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 t
o 0
def add(x, y):
    # HERE
    return 1 + add(x, y-1)
```

What was wrong?:

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
```

In [ ]:
```
# examples
#
print multiplyAll([1,2,3])
# should print 6

print multiplyAll([4, 0, 2])
# should print 0

print multiplyAll([1, 1, 1, 1, 1])
# should print 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 [ ]:
```
def isSorted(L):
    # write your code here
```

In [ ]:
```
# examples
#
print isSorted([1,2,3,4,5])
# should print True

print isSorted([1, 2, 3, 5, 4])
# should print 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
```

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

print sort11([1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 10])
# should print [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
```

In [ ]:
```
# examples
#
print sort20([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8,
9, 10])
# should print [1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9,
 10, 10]

print sort20([1, 2, 1, 2, 3, 4, 3, 4, 5, 6, 5, 6, 7, 8, 7, 8, 9, 10,
10, 9])
# should print [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 are nonnegative**. The return value is a list of length 4, with the values of `x,y,z,w` in the solution. If you `positive_solve4` is given any equation with a negative coefficient, it fails and just outputs `None`. `eqs` here is a list of 4 lists, and each of those lists contains 5 numbers. For example, if the equations are

```
ax + by + cz + dw + e = 0

fx + gy + hz + iw + j = 0

kx + ly + mz + nw + o = 0

px + qy + rz + sw + t = 0
```

then `eqs` will be `[ [a,b,c,d,e], [f,g,h,i,j], [k,l,m,n,o], [p,q,r,s,t] ]`. `positive_solve4` will only solve the system if equations if `a,b,c,d,f,g,h,i,k,l,m,n,l,p,q,r,s` are all at least 0. That is, the coefficients need to be nonnegative. It is OK if `e`, `j`, `o`, or `t` are negative.

Use `positive_solve4` to implement `solve2`. You should not implement `solve2` from scratch. Instead, `solve2` should transform its input into a new system of equations with twice as many variables and all nonnegative coefficients such that once you receive the `return` value from `positive_solve4`, you should be able to easily convert that into the solutions for `x,y` in `solve2`.

`solve2(a,b,c,d,e,f)` should give the solution to the system of equations

```
ax + by + c = 0

dx + ey + f = 0
```

In [25]:
```
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)]

def solve2(a,b,c,d,e,f): # solve ax + by + c = 0 , dx + ey + f = 0
    eqs = []
    eqs += [ [1, 1, 0, 0, 0] ] # x + x' = 0
    eqs += [ [0, 0, 1, 1, 0] ] # y + y' = 0
    eq1 = [0, 0, 0, 0, c]

    if a >= 0:
        eq1[0] = a
    else:
        eq1[1] = -a
    if b >= 0:
        eq1[2] = b
    else:
        eq1[3] = -b

    eqs += [eq1]

    eq2 = [0, 0, 0, 0, f]

    if d >= 0:
        eq2[0] = d
    else:
        eq2[1] = -d
    if e >= 0:
        eq2[2] = e
    else:
        eq2[3] = -e

    eqs += [eq2]

    res = positive_solve4(eqs)
    # YOU SHOULD FIGURE OUT WHAT TO RETURN BELOW
    return [res[0], res[2]]
```

In [26]:
```
# examples
#
# the example from above:
print solve2(1,1,-10,1,-1,-4)
# should print [7.0, 3.0]
```

```
[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 equ
ation 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
```