6.00 Introduction to Computer Science and Programming
Fall 2008

Here are some practice quiz questions (taken mostly from last year's final).

1) Is each of the following True or False

1.1. In Python, the method `sort` of class `list` has a side effect.

1.2. When run on the same inputs, an exponential algorithm will always take longer than a polynomial algorithm.

1.3. Newton's method is based upon successive approximation.

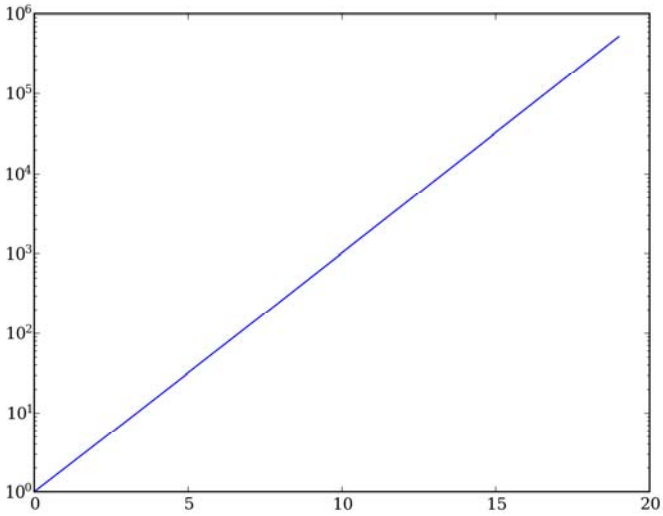
1.4. Constructing a black box test suite involves looking at the control flow of the code to be tested.

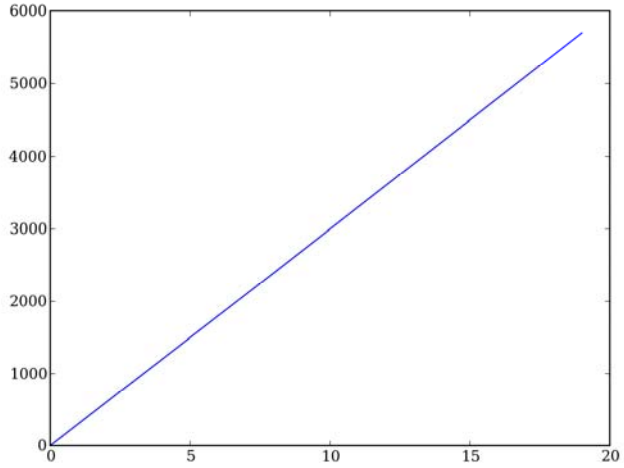
1.5. Python classes enforce data hiding.

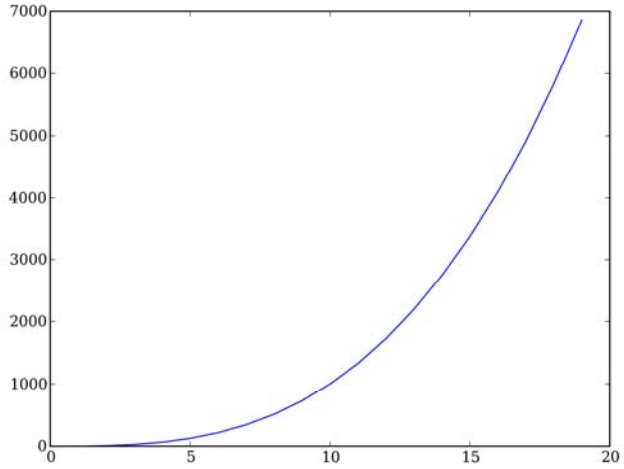
2) The code below produces three plots. Match each of the plots on the next page with the appropriate figure (Figure 1, Figure 2, or Figure 3).

```
import pylab

y1 = []
y2 = []
y3 = []
for i in range(20):
    y1.append(300*i)
    y2.append(i**3)
    y3.append(2**i)
pylab.figure(1)
pylab.plot(y1)
pylab.figure(2)
pylab.plot(y2)
pylab.figure(3)
pylab.plot(y3)
pylab.semilogy()
```


$10^6$
$10^5$
$10^4$
$10^3$
$10^2$
$10^1$
$10^0$
0
5
10
15
20


6000
5000
4000
3000
2000
1000
0
0
5
10
15
20


7000
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20

3) Can the following specification be implemented? If not, explain why.

```
def f(x, e):
    """ x and e are floats
        returns y, a float, such that x-e <= y*y <= x+e"""
```

4) Write a Python program that satisfies the following specification.

```
def f(L1, L2):
   """ L1 and L2 are lists of integers
       returns the element-wise product of L1 and L2.
       If the lists are of different length, it uses 1
       for the missing coefficients.
       E.g., if L1 = [2, 2, 3] and L2 = [7, 5, 6, 7]
       it returns [14, 10, 18, 7]
    side effects: none """
```

The following questions all refer to the code you were asked to study in preparation for this exam. They also assume that the following code is used to test the program. (For your convenience, a copy of the posted code is at the end of this quiz. Feel free to separate it from the rest of the quiz.)

**Warning: The code at the end of this is NOT the same as the code you have been given to study. It may not be worth your time to study this code carefully enough to answer all of the practice questions, but these questions should give you the flavor of the kinds of questions we might ask about the code supplied this term.**

```
numDays = 500
bias = 0.11/200.0
numSectors = 1
sectorSize = 500
numTrials = 5

print 'Testing Model on', numDays, 'days'
mean = 0.0
for i in range(numTrials):
    pylab.figure(1)
    mkt = generateMarket(numSectors, sectorSize, bias)
    endPrices = simMkt(mkt, numDays)
    mean += endPrices[-1]
    title = 'Closing Prices With ' + str(numSectors) + ' Sectors'
    plotValueOverTime(endPrices, title)
    pylab.figure(2)
    plotDistributionAtEnd(mkt, title)
meanClosing = mean/float(numTrials)
print 'Mean closing with', numSectors, 'sectors is', meanClosing
```

5.1) If, in class Stock, the line

```
self.price = self.price * (1.0 + baseMove)
```

were replaced with the line

```
self.price = self.price * baseMove
```

which of the following best describes the expected mean value of the market after 200 days:

a. Close to 111

b. Close to 100

c. Close to 50

d. Close to 0

e. About the same as with the original code

5.2) Consider running the above test code twice, once with `numSectors = 1` and once with `numSectors = 20`. Each will produce a Figure 1. Now consider running a linear regression to fit a line to each of the curves produced in the Figure 1's for each test. Would you expect the average mean square error of the fits for `numSectors = 20` to be:

a. Smaller than the average mean square error of the fits for `numSectors = 1`.

b. Larger than the average mean square error of the fits for `numSectors = 1`.

c. About the same as the average mean square error of the fits for `numSectors = 1`.

d. None of the above.

5.3) Characterize the algorithmic efficiency of `generateMarket`. (10 points)

5.4) If in Stock.makeMove, the line

```
baseMove=bias+random.uniform(-self.volatility, self.volatility)
```

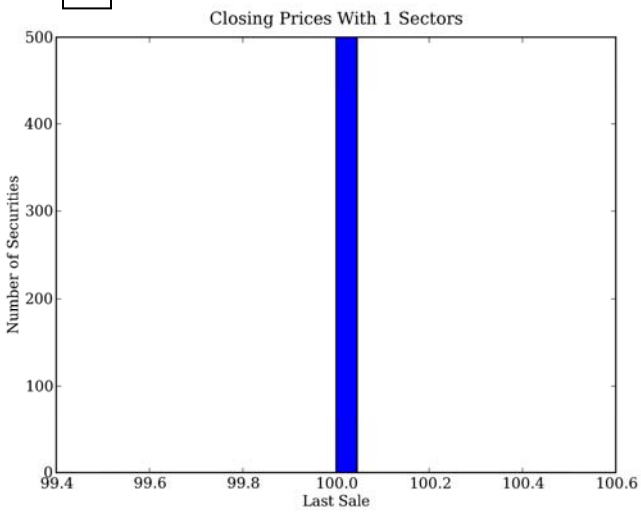
were replaced by the line

```
baseMove = bias
```

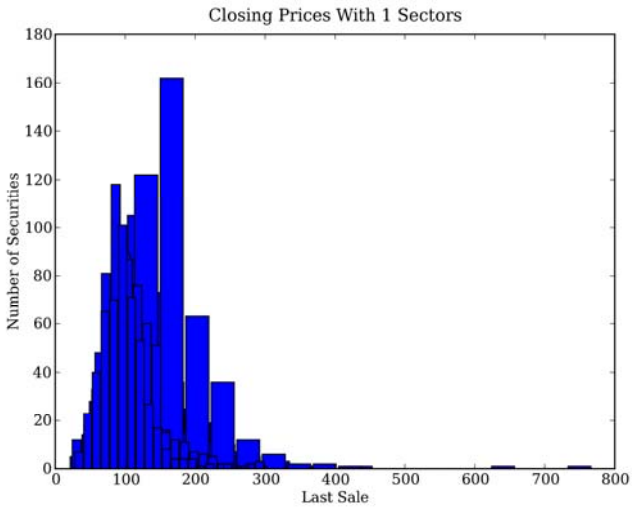
which of the following Figure 2's is most likely to be produced by the test code? (10 points)

A

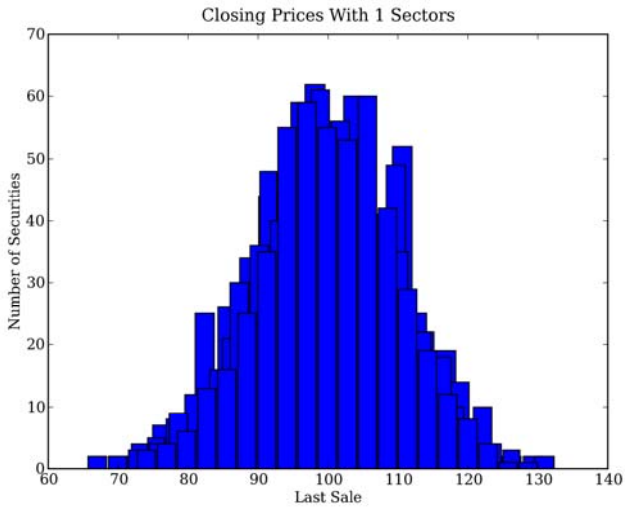

B




C




D



```
import pylab
import random

# Global constant
TRADING_DAYS_PER_YEAR = 200

class Stock:
    def __init__(self, ticker, volatility):
        self.volatility = volatility
        self.ticker = ticker
        self.price = None
        self.history = []
    def setPrice(self, price):
        self.price = price
        self.history.append(price)
    def getPrice(self):
        return self.price
    def getTicker(self):
        return self.ticker
    def makeMove(self, bias):
        if self.price == 0.0:
            return
        baseMove = bias + random.uniform(-self.volatility, self.volatility)
        self.price = self.price * (1.0 + baseMove)
        if self.price < 0.01:
            self.price = 0.0
        self.history.append(self.price)


class Market:
    def __init__(self):
        self.sectors = []
        self.tickers = set()
    def addSector(self, sect):
        if sect.getName() in self.sectors:
            raise ValueError('Duplicate sector')
        for t in sect.getTickers():
            if t in self.tickers:
                raise ValueError('A ticker in sect already in market')
            else: self.tickers.add(t)
        self.sectors.append(sect)
    def getSectors(self):
        return self.sectors
    def getStocks(self):
        stocks = []
        for s in self.sectors:
            stocks = stocks + s.getStocks()
        return stocks
    def moveAllStocks(self):
        vals = []
        for s in self.sectors:
            vals = vals + s.moveAllStocks()
        return vals
```

```
class Sector:
    def __init__(self, sectorName, bias):
        self.tickers = set()
        self.sectorName = sectorName
        self.stocks = []
        self.bias = bias
        self.origBias = bias
    def addStock(self, stk):
        if stk.getTicker() in self.tickers:
            raise ValueError('Duplicate ticker')
        self.tickers.add(stk.getTicker())
        self.stocks.append(stk)
    def getStocks(self):
        return self.stocks
    def getTickers(self):
        return self.tickers
    def setBias(self, newBias):
        self.bias = newBias
    def getBias(self):
        return self.bias
    def getOrigBias(self):
        return self.origBias
    def sameSector(self, other):
        return self.sectorName == other.sectorName
    def getName(self):
        return self.sectorName
    def moveAllStocks(self):
        vals = []
        for stock in self.stocks:
            stock.makeMove(self.bias)
            vals.append(stock.getPrice())
        return vals


def generateSector(sectorSize, sectorName, bias):
    sect = Sector(sectorName, bias)
    for i in range(sectorSize):
        ticker = sectorName + 'Ticker ' + str(i)
        volatility = random.uniform(0.01, 0.04)
        stk = Stock(ticker, volatility)
        stk.setPrice(100)
        try:
            sect.addStock(stk)
        except ValueError:
            # Tickers are all different by construction, so this should never
            # happen
            print 'Error in generate stocks, should never reach here.'
            raise AssertionError
    return sect

def generateMarket(numSectors, sectorSize, bias):
    mkt = Market()
    for n in range(numSectors):
        sect = generateSector(sectorSize, 'Sector ' + str(n), bias)
        mkt.addSector(sect)
    return mkt
```

```
def simMkt(mkt, numDays):
    endPrices = []
    for i in range(numDays):
        if i%(TRADING_DAYS_PER_YEAR/4) == 0:
            for s in mkt.getSectors():
                newBias = s.getOrigBias() + random.gauss(0, 2*s.getOrigBias())
                s.setBias(newBias)
        vals = mkt.moveAllStocks()
        vals = pylab.array(vals)
        mean = vals.sum()/float(len(vals))
        endPrices.append(mean)
    return endPrices

def plotValueOverTime(endPrices, title):
    pylab.plot(endPrices)
    pylab.title(title)
    pylab.xlabel('Days')
    pylab.ylabel('Price')

def plotDistributionAtEnd(mkt, title):
    prices = []
    for s in mkt.getStocks():
        prices.append(s.getPrice())
    pylab.hist(prices, bins = 20)
    pylab.title(title)
    pylab.xlabel('Last Sale')
    pylab.ylabel('Number of Securities')
```