Mock Final Exam. Work alone for the first 30 minutes. (This will be graded for participation only.)
NOTE: These problems were written by the instructor.

1. (Linked Lists) Use the `LinkedList` and `Node` class definitions below to answer the question. You may directly access the attributes or use the getters defined below.

```
class Node:                              class LinkedList:
    def __init__(self, value):               def __init__(self):
        self._value = value                      self._head = None
        self._next = None
                                             def add(self, new):
    def get_next(self):                          new._next = self._head
        return self._next                        self._head = new

                                             def get_head(self):
                                                 return self._head
```

Write a function `compare(llistA, llistB)` that takes linked lists `llistA` and `llistB` as arguments and returns the following:

> -1 if `llistA` is shorter than `llistB`
> 1 if `llistB` is shorter than `llistA`
> 0 if the lists are the same length

Your solution **must not** simply count the lengths of the lists and compare them, since the complexity would be proportional to the sum of the lengths of the lists.

Instead, the complexity of your solution **must** be proportional to the length of the smaller list. *Use an iterative solution.*

ANS:

```
def compare(llistA, llistB):
    cur_a = llistA.get_head()          # or use llistA._head
    cur_b = llistB.get_head()          # or use llistB._head

    while cur_a != None and cur_b != None:
        cur_a = cur_a.get_next()       # or use cur_a._next
        cur_b = cur_b.get_next()       # or use cur_b._next

    if cur_a != None:
        return 1
    if cur_b != None:
        return -1
```

```
    return 0
```

2. (Trees) Consider the following definition of a BinaryTree class:

```
class BinaryTree:
    def __init__(self, value):
        self._value = value
        self._left = None
        self._right = None
```

Use this class to write the solution to the problem below. You may directly access the attributes of the `BinaryTree` class when writing your function, or you may use typical names for getters/setters of the attributes.

Write a function `notate_interior(tree)` that takes a binary tree `tree` and produces an *inorder* traversal of the tree, with "I" following any interior node. The output should print the value of each node, one per line. For example, for the following tree
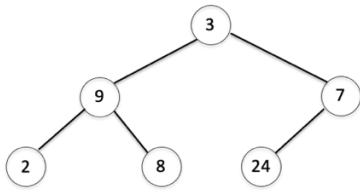
the output would be

```
2
9 I
8
3 I
24
7 I
```

ANS:

```
def notate_interior(tree):
    if tree == None:
        return
    notate_interior(tree._left)
    if tree._left == None and tree._right == None:
        print(tree._value)
    else:
        print(tree._value, "I")
    notate_interior(tree._right)
```

3. (Recursion) Write a **recursive** function `every_third(alist)` that returns a list consisting of the elements at every 3rd position of the Python list `alist`. The first element is at position 0; "every 3rd position" means the elements at positions `0, 3, 6, 9,...`. For example, the call `every_third([11,22,33,44,55,66,77,88])` should return the list `[11,44,77]`. Assume there is at least one element in the list.

ANS:

```
def every_third(alist):
    if alist == []:
        return []
    if len(alist) <= 3:
        return [alist[0]]
    else:
        return [alist[0]] + every_third(alist[3:])
```

4. (Short answer.)
   a) Some data types in Python are *immutable*. Describe in one sentence what that means and give an example of an immutable data type.

ANS:
Immutable data types cannot be modified. For example, an object of type `str` cannot be modified, meaning that the individual characters of the string cannot be changed.

   b) What method is defined in a class so that two objects of that class can be compared with the "==" operator?

ANS:

`_ _eq_ _(self, other)`

   c) How is a binary search tree different from a regular binary tree?

ANS:
A binary search tree satisfies the property that the value of every node (or tree) on the left side of the tree is less than the tree's value, and the value of every node (or tree) on the right side of the tree is greater that the tree's value.

5. (References) Write the code that would produce the data objects in the diagram below:
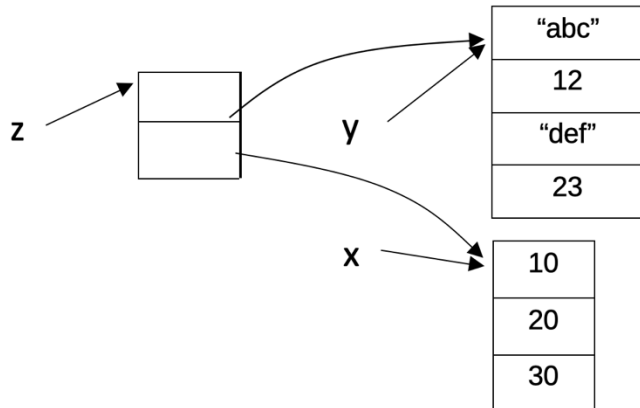
Put your code below:

ANS:

```
y = ["abc", 12, "def", 23]
x = [10,20,30]
z = [y,x]
```

6. (Complexity) Consider the function defined below:

```
def has_overlaps(list_a, list_b):
    for item in list_b:
        if item in list_a:
            return True
    return False
```

a) Explain why the code below is $O(n^2)$.

ANS:
The code has two nested loops. The outer loop iterates over every element of `list_a` and compares each element to every element of `list_b`. If there are no elements in common, each loop will iterate over each of its elements. If we say that the lists each have size n, then the big-O complexity is $O(n^2)$.

b) Give an example of two lists that demonstrate the worst-case complexity.

ANS:

```
list_a = [1,3,5,7,9]
list_b = [2,4,6,8,10]
```

7. (Complexity) Under what condition is searching for an item in a list O(log(n))?

    ANS:
    If the list is sorted. (Use binary search algorithm to search the list.)

8. (General programming) Write a function `get_min(invent_dict)` that takes a dictionary of strings that map to integers and returns a list containing a tuple of the key/value pair that has the minimum value. Assume that all values are 0 or greater. **If there are ties, the function returns all of the key/value pairs that have the same minimum.**

    For example, given the dictionary `d` below

    `d = {"spoons": 7, "knives" : 8, "forks": 6, "plates": 10, "cups": 6}`

    The call `get_min(d)` returns `[ ("forks", 6), ("cups", 6) ]`.

    Restrictions: You may not use list comprehensions or the built-in functions `min()` or `sort()`. Your function must have complexity O(n) and must only iterate through the dictionary *one time*.

    ANS:

```
def get_min(invent_dict):

    min_key, min_val = None, None
    result = []

    for key, value in invent_dict.items():
        if min_key is None:
            # initialize with the first key/value pair we see
            min_key, min_value = key, value
            result = [ (min_key, min_value) ]
        elif value == min_value:
            result.append((key, value))
        elif value < min_value:
            # reset the minimums and the result list
            min_key, min_value = key, value
            result = [ (min_key, min_value) ]

    return result
```