# 15-112: Practice Final Exam Questions

Angela Zhang (afzhang) and Michael Choquette (mchoquet)

May 3, 2014

## True or False

a. TRUE or FALSE: Anything computed with recursion can also be computed without recursion.

b. TRUE or FALSE: If we changed floodFill so that sometimes it recursed in the order up/down/left/right, and other times it recursed in the order up/left/right/down, it would sometimes fail to completely floodFill some regions.

c. TRUE or FALSE: A set can be a key in a dictionary, but a dictionary cannot be added to a set.

d. TRUE or FALSE: In our version of Model-View-Controller animations, we stored our model in canvas.data, and we called various canvas methods in our view.

e. TRUE or FALSE: Any binary number that ends in a 1 must be odd.

f. TRUE or FALSE: Without canvas.delete(ALL) in our redrawAll function, our animation would get progressively slower.

g. TRUE or FALSE: Any binary number that contains exactly one 1 must be a power of 2.

h. TRUE or FALSE: Lambda functions cannot call themselves.

i. TRUE or FALSE: If x and y are positive integers where x < y, then (x / y) will always equal zero and (x % y) will always equal x.

j. TRUE or FALSE: Any list of N integers, where they are all between 1 and 5, can be sorted in O(N) time.

# (Very) Short Answer

a. If (as in the notes) we consider a Level 0 Koch Snowflake to be just a straight line (not a triangle, so this is really one-third of a snowflake), then sketch a Level 2 Koch Snowflake and clearly identify all the Level 1 Koch Snowflakes within it.

b. For positive int values x, k, write an equivalent expression to (x%(2**k)) that does not use %, *, or /.

c. What is steganography?

d. When you run a script in python, you can include the -i flag, like this: *python -i foo.py*. What does the -i flag do?

e. Convert 112 into binary.

f. Given the tuple T containing an odd number of int values, write a single expression, not a statement, that computes the median value in T. For half credit, you may write a function instead.

g. Prove any version of DeMorgans Law using truth tables.

h. Write an expression that gives you a 15 by 112 2D list.

i. What is the runtime of the recursive Fibonacci function?

j. Sets cannot be used as a key in a dictionary. If you try using set as a key in a dictionary, an error saying: (unhashable type: 'set') will appear. Explain what this means and why a set is unhashable!

k. What is the purpose of the __init__ method?

l. Draw the first 3 levels of the Sierpinski triangle.

m. Briefly describe how A* differs from BFS.

n. What is -5 >> 2?

o. What is the big-Oh of mergesort? Why?

p. Why do we use the least significant bits in steganography?

# Free Response

1. Write a function, **integerSquareRoot(x)**, that takes a possibly-floating-point non-negative number and returns the integer value that is nearest to its actual square root.

2. Write the function **hasBalancedParentheses(s)**, which takes a string and returns True if that code has balanced parentheses and False otherwise (ignoring all non-parentheses in the string). We say that parentheses are balanced if each right parenthesis closes (matches) an open (unmatched) left parenthesis, and no left parentheses are left unclosed (unmatched) at the end of the text. So, for example, "( ( ( ) ( ) ) ( ) )" is balanced, but "( ) )" is not balanced, and "( ) ) (" is also not balanced.

3. Write the function **missingNumber(a)** that takes an unsorted list of size n that contains all but one of the integers from 1 to (n+1). Return the missing number. Your function must run in O(n) time, so it cannot sort the list.

4. Write the function **smallestDifference(a)** that takes a list of integers and returns the smallest absolute difference between any two integers in the list. If the list is empty, return -1.

5. Write the function **map(f, l)**, which does not use the builtin map function, and which takes a function f and a list l, and returns a new list containing f(x) for each value x in l. You may assume all elements of l are of the same type. For example, say you defined a function plus3(x) that returns (x+3). Then, map(plus3, [2,4,7]) returns [5,7,10].

6. Write the function **reduce(f, b, l)** which takes a function f, a base case b, and a list l. You may assume all elements of l are of the same type. reduce should take the list l and return a single item of the same type as the elements of l, using f to combine the elements with b as the base case. For example, say you have a function add(x, y) that returns x + y. reduce(add, 0, [1,2,3]) returns 6.

7. Write the 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. One 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, what should inverseD[v] equal? Answer: map the original values back to the set of keys that originally mapped to them. Thus, in this example, inverseD[v] maps to a set containing both s1 and s2.

8. Write a function **countKosbie(s)** that takes a string s and returns how many instances of kosbie (case-sensitive) occur in the string. Note, however, that koskosbiebie should return 2 since after you remove the nested kosbie from the string it contains another instance of kosbie.

9. Write a function **uniqueLetterCount(s)** that takes a string s and returns the number of unique letters, ignoring case, that occur in that String. For example, "Wowee" contains the letters 'e', 'o', and 'w' (ignoring case), so uniqueLetterCount("Wowee") should return 3. Ignore all non-letters. The null string contains no unique letters.

10. Write a recursive **fib(n)** function that returns the nth Fibonacci number. Then, using a dictionary (and NOT @memoized) write a function **memoizedFib(n)** that computes the Fibonacci numbers in O(n) time. Make sure to store the intermediate values, so when you compute fib(15) you should also be able to compute fib(12) immediately!

11. Without loops or using any built-in functions, write a function **listSum(a)** that returns the sum of the items in a. You may assume a is a list containing only ints.

12. Write the recursive function **flatten(a)**, which takes a list which may contain lists (which themselves may contain lists, and so on), and returns a single list (which does not contain any other lists) which contains each of the non-lists, in order, from the original list.

# Big-Oh Questions

## Basics/General

```
def f0(n): #backtobasics
    return str(n) * n

def f1(n): #basicfor
    total = 0
    for i in xrange(0, n, 3):
        for j in xrange(0, i, 2):
            total += i
    return total

def f2(n): #basicwhile
    total = 0
    while (n > 0):
        i = n / 2
        while (i < n):
            i += 1
            total += i
        n -= 1
    return total

def f3(n): #carefulwhenloopcounting
    total = 0
    for i in xrange(0, n, n/50):
        for j in xrange(0, i, 50):
            total += j
    return total

def f4(n): #sorting
    a = range(n - 1, -1, -1)
    return sorted(a)

def f5(n): #moresorting
    a = range(n**2 - 1, -1, -1)
    return sum(sorted(a)[1:n:2])

def f6(n): #sets
    s = set()
    for c in str(n):
        s.add(c)
    return len(s)
```

## Loops

```
def f1(n): #weirdinnerloop
    total = 0
    for i in xrange(n):
        for j in xrange(i**2):
            total += 1
    return total

def f2(n): #weirdinnerloop
    total = 0
    for i in xrange(0, n, 3):
        x = i
        while (x > 0):
            total += x
            x /= 2
    return sum(total)

def f3(n): #bitopsarecool
    x = 1
    total = 0
    while (x < n):
        x <<= 1
        y = 1
        total += y
        while (y < n):
            y <<= 1
            total += y
    return total

def f4(n): #whileloopsarecool
    a = range(n)
    i = 0
    while (i < len(a)):
        if (i % 2 == 0): a.append(i)
    return len(a)

def f5(n): #weirdinnerloop #outofscope
    total = 0
    for i in xrange(0, n, 3):
        x = i
        while (x < n):
            total += x
            x *= 2
    return total
```

```
def f6(n): #lol #outofscope
    x = 1
    total =
    while (x < n):
        x *= 10
        y = 1
        total += str(y)
        while (y < x):
            y *= 10
            total += str(y)
    return len(total)
```

### Recursion

```
def f1(n): #standard
    return 1 if n < 2 else f1(n-1)

def f2(n): #standard
    return 1 if n < 2 else f2(n / 2)

def f3(n): #seemsfarmiliar
    return 1 if (n < 2) else f3(n-1) + f3(1) + f3(n-1)

def f4(n): #seemsfarmiliar
    if (n <= 0): return 1
    if (n == 1): return 2
    return f4(n - 1) * f4(n - 2)

def f5(nums): #weirdshuffle #prettyhard
    # nums is a list of length n
    if (len(nums) < 2): return nums[:]
    left = nums
    right = []
    for i in xrange(len(nums) - 1, -1, -1):
        if (i % 2 == 0): right.append(left.pop(i))
    return f5(left) + f5(right)

def f6(n): #O.O #outofscope
    return 1 if (n < 1) else sum([f6(i) for i in xrange(n)])
```

### Other

```
def f1(n): #montecarlo
    numTrials = 1000000
```

```
numSuccesses = 0
for t in xrange(numTrials):
    a = range(n)
    random.shuffle(a)  # runs in O(n)
    if (isSorted(a)): # runs in O(n)
        numSuccesses += 1
return 1.0 * numSuccesses / numTrials
```