Name:_____ andrewID:_____ Recitation:___

15-112 F24
Quiz9 version B (25 min)

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 4pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 9.
    - Do not use OOP.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. Assume almostEqual(x, y) and rounded(n) are both supplied for you.
    - You must write all other helper functions you wish to use, unless we specify otherwise.
9. Good luck!

Free Response / FR1: elementCount(L) [30 pts]

Hint: this problem is similar to flatten, which we covered in recitation.

Without using iteration (for or while loops), write the recursive function elementCount(L) that takes a list L that may contain sublists (which themselves may further contain sublists), and returns a count of the number of non-list values in L.

For example:

```
assert(elementCount([1, 1, 1]) == 3)
assert(elementCount(['dog', ['cat', 'mouse']]) == 3)
assert(elementCount([1, 2, [3, [4, 5], 6], 7]) == 7)
assert(elementCount([ ]) == 0)
assert(elementCount([ [ ], [ ] ]) == 0)
```

Begin your FR1 answer on the next page.

Begin your answer to FR1 here:

Free Response / FR2: getTotalsMap(countsList) [35 pts]

Background: This problem uses a "counts list" (a coined term), which is a possibly-empty list of tuples of (label, count), like so:

```
[ ('a', 4), ('b', 3), ('b', 5), ('a', 2), ('z', 1) ]
```

Here, 'a' has a count of 4, 'b' has a count of 3, 'b' has another count of 5, and so on.

From this, we can construct a "totals map" (another coined term), which is a dictionary mapping each label in a counts list to the sum of all of its counts. In the previous example, since 'a' has counts of 4 and 2, its total is 6. Here is the full totals map for the counts list above:

```
{ 'a':6, 'b':8, 'z':1 }
```

With this in mind, without using iteration (for or while loops), write the function getTotalsMap(countsList) that takes a countsList as described and returns its corresponding totals map.

Here are some examples:

```
countsList = [ ('a', 4), ('b', 3), ('b', 5),
               ('a', 2), ('z', 1) ]
totalsMap = { 'a':6, 'b':8, 'z':1 }
assert(getTotalsMap(countsList) == totalsMap)

countsList = [ ]
totalsMap = { }
assert(getTotalsMap(countsList) == totalsMap)
```

Hint: we found it helpful to use a wrapper function here.

Begin your FR2 answer on the next page.

Begin your answer to FR2 here:

Big O [10 pts, 2.5 pts each]

For each problem, indicate the Big O (in simplified form) of the code.
Place your answer (and nothing else) in the box to the right of the code.
Assume L is a list with N integers.
Assume S is a set with N integers.
Assume N is a positive integer.

 # Big O #1:

```
b = (v in L) and (v in S)
```

 # Big O #2:

```
total = 0
while N > 0:
    total += N
    N //= 2
```

 # Big O #3:

```
for x in range(N):
    for y in range(x+1, N):
        L.append(x+y)
        L.pop(0)
```

 # Big O #4:

```
for v in sorted(S):
    for k in range(42):
        print(L.count(v))
        print(L.count(k))
```

Code Tracing (CT) [10 pts, 5 pts each]
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

CT1:

```
def ct1(n):
    if n < 10:
        return [n]
    else:
        d = n%10
        if d % 2 == 0:
            return ct1(n//10) + [d]
        else:
            return [d] + ct1(n//10)
print(ct1(7654))
```

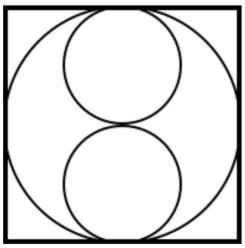
CT2:

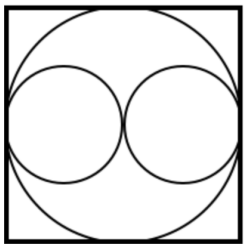
```
def ct2(L):
    if len(L) < 2:
        return L
    else:
        i = len(L)//2
        left = L[:i]
        right = L[i:]
        return ct2(left) + [sum(L)] + ct2(right)
print(ct2([1,2,4]))
```

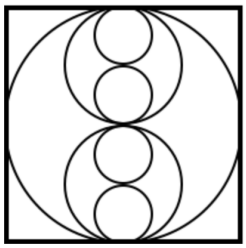
Fractal Multiple Choice [5 pts]
Consider the following code:

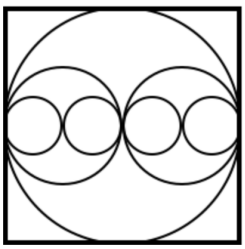
```
from cmu_graphics import *

def drawFractal(level, cx, cy, r):
    if level == 0:
        drawCircle(cx, cy, r, fill=None, border='black')
    else:
        drawCircle(cx, cy, r, fill=None, border='black')
        drawFractal(level-1, cx-r/2, cy, r/2)
        drawFractal(level-1, cx+r/2, cy, r/2)

def redrawAll(app):
    drawRect(100, 100, 200, 200, fill=None,
             border='black', borderWidth=4)
    drawFractal(2, 200, 200, 100)

runApp()
```

Which one of the following will be drawn when the code above runs?
Assume each box is the 200x200 rectangle drawn in the code.
(Circle the letter corresponding to your answer.)

A 

B 

C 

D 

E None of these

Recursion Short Answer [10 pts]

Rewrite the following function so it works identically only using recursion and without iteration.

```
def f(L):
    t = 1
    for v in L:
        t *= v**2
    return t
```

Place your answer in the box:

Bonus Code Tracing (BonusCT)) [Optional, 4 pts, 2 pts each]
Bonus problems are not required.
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

BonusCT1:

```
def bonusCt1(n):
    def f(n):
        return sum(range(n+1)) + f(n-1) if n else n
    def g(n):
        return f(n)//(n+2)
    return g(n)
print(bonusCt1(5))
```

BonusCT2:

```
def bonusCt2(n):
    def h(n, g=2, f=0):
        if g == n:
            return [ ] if f else [n]
        else:
            return h(n, g+1, f+int(n%g==0))
    return [ ] if n <= 15 else h(n) + bonusCt2(n-1)
print(bonusCt2(25))
```