6.00 Final Exam, December 16, 2013

Name

1. _______/24
2. _______/16
3. _______/10
4. _______/20
5. _______/10
6. _______/20

Total _______/100

This exam is open book and open notes, but do not use a computer. **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 (24 points)

1.1. The code `print 3.0 + 'a'` has incorrect syntax.

1.2. In Python, an argument to a function cannot be another function.

1.3. All $O(1)$ functions take exactly the same amount of time to run.

1.4. In Python, any class is a subclass of `object`.

1.5. Consider an undirected graph with non-negative weights that has an edge between each pair of nodes. The shortest distance between any two nodes is always the path that is the edge between the two nodes.

1.6. A decision tree does not have to be binary. In other words, for a decision node, you can have more than 2 children representing more than 2 decisions.

1.7. If we let the k-means clustering algorithm run for a very long time, we will eventually end up with all the data points in one cluster.

1.8. Training an algorithm on data set A and then testing it on a completely separate data set B is an example of unsupervised learning.
2) [16 points] What does the following code print? If you think the code throws an error, write “error”.

a) [4 points]
a = [1, 2, 3]
b = a
a.append(0)
print(b)

b) [4 points]
t = (2, 'ab', 5, (3,4), 'return')
print(t[-2])

c) [4 points]
d = {'1':'2', '3':'4', '5':'6'}
d['3'] = 3
print(d['3'] * d['5'])

d) [4 points]
try:
    1/0
except:
    print('Not again!')
finally:
    print('Good!')
3) [10 points] Consider being given a list of positive (there is at least one positive) and negative numbers. You are asked to find the maximum sum of a contiguous subsequence. For example, given the list [3, 4, -1, 5, -4], the maximum sum is 3+4-1+5 = 11 or as another example, given the list [3, 4, -8, 15, -1, 2], the maximum sum is 15-1+2 = 16.

a) [2 points] The brute force algorithm goes through all possible subsequences and compares the sums of each contiguous subsequence with the largest sum it has seen. What is the time complexity of this algorithm in terms of the length of the list N?

b) [8 points] Now implement the algorithm using dynamic programming and recursion. Instead of looking at all possible contiguous subsequences, we will incrementally look at each element in the array. We will keep a variable representing the best sum we have seen so far and a variable representing the current sum (instead of computing the sum for all possible subsets of the list). The trick is apparent in the second example above. Notice that when current sum reaches negative (3+4-8 < 0) then no matter how large the rest of the numbers in the list are, if we add this negative sum to the rest of the numbers, the result will always be less than if we dropped the negative sum and reset to 0.

```python
def maxContigSumRec(L):
    def maxContigSumRecHelper(best, cur, index, L):
        # base case is when index reached the end of the list L
        if
            # recursive case
        else:
            # new current sum, keeping track of whether it
            # needs to be reset to 0 because it went negative

            # new best sum

            # return the recursive call

        # call to recursive helper
    return maxContigSumRecHelper(0, 0, 0, L)
```
4) [20 points] In this question, you will be writing two versions of a function, one iterative and one recursive. The function will take as input two strings s1 and s2 and will alternate letters from the two strings, starting with the first letter of s1. If the strings are not the same length then the remaining letters after interleaving will be appended to the end of the result string.
For example, if s1="abcde" and s2="fgh" then the function will return "afbgchde"

a) [10 points] Write the iterative version of the function.

```python
def interleaveIter(s1, s2):
    
    """
    Input strings s1 and s2 are not necessarily the same length. Returns a string representing the weaved strings """
```
b) [10 points] Write the **recursive** version of the function.

```python
def interleaveRecur(s1, s2):
    """ Input strings s1 and s2 are not necessarily the same length. Returns the interleaved strings, starting with s1 and any extra elements appended at the end """
    def helpInterleave(s1, s2, out):
        """ helper function """
        ## YOUR CODE HERE
    return helpInterleave(s1, s2, '')
```

```python
return helpInterleave(s1, s2, '')
```
5) [10 points] Suppose we want to write a function that calculates the minimum number of coins needed to form a certain number greater than 0. Assume that the coins available are worth 1, 5, 8, and 10 and you can use as many coins as you need.

a) [6 points] Assume a greedy approach to this problem, where you try to find how many coins you need of the largest denomination, then of the second largest, and so on. Write a function that returns the number of coins needed.

```python
def greedyCoins(N):
    """ Input is number N that the coins should add up to. Assume the denominations available are 1, 5, 8, 10. Use a greedy approach to determine how many of each denomination is needed to add up to N. Return the total number of coins needed. """
    denom = [1, 5, 8, 10]
    # Your code here
```

b) [3 points] Will the greedy approach always give the smallest number of coins needed? If yes, explain. If no, provide a counterexample.
6) [20 points] 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 exam. Feel free to detach it.

a) [2 points] What is the worst-case time complexity of the function dijkstra(graph, source) in terms of the number of nodes N?

b) [8 points] The function dijkstra in the study code finds the shortest paths between a start node and all other nodes. It returns a dictionary of the shortest distances and a dictionary of every node’s parents to yield the shortest distance. Write a function called dijkstraSP, which takes as arguments a graph, start node, and end node, and prints the shortest path route (using the function printPath) found by the function dijkstra.

def dijkstraSP(graph, startNode, endNode):
    """ graph is a Graph, startNode and endNode are Node objects representing the nodes to find the shortest path between. Prints the path of the form: startNode->node#->...->endNode , representing the shortest path found by dijkstra """
c) [10 points] Write a function that finds the average of the shortest paths between each pair of nodes in a graph. Do not include the path from the node to itself in the calculation for the average. Use the result from the function dijkstra.

def averageSP(graph):
    """ Input graph is a Graph object. Calculates the shortest path between all node pairs and returns the average length """
## Please study the code below in preparing for the 6.00 final exam. The exam will contain questions related to this code. Trying to understand the code in realtime during the exam would not be a good idea. We suggest that you read it, run it, and try and modify it in simple ways.

```python
class Node(object):
    def __init__(self, name):
        self.name = str(name)
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
    def __hash__(self):
        return hash(self.name)
    def __eq__(self, other):
        return self.name == other.name

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

class WeightedEdge(Edge):
    def __init__(self, src, dest, weight = 1.0):
        self.src = src
        self.dest = dest
        self.weight = weight
    def getWeight(self):
        return self.weight
    def __str__(self):
        return str(self.src) + '->(' + str(self.weight) + ')' + str(self.dest)

class Digraph(object):
    def __init__(self):
        self.nodes = []
        self.edges = {}
        self.distances = {}
    def getNodes(self):
        return self.nodes
    def getEdges(self):
        return self.edges
    def addNode(self, node):
        if node in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.append(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(dest)
    def addWeightedEdge(self, edge, weight = 1):
        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(dest)
self.distances[(src, dest)] = weight
def childrenOf(self, node):
    return self.edges[node]
def hasNode(self, node):
    return node in self.nodes
def __str__(self):
    res = ''
    for k in self.edges:
        for d in self.edges[k]:
            res = res + str(k) + '->' + str(d) + '\n'
    return res[:-1]
class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
    def addWeightedEdge(self, edge, weight):
        Digraph.addWeightedEdge(self, edge, weight)
        rev = WeightedEdge(edge.getDestination(), edge.getSource())
        Digraph.addWeightedEdge(self, rev, weight)
    def printPath(path):
        # a path is a list of nodes
        result = ''
        for i in range(len(path)):
            if i == len(path) - 1:
                result = result + str(path[i])
            else:
                result = result + str(path[i]) + '->'
        print result
    def dijkstra(graph, source):
        distances = {source: 0}
        parents = {}
        nodes = set(graph.nodes)
        while nodes:
            smallest = None
            for node in nodes:
                if node in distances:
                    if smallest is None:
                        smallest = node
                    elif distances[node] < distances[smallest]:
                        smallest = node
                if smallest is None:
                    break
            nodes.remove(smallest)
            weight = distances[smallest]
            for node in graph.edges[smallest]:
                newWeight = weight + graph.distances[(smallest, node)]
                if node not in distances or newWeight < distances[node]:
                    distances[node] = newWeight
                    parents[node] = smallest
        return distances, parents