Computer Science 15-112, Fall 2011
Class Notes: Recursion

Recursion

1. **Popular Recursion**
    1. **"Recursion":** See "Recursion".
    2. **Google search:** Recursion
    3. **Recursion comic:** http://xkcd.com/244/
    4. **Droste Effect:** See the Wikipedia page and this Google image search
    5. **Fractals:** See the Wikipedia page and this Google image search and this infinitely-zooming video
    6. **The Chicken and Egg Problem** (mutual recursion)
    7. **Sourdough Recipe:** First, start with some sourdough, then...
    8. **Books:** Godel, Escher, Bach; Metamagical Themas;

2. **General Recursive Form**

```
def recursiveFunction():
    if (this is the base case):
        # no recursion allowed here!
        do something non-recursive
    else:
        # this is the recursive case!
        do something recursive
```

See http://en.wikipedia.org/wiki/Recursion_(computer_science)

3. **Basic Examples**

1. **rangeSum**

```
def rangeSum(lo, hi):
    if (lo > hi):
        return 0
    else:
        return lo + rangeSum(lo+1, hi)

print rangeSum(10,15) # 75
```

2. **listSum**

```
def listSum(list):
    if (len(list) == 0):
        return 0
    else:
        return list[0] + listSum(list[1:])

print listSum([2,3,5,7,11]) # 28
```

3. **power**

```
def power(base, expt):
    # assume expt is non-negative integer
    if (expt == 0):
        return 1
    else:
        return base * power(base, expt-1)

print power(2,5) # 32
```

4. **interleave**

```
def interleave(list1, list2):
    # assume list1 and list2 are same-length lists
    if (len(list1) == 0):
        return []
    else:
        return [list1[0] , list2[0]] + interleave(list1[1:], list2[1:])

print interleave([1,2,3],[4,5,6])
```

4. **Examples with Multiple Base or Recursive Cases**
    1. **power with negative exponents**

```
def power(base, expt):
    # This version allows for negative exponents
    # It still assumes that expt is an integer, however.
    if (expt == 0):
        return 1
    elif (expt < 0):
        return 1.0/power(base,abs(expt))
    else:
        return base * power(base, expt-1)

print power(2,5) # 32
print power(2,-5) # 1/32 = 0.03125
```

    2. **interleave with different-length lists**

```
def interleave(list1, list2):
    # This version allows for different-length lists
    if (len(list1) == 0):
        return list2
    elif (len(list2) == 0):
        return list1
    else:
        return [list1[0] , list2[0]] + interleave(list1[1:], list2[1:])

print interleave([1,2,3],[4,5,6,7,8])
```

5. **Examples with Multiple Recursive Calls**

1. **fibonacci**

A) First attempt

```
# Note: as written, this function is very inefficient!
# (We need to use "memoization" to speed it up! See below for details!)
def fib(n):
    if (n < 2):
        # Base case:  fib(0) and fib(1) are both 1
        return 1
    else:
        # Recursive case: fib(n) = fib(n-1) + fib(n-2)
        return fib(n-1) + fib(n-2)

for n in range(15): print fib(n),
```

B) Once again, printing call stack using recursion depth:

```
def fib(n, depth=0):
    print "   "*depth, "fib(", n, " )"
    if (n < 2):
        # Base case:  fib(0) and fib(1) are both 1
        return 1
    else:
        return fib(n-1, depth+1) + fib(n-2, depth+1)

fib(4)
```

C) Even better (printing result, too):

```
def fib(n, depth=0):
    print "   "*depth, "fib(", n, " )"
    if (n < 2):
        result = 1
        # Base case:  fib(0) and fib(1) are both 1
        print "   "*depth, "-->", result
        return result
    else:
        result = fib(n-1, depth+1) + fib(n-2, depth+1)
        print "   "*depth, "-->", result
        return result

fib(4)
```

D) Finally, not duplicating code:

```
def fib(n, depth=0):
    print "   "*depth, "fib(", n, " )"
    if (n < 2):
        result = 1
    else:
        result = fib(n-1, depth+1) + fib(n-2, depth+1)
    print "   "*depth, "-->", result
    return result

fib(4)
```

2. **towersOfHanoi**

A) First attempt (without Python)

```
# This is the plan we generated in class to solve Towers of Hanoi:
magically move(n-1, frm, via, to)
          move(  1, frm, to,  via)
magically move(n-1, via, to,  frm)
```

B) Turn into Python (The "magic" is recursion!)

```
def move(n, frm, to, via):
    move(n-1, frm, via, to)
    move(  1, frm, to,  via)
```

```
    move(n-1, via, to,  frm)

move(2, 0, 1, 2) # Does not work -- infinite recursion
```

C) Once again, with a base case

```
def move(n, frm, to, via):
    if (n == 1):
        print (frm, to),
    else:
        move(n-1, frm, via, to)
        move(  1, frm, to,  via)
        move(n-1, via, to,  frm)

move(2, 0, 1, 2)
```

D) Once more, with a nice wrapper:

```
def move(n, frm, to, via):
    if (n == 1):
        print (frm, to),
    else:
        move(n-1, frm, via, to)
        move(  1, frm, to,  via)
        move(n-1, via, to,  frm)

def hanoi(n):
    print "Solving Towers of Hanoi with n =",n
    move(n, 0, 1, 2)
    print

hanoi(4)
```

E) And again, printing call stack and recursion depth:

```
def move(n, frm, to, via, depth=0):
    print (" " * 3 * depth), "move", n, "from", frm, "to", to, "via", via
    if (n == 1):
        print (" " * 3 * depth), (frm, to)
    else:
        move(n-1, frm, via, to, depth+1)
        move(1, frm, to, via, depth+1)
        move(n-1, via, to, frm, depth+1)

def hanoi(n):
    print "Solving Towers of Hanoi with n =",n
    move(n, 0, 1, 2)
    print

hanoi(4)
```

F) Iterative Towers of Hanoi (just to see it's possible)

For a good explanation of iterative solutions to Towers of Hanoi, look at [this part of the Towers of Hanoi Wikipedia page](#).

```
def iterativeHanoi(n):
    def f(k): return (k%3) if (n%2==0) else (-k%3)
    return [(f(move&(move-1)), f((move|(move-1))+1)) for move in xrange(1,1<<n)]

def recursiveHanoi(n, frm=0, to=1, via=2):
    if (n == 1):
        return [(frm, to)]
    else:
        return (recursiveHanoi(n-1, frm, via, to) +
                recursiveHanoi(  1, frm, to,  via) +
                recursiveHanoi(n-1, via, to,  frm))

def compareIterativeAndRecursiveHanoi():
    for n in xrange(1,20):
        assert(iterativeHanoi(n) == recursiveHanoi(n))
    print "iterative and recursive solutions match exactly in all tests!"
```

```
compareIterativeAndRecursiveHanoi()
```

6. **Examples Comparing Iteration and Recursion**

Function	Iterative Solution	Recursive Solution	Recursive Solution with Stack Trace

factorial	def factorial(n):
	  factorial = 1
	  for i in range(2,n+1):
	  factorial *= i
	  return factorial
	
	print factorial(5)
		def factorial(n):
		  if (n < 2):
		  return 1
		  else:
		  return n*factorial(n-1)
		
		print factorial(5)
			def factorial(n, depth=0):
			  print " "*depth, "factorial(",n,"):"
			  if (n < 2):
			  result = 1
			  else:
			  result = n*factorial(n-1,depth+1)
			  print " "*depth, "-->", result
			  return result
			
			print factorial(5)

reverse	def reverse(s):
	  reverse = ""
	  for ch in s:
	  reverse = ch + reverse
	  return reverse
	
	print reverse("abcd")
		def reverse(s):
		  if (s == ""):
		  return ""
		  else:
		  return reverse(s[1:]) + s[0]
		
		print reverse("abcd")
			def reverse(s, depth=0):
			  print " "*depth, "reverse(",s,"):"
			  if (s == ""):
			  result = ""
			  else:
			  result = reverse(s[1:], depth+1) + s[0]
			  print " "*depth, "-->", result
			  return result
			
			print reverse("abcd")

gcd	def gcd(x,y):
	  while (y > 0):
	  oldX = x
	  x = y
	  y = oldX % y
	  return x
	
	print gcd(500, 420) # 20
		def gcd(x,y):
		  if (y == 0):
		  return x
		  else:
		  return gcd(y,x%y)
		
		print gcd(500, 420) # 20
			def gcd(x,y,depth=0):
			  print " "*depth, "gcd(",x, ",", y, "):"
			  if (y == 0):
			  result = x
			  else:
			  result = gcd(y,x%y,depth+1)
			  print " "*depth, "-->", result
			  return result
			
			print gcd(500, 420) # 20

7. **Iteration vs Recursion Summary**

	Recursion	Iteration
Elegance	**++**	**--**
Performance	**--**	**++**
Debugability	**--**	**++**

Note: These are general guidelines. For example, it is possible to use recursion with high performance, and it is certainly possible to use (or abuse) iteration with very low performance.

Conclusion (for now): Use iteration when practicable. Use recursion when required (for "naturally recursive problems").

8. **Expanding the Stack Size and Recursion Limit (callWithLargeStack)**
   1. **The problem**

```
def rangeSum(lo, hi):
    if (lo > hi):
        return 0
    else:
        return lo + rangeSum(lo+1, hi)
```

```
print rangeSum(1,1234)  # RuntimeError: maximum recursion depth exceeded
```

2. **The solution**

```
def rangeSum(lo, hi):
    if (lo > hi):
        return 0
    else:
        return lo + rangeSum(lo+1, hi)

def callWithLargeStack(f,*args):
    import sys
    import threading
    threading.stack_size(2**27)  # 64MB stack
    sys.setrecursionlimit(2**27) # will hit 64MB stack limit first
    # need new thread to get the redefined stack size
    def wrappedFn(resultWrapper): resultWrapper[0] = f(*args)
    resultWrapper = [None]
    #thread = threading.Thread(target=f, args=args)
    thread = threading.Thread(target=wrappedFn, args=[resultWrapper])
    thread.start()
    thread.join()
    return resultWrapper[0]
```

```
print callWithLargeStack(rangeSum,1,123456) # prints 7620753696
```

9. **Improving Efficiency with Memoization**
    1. **The problem**

```
def fib(n):
    if (n < 2):
        return 1
    else:
        return fib(n-1) + fib(n-2)

import time
def testFib(maxN=40):
    for n in xrange(maxN+1):
        start = time.time()
        fibOfN = fib(n)
        ms = 1000*(time.time() - start)
        print "fib(%2d) = %8d, time =%5dms" % (n, fibOfN, ms)

testFib() # gets really slow!
```

    2. **The solution**

```
def memoized(f):
    import functools
    cachedResults = dict()
    @functools.wraps(f)
    def wrapper(*args):
        if args not in cachedResults:
            cachedResults[args] = f(*args)
        return cachedResults[args]
    return wrapper

@memoized
def fib(n):
    if (n < 2):
        return 1
    else:
        return fib(n-1) + fib(n-2)

import time
def testFib(maxN=40):
    for n in xrange(maxN+1):
        start = time.time()
        fibOfN = fib(n)
        ms = 1000*(time.time() - start)
        print "fib(%2d) = %8d, time =%5dms" % (n, fibOfN, ms)
```

```
testFib() # ahhh, much better!
```

10. **Some Interesting Recursion Examples**
    1. **powerset (all subsets)**

```
def powerset(a):
    # returns a list of all subsets of the list a
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = [ ]
        for subset in powerset(a[1:]):
            allSubsets += [subset]
            allSubsets += [[a[0]] + subset]
        return allSubsets

print powerset([1,2,3])
```

    2. **permutations**

```
def permutations(a):
    # returns a list of all permutations of the list a
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = [ ]
        for subPermutation in permutations(a[1:]):
            for i in xrange(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms

print permutations([1,2,3])
```

    3. **printFiles (with os module)**

```
import os
def printFiles(path):
    if (os.path.isdir(path) == False):
        # base case:  not a folder, but a file, so print its path
        print path
    else:
        # recursive case: it's a folder
        for filename in os.listdir(path):
            printFiles(path + "/" + filename)

# To test this, download and expand this zip file in the same directory
# as the Python file you are running:  sampleFiles.zip
# Note: if you see .DS_Store files in the sampleFiles folders, or in the
# output of your function (as often happens with Macs, in particular),
# download removeDsStore.py, place it in the same directory, and run it,
# and you should see your .DS_Store files removed.

printFiles("sampleFiles")

Produces this output:

sampleFiles/emergency.txt
sampleFiles/folderA/fishing.txt
sampleFiles/folderA/folderC/folderD/misspelled.txt
sampleFiles/folderA/folderC/folderD/penny.txt
sampleFiles/folderA/folderC/folderE/tree.txt
sampleFiles/folderA/folderC/giftwrap.txt
sampleFiles/folderA/widths.txt
sampleFiles/folderB/folderH/driving.txt
sampleFiles/folderB/restArea.txt
sampleFiles/mirror.txt
```

    4. **listFiles (with os module)**

```
import os
def listFiles(path):
    if (os.path.isdir(path) == False):
        # base case:  not a folder, but a file, so return singleton list with its path
```

```
            return [path]
        else:
            # recursive case: it's a folder, return list of all paths
            files = [ ]
            for filename in os.listdir(path):
                files += listFiles(path + "/" + filename)
            return files

# To test this, download and expand this zip file in the same directory
# as the Python file you are running:  sampleFiles.zip

print listFiles("sampleFiles")
```

Produces this output:

```
['sampleFiles/emergency.txt', 'sampleFiles/folderA/fishing.txt', 'sampleFiles/fo
lderA/folderC/folderD/misspelled.txt', 'sampleFiles/folderA/folderC/folderD/penn
y.txt', 'sampleFiles/folderA/folderC/folderE/tree.txt', 'sampleFiles/folderA/fol
derC/giftwrap.txt', 'sampleFiles/folderA/widths.txt', 'sampleFiles/folderB/folde
rH/driving.txt', 'sampleFiles/folderB/restArea.txt', 'sampleFiles/mirror.txt']
```

5. **floodFill (with PhotoImage pixels)**

A) Basic idea

```
def floodFill(x, y, color):
    if ((not inImage(x,y)) or (getColor(img, x, y) == color)):
        # do nothing in the base case!
        pass
    else:
        img.put(color, to=(x, y))
        floodFill(x-1, y, color)
        floodFill(x+1, y, color)
        floodFill(x, y-1, color)
        floodFill(x, y+1, color)
```

B) Full Program (pixel-based version, without animation)

```
# FloodFill using Tkinter
# pixel-based without animation

from Tkinter import *
root = Tk()
canvas = Canvas(root, width=250, height=250)
canvas.pack()

canvas.create_text(125,20,text="FloodFill Demo",font="Helvetica 16 bold")
canvas.create_text(125,40,text="left click = draw",font="Helvetica 12 italic")
canvas.create_text(125,60,text="shift-left or right click = fill",font="Helvetica 12 italic")

imgLeft = 75
imgTop = 75
imgWidth = 100
imgHeight = 100

img = PhotoImage(width=imgWidth, height=imgHeight)
canvas.create_image(imgLeft, imgTop, image=img, anchor=NW)

color1 = "#0000ff"
color2 = "#00ff00"
canvas.fillColor = color1
for x in range(imgWidth):
    for y in range(imgHeight):
        img.put(color1, to=(x,y))

def inImage(x, y):
    return ((x >= 0) and (x < imgWidth) and \
            (y >= 0) and (y < imgHeight))

def drawDot(x, y, color):
    r = 5
    for dx in range(-r,+r):
        for dy in range(-r,+r):
            if ((dx**2 + dy**2 <= r**2) and inImage(x+dx,y+dy)):
```

```
                img.put(color, to=(x+dx,y+dy))

def getColor(img, x, y):
    hexColor = "#%02x%02x%02x" % getRGB(img, x, y)
    return hexColor

def getRGB(img, x, y):
    value = img.get(x, y)
    return tuple(map(int, value.split(" ")))

def mousePressed(event, doFlood):
    x = event.x-imgLeft
    y = event.y-imgTop
    if (inImage(x,y)):
        color = getColor(img, x, y)
        if (color == color1):
            canvas.fillColor = color2
        else:
            canvas.fillColor = color1
        if (doFlood):
           floodFillWithLargeStack(x, y)
        else:
           drawDot(x, y, canvas.fillColor)
```

```
def leftMousePressed(event):
    shiftDown = ((event.state & 0x0001) == 1)
    mousePressed(event, shiftDown)

def leftMouseMoved(event):
    x = event.x-imgLeft
    y = event.y-imgTop
    if (inImage(x, y)):
        drawDot(x, y, canvas.fillColor)

def floodFill(x, y, color):
    if ((not inImage(x,y)) or (getColor(img, x, y) == color)):
        return
    img.put(color, to=(x, y))
    floodFill(x-1, y, color)
    floodFill(x+1, y, color)
    floodFill(x, y-1, color)
    floodFill(x, y+1, color)

def callWithLargeStack(f,*args):
    import sys
    import threading
    threading.stack_size(2**27)  # 64MB stack
    sys.setrecursionlimit(2**27) # will hit 64MB stack limit first
    # need new thread to get the redefined stack size
```

```
    def wrappedFn(resultWrapper): resultWrapper[0] = f(*args)
    resultWrapper = [None]
    #thread = threading.Thread(target=f, args=args)
    thread = threading.Thread(target=wrappedFn, args=[resultWrapper])
    thread.start()
    thread.join()
    return resultWrapper[0]

def floodFillWithLargeStack(x,y):
    callWithLargeStack(floodFill, x, y, canvas.fillColor)

def rightMousePressed(event):
    mousePressed(event, True)

canvas.bind("<Button-1>", leftMousePressed)
canvas.bind("<B1-Motion>", leftMouseMoved)
canvas.bind("<Button-3>", rightMousePressed)
root.mainloop()
```

C) Full Program (grid-based version, with animation)

```
# FloodFill using Tkinter
# grid-based (not pixel-based), with animation
# and numeric display of depth of recursion
```

```
# also, this version is based on our barebones animation code

from Tkinter import *
import time # for time.sleep()

def mousePressed(event, doFlood):
    clearDepths()
    (row,col) = getCell(event.x, event.y)
    if ((row >= 0) and (row < canvas.data.rows) and
        (col >= 0) and (col < canvas.data.cols)):
        color = canvas.data.board[row][col]
        if (color == "cyan"):
            canvas.data.fillColor = "green"
        else:
            canvas.data.fillColor = "cyan"
        if (doFlood):
            floodFillWithLargeStack(row, col)
        else:
            canvas.data.board[row][col] = canvas.data.fillColor

def leftMousePressed(event):
    shiftDown = ((event.state & 0x0001) == 1)
    mousePressed(event, shiftDown)
    redrawAll()

def leftMouseMoved(event):
    (row,col) = getCell(event.x, event.y)
    if ((row >= 0) and (row < canvas.data.rows) and
        (col >= 0) and (col < canvas.data.cols)):
        canvas.data.board[row][col] = canvas.data.fillColor
    redrawAll()

def rightMousePressed(event):
    mousePressed(event, True)
    redrawAll()

def getCell(x, y):
    # return row,col containing the point x,y
    row = (y - 100)/canvas.data.cellSize
    col = x / canvas.data.cellSize
    return (row, col)

def getCellBounds(row, col):
    # return (left, top, right, bottom) of this cell
    left = col * canvas.data.cellSize
    right = (col+1) * canvas.data.cellSize
    top = 100 + row * canvas.data.cellSize
    bottom = 100 + (row+1)*canvas.data.cellSize
    return (left, top, right, bottom)

def floodFill(row, col, color, depth=0):
    if ((row >= 0) and (row < canvas.data.rows) and
        (col >= 0) and (col < canvas.data.cols) and
        (canvas.data.board[row][col] != color)):
        canvas.data.board[row][col] = color
        canvas.data.depth[row][col] = depth
        redrawAll()
        canvas.update()
        time.sleep(0.05 if (depth < 25) else 0.005)
        floodFill(row-1, col, color, depth+1)
        floodFill(row+1, col, color, depth+1)
        floodFill(row, col-1, color, depth+1)
        floodFill(row, col+1, color, depth+1)

def callWithLargeStack(f,*args):
    import sys
    import threading
    threading.stack_size(2**27)  # 64MB stack
    sys.setrecursionlimit(2**27) # will hit 64MB stack limit first
    # need new thread to get the redefined stack size
    def wrappedFn(resultWrapper): resultWrapper[0] = f(*args)
    resultWrapper = [None]
    #thread = threading.Thread(target=f, args=args)
```

```
        thread = threading.Thread(target=wrappedFn, args=[resultWrapper])
        thread.start()
        thread.join()
        return resultWrapper[0]

    def floodFillWithLargeStack(row, col):
        callWithLargeStack(floodFill, row, col, canvas.data.fillColor)

    def redrawAll():
        canvas.delete(ALL)
        xmid = canvas.data.width/2
        font16b = "Helvetica 16 bold"
        font12i = "Helvetica 12 italic"
        canvas.create_text(xmid,20,text="FloodFill Demo",font=font16b)
        canvas.create_text(xmid,40,text="left click = draw",font=font12i)
        canvas.create_text(xmid,60,text="shift-left or right click = fill",font=font12i)
        canvas.create_text(xmid,80,text="Do not click during floodFill animation!",font=font12i)
        for row in xrange(canvas.data.rows):
            for col in xrange(canvas.data.cols):
                (x0,y0,x1,y1) = bounds = getCellBounds(row, col)
                canvas.create_rectangle(bounds, fill=canvas.data.board[row][col])
                if (canvas.data.depth[row][col] != -1):
                    canvas.create_text((x0+x1)/2,(y0+y1)/2,text=str(canvas.data.depth[row][col]))

    def clearDepths():
        canvas.data.depth =[([-1]*canvas.data.cols) for row in xrange(canvas.data.rows)]

    def init():
        canvas.data.board = [(["cyan"]*canvas.data.cols) for row in xrange(canvas.data.rows)]
        clearDepths()

    def run():
        # create the root and the canvas
        global canvas
        root = Tk()
        class Struct: pass
        data = Struct()
        data.rows = 20
        data.cols = 30
        data.cellSize = 25 # pixels
        data.width = data.cols * data.cellSize
        data.height = data.rows * data.cellSize + 100 # room for text at top
        canvas = Canvas(root, width=data.width, height=data.height)
        canvas.data = data
        canvas.pack()
        init()
        # set up events
        root.bind("<Button-1>", leftMousePressed)
        root.bind("<B1-Motion>",leftMouseMoved)
        root.bind("<Button-3>", rightMousePressed)
        # and launch the app
        redrawAll()
        root.mainloop()  # This call BLOCKS (so your program waits until you close the window!)

    run()
```

6. **kochSnowflake (with Turtle)**

A) Basic idea
(build up k4 snowflake non-recursively)

```
import turtle

def k1(length):
    turtle.forward(length)

def k2(length):
    turtle.forward(length/3.0)
    turtle.left(60)
    turtle.forward(length/3.0)
    turtle.right(120)
    turtle.forward(length/3.0)
```

```
    turtle.left(60)
    turtle.forward(length/3.0)

def k2(length):
    k1(length/3.0)
    turtle.left(60)
    k1(length/3.0)
    turtle.right(120)
    k1(length/3.0)
    turtle.left(60)
    k1(length/3.0)

def k3(length):
    k2(length/3.0)
    turtle.left(60)
    k2(length/3.0)
    turtle.right(120)
    k2(length/3.0)
    turtle.left(60)
    k2(length/3.0)

def k4(length):
    k3(length/3.0)
    turtle.left(60)
    k3(length/3.0)
    turtle.right(120)
    k3(length/3.0)
    turtle.left(60)
    k3(length/3.0)

turtle.delay(0)
turtle.speed(0)
turtle.penup()
turtle.goto(-300,0)
turtle.pendown()
turtle.pensize(2)

turtle.pencolor("black")
k1(150)
turtle.pencolor("red")
k2(150)
turtle.pencolor("green")
k3(150)
turtle.pencolor("blue")
k4(150)

turtle.done()
```

B) Recursive solution
(convert obvious recurring pattern from previous examples into a recursive solution)

```
import turtle

def kN(length, n):
    if (n == 1):
        turtle.forward(length)
    else:
        kN(length/3.0, n-1)
        turtle.left(60)
        kN(length/3.0, n-1)
        turtle.right(120)
        kN(length/3.0, n-1)
        turtle.left(60)
        kN(length/3.0, n-1)

def kochSnowflake(length, n):
    for step in range(3):
        kN(length, n)
        turtle.right(120)

turtle.delay(0)
turtle.speed(0)
turtle.penup()
```

```
turtle.goto(-300,100)
turtle.pendown()

turtle.pencolor("black")
kN(300, 4) # same as k4(300)

turtle.pencolor("blue")
kochSnowflake(300, 4)

turtle.penup()
turtle.goto(-250,50)
turtle.pendown()
turtle.pencolor("red")
kochSnowflake(200, 7)
turtle.done()
```

7. **sierpinskiTriangle (with Tkinter)**

```
from Tkinter import *

def drawSierpinskyTriangle(canvas, x, y, size, level):
    # (x,y) is the lower-left corner of the triangle
    # size is the length of a side
    x = float(x)
    y = float(y)
    if (level == 0):
        canvas.create_polygon(x, y,
                              x+size, y,
                              x+size/2, y-size*(3**0.5)/2,
                              fill="black")
    else:
        drawSierpinskyTriangle(canvas, x, y, size/2, level-1)
        drawSierpinskyTriangle(canvas, x+size/2, y, size/2, level-1)
        drawSierpinskyTriangle(canvas, x+size/4, y-size*(3**0.5)/4, size/2, level-1)

def keyPressed(event):
    if (event.keysym in ["Up", "Right"]):
        canvas.data.level += 1
    elif ((event.keysym in ["Down", "Left"]) and (canvas.data.level > 0)):
        canvas.data.level -= 1
    redrawAll()

def redrawAll():
    canvas.delete(ALL)
    drawSierpinskyTriangle(canvas, 25, 450, 450, canvas.data.level)
    canvas.create_text(250, 25,
                       text = "Level %d Sierpinsky Triangle" % (canvas.data.level),
                       font = "Arial 26 bold")
    canvas.create_text(250, 50,
                       text = "Use arrows to change level",
                       font = "Arial 10")

def init():
    canvas.data.level = 1
    redrawAll()

def run():
    # create the root and the canvas
    global canvas
    root = Tk()
    canvas = Canvas(root, width=500, height=500)
    canvas.pack()
    # Set up canvas data and call init
    class Struct: pass
    canvas.data = Struct()
    init()
    # set up events
    #root.bind("<Button-1>", mousePressed)
    root.bind("<Key>", keyPressed)
    #timerFired()
    # and launch the app
    root.mainloop()  # This call BLOCKS (so your program waits until you close the window!)
```

```
run()
```

11. **More Advanced Recursion Examples**
    Note: some of these algorithms also have reasonably straightforward iterative formulations, or comparable iterative alternatives.
    1. **Sorting (mergesort, quicksort)**
    2. **Backtracking (solving mazes, Sudoku-like puzzles, and, in general, "constraint satisfaction problems")**
    3. **Top-Down Dynamic Programming**
       (Problem-solving by divide-and-conquer while reusing solutions to overlapping subproblems. Many items in this list are really examples of Dynamic Programming.)
    4. **Top-Down (Recursive Descent) Parsing**
    5. **Evaluating (Lisp-Like) Expressions**
    6. **String Matching (Longest common subsequence, ...)**
    7. **Bioinformatics (sequence alignment, ...)**
    8. **Music (beat tracking, dynamic time warping, ...)**
    9. **AI for 2-player board games (Minimax [with Alpha-Beta Pruning])**
    10. **Image Manipulation (seam carving,...)**
    11. **Math (Fast Fourier Transform, Fast Matrix Multiplication (Strassen), Fast Scalar Multiplication (Karatsuba), ...)**
    12. **And many, many others...**
