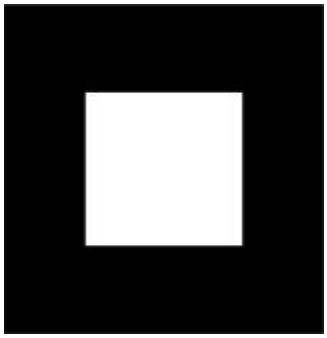
Practice Final

Short Answers

1. When is memoization useful?

2. Draw a level 2 Serpinski carpet, noting that a level 1 Serpinski carpet looks as depicted in the picture below:



3. State and prove the average case big O of quicksort

4. What is one advantage of using lists over sets?

5. Name a problem we can solve using Monte Carlo methods.

6. If we start with the list [9, 5, 4, 8, 0, 3, 5, 1], what is the resulting list after two rounds of mergesort?

7. Here is a buggy implementation of fib. Name a case when this will recurse forever, and suggest a way to fix it.

```
def fib(n):
    if n == 1 or n == 0:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

8. 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.

9. Find the Big-Oh of the following function:

```
def foo(n):
    x = y = 0   # hint: note that this is outside both loops!
    while (x < n):
        while (y < n):
            print("y", end="")
            y += 3
        print("x", end="")
        x += 4
```

Code Tracing

Indicate what each of the following functions will print:

Statement(s)	**Prints**

`def ct1(s, x):`
`  print(s, x)`
`  if x == 1:`
`  return 1`
`  elif (x % 5 == 0):`
`  return ct1(s + str(x), x // 5)`
`  else:`
`  return ct1(str(x) + s, x + 1)`
 	
`print(ct1("CA", 19))`	

`def ct2(s):`
`  count = 1`
`  for i in range(len(s)):`
`  for j in xrange(i + 1, len(s), 2):`
`  if s[i].islower() and s[j].isupper() and s[i] == s[j].lower():`
`  s = s.replace(s[j], chr(ord(s[j]) + count))`
`  count += 1`
`  return s`
`print ct2('aAbAA')`	

`def ct4(i, t=""):`
`  print ((i,t), end=" ") # don't miss this!`
`  if (i == 0): return t`
`  else: return chr(ord('A')+i) + ct4(i//2, t+str(i))`
`print (ct4(3))`	

```
import copy

def ct3(a):
      b,c,d = a, copy.copy(a),
      copy.deepcopy(a)
      b[0][0] += 1
      c[0][1] += 2
      d[1][0] += 3
      a[1] = [c[0][0]] +
      [c[0][1]]
      d[0][0] += 5
      a = [[1, 1], [0, 0]]
      print b
      print c
      print d
a = [[1,2],[3,4]]
ct3(a)
print a
```

Reasoning over code

For each function, find values of the parameters so that the function will return True.

```
def roc1(x, y = 0):
      if x <= 0:
            return y == 93
      else:
            if (x % 2 == 0):
                  return roc1(x-1, 2*y)
            else:
                  return roc1(x-1, 2*y + x)
```

x = _____

```
def roc2(L):
      if L[0] <= 0: return L == [0, 3, 1, 4, 2]
      if L[0] % 2 == 0:
            return roc2(L[1:] + [L[0]])
      else:
            return roc2(L[::2] + L[1::2])
```

L = _____

def roc3(a):
  b = [ ]
  for i in range(1, len(a)):
  if (a[i] == a[i-1]-1):
  b.extend([i, a[i]])
  return (b == [1, 6, 3, 5])
	a = _____

1. Free Response: transpose

While we probably would not do this, we could represent a 2d list of numbers as a triple: (rows, cols, values), where we first include the dimensions and then we include a 1d list of the values as read from top-to-bottom and then left-to-right. For example, consider this 2d list:

```
5  8  1
0  7  4
```

This has 2 rows and 3 columns, so we would represent this as: (2, 3, [5, 8, 1, 0, 7, 4]). Now, the transpose of a 2d list is constructed by swapping rows and columns. For example, the transpose of the list above is this list:

```
5  0
8  7
1  4
```

This would be represented as (3, 2, [5, 0, 8, 7, 1, 4]). With this in mind, write the function transpose(L) that takes a triple L representing a 2d-list as just described, and returns another triple representing the transpose of that list.

Note that a "triple" is a tuple, not a list, with 3 elements. So transpose takes one parameter, but that parameter is a triple containing 3 values.

2. Free Response: bestConnection

You are given the following CSV as shown below. Each line in the CSV specified a city and all the flights from that city and the associated cost. So for example, reading from the CSV below, we can see that from Pittsburgh we can go to Chicago for $190, Boston for $80, and New York for $150.

```
CSV = """boston,new york,280,pittsburgh,120,chicago,325
new york,pittsburgh,220,chicago,340,boston,220
pittsburgh,chicago,190,boston,80,new york,150
chicago,boston,100,new york,120, pittsburgh,75"""
```

Now let's assume we want to fly from Boston to New York. We can see that from boston, there's a direct flight to new york for $280 but is there any cheaper way? We can also fly from Boston to Pittsburgh then to New York and that would cost us $120 + $150 = $270, which is cheaper than the direct flight Boston - New York.

With this in mind, write the function bestConnection(costCSV, fromCity, toCity) that takes a CSV as described above and returns the tuple (connectingCity, nonstopCost, costWithConnection). The layover city should be the one that saves the most money over flying non-stop between the two cities. Resolve ties however you choose. If there is no way to get from the fromCity to the toCity with a layover or flying through a layover city costs more than the direct flight then return None.

3. Free Response: cosineArea()

Write the function cosineArea() which takes no parameters. cosineArea should use Monte Carlo methods to determine the following: if a point (x,y) is randomly chosen (with the requirement that 0 <= x <= 1 and 0 <= y <= 1), what the probability that y < cos(x)?

4. Free Response: longestAnagramList

Given a list of strings, write the function longestAnagramList(L) that returns the sublist with the maximum number of elements such that each of the elements in this sublist is an anagram of the previous element with a missing letter.

For example,

longestAnagramList([“abcd”, “dca”, “ca”, “bac”, “ba”, “b”]) returns [“abcd”, “bac”, “ba”, “b”]

Notice that “bac” is an anagram of “abcd” without “d” (i.e “abcd” with a missing letter), “ba” is an anagram of “bac” with a missing letter and “b” is an anagram of “ba” with a missing letter.

We could have also formed the following sublist [“abcd”, “bac”, “ca”], but clearly the previous sublist is longer.

The returned sublist can only contain each element once.

Here’s a few more test cases:

```
assert(longestAnagramList([] == [])
assert(longestAnagramList([“abcd”, “bcl”] == [“abcd”])
assert(longestAnagramList([“abcd”, “bcl”] == [“abcd”])
assert(longestAnagramList([“xyzw”, “mzw”, “zw”, “z”] == [“mzw”, “zw”, “z”])
assert(longestAnagramList([“xyzw”, “mzl”, “zw”, “z”] == [“zw”, “z”])
```

You can resolve ties whoever you wish!

5. Free Response: xmasDecor

Write the function xmasDecor which keeps track of the number of "joy-givers" in the returned results of f(*args) where a "joy-giver" is defined as an integer that is divisible by the sum of its digits.

For example:
18 is a joy-giver, because 1 + 8 = 9 and 18 is divisible by 9. 19 is not a joy-giver, because 19 is not divisible by 10.

After writing the xmasDecor and all other necessary python functions, the following test cases should pass:

```
def testXmasDecor():
    print("Testing xmasDecor...", end = "")

    @xmasDecor
    def f(x,y): return [18]
    assert(getJoyCount(f) == 0)
    assert(f(2,3) == [18])
    assert(getJoyCount(f) == 1)
    assert(f(3,2) == [18])
    assert(getJoyCount(f) == 2)

    @xmasDecor
    def h(a, b, c, d ,e): return [19]
    assert(getJoyCount(h) == 0)
    assert(h(1,2,3,4,5) == [19])
    assert(getJoyCount(h) == 0)
    print("Passed!")

testXmasDecor()
```

6. Free Response: HolidayPlaylist

Implement the classes and their corresponding methods that would make the following code work properly. For full credit, you will have to use OOP properly, including, for example, using super() where appropriate, and only overriding those methods that need to be overridden.

```
song0 = HolidaySong("Silent Night", "Michael Buble")
song1 = HolidaySong("Silent Night", "Michael Buble")
assert(song1.title == "Silent Night" and song1.artist == "Michael
Buble")
assert(str(song1) == "Song Title: Silent Night, Artist: Michael
Buble")
song2 = HolidaySong("Do They Know It's Christmas", "Band-Aid")
assert (song1 == song0) # same title and artist
assert (song1 != song2)

song3 = SleighRide("Ella Fitzgerald")
assert(isinstance(song3, HolidaySong))
assert(str(song3) == "Song Title: Sleigh Ride, Artist: Ella
Fitzgerald")
song4 = SleighRide("Pentatonix")
assert(str(song4) == "Song Title: Sleigh Ride, Artist: Pentatonix")
assert(song3 != song4)     # still not equal!
assert(song3 != "White Christmas")    # does not crash!

p1 = HolidayPlaylist("A Lot Like Christmas")
assert(p1.playNextSong() == "No songs yet.")
assert(p1.getCurrentSong() == None)

p1.addSong(song3)

assert(str(p1) == (
    """Playlist: A Lot Like Christmas
Songs: {Song Title: Sleigh Ride, Artist: Ella Fitzgerald}
Current Song: None"""))

assert(p1.playNextSong() == "Playing.")
assert(p1.getCurrentSong() == song3)
assert(len(p1.getSongs()) == 1)
```

```
p1.addSong(song4)
p1.addSong(song1)
p1.addSong(song2)
p1.addSong(song0)

assert(song4 in p1.getSongs())
assert(song1 in p1.getSongs())
assert(song2 in p1.getSongs())
assert(len(p1.getSongs()) == 4)

assert(p1.playNextSong() == "Playing.")
# When selecting next song, choose one that does not have the same
title!
assert(p1.getCurrentSong() != song4)
p1.playNextSong()
p1.playNextSong()
assert(p1.playNextSong() == "Playlist done.")

p2 = HolidayPlaylist("Chesnuts Roasting")
p2.addSong(song3)
assert(p2.playNextSong() == "Playing.")
assert(p2.getCurrentSong() == song3)
p2.addSong(song4)
assert(p2.playNextSong() == "Playing.")
# Still play next song even if the same title, since there's nothing
else!
assert(p2.getCurrentSong() == song4)
assert(p2.playNextSong() == "Playlist done.")
```

7. Free Response: bouncingSquares

Assuming the run() function from this semester's events-example0.py is already written, write the remaining functions required so that when run(400,400) is called, the following animation runs:

When the mouse is clicked for the first time 4 squares are created, all centered at the clicked position, with side lengths randomly chosen from 10 to 50, each heading randomly in one of the following 4 directions: North-East, North-West, South-East, and South-West. The direction of each of the squares is unique. The squares bounce back and forth each in its specified direction. The squares that are initialized moving in the NW or NE should flash every half a second, i.e their colors should alternate between any two colors of your choosing, while the side of the squares moving in the SW and SE should increase by 1 every half a second.
If the user clicks in any of the squares, the square stops moving. If clicked again, the square starts moving again.
The game lasts for 10 seconds: a timer should be displayed at the right top corner and should be counting down from 10. After 10 seconds, all the squares should stop moving and the text "Game Over" should be displayed in the center of the screen.
If the user press "r", the game start over (i.e all the squares disappear and the user can begin a new game).

8. Free Response: Lines of Code

Write the function linesOfCode that takes a path to a folder and returns the number of lines of code contained in all of the Python files (files that end in .py) in that folder (and its subfolders). Since we only want to count actual lines of code, lines that just contain whitespace and lines that are just comments (ones that begin with a #) should be ignored. You must use recursion in a meaningful way for this problem. You may assume that readFile(path) is written for you and it returns a string with the content of the given file.