**15-112 Spring 2021 Exam 2**

**Name:**

**Andrew ID:**

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- All code samples run without crashing. Assume any imports are already included as required.
- You may assume that math, string, and copy are imported; do not import any other modules.

**1. Efficiency [20 pts]** For each function shown below, write...

1. The overall Big-O runtime of the function in terms of N, the length of `L`.
2. An explanation of your answer.

All Big-O answers must be simplified: Do not include lower-order terms.

**A.**

```
def r1(L, z):
    n = len(L)
    a = max(L)
    for x in range(n):
        a += x
    return a
```

Big-O:

Explanation:

**B.**

```
def r2(L, z):
    a = max(len(L)-1, z)
    L[0] += L[a]
    return max(L[a],z)
```

Big-O:

Explanation:

**C.**

```
def r3(L, z):
    n = len(L)
    x = 0
    y = 0
    while (x < n):
        while (y < n):
            y += 1
            z += y
        x += 1
        z += x
    return z
```

Big-O:

Explanation:

**D.**

```
def r4(L,z):
    for i in range(len(L)):
        if L[i] == sorted(L)[z]:
            return i
    return False
```

Big-O:

Explanation:

**2. Free Response – OopyWars [20 pts]** Write the classes ForceUser and Jedi to pass the test cases shown below. Do not hardcode against the testcase values, though *you can assume the testcases cover needed functionality.*

For full credit you must use inheritance appropriately, including not needlessly duplicating code.

```
# A ForceUser has a name. It also has a lightsaber,
# which can either be None (no lightsaber) or a
# string (the color of the lightsaber).
# The default lightsaber is None.
assert(str(ForceUser('Luke')) == 'Force User Luke')
assert(str(ForceUser('Yoda', 'green')) == 'Force User Yoda, green lightsaber')

# You can give a lightsaber to a ForceUser,
# but only if they don't have one already
rey = ForceUser('Rey')
assert(str(rey) == 'Force User Rey')
assert(rey.giveLightsaber('blue') == 'Rey got a lightsaber!')
assert(str(rey) == 'Force User Rey, blue lightsaber')
assert(rey.giveLightsaber('red') == 'Rey already has a lightsaber!')
assert(str(rey) == 'Force User Rey, blue lightsaber')

# Here are a variety of things that should
# work for a ForceUser
luke = ForceUser('Luke')
assert(luke == ForceUser('Luke', None))
assert(luke != ForceUser('Leia'))
assert(luke != 'do not crash here!')
s = set()
assert(luke not in s)
s.add(luke)
assert(ForceUser('Luke') in s)
assert(ForceUser('Luke', 'green') not in s)

# A Jedi is a ForceUser who can become a ghost
# if you strike them down.
# But not all Jedi are ghosts!
obiWan = Jedi('Obi-Wan', 'blue')
assert(isinstance(obiWan, ForceUser) == True)
assert(obiWan.giveLightsaber('purple') == 'Obi-Wan already has a lightsaber!')
assert(obiWan.isGhost() == False)
obiWan.strikeDown()
assert(obiWan.isGhost() == True)

# you can't strike down a non-Jedi, because
# ForceUser doesn't implement strikeDown.
crashed = False
try:
    rey.strikeDown()
except:
    crashed = True
assert(crashed == True)
```

**3. Free Response – Recursion [15 pts]** **Your solution for this problem must use recursion. If you use any loops, comprehensions, or iterative functions, you will receive no points on this problem. Note that the built-in Python function `count` is iterative, so you can't use it.**

Write the recursive function `wordCounter(wordList, word)` which, given a list of words and a word, returns how many times word appears in the list.

For example, the following code:

```
wordList = ["father","bless","of","golf","of","golf","father","of"]
print(wordCounter(wordList, "of"))
print(wordCounter(wordList, "golf"))
print(wordCounter(wordList, "fred"))
```

prints

```
3
2
0
```

**4. Free Response – Invert Dictionary [20 pts]** Write the non-destructive function `invertDictionary(d)` that takes a dictionary `d` that maps keys to values and returns a dictionary of its inverse, that maps the original values back to their keys.

This has a complication: There can be duplicate values in the original dictionary. That is, there can be keys k1 and k2 such that `d[k1] == v` and `d[k2] == v` for the same value `v`. In that case, `invertDictionary` should map the original value back to the *set* of all the keys that originally mapped to it.

Consider the following example:

```
d = {'cat':5, 'dog':6, 'bunny':5, 'bird':7}
print(invertDictionary(d))
```

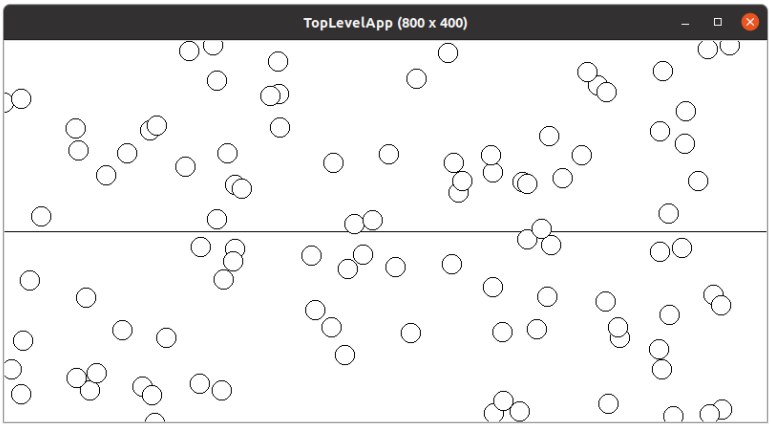
prints:

```
{5: {'cat', 'bunny'}, 6: 'dog', 7: 'bird'}
```

Note: You may assume that all keys and values used will be immutable.

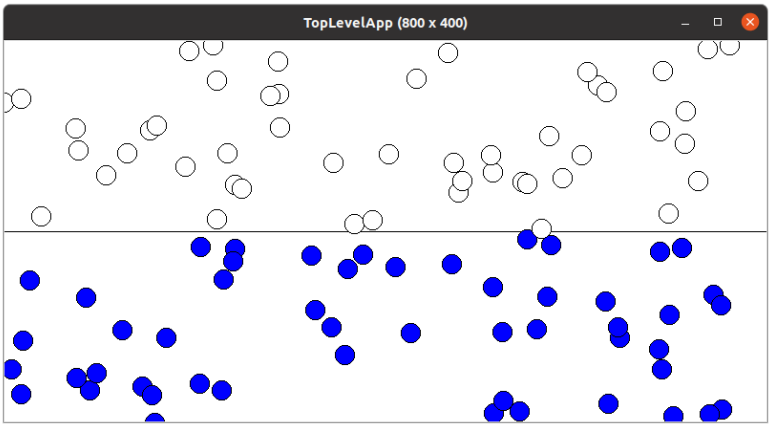
**5. Free Response - Clicky Dottie Game [25 pts]** Write the functions `appStarted(app)`, `keyPressed(app, event)`, `mousePressed(app, event)`, `redrawAll(app, canvas)`, and any other helper functions you might need in order to implement the following (boring) game.

When game starts, the board is split in half horizontally and 100 circles are placed randomly on the board. The circles each have a radius of 10.

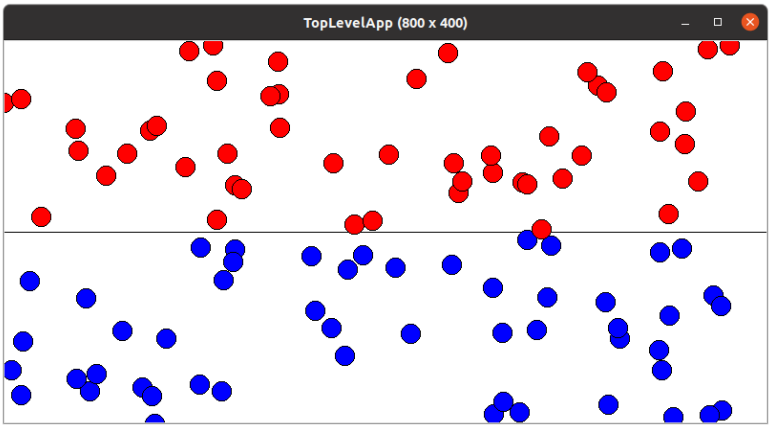



If the user clicks on the top half of the board, then all the circles whose centers are in the top half of the board turn red. If the user clicks on the bottom half of the board, then all the circles whose centers are in the bottom half of the board turn blue. The user can click on the top-half or bottom-half in any order.

Here is the view after the user clicks on the bottom half of the board:




If the user then clicks on the top half of the board, it looks like:



At anytime, the user can press the “r” key to reset the board with new circles and play again, like so: