. [14 MARKS]

Part (a) [6 MARKS]

For part (a) only, correct answers are worth 2 marks, incorrect answers receive -1 mark penalty, and no answer is worth 0 marks. (The minimum grade for part (a) is 0.)

Circle True or False for each statement.

(2 marks) $log_{10}(n^4)$ is $O(log_2(n))$.	[✓ True]	False
(2 marks) $n(n-4)$ is $O(n^3)$.	[✓ True]	False
(2 marks) n is $O(1)$.	True	[✓ False]

Part (b) [2 MARKS]

Using big-oh notation, state the worst-case time complexity of the following function in terms of n, the length of the input list. Give the strongest answer that you can, that is, the tightest bound. You do not have to justify your answer.

Answer: $\boxed{O(n * log(n))}$

```
def func(inlist):
   n = len(inlist)
   sum = 0
   for i in range(n):
       cur = n
       sum = sum + cur
       while cur > 0:
           cur = cur / 2
```

Part (c) [2 MARKS]

Using big-oh notation, state the worst-case time complexity of the following function in terms of n, the length of the input list. Give the strongest answer that you can, that is, the tightest bound. You do not have to justify your answer.

Answer: $\boxed{O(n)}$

```
def func(inlist):
   n = len(inlist)
   sum = 0
   cur = n
   for i in range(n):
       sum = sum + cur
       while cur > 0:
           cur = cur / 2
```

Part (d) [4 MARKS]

Using big-oh notation, state the worst-case time complexity of the following function in terms of n, the length of the input list. Give the strongest answer that you can, that is, the tightest bound. Assume that all operations on the `Queue` used in the function take $O(1)$ time. **Briefly justify your answer in plain english, using no more than a few sentences.**

```
def func(inlist):
   '''
   Perform some computation using inlist,
   where inlist is a list of non-negative integers.
   '''

   n = len(inlist)
   q.enqueue(inlist[0])
   idx = 1

   while not q.isEmpty():
       elt = q.dequeue()              (*)
       while idx < elt and idx < n:
           q.enqueue(inlist[idx])
           idx = idx + 1
```

SOLUTION: Worst-case time complexity is $O(n)$. Here's why:
One item is added to the queue at the beginning of the function, and the rest are added to the queue in the inner-loop. The inner-loop executes at most $O(n)$ times in total, which follows from the loop termination condition and the fact that `idx` starts at 1 and is incremented in each iteration of the inner-loop. This implies that at most $O(n)$ items can be added to the queue, which, in turn, implies that at most $O(n)$ items can be removed from the queue. Since the outter-loop dequeues a single item in each iteration at line (*) and terminates when there are no more items in the queue, it follows that the function terminates after at most $O(n)$ steps.

Question 4. [10 MARKS]

Consider the class `BTNode` that could be used to implement a binary tree.

```
class BTNode:
    def __init__(self, label):
        self.label = label            # the label for this tree node
        self.left  = None             # the root node of the left subtree
        self.right = None             # the root node of the right subtree
```

Write a recursive function `construct(preorder, inorder)` that takes as arguments two lists of labels. The first argument, `preorder`, is the sequence in which nodes are visited in a preorder traversal of some tree, and the second argument, `inorder`, is the sequence in which nodes are visited in an inorder traversal of the same tree. The return value from the function is the root `BTNode` instance of the tree defined by the traversals.

For example, after the following call

```
btroot = construct(['A','B','C'], ['B','A','C'])
```

the following will hold:

```
btroot.label == 'A'
btroot.left.label == 'B'
btroot.left.left == None
btroot.left.right == None
btroot.right.label == 'C'
btroot.right.left == None
btroot.right.right == None
```

Begin writing the function on the next page.

SOLUTION:

```
def construct(preorder, inorder):
    # the base case is when the traversals contain no elements
    if preorder == []:
        return None

    # the root element is the first element of the preorder list.

    root = BTNode(preorder[0])

    # Scan the inorder list to see where preorder[0] occurs.
    # This will define the dividing point between the
    # traversals of the left and the right subtrees.

    i = 0
    while inorder[i] != preorder[0]:
        i = i + 1

    root.left = construct(preorder[1:i+1], inorder[:i])
    root.right = construct(preorder[i+1:], inorder[i+1:])

    return root
```

Question 5. [10 MARKS]

Part (a) [5 MARKS]

Assume that we insert the following items into a min heap, in order: 3, 6, 9, 12, 15, 1. Draw the tree representation:

SOLUTION:

```
      1
    /   \
   6     3
  / \   /
 12 15 9
```

Fill in the Python list that stores this heap:

`[ 1 , 6 , 3 , 12 , 15 , 9 ]`

Part (b) [5 MARKS]

Assume that we insert the following items into a min heap, in order: 21, 18, 15, 12, 9, 6, 3. After inserting these items, we remove the min element from the heap twice. Draw the tree representation once these operations are finished (you may wish to use one of the rough-work pages at the end of the exam):

SOLUTION:

```
      9
    /   \
   12    18
  / \
 21 15
```

Fill in the Python list that stores this heap:

`[ 9 , 12 , 18 , 21 , 15 ]`

Question 6. [10 MARKS]

Consider the classes `LinkedList` and `Node` that are used to implement a linked list, and the function `search` that can be used for searching linked lists.

```
class Node:
    def __init__(self, data):
        self.data = data              # the data stored in this node
        self.next = None              # the next item in the list

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

    # implementation details of other list methods omitted ...

def search(llist, data):
    '''
    Return a 2-element tuple (prev, node), where node is the Node instance
    containing data and prev precedes node in llist.
    '''

    # implementation omitted ....
```

Using the `search` function, write a function `swap(llist, data1, data2)` that takes the `LinkedList` `llist` as an argument and swaps the **positions of the nodes** containing data `data1` and `data2`. You may assume in your function that nodes containing `data1` and `data2` exist and are unique. Furthermore, to simplify your code, you may assume that the node containing `data1` is earlier in `llist` than the node containing `data2`.

Note: Do **not** simply swap the data in the two nodes. This is not what the question is asking for. It is strongly recommended that you draw a diagram to help you out. Also, think carefully about the different cases that you have to handle.

Implement the method on the next page.

SOLUTION:

```
def swap(llist, data1, data2):

    (prev1, node1) = search(llist, data1)
    (prev2, node2) = search(llist, data2)

    if prev1 is None:
        # node1 is the head of the list.
        # Update node2 to be the new head of the list
        llist.head = node2
    else:
        prev1.next = node2

    if node1 != prev2:
        # nodes are not adjacent

        # swap node1.next and node2.next (can also be done with a tmp variable)
        node1.next, node2.next = node2.next, node1.next

        prev2.next = node1
    else:
        # nodes are adjacent

        node1.next = node2.next
        node2.next = node1
```

Question 7. [8 marks]

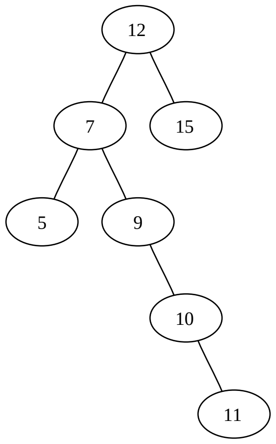
Part (a) [2 marks]

In a binary search tree, given a node n with two children, where is n's successor node? Answer in one sentence.

Node n's successor is the leftmost child of the right subtree.

Part (b) [3 marks]

You are given a binary search tree below. Draw the tree after deleting node 9.

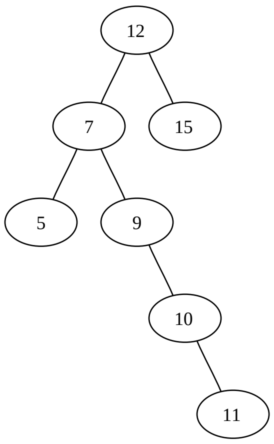


SOLUTION:

```
          12
         /  \
        7    15
       / \
      5   10
            \
             11
```

Part (c) [3 marks]

You are given a binary search tree below. Draw the tree after deleting node 7. (Use either the delete method we studied in lecture, or the modification you implemented in the lab.)



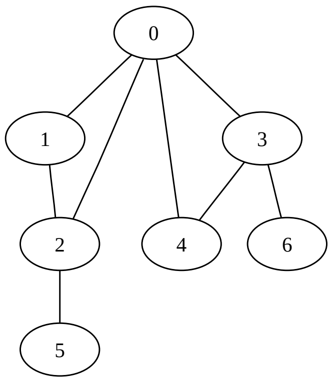
SOLUTION:

```
        12                   12
       /  \                 /  \
      9    15     or       5    15
     / \                    \
    5   10                   9
          \                   \
           11                  10
                                 \
                                  11
```

Question 8. [12 MARKS]

Part (a) [8 MARKS]

Define a *triangle* in an undirected graph to be a set of 3 vertices such that there is an edge between each pair of vertices in the set. For example, the following graph has two triangles: $\{0, 1, 2\}$ and $\{0, 3, 4\}$.



We say that a vertex `v` *is in a triangle* if there is a triangle in the graph that contains vertex `v`. For example, in the above graph, 5 and 6 are the only vertices **not** in any triangle.

You are given the following `Graph` class and its `__init__` method:

```
class Graph:
   '''
   An undirected graph represented using an adjacency matrix.
   '''

   def __init__(self, n):
       '''Create an empty graph with n vertices.'''
       self.n = n
       self.matrix = [[0]*n for i in range(n)]

   # other methods for adding edges, getting neighbours, etc. omitted
```

In the above code, the `matrix` instance variable is an adjacency matrix for an undirected graph. Vertices are non-negative integers less than `self.n`, and entries `self.matrix[v1][v2]` and `self.matrix[v2][v1]` equal 1 if and only if there is an edge between vertices `v1` and `v2` in the graph.

Add to the `Graph` class a method `is_in_triangle(self, v)` that returns `True` if the vertex `v` is in a triangle, and `False` otherwise. Your method should take time $O(n^2)$, where n is the number of vertices in the graph.

Implement the method on the next page. Hint: Your method should be no more than 7 - 8 lines of code. If it's any longer, you're probably on the wrong track.

SOLUTION:

```
def is_in_triangle(self, v):

    # test every distinct triple of vertices containing v to
    # see if they form a triangle

    for i in range(self.n):
        if self.matrix[v][i] == 1:
            for j in range(self.n):
                if self.matrix[i][j] == 1 and self.matrix[v][j] == 1:
                    return True
    return False
```

Part (b) [4 MARKS]

Define a *sink* to be a vertex in a directed graph that has no outgoing edges and at least one incoming edge. Define a *source* to be a vertex in a directed graph that has no incoming edges and at least one outgoing edge. Draw a **directed** graph that has **exactly one source and one sink**, but for which there is no path between the two. Label the source and the sink in your graph.

SOLUTION:

```
       OOO ---->  OOO        OOO --------> OOO
                  /\|        /\|
                  | |        | |
                  | |        | |
                  | \/       | \/
                  OOO        OOO

Each line of 3 O's represents a node. The source is node
on the far left, and the sink is the node on the far right.
```

Question 9. [16 marks]

Part (a) [4 marks]

```
def mystery(inlist):
    '''
    Perform some computation on inlist, where inlist
    is a non-empty list of integers.
    '''

    sublists = [[]]
    last = inlist[0]
    sublists[0].append(last)

    idx = 1
    n = len(inlist)

    while idx < n:
        if inlist[idx] < last:
            sublists.append([])
        sublists[-1].append(inlist[idx])
        last = inlist[idx]
        idx = idx + 1

    return sublists
```

```
def mysteryprint(inlist):
    '''
    Print some information based on inlist, where inlist
    is a non-empty list of integers.
    '''

    sublists = mystery(inlist)

    mysublist = sublists[0]
    mylen = len(sublists[0])
    for b in sublists:
        if len(b) > mylen:
            mylen = len(b)
            mysublist = b

    print mysublist
```

Describe in plain english using **one sentence** what `mysteryprint` **prints to the screen**. Be specific about how the output relates to the original input list.

SOLUTION: `mysteryprint` prints the first occurrence of the longest contiguous sequence of sorted integers in `inlist`.

Part (b) [12 MARKS]

It turns out that we can use the return value from `mystery` to implement a sorting method as follows:

```
def mysterysort(inlist):
    ''' Return a sorted list of items consisting of the items in inlist. '''

    sublists = mystery(inlist)

    while len(sublists) > 1:
        newlists = []
        curlist = 0
        while curlist + 1 < len(sublists):
            list1 = sublists[curlist]
            list2 = sublists[curlist+1]
            tmp = mysteryhelper(list1, list2)
            curlist = curlist + 2
            newlists.append(tmp)

        if curlist < len(sublists):
            newlists.append(sublists[curlist])

        sublists = newlists

    return sublists[0]

def mysteryhelper(list1, list2):
    tmp = []
    i = 0
    j = 0

    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            tmp.append(list1[i])
            i = i + 1
        else:
            tmp.append(list2[j])
            j = j + 1

    while i < len(list1):
        tmp.append(list1[i])
        i = i + 1
    while j < len(list2):
        tmp.append(list2[j])
        j = j + 1

    return tmp
```

Question is continued on the next page.

This sorting method is very similar to a sorting method that we studied in class.

(2 marks) What is that sorting method? $\boxed{\text{Merge sort}}$

(1 mark) Circle one of the following. Asymptotically, the efficiency of `mysterysort` is

[better than / [✓ the same as] / worse than]

the efficiency of the sorting method to which `mysterysort` is similar.

(2 marks) What is the main drawback that `mysterysort` suffers from in terms of its usage of space?

$\boxed{\text{It requires a lot more space in addition to the original list being sorted (i.e., it is not “in-place”).}}$

(1 mark) Name one sorting method that we studied in class that does not have this drawback:

$\boxed{\text{Any one of quicksort, bubble sort, selection sort, insertion sort}}$

(3 marks) Is `mysterysort` still a correct sorting method if `mystery` is replaced with the definition below?

[✓ Yes] No (circle one)

In case it is no longer a correct sorting method, briefly describe why not.

```
def mystery(inlist):
   ret = []
   for i in range(len(inlist)):
       ret.append([inlist[i]])
   return ret
```

(3 marks) Is `mysterysort` still a correct sorting method if `mystery` is replaced with the definition below?

Yes [✓ No] (circle one)

In case it is no longer a correct sorting method, briefly describe why not.

```
def mystery(inlist):
   mid = len(inlist) / 2
   return [inlist[0:mid], inlist[mid:]]
```

SOLUTION: Mergesort requires that the sublists it merges are sorted. The two sublists returned by this implementation of `mystery` may not satisfy that requirement.

[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 = 100