CMPUT 275 - Introduction to Tangible Computing II
Practice Final Exam - Winter 2018
Department of Computing Science, University of Alberta
Instructors: Omid Ardakanian & Zachary Friggstad

Last Name: _____

ID: _____

Instructions

- Print your last name and ID clearly above.
- You have 180 minutes to complete the final, although it was designed to be completed 120 minutes.
- There should be 8 questions and 13 pages in this exam booklet. You are responsible for checking that your exam booklet is complete.
- This is an **OPEN BOOK** exam. You may use your notes, laptop, Arduino, or an internet search. You **may not directly query anyone** (through email, IM, text message, etc.) for answers: all answers must be your own. As usual, you should document any and all sources used including.
- Place most answers in the spaces provided on the question pages. The answers to all questions are straight forward and brief. Written answers should fit easily into the space provided. Code answers will be submitted via eClass, but they should not require overly complex programming. Think about each question a bit before answering it. Don't forget to document your code.
- This exam counts 30% toward your final grade in this course. This exam is worth 100 points. The weight of each question is indicated in square brackets by the question number.

Question	1	2	3	4	5	6	7	8	Total
Out of	12	12	9	12	12	10	12	12	91

Question 1 [12 marks]: You are given the following function:

```
import math

def bar(n):
    a = []
    for i in range(2,int(math.sqrt(n))+1):
        if not i in a:
            for j in range(i*i, n+1, i):
                a.append(j)
    return [i for i in range(2,n) if i not in a]
```

Part 1.1 [2 marks]: What is the output of `bar(30)`?

Part 1.2 [4 marks]: Describe in *one sentence* what this function computes.

Part 1.3 [4 marks]: What small change(s) to this implementation would improve this function's running time?

Part 1.4 [2 marks]: Explain in *one sentence* how and why your change(s) help(s).

Question 2 [9 marks]: Quick Coding. Each question below should be solved in one or two lines of python code. Write your code in the space below each subquestion. Note: an *expression* is a **single** line of python which can be evaluated to get a value.

Part 2.1 [3 marks]: You are given a list `my_list`. Write an *expression* which evaluates to a new list containing all but the first and last elements of `my_list`. Note that this may return an empty list in the cases where `my_list` has only 1 or 2 elements.

Part 2.2 [3 marks]: Write an *expression* that evaluates to a dictionary which maps the lowercase letters of the alphabet to their position in the alphabet. In this case, `a` should map to `0`, `b` to `1`, etc. It may be helpful to use `string.ascii_lowercase`. You can assume the string library has already been imported.

Part 2.3 [3 marks]: Supppose you are representing books as a tuple `(book_title, num_pages)`. You are given a variable `library` that is a list of book tuples. Write an *expression* that sorts the list `library` from shortest to longest based on each book's number of pages.

Question 3 [12 marks]:

Indicate whether each statement is `True` or `False`. Write the whole word.

Part 3.1 [2 marks]: $n \log_2 n \in O(n^2)$

Part 3.2 [2 marks]: $6n^2 - 3n - \sqrt{n} - 1 \in \Theta(n^2)$

Part 3.3 [2 marks]: $100n + \frac{n^2}{100} \in \Theta(n)$

Part 3.4 [2 marks]: $1 + \frac{1}{n} \in \Theta(1)$

Part 3.5 [2 marks]: $n! \in O(2^n)$ where $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$

Part 3.6 [2 marks]: $f(n) + g(n) \in O(\max(f(n), g(n))$ for any eventually-positive functions f, g.

Question 4 [12 marks]:

A string `s` containing only opening and closing parenthesis is called *balanced* if for for every index `i`, the substring `s[0:i]` (the first `i` characters) contains at least as many '`(`' characters as it does '`)`' characters and `s` itself contains an equal number of '`(`' characters as it does '`)`' characters.

For example, `(())()(()())` is balanced but `(()())))()` is not because the substring `s[0:7] = (()()))` contains more `)` characters than `(` characters.

Let $b(n)$ denote the number of balanced strings of length n. Say that $b(0) = 1$ (because there is only one "empty" string). The following recurrence computes $b(n)$ for integers $n \geq 0$.

For integers $n \geq 0$:

$$b(n) = \begin{cases} \sum_{k=2}^{n} b(k-2) \cdot b(n-k) & \text{if } n \geq 2 \text{ and } n \text{ is even} \\ 1 & \text{if } n = 0 \\ 0 & \text{if } n \text{ is odd} \end{cases}$$

Part 4.1 [9 marks]: Write a Python function `count_balanced` that takes a nonnegative integer n and returns the value of $b(n)$ using *dynamic programming*. You may use either a bottom-up or a top-down approach.

Part 4.2 [3 marks]: What is the running time of computing $b(n)$ using the memoized implementation of `count_balanced`? Give the best possible running time bound as a function of n.

Question 5 [12 marks]:

Consider the following class which manages a site license or a certain number of individual licenses and creates a new license only if the total number of license created so far does not exceed a given quota:

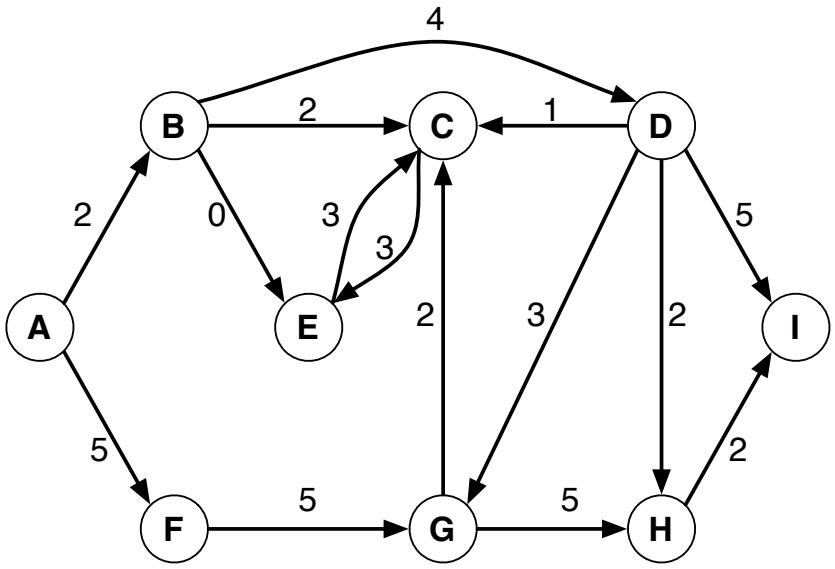
```
class LicenseManager:
  __instance_count = 0  # class attribute
  def __new__(cls):
    return super().__new__(cls)

  def __init__(self, lname):
    self.licenseNumeber = lname    # instance attribute

  def print_user(self):
    print("The license number is", self.licenseNumeber)
```

Here *__new__(cls)* is the class constructor which is responsible for creating a new instance and returning it. The object creation is done by calling the constructor of the super class.

Part 5.1 [6 marks]: Suppose the CS department has a site license for MATLAB which can be used by all users in our department. Modify the *__new__(cls)* method to ensure that only one instance of this class is instantiated during runtime. If the user tries to create a second instance, the constructor should return the first instance that has been created.

Part 5.2 [6 marks]: Suppose the CS department has purchased $K \geq 1$ individual MATLAB licenses, where K is a global variable which can be seen inside the LicenseManager class. Hence, the user can create at most K instances of this class and a further attempt to create an instance will not result in object creation. Modify the LicenseManager class to guarantee that the constructor simply returns the last created instance if K instances have been created so far. Otherwise, a new instance should be created and returned to the user.

Question 6 [10 marks]: Consider the following directed graph. The number beside each edge indicates the length of the edge.



We assume that the outgoing neighbours of a vertex are listed in alphabetic order. For example, the outgoing neighbours of **G** are [**C**, **H**].

Ignore the edge weights in the first two questions below.

Part 6.1 [3 marks]: Consider a depth-first search traversal of the graph starting from vertex **A**. What order would the vertices be first removed from the stack? When iterating through the neighbours of a recursive call for a specific vertex, iterate through them in alphabetic order.

Part 6.2 [3 marks]: Now consider a breadth-first search traversal of the graph starting from vertex **A**. What order would the vertices be first removed from the queue? Again, when adding the neighbours of a vertex to the queue, consider them in alphabetic order.

Part 6.3 [4 marks]: Now run Dijkstra's algorithm starting from vertex **A**. List the nodes in order of their first removal from the heap. Also write the length of the shortest path from **A** to every other node. If you ever encounter a tie for the vertex that should be expanded next, pick the one that is least in alphabetic order.

Question 7 [12 marks total]: You are given two lists of length n. One list, `inc_list`, is sorted in increasing order. The other list, `dec_list`, is sorted in decreasing order. For example,

```
inc_list = [2,2,6,8]
dec_list = [9,7,6,2]
```

gives an example of a valid pair of lists, as all the elements are increasing in `inc_list`, and all the elements are decreasing in the `dec_list`.

Note that there is no relation between these lists other than that they are both of length n – they may or may not share common elements. Write a function `find_intersection` to determine if there exists an index `i` such that `inc_list[i]` and `dec_list[i]` are equal. If such an index exists, return `i`. If no such index exists, return `None`. Your code should run in $O(\log(n))$ time. Submit your code in a file named `"intersection.py"` via eclass. Nothing on this page will be marked.

Question 8 [12 marks]:

Part 8.1 [3 marks]: You are given elements:

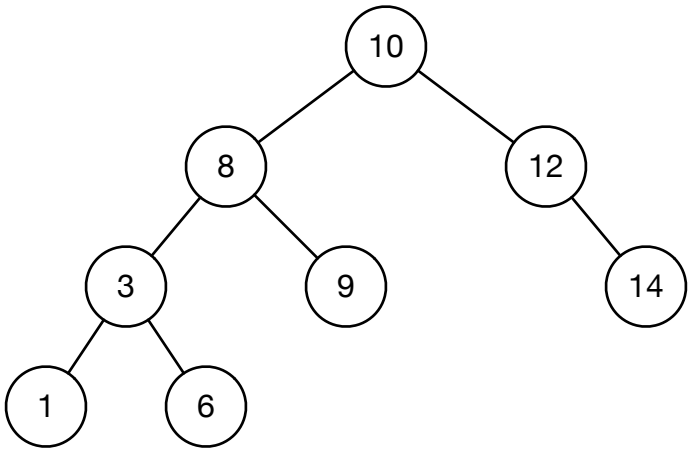
```
1, 4, 5, 7, 11, 13, 14, 15, 16, 21, 24, 28, 33, 35, 36
```

Draw a binary search tree containing these elements as keys with a *height* of 3.

Part 8.2 [3 marks]: What order would you insert the elements above in order to construct the binary search tree you drew above? (Note: Assume you're using a plain binary search tree implementation, not a self-balancing tree like an AVL tree).

Order:

Part 8.3 [3 marks]: The following depicts an AVL tree (only the keys are shown). Insert a new node with key 2 using the insertion algorithm from the lectures and then use rotations to rebalance the tree. Show the tree before and after every single rotation. Clearly indicate which rotation you are using in each step.



Part 8.4 [3 marks]: Using the same tree depicted on the previous page (the original tree, not your final tree), remove the node with key 12. You should delete it using the algorithm we discussed in class for deleting a node from an AVL tree. Depict the tree immediately after the key is removed, and then show the tree after every single rotation. Again, clearly indicate which rotation you are using in each step.