Max List Function:

```
def main():
   list = [23,54,12,34,65]
#Should return 65
   print(maximum(list))
def maximum(list):
   if len(list) == 1:
      return list[0]
   else:
      return max(list[0],
maximum(list[1:]))
main()
```

Recursive Fractal Mountain:

```
import turtle
import random

def main():
   t = turtle.Turtle()
   x1 = -100
   y1 = 0
   x2 = 200
   y2 = 0
   iterations = 2
   m(x1, y1, x2, y2, t,
iterations)
   turtle.done()

def m(x1, y1, x2, y2, t,
it):
   if it == 0:
      t.up()
      t.goto(x1,y1)
      t.down()
      t.goto(x2, y2)
   else:
      midx = (x1+x2)//2
      midy = (y1+y2)//2
+ random.randint(0,50)
      m(x1, y1, midx,
midy, t, it-1)
      m(midx, midy, x2,
y2, t, it-1)

main()
```

C-Curve:

```
from turtle import Turtle, tracer, update
import turtle

def cCurve(t, x1, y1, x2, y2, level):

  def drawLine(x1, y1, x2, y2):
    """Draws a line segment between the endpoints."""
    t.up()
    t.goto(x1, y1)
    t.down()
    t.goto(x2, y2)

  if level == 0:
    drawLine(x1, y1, x2, y2)
  else:
    xm = (x1 + x2 + y1 - y2) // 2
    ym = (x2 + y1 + y2 - x1) // 2
    cCurve(t, x1, y1, xm, ym, level - 1)
    cCurve(t, xm, ym, x2, y2, level - 1)

def main():
  level = int(input("Enter the level (0 or greater): "))
  t = Turtle()
  if level > 8:
    tracer(False)
  t.pencolor("blue")
  t.speed(0)
  t.hideturtle()
  cCurve(t, 50, -50, 50, 50, level)
  if level > 8:
    update()
  turtle.done()


if __name__ == "__main__":
  main()
```

Change Program:

```
def
dpMakeChange(coinValueList,ch
ange,minCoins,coinsUsed):
  for cents in range(change+1):
    coinCount = cents
    newCoin = 1
    for j in [c for c in
coinValueList if c <= cents]:
        if minCoins[cents-j] + 1 <
coinCount:
          coinCount =
minCoins[cents-j]+1
          newCoin = j
    minCoins[cents] = coinCount
    coinsUsed[cents] = newCoin
  return minCoins[change]

def
printCoins(coinsUsed,change):
  coin = change
  while coin > 0:
    thisCoin = coinsUsed[coin]
    print(thisCoin)
    coin = coin - thisCoin

def main():
   amnt = 63
   clist = [1,2,5,8,10,25]
   coinsUsed = [0]*(amnt+1)
   coinCount = [0]*(amnt+1)

   print("Making change
for",amnt,"requires")

print(dpMakeChange(clist,amnt,
coinCount,coinsUsed),"coins")
   print("They are:")
   printCoins(coinsUsed,amnt)
   print("The used list is as
follows:")
   print(coinsUsed)

main()
```

17. (2 points) Give the Big-O performance of the following code fragment:

-1

```
for i in range(n):
    for j in range(n*n):
        for k in range(n):
            k = 2 + 2
```

n^3 $O(n^4)$

18. (4 points) Turn the following infix expressions to postfix expressions

A * (B + C) + (D – E)

-1

A DE- BC+ + * X

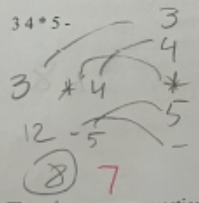
(A – B) * C – D / E

AB - C * DE/ -

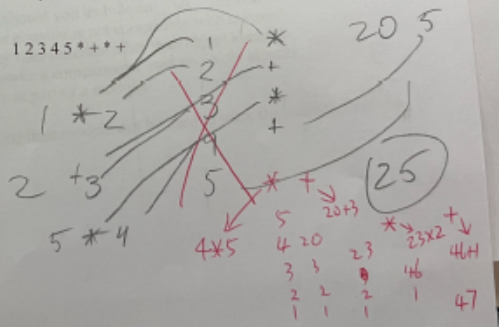
19. (4 points) Evaluate the following two postfix expressions. Show the stack as each operand and operator is processed.

-2

34*5-



12345*+*+



There is one more question on the next page.

```
def koch(t, length, n):
    if n < 1:
        t.forward(length)
    else:
        koch(t, length/3, n-1)
        t.left(60)
```

-1

koch(t, lenght/3, n-1)

t.right(~~60~~) 120

koch(t, length/3, n-1)

t.left(60)

koch(t, length/3, n-1)

4. (7 points) Write a recursive function to reverse a list.

```
def revList(list, rlist, n):
    if len(list) == len(rlist)
        return rlist
    else
        rlist.append(list[n-1])
        revList(list, rlist, n-1)

def main():
    n = 0
    list = [1, 2, 3, 4, 5]
    rlist = [ ]
    revList(list, rlist, n)
```

It might actually work.

Binary Search

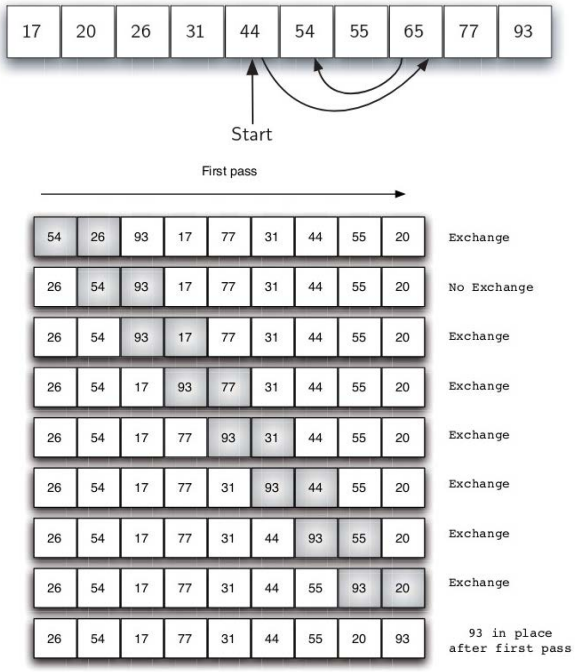


Figure 5.13: `bubbleSort`: The First Pass

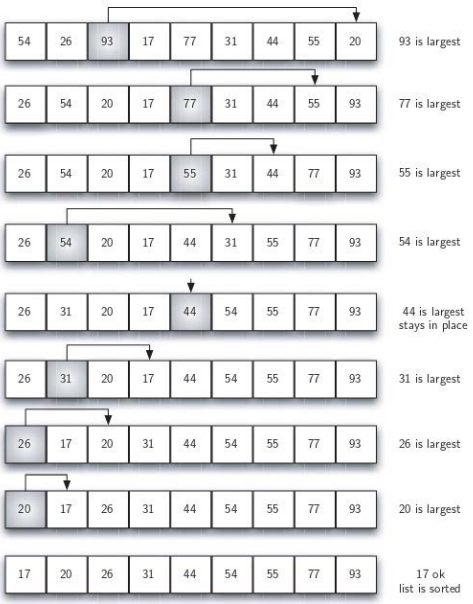


Figure 5.15: `selectionSort`

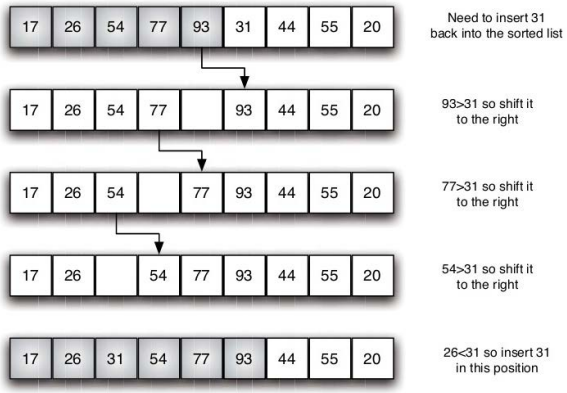


Figure 5.17: `insertionSort`: Fifth Pass of the Sort

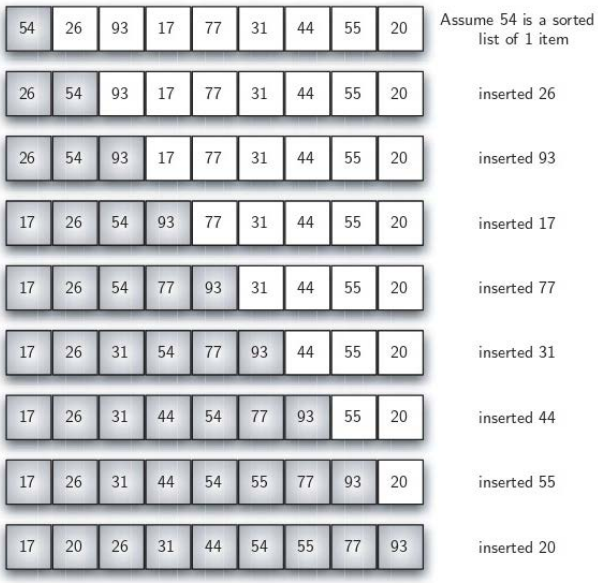


Figure 5.16: `insertionSort`

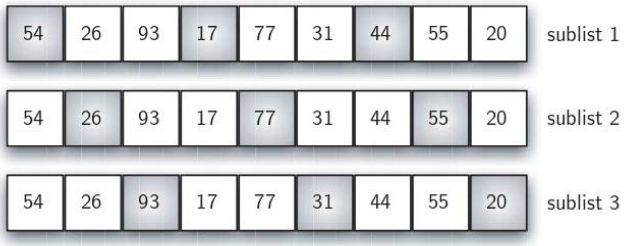


Figure 5.18: A Shell Sort with Increments of Three

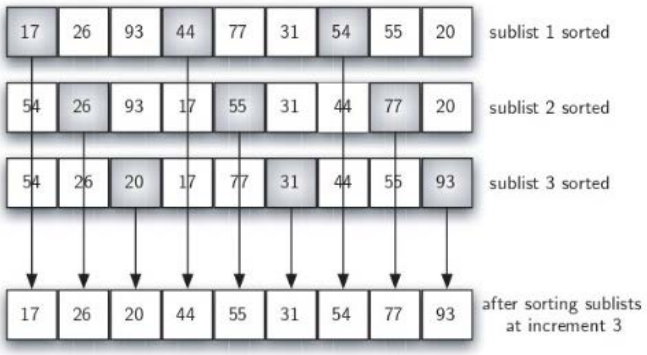


Figure 5.19: A Shell Sort after Sorting Each Sublist

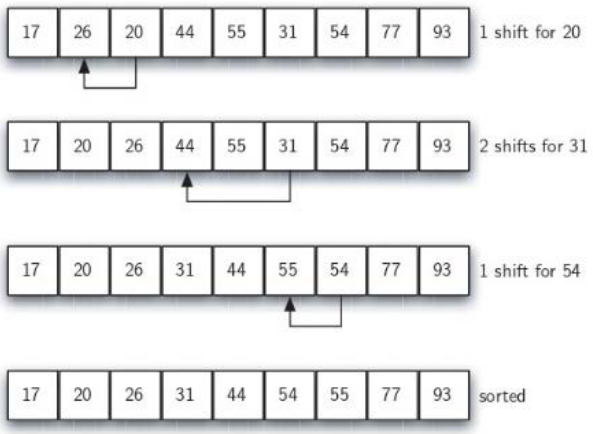
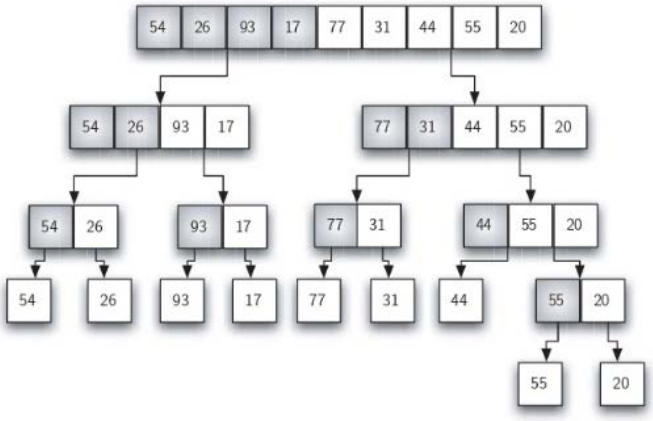
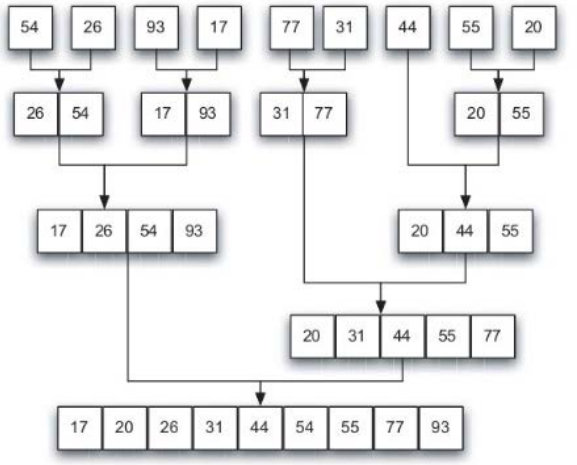


Figure 5.20: ShellSort: A Final Insertion Sort with Increment of 1



(a) Splitting the List in a Merge Sort



(b) Lists as They Are Merged Together

Queue:

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)
```

Stack:

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

Deque:

```
class Deque:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def addFront(self, item):
        self.items.append(item)

    def addRear(self, item):
        self.items.insert(0,item)

    def removeFront(self):
        return self.items.pop()

    def removeRear(self):
        return self.items.pop(0)

    def size(self):
        return len(self.items)
```