6.00 Quiz 3, Spring 2011 ----------

Name

1. _____/15
2. _____/10
3. _____/15
4. _____/18
5. _____/5
6. _____/17
7. _____/5
8. _____/5
9. _____/10

Total _____ /100

This quiz is open book and open notes, but do not use a computer (or cell phone!). You have 120 minutes.

Please **write your name on the top of each page**. Answer all questions in the boxes provided.

1) Are each of the following True or False? (15 points)

True	1.1. Dynamic programming provides an `O(n)` solution to the 0/1 knapsack problem, where `n` is the number of items to be considered.
True	1.2. K-means clustering is more computationally efficient than hierarchical clustering.
True	1.3. The depth of a decision tree is related to the number of independent decisions it records.
True	1.4. Hierarchical clustering is deterministic, but k-means clustering is not.
false	1.5. When used to find a root of a polynomial, Newton's method converges in `O(log(n))` iterations, where n is the degree of the polynomial.

Name

2) Replace the "?" in the code below by an expression that guarantees that if the first assert does not raise an exception, the **second** assert will not raise an exception. The expression should not include a call to comp. (10 points)

```
def comp(s):
    res = 0
    for c1 in s:
        res += 2
        for c2 in s:
            res -= 1
    return res
assert type(s) == str
assert comp(s) == ?
```

len(s) * len(s) * (-1) + len(s) * 2

```
assert comp(s) == len(s)*2 - len(s)**2
```

3) Consider the following code.

```
def throwNeedles(fcn, numNeedles = 100000):
    inCircle = 0
    estimates = []
    for Needles in xrange(1, numNeedles + 1, 1):
        x = fcn(0, 1)
        y = fcn(0, 1)
        if (x*x + y*y) <= 1.0:
            inCircle += 1
    return 4*(inCircle/float(numNeedles))

print throwNeedles(random.uniform) + [a,b] (a,b)
print throwNeedles(random.gauss) - and + (mu, sigma)
print throwNeedles(random.random) Error
```

b

3.1. With high probability, it will first print a value that is approximately equal to
- a. `0.5*pi`
- b. `pi`
- c. `2*pi`
- d. `4*pi`
- e. None of the above

a

3.2. With high probability, it will next print a value that is
- a. less than the first value printed
- b. about equal to the first value printed
- c. greater than the first value printed

e

3.3. With high probability, it will next print a value that is approximately equal to
- a. `0.5*pi`
- b. `pi`
- c. `2*pi`
- d. `4*pi`
- e. None of the above

(15 points)

Name ----------

4) Next to each item in the left column write the letter labeling the item in the right column that best matches the item in the left column. No item in the right column should be used more than once. (18 points)

[g]	objective function	a) digraph
[h]	confidence interval	b) undirected graph
[f]	depth first search	c) local optima
[d]	hierarchical clustering	d) linkage criterion
[c]	greedy algorithm	e) unit testing
[i]	feature vector	f) backtracking
		g) optimization
		h) standard deviation
		i) normalization

Name

5) To get a sense of the quality of its dormitories, MIT surveyed all students living in a dorm. This is an example of (select one): (5 points)

b

a) The Texas sharpshooter fallacy

b) Sample bias

c) GIGO

d) A well-designed study

The following questions all refer to the code you were asked to study in preparation for this exam. A copy of the posted code is at the end of this quiz. Feel free to detach it.

6.1) Ryan believes that if a LIFO queue were used at the bus stops, the average wait time for buses with a small capacity would decline. Implement a class LIFO and indicate what you modification you would make elsewhere in the code to test Ryan's conjecture. (10 points)

```
class LIFO(JobQueue):
    def depart(self):
        try:
            return self.jobs.pop(-1)
        except:
            raise ValueError('Empty Queue')
```

```
#Changed Code
class BusStop(LIFO):
    pass
```

Run test with a list of decreasing capacities list for LIFO and FIFO, save datas. Compare datas at plots of capacities.

6.2) Do you believe Ryan's conjecture? Why or why not? (7 points)

Yes, because we are computing boarded passengers average wait times. Busses are small, so people get behind are those waited longer.

7) If a bus leaves a stop with 5 passengers on it, how many passengers will get off at the next stop? Hint: The answer is an integer in `range(6)`. (5 points)

1

8) Indicate how you would modify `simBus` so that it returns the ratio of the total number of passengers carried to the total number of passengers who arrive at bus stops but never board a bus. (5 points)

return float(totPassengers)/leftWaiting

Name

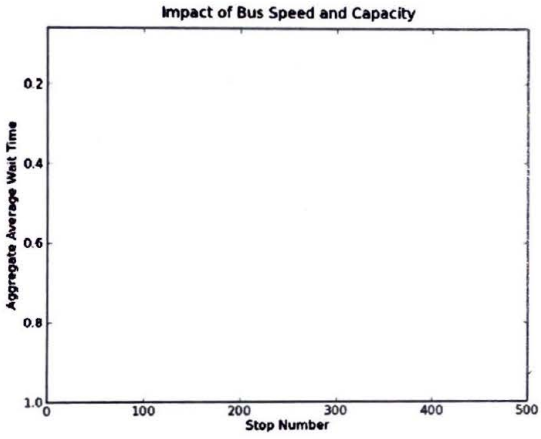
9) Match each of the plots below with a test that could have produced it, assuming that the statement `pylab.legend(loc = 'best')` were removed from `test`. You may not use the same plot more than once. (10 points)

`test([20], [0], 20)` C

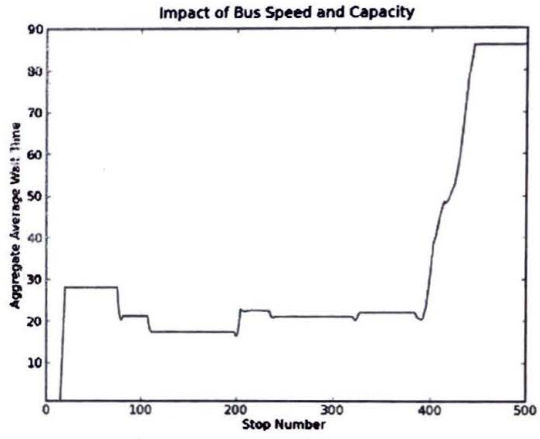
`test([0], [20], 20)` A

`test([1], [200], 1)` B

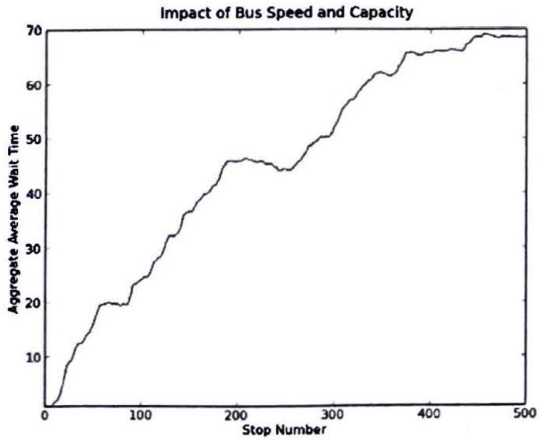
A



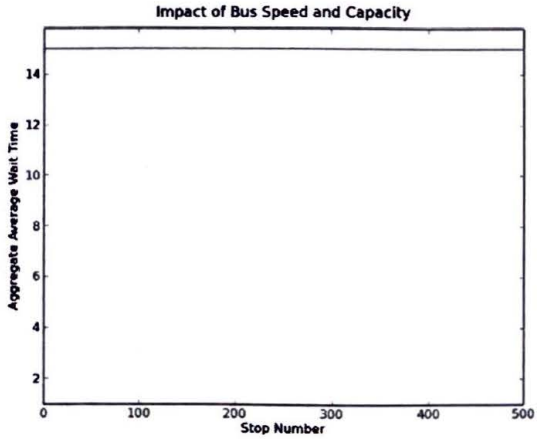
B



C



D



```
import random, pylab, math


class Job(object):
    def __init__(self, meanArrival, meanWork):
        #arrival rate of jobs
        self.arrival = random.expovariate(1.0/meanArrival)
        #time required to perform job, other distributions worth considering
        self.wk = random.gauss(meanWork, meanWork/2.0)
        #Next attribute used to keep track of waiting time for job
        self.timeQueued = None
    def interArrival(self):
        return self.arrival
    def work(self):
        return self.wk
    def queue(self, time):
        self.timeQueued = time
    def queuedTime(self):
        return self.timeQueued

class Passenger(Job):
    #Arrival rate is for passenger to arrive at bus stop
    #Work is time for passenger to board bus
    pass

class JobQueue(object):
    def __init__(self):
        self.jobs = []
    def arrive(self, job):
        self.jobs.append(job)
    def length(self):
        return len(self.jobs)

class FIFO(JobQueue):
    def depart(self):
        try:
            return self.jobs.pop(0)
        except:
            print 'depart called with an empty queue'
            raise ValueError('EmptyQueue')

class SRPT(JobQueue):
    def depart(self):
        try:
            leastIndx = 0
            for i in range(len(self.jobs)):
                if self.jobs[i].work < self.jobs[leastIndx].work:
                    leastIndx = i
            return self.jobs.pop(leastIndx)
        except:
            print 'depart called with an empty queue'
            raise ValueError('EmptyQueue')

class BusStop(FIFO):
    pass
```

```
class Bus(object):
    def __init__(self, capacity, speed):
        self.cap = capacity
        self.speed = speed
        self.onBus = 0
    def getSpeed(self):
        return self.speed
    def getLoad(self):
        return self.onBus
    def enter(self):
        if self.onBus < self.cap:
            self.onBus +=1
        else:
            raise ValueError('full')
    def leave(self):
        if self.onBus > 0:
            self.onBus -= 1
    def unload(self, num):
        while num > 0:
            self.leave()
            num -= 1
```

```
def simBus(bus, numStops = 6, loopLen = 1200, meanArrival = 90,
           meanWork = 10, simTime = 50000):
    assert loopLen%numStops == 0
    stops = []
    for n in range(numStops):
        stops.append(BusStop())
    time, totWait, totPassengers, lastArrival = [0.0]*4
    aveWaitTimes = []
    nextStop, busLoc, time = [0]*3
    nextJob = Passenger(meanArrival, meanWork)
    while time < simTime:
        #advance time and move bus
        time += 1
        for i in range(bus.getSpeed()):
            busLoc += 1
            if (busLoc)%(loopLen/numStops) == 0:
                break
        #see if there is a passenger waiting to enter queue
        if lastArrival + nextJob.interArrival() <= time:
            #passengers arrive simultaneously at each stop
            for stop in stops:
                stop.arrive(nextJob)
            nextJob.queue(time)
            lastArrival = time
            nextJob = Passenger(meanArrival, meanWork)
        #see if bus is at a stop
        if (busLoc)%(loopLen/numStops) == 0:
            #some passengers get off bus
            bus.unload(math.ceil(bus.getLoad()/float(numStops)))
            #all passengers who arrived prior to the bus's arrival
            #attempt to enter bus
            while stops[nextStop%numStops].length() > 0:
                try:
                    bus.enter()
                except:
                    break
                p = stops[nextStop%numStops].depart()
                totWait += time - p.queuedTime()
                totPassengers += 1
                time += p.work() #advance time, but not bus
            try:
                aveWaitTimes.append(totWait/totPassengers)
            except ZeroDivisionError:
                aveWaitTimes.append(0.0)
            #passengers might have arrived at stops while bus is loading
            while lastArrival + nextJob.interArrival() <= time:
                for stop in stops:
                    stop.arrive(nextJob)
                nextJob.queue(time)
                lastArrival += nextJob.interArrival()
                nextJob = Passenger(meanArrival, meanWork)
            nextStop += 1
    leftWaiting = 0
    for stop in stops:
        leftWaiting += stop.length()
    return aveWaitTimes, leftWaiting
```

```
def test(capacities, speeds, numTrials):
    random.seed(0)
    for cap in capacities:
        for speed in speeds:
            totWaitTimes = pylab.array([0.0]*500) #keep track of 1st 500 stops
            totLeftWaiting = 0.0
            for t in range(numTrials):
                aveWaitTimes, leftWaiting = simBus(Bus(cap, speed))
                totWaitTimes += pylab.array(aveWaitTimes[:500])
                totLeftWaiting += leftWaiting
            aveWaitTimes = totWaitTimes/numTrials
            leftWaiting = int(totLeftWaiting/numTrials)
            lab = 'Spd = ' + str(speed) + ', Cap = ' + str(cap)\
                  + ', Left = ' + str(leftWaiting)
            pylab.plot(aveWaitTimes, label = lab)
    pylab.xlabel('Stop Number')
    pylab.ylabel('Aggregate Average Wait Time')
    pylab.title('Impact of Bus Speed and Capacity')
    ymin, ymax = pylab.ylim()
    if ymax - ymin > 200:
        pylab.semilogy()
    pylab.ylim(ymin = 1.0)
    pylab.legend(loc = 'best')
```