6.00 Quiz 3 ----------

Name

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

Athena User Name

Recitation hour

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**, and your user name and the hour of the recitation you attend on the first page. Answer all questions in the boxes provided.

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

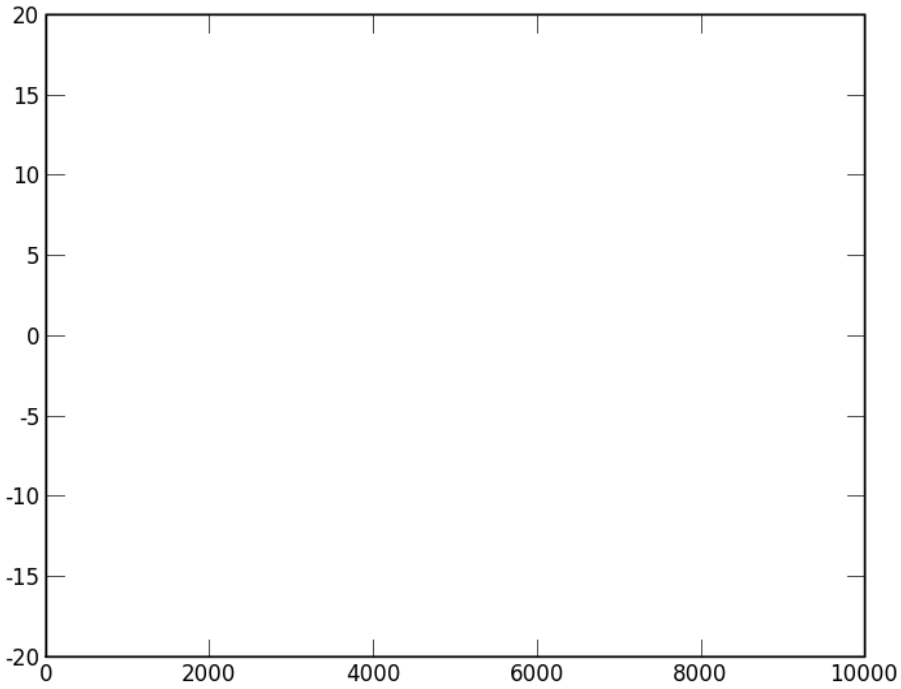
	1.1. The result of agglomerative hierarchical clustering depends upon the linkage criterion used.
	1.2. K means clustering is usually faster than agglomerative hierarchical clustering.
	1.3. When run on a set of data, the result of k-means clustering does **not** depend on the initial centroids.
	1.4. Agglomerative hierarchical clustering is a deterministic algorithm.
	1.5. The continuous knapsack problem **cannot** be solved in O(n log n) time.

2) Consider the following code.

```
yVals = []
for i in range(10000):
    yVals.append(random.gauss(0, 4))
xVals = pylab.arange(10000)
a, b, c = pylab.polyfit(xVals, yVals, 2)
print round(a)
print round(b)
print round(c)
pylab.plot(sorted(yVals, reverse = True))
pylab.xlim(0, 10000)
pylab.ylim(-20, 20)
```

2.1. What does it print? (8 points)

2.2. Draw an approximation to the plot it is likely to produce (7 points)

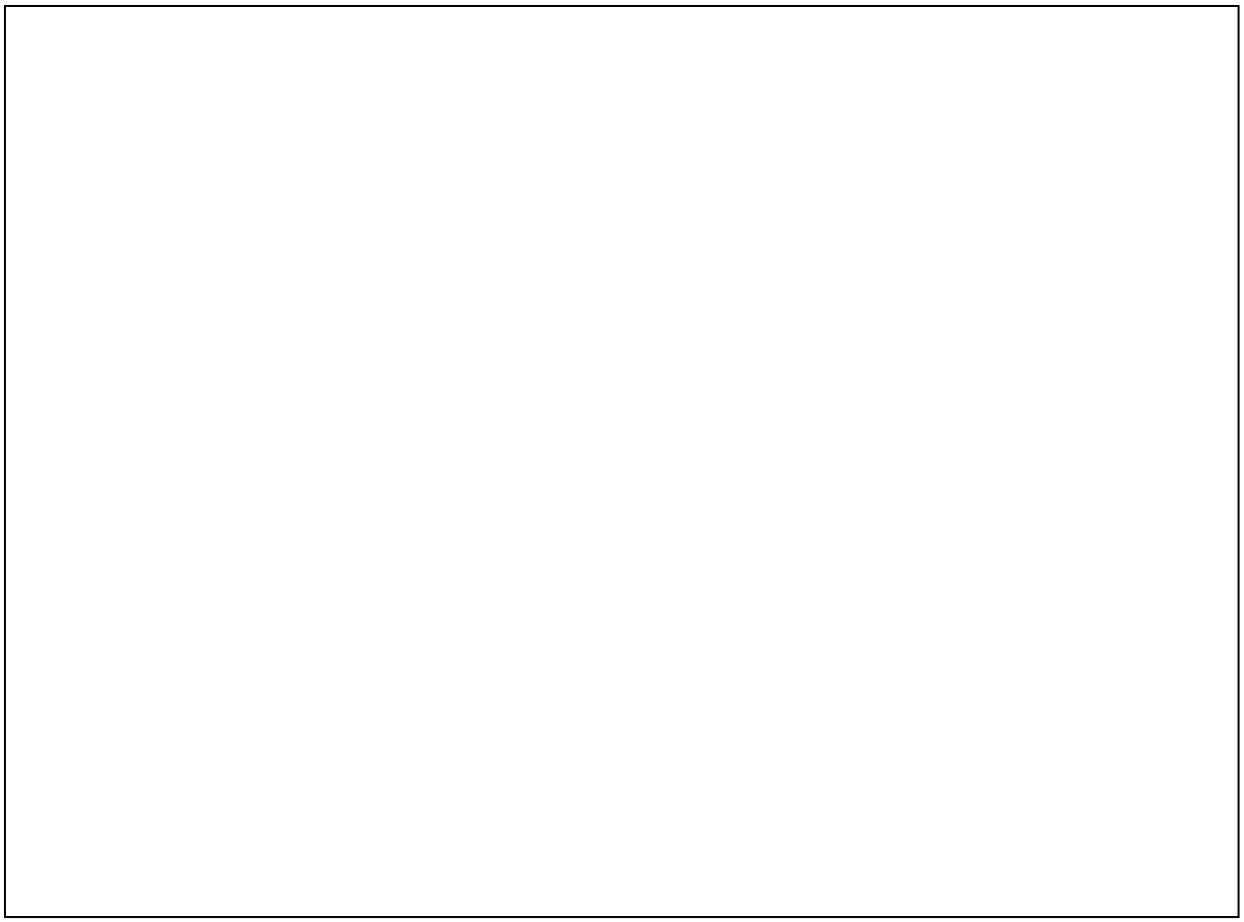


3) 1000 students took an online course. ¼ of them were from Africa, ¼ from Europe , ¼ from South America, and ¼ from Asia. At the end of the course, the instructor observed that of the top 100 grades, 35 belonged to students from one geographical area (South America). He argued that since the expected number of students from each area in the top 100 was 25, this was unlikely to have happened by pure chance. Write a program that returns an estimate of the probability of this happening purely by chance. (15 points)

4) Consider the following two implementations. Would you expect `f2` to be appreciably faster than `f1`? Explain why or why not. (10 points)

```
def f1(x, y, g):
    if x == 0:
        return 0
    result = 0
    for i in range(x):
        result += g(i, y) + f1(x-1, y, g)
    return result

def f2(x, y, g, memo = {}):
    print memo
    if x == 0:
        return 0
    try:
        return memo[(x, y)]
    except:
        result = 0
        for i in range(x):
            result += g(i, y) + f2(x-1, y, g, memo)
        memo[(x, y)] = result
    return result
```



The following questions all refer to the code you were asked to study in preparation for this exam. (For your convenience, a copy of the posted code is available in a separate document today.
AS: Study code is at the end of the exam)

5) If the line

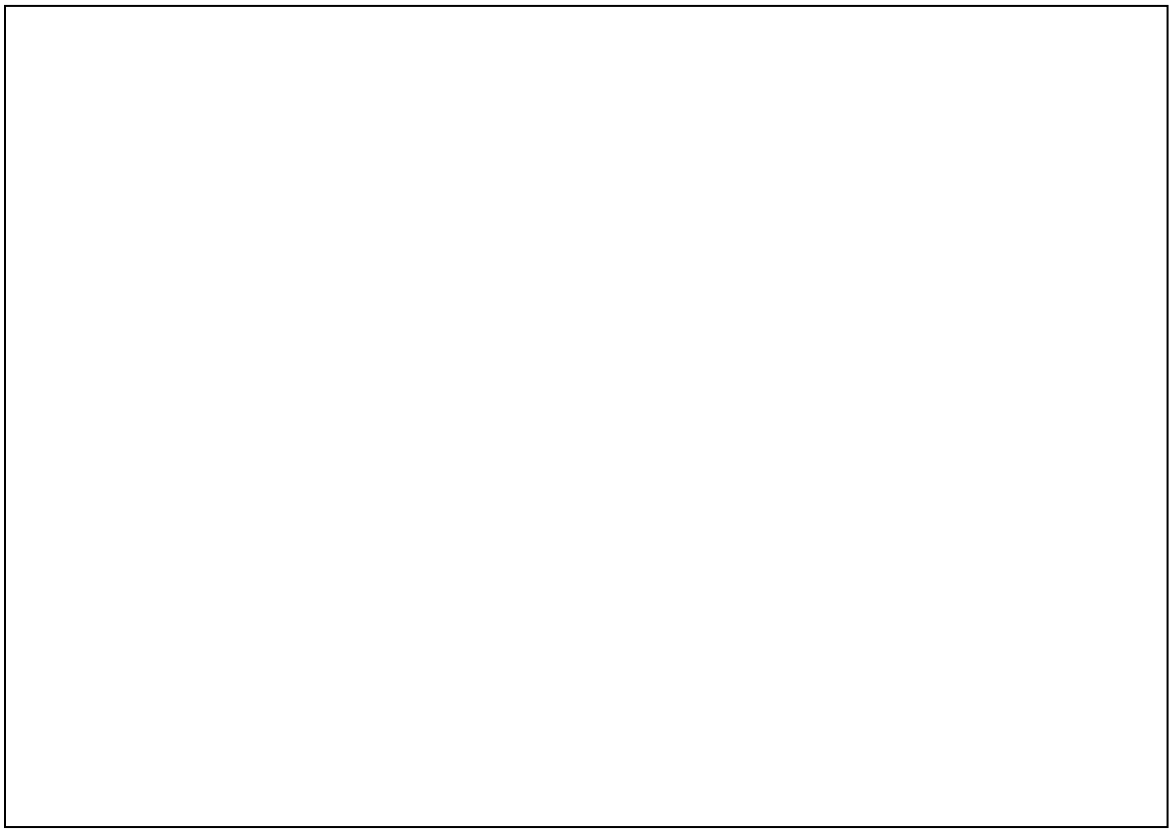
```
class Map(Digraph):
```

were replaced by

```
class Map(Graph):
```

would you expect `bigTest` to take substantially longer to execute. Explain why or why not.

(10 points)



6) Show how to change `findMinWP` to find a minimum path that does not contain any nodes where the quality of the food is `'awful'`. (10 Points)

```
def findMinWP(graph, start, end, weightFcn):
    def minWeightPath(graph, end, weightFcn, path):
        currentNode = path.getLastNode()
        if currentNode == end:
            return path
        shortest = None
        for edge in graph.edgesOf(currentNode):
            if not path.contains(edge.getDestination()):
                newPath = minWeightPath(graph, end, weightFcn, path + edge)
                if newPath != None:
                    if shortest == None or\
                       newPath.getWeight(weightFcn)\
                       < shortest.getWeight(weightFcn):
                        shortest = newPath
        return shortest
    if not (graph.hasNode(start) and graph.hasNode(end)):
        raise ValueError, 'Start or end not in graph.'
    result = minWeightPath(graph,  end, weightFcn, Path(start))
    if result == None:
        raise ValueError, 'Cannot get from ' + str(start) + ' to ' + str(end)
    return result
```

7) Write a function that meets the specification below. (15 points)

```
def cheapestTrip(g, src, dest, gasPrice):
    """returns a path from src to dest that minimizes
       the total cost (money spent on gas + tolls) of the trip.
       returns None if no path exists"""
```

8) Which of the following is an accurate statement about the computational complexity of `findMinWP`? (5 points)

A. If `findMinWP` is called with an **acyclic** graph, its complexity is linear in the number of edges in the graph.

B. If `findMinWP` is called with a graph that **contains cycles**, its complexity is linear in the number of edges in the graph.

C. `FindMinWP` is quadratic in max(number of nodes, number of edges).

D. None of the above.

9) Which of the following best describes the result of the invocation `bigTest(0, 1)`? (5 points)

A. An exception will be raised in `buildRandomGraph`.

B. An exception will be raised in `findMinWP`.

C. An exception will be raised in `bigTest`.

D. No exception will be raised.

```
##Please study the code below in preparing for the 6.00 final
##exam.  The exam will contain several questions related to this
##code.  Trying to understand the code in realtime during the exam
##would not be a good idea.  I suggest that you read it, run it,
##and try and modify it in simple ways.

import random

class Node(object):
    def __init__(self, name, foodQuality):
        self.name = name
        self.foodQuality = foodQuality
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
    def __eq__(self, other):
        return self.name == other.getName()

class Weight(object):
    pass

class Edge(object):
    def __init__(self, src, dest, weight):
        self.src = src
        self.dest = dest
        self.weight = weight
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def getWeight(self):
        return self.weight
    def __str__(self):
        return str(self.src) + '->' + str(self.dest)\
               + '(' + str(self.weight) + ')'
```

```
class Digraph(object):
    def __init__(self):
        self.nodes = set([])
        self.edges = {}
    def addNode(self, node):
        if node in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.add(node)
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not(src in self.nodes and dest in self.nodes):
            raise ValueError('Node not in graph')
        self.edges[src].append(edge)
    def numNodes(self):
        return len(self.nodes)
    def childrenOf(self, node):
        result = []
        for e in self.edges[node]:
            if not e.getDestination() in result:
                result.append(e.getDestination())
        return result
    def edgesOf(self, node):
        result = []
        for e in self.edges[node]:
            result.append(e)
        return result
    def hasNode(self, node):
        return node in self.nodes
    def __str__(self):
        res = ''
        for k in self.edges:
            for e in self.edges[k]:
                res = res + str(e) + '\n'
        return res[:-1]

class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource(), edge.getWeight())
        Digraph.addEdge(self, rev)
```

```
class DriveInfo(Weight):
    def __init__(self, distance, speed, mpg, tollsPaid):
        try:
            self.time = float(distance)/speed
            self.gasUsed = float(distance)/mpg
        except ZeroDivisionError:
            self.time, self.gasUsed = 0.0, 0.0
        self.tollsPaid = tollsPaid
    def getTime(self):
        return self.time
    def getGasUsed(self):
        return self.gasUsed
    def getTollsPaid(self):
        return self.tollsPaid

class Drive(Edge):
    def __init__(self, city1, city2, info):
        Edge.__init__(self, city1, city2, info)

class Map(Digraph):
    pass

class Path(object):
    def __init__(self, start):
        assert type(start) == Node
        self.val = [(start, DriveInfo(0.0, 0.0, 0.0, 0.0))]
        self.weights = {}
    def getStart(self):
        return self.val[0][0]
    def getWeight(self, weightFcn):
        try:
            result = self.weights[weightFcn]
        except:
            result = 0.0
            for step in self.val:
                result += weightFcn(step[1])
            self.weights[weightFcn] = result
        return result
    def getLength(self):
        return len(self.val) - 1
    def __add__(self, edge):
        result = Path(self.getStart())
        for elem in self.val[1:]:
            result.val.append(elem)
        result.val.append((edge.getDestination(), edge.getWeight()))
        return result
    def contains(self, node):
        for step in self.val:
            if step[0] == node:
                return True
        return False
    def __str__(self):
        result = ''
        for step in self.val:
            result = result + '->' + str(step[0])
        return result[2:]
```

```
def findMinWP(graph, start, end, weightFcn):
    def minWeightPath(graph, start, end, weightFcn, path, edge):
        if not (graph.hasNode(start) and graph.hasNode(end)):
            raise ValueError, 'Start or end not in graph.'
        if path == None:
            path = Path(start)
        else:
            path = path + edge
        if start == end:
            return path
        shortest = None
        for edge in graph.edgesOf(start):
            if not path.contains(edge.getDestination()):
                newPath = minWeightPath(graph, edge.getDestination(),
                                        end, weightFcn, path, edge)
                if newPath != None:
                    if shortest == None or\
                       newPath.getWeight(weightFcn)\
                       < shortest.getWeight(weightFcn):
                        shortest = newPath
        return shortest
    result = minWeightPath(graph, start, end, weightFcn, None, None)
    if result == None:
        raise ValueError, 'Cannot get from ' + str(start) + ' to ' + str(end)
    return result

def buildRandomGraph(numNodes, numEdges):
    nodes = []
    for name in range(numNodes):
        foodQuality = random.choice(('excellent', 'good', 'ok', 'poor', 'awful'))
        nodes.append(Node(str(name), foodQuality))
    g = Map()
    for n in nodes:
        g.addNode(n)
    for e in range(numEdges):
        src = nodes[random.choice(range(0, len(nodes)))]
        dest = nodes[random.choice(range(0, len(nodes)))]
        distance = random.randint(1, 200)
        speed = random.random()*80
        mpg = random.gauss(24, 2)
        tollsPaid = random.random()*75
        weight = DriveInfo(distance, speed, mpg, tollsPaid)
        g.addEdge(Drive(src, dest, weight))
    return g, nodes
```

```
def bigTest(numNodes, numEdges):
    g, nodes = buildRandomGraph(numNodes, numEdges)
    try:
        shortest = findMinWP(g, nodes[1], nodes[3], DriveInfo.getTime)
        print 'The minimum time path is', shortest
        print 'The minimum time is', round(shortest.getWeight(DriveInfo.getTime),
2), 'hours'
        print 'The gas used is', round(shortest.getWeight(DriveInfo.getGasUsed),
2),\
              'gallons'
    except ValueError, s:
        print s
    try:
        shortest = findMinWP(g, nodes[1], nodes[3], DriveInfo.getGasUsed)
        print 'The minimum gas used path is', shortest
        print 'The minimum gas used is',
round(shortest.getWeight(DriveInfo.getGasUsed), 2),\
              'gallons'
        print 'The time is', round(shortest.getWeight(DriveInfo.getTime), 2),
'hours'
    except ValueError, s:
        print s

bigTest(20, 70)
```