Q1. Suppose you have a weighted directed graph and want to find a path between nodes Y and Z with the smallest total weight. Which of the following are True? Check all that apply.

- If the graph is acyclic and some edges have negative weights, depth-first search finds a correct solution. (True)
- If all edges have the SAME positive weight, depth-first search guarantees that the first path found is the shortest path. (False)
- If the graph is acyclic and some edges have negative weights, the first path found by breadth-first search is the correct solution. (False)
- If all edges in a graph have the SAME positive weight, the first path found by breadth-first search is the correct solution. (True)

Q2. A Monte Carlo simulation with 1000 trials estimates the value of a constant K. The mean estimate of K is 5, and the standard deviation of the 1000 estimates is 1. Which of the following can be concluded given this information? Check all that apply.

- If the simulation were run again, with a probability greater than 0.9 the estimate of K would be between 3.04 and 6.96. (True)
- With a probability greater than 0.9, the true value of K is between 3.04 and 6.96. (False)
- All values between 3.04 and 6.96 are equally likely. (False)
- If we run the Monte Carlo simulation with 2000 trials, the standard deviation of the 2000 estimates is less than 1 with high probability. (False)

Q3. The following questions all assume sampling is done with replacement. Which of the following are True? Check all that apply.

- Given a random sample, decreasing the width of the confidence interval always increases the confidence level. (False)
- A random sample from a population will never have the same mean as the actual population. (False)
- Assume you have a population of size N. You take one sample of size n. As n increases, the standard error decreases. (True)
- Assume you have a population of size N. You take M random samples of size n. As n grows, the distribution of the means of the M samples approaches a normal distribution. (True)

Q4: Which of the following are true? Check all that apply.

- Assume that there are 2*k examples in the data. You train a model and find the mean squared error on the same data. If the mean squared error for a polynomial of degree k is 0 then the mean squared error for a polynomial of degree k+1 might be positive. (False)
- If you fit a model to some data, it is possible that the model captures experimental error in the data. (True)
- You separate data into training and test sets. You train a model on the training set. The mean squared error of the model on the training data will always be smaller than the mean squared error of the same model on the testing data. (False)

Q5: Write a function that meets the following specification. Only use libraries as imported for you below.

```
import random
def prob_ordered(L, n):
    """ L is a non-empty list of numbers
        n is a positive int 0 < n <=len(L)
    Returns an estimate of the probability that if n elements are chosen
    at random (with replacement) from L, the n elements are chosen in STRICTLY
    increasing order. Use a Monte Carlo simulation with 100000 trials. """
    ## ANSWER
    success = 0
    ntrials = 100000
    for i in range(ntrials):
        inorder = True
        old = random.choice(L)
        for i in range(1,n):
            draw = random.choice(L)
            if draw <= old:
                inorder = False
            old = draw
        if inorder:
            success += 1
    return success/ntrials

# For example
L = [0,1,2,3,4,5,6,7,8,9]
n = 4
prob_ordered(L, n)  # prints 0.02

L = [12,17,3]
n = 1
prob_ordered(L, n)  # prints 1.0

L = [5,5,5]
n = 2
prob_ordered(L, n)  # prints 0.0
```

Q6: Write a function that meets the following specification. You may not import any libraries.

```
def to_take(d, n_allowed):
    """ d is a dict of coins that maps an int (representing the value of a coin)
           to a positive int (representing how many coins of that value are available)
       n_allowed is a non-negative int, representing the total number of coins to take
    Returns the maximum total value of taking n_allowed coins from d.
    You may mutate d and you can use brute force. """
    ## ANSWER
    flattened_items = []
    for k, count in d.items():
        flattened_items += [k for _ in range(count)]

    def to_take_helper(d, n_allowed):
        if len(d) == 0 or n_allowed == 0:
            return []
        # Take first item
        result_with = [d[0]] + to_take_helper(d[1:], n_allowed - 1)
        # Do not take first item
        result_without = to_take_helper(d[1:], n_allowed)
        if result_with == []:
            return result_without
        elif result_without == []:
            return result_with
        else:
            if sum(result_with) > sum(result_without):
                return result_with
            else:
                return result_without

    items_selected = to_take_helper(flattened_items, n_allowed)
    return sum(items_selected)

# For example:
print(to_take({1:2, 3:1, 4:2}, 0)) # prints 0
print(to_take({1:2, 4:2, 3:1}, 3)) # prints 11
```

Q7: Write a function that meets the following specification. You may only use libraries as imported for you below.

```
import numpy as np

## GIVEN THIS FUNCTION -- DO NOT MODIFY
def rSquared(pred, meas):
     pred = np.array(pred)
     meas = np.array(meas)
     est_err = ((pred - meas)**2).sum()
     meas_mean = meas.sum()/len(meas)
     var = ((meas - meas_mean)**2).sum()
     return 1 - est_err/var

## WRITE THIS FUNCTION
def find_fit(exp1, exp2, degrees):
    """ * degrees is a non-empty list of positive integers.
        * exp1 and exp2 represent experiments used to
                 try and understand the same process.
          -exp1, and exp2 are tuples of length 2.
          -Each element of each tuple is a list of floats.
          -All of the lists are of the same length. The first list in each
           tuple contains the independent values of an experiment. The second
           list in each tuple contains the corresponding dependent values.

        Returns the degree (chosen from degrees) of the polynomial that best
        characterizes the single process used to generate the relationship between
        the independent values and the dependent values in the experiments,
        based on linear regression and the least squares objective function. """
    ## ANSWER
    degrees.sort()
    rSquared_vals = dict()
    for d in degrees:
        model_1 = np.polyfit(exp1[0], exp1[1], d)
        model_2 = np.polyfit(exp2[0], exp2[1], d)
        tot_test_r2 = rSquared(np.polyval(model_1, exp2[0]), exp2[1])
        tot_test_r2 += rSquared(np.polyval(model_2, exp1[0]), exp1[1])
        rSquared_vals[d] = tot_test_r2/2
    best_degree = degrees[0]
    best_val = rSquared_vals[degrees[0]]
    for d in rSquared_vals:
        if rSquared_vals[d] > best_val:
            best_degree = d
            best_val = rSquared_vals[d]
    return best_degree

# For example:
exp1 = ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19], [0, 1,
15, 89, 279, 725, 1428, 2547, 3971, 6079, 10749, 14723, 19325, 28803, 35720, 48941,
73815, 87986, 90323, 137742])

exp2 = ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19], [8, 3,
23, 83, 259, 623, 1292, 2400, 4096, 6556, 9999, 14636, 20743, 28555, 38422, 50624,
65527, 83517, 104980, 130319])

degrees = [5, 6, 4, 3]

print(find_fit(exp1, exp2, degrees))  # prints 4
```

Q8: Consider the following class, Example and the code for findKNearest (same as the one in class). Write the function KNearestLabel according to the specifications below. You may not import any libraries.

```
# YOU ARE GIVEN THIS CLASS
class Example(object):
    def __init__(self, name, feature_vec, label = None):
        self.name = str(name)
        self.feature_vec = feature_vec
        self.label = label
    def distance(self, other):
        dist = 0
        for i in range(len(self.feature_vec)):
            dist += abs(self.feature_vec[i] - other.feature_vec[i])**2
        return dist**0.5
    def get_label(self):
        return self.label
    def __str__(self):
        return self.name + ':' + str(self.feature_vec) + ':' + str(self.label)

# YOU ARE GIVEN THIS FUNCTION, SAME AS IN CLASS
def findKNearest(example, exampleSet, k):
    kNearest, distances = [], []
    #Build lists containing first k examples, and their distances
    for i in range(k):
        kNearest.append(exampleSet[i])
        distances.append(example.distance(exampleSet[i]))
    maxDist = max(distances) #Get maximum distance
    #Look at examples not yet considered
    for e in exampleSet[k:]:
        dist = e.distance(example)
        if dist < maxDist:
            maxIndex = distances.index(maxDist)
            kNearest[maxIndex] = e
            distances[maxIndex] = dist
            maxDist = max(distances)
    return kNearest, distances

# IMPLEMENT THIS FUNCTION
def KNearestLabel(training, example, k):
    """ training is a list of elements of type Example
        example is a value of type Example
        k is an odd int > 0

    Uses k-nearest neighbors (with Euclidean distance) to assign a label
    of True or False for `example` based on the training data in `training`.
    Returns a Boolean representing the predicted label of `example`. """
    ## ANSWER
    nearest, distances = findKNearest(example, training, k)
    true_count, false_count = 0.0, 0.0
    for i in range(len(nearest)):
        if nearest[i].get_label() == True:
            true_count += 1
        else:
            false_count += 1
    if true_count > false_count:
        return True
    elif false_count > true_count:
```

```
        return False
    else:
        raise ValueError

# For example:
ex1 = Example('ex1', [1], True)
ex2 = Example('ex2', [1], True)
ex3 = Example('ex3', [1], True)
ex4 = Example('ex4', [0], False)
examples = [ex1, ex2, ex3, ex4]

ex0 = Example('test', [0.01])
print(KNearestLabel(examples, ex0, 3))  # prints True
print(KNearestLabel(examples, ex0, 1))  # prints False
```