Quiz 2

Name:

Grade:

Region:

Please answer the following questions.

Problem 1

Write a function hasSolution1(a, b) which returns `True` if `ax = b` has a unique solution for `x`, and otherwise returns `False`

In [ ]:

```
def hasSolution1(a,b):
    # write your code here
```

Problem 2

Describe what is wrong with the following code? Also change the code below by replacing `# HERE` with something else to make it work. The code you add there should not use + at all.

In [ ]:

```
# returns the sum x+y, where x and y are both greater than or equal to 0
def add(x, y):
    # HERE
    return 1 + add(x, y-1)
```

What was wrong?:

write your answer here

Problem 3

Use recursion to implement a function `multiplyAll` which takes as input a list `L` of integers and returns the product of all elements of `L`.

In [ ]:

```
def multiplyAll(L):
    # write your code here
```

Examples:

In [4]:

```
multiplyAll([1,2,3])
```

Out[4]:

```
6
```

```
In [5]: multiplyAll([4, 0, 2])

Out[5]: 0

In [6]: multiplyAll([1, 1, 1, 1, 1])

Out[6]: 1
```

Problem 4

Write a function `isSorted` which takes as input a list `L` of integers and returns `True` if `L` is sorted and `False` if it is not sorted.

```
In [7]: def isSorted(L):
            # write your code here

          File "<ipython-input-7-126383529f06>", line 2
            # write your code here
                                  ^
        SyntaxError: unexpected EOF while parsing
```

Examples:

```
In [10]: isSorted([1,2,3,4,5])

Out[10]: True

In [11]: isSorted([1, 2, 3, 5, 4])

Out[11]: False
```

Problem 5

Suppose you are given an implementation of `sort10` as below, which sorts an input list of 10 numbers. Use it to implement `sort11`, which sorts a list of 11 numbers. Use the selection sort algorithm. Your code should not use the built-in `sorted()` function in `sort11`.

```
In [ ]: def sort10(L):
            return sorted(L)

        def sort11(L):
            # write your code here, using sort10
```

Examples

```
In [13]: sort11([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

Out[13]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

In [14]: sort11([1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 10])

Out[14]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12]
```

Problem 6

Suppose you are given implementations of `sort10` and `merge_lists` as below. `sort10` sorts an input list of 10 numbers. `merge_lists` takes two sorted lists `L` and `R` as input and outputs the merged sorted list containing all the elements of both `L` and `R`. Use `sort10` and `merge_lists` to implement `sort20`, which sorts a list of 20 numbers. Use the merge sort algorithm. Your code should not use the built-in `sorted()` function in `sort20`.

```
In [ ]: def sort10(L):
            return sorted(L)

        def merge_lists(L, R):
            return sorted(L + R)

        def sort20(L):
            # write your code here, using sort10 and merge_lists
```

Examples

```
 In [17]: sort20([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
Out[17]: [1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10]
```

```
 In [18]: sort20([1, 2, 1, 2, 3, 4, 3, 4, 5, 6, 5, 6, 7, 8, 7, 8, 9, 10, 10, 9])
Out[18]: [1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10]
```

Problem 7

You are given the function `positive_solve4(eqs)` which solves systems of 4 equations with 4 variables, but **only if all the coefficients of the variables are nonnegative**. On input a list `eqs` of 4 equations (each of which is a list of 5 coefficients), `positive_solve4(eqs)` will return the same value as `solve(eqs)` (i.e., a list of length 4 of the solutions) if all coefficients of variables are non-negative and return `None` otherwise. (It's OK if the constant coefficients are negative.)

You need to implement the function `solve2(a,b,c,d,e,d)` which will solve 2 equations with 2 variables (with potentially negative coefficients) of the form $ax + by + c = 0, cx + dy + e = 0$ using a call to `positive_solve4(eqs)`.

```
In [20]: ## Helper function: positive_solve4
         from __future__ import division
         import numpy as np

         def positive_solve4(eqs):
             for eq in eqs:
                 if eq[0]<0 or eq[1]<0 or eq[2]<0 or eq[3]<0:
                         return None
             A = np.ndarray([4,4])
             for i in range(4):
                 for j in range(4):
                     A[i,j] = eqs[i][j]
             b = np.ndarray([4,1])
             for i in range(4):
                 b[i,0] = -eqs[i][4]
             C = np.linalg.inv(A)
             sol = np.dot(C,b)
             return [round(sol[i,0],3) for i in range(4)]
```

```
In [25]: def solve2(a,b,c,d,e,f): # solve ax + by + c = 0 , dx + ey + f = 0
             eqs = []

             # YOUR CODE HERE

             res = positive_solve4(eqs)

             # YOUR CODE HERE TO COMPUTE RETURN VALUE
```

```
In [24]: #staff solution - should delete!!!!
         def pair(a):
             return [a if a>= 0 else 0, -a if a< 0 else 0]

         def solve2(a,b,c,d,e,f): # solve ax + by + c = 0 , dx + ey + f = 0
             eqs = [ [1,1,0,0,0], [0,0,1,1,0]]
             eqs.append(pair(a)+pair(b)+[c])
             eqs.append(pair(d)+pair(e)+[f])
             res = positive_solve4(eqs)
             return [res[0], res[2]]
```

Examples

```
In [25]: solve2(1,1,-10,1,-1,-4)

Out[25]: [7.0, 3.0]
```

Problem 8

Suppose n is about one billion (1,000,000,000). Which of these numbers is closest to the number of steps that solve will make on a system of n equations with n variables? Also please explain your solution. We have included the code for `solve` below for your convenience.

A. n

B. n^2

C. n^3

D. n^4

Answer:

Justification for answer:

In [ ]:

```
from __future__ import division

def solve(eqs):
    n = len(eqs)
    make_first_coeff_nonzero_general(eqs)  # make 1st coef of 1st equation nonzero

    eqs[0] = multiply_equation(eqs[0],1/eqs[0][0])
    # make 1st coef of 1st  equation equal 1

    for i in range(1,n-1):
        eqs[i] = add_equations(eqs[i],multiply_equation(eqs[0],-eqs[i][0])) # zero
out first coefficient in eqs 1,2
    # make 1st coef of 2nd .. n-th equation equal zero

    rest_equations = []
    for i in range(1,n):
        rest_equations.append(eqs[i][1:n+1])

    solutions = solve(rest_equations)
    # solve remainder of  equations for remainder of  variables

    x =  - eqs[0][n]
    for i in range(1,n):
        x -= eqs[0][i]*solutions[i-1]
    # solve 1st variable using solution for 2nd and 3rd variable

    return [x] + solutions
```