fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

Midterm2 version B

You **MUST** stop writing and hand in this **entire** exam when instructed in lecture.

- You may not unstaple any pages.
- Failure to hand in an intact exam will be considered cheating. Discussing the exam with anyone in any way, even briefly, is cheating. (You may discuss it only once the exam has been posted to the course website.)
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper exam, and we will not grade it.
- You may not ask questions during the exam, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in weeks 1-13.
- We may test your code using additional test cases.
- Assume almostEqual(x, y) and rounded(n) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

CT1: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
import copy
def ct1(L, M):
    N = copy.deepcopy(L)
    xl = L.pop(0)
    xm = M.pop(0)
    xn = N.pop(0)
    L.append(xn)
    M[0].append(xm)
    N.append(xn[0])
    return N
L = [[4],[3]]
M = copy.copy(L)
N = ct1(L, M)
print(L)
print(M)
print(N)
```

CT2: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct2(d):
    s = set([k for k in d])
    t = set([d[k] for k in d])
    L = sorted(s - t)
    M = sorted(t - s)
    e = dict()
    for i in range(len(L)):
        e[L[i]] = i
    M.append(e)
    return M
print(ct2({1:3, 2:4, 4:3}))
```

CT3: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct3(L, d=1):
    if L == [ ]:
        return [ ]
    else:
        i = len(L)//2
        A = L[:i]
        B = [(L[i], d)]
        C = L[i+1:]
        return B + ct3(C, d+1) + ct3(A, d+1)
print(ct3([1,2,3,4]))
```

CT4: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
class A:
    def __init__(self, x, y=0):
        self.x = x
        self.y = y
    def __eq__(self, other):
        return self.f() == other.f()
    def f(self):
        return sorted([self.x, self.y])

def ct4(L):
    M = [ ]
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            M.append(int(L[i] == L[j]))
    return M

L = [ A(0, 1), A(0), A(1)]
print(ct4(L))
```

Short Answer (7pts total, 1pt ea.)

Write your answer as an integer on the line provided.

SA1._____ How many passes through the list will selectionSort require to sort a list containing 1024 values?

SA2._____ How many passes through the list will mergeSort require to sort a list containing 1024 values?

SA3. What is the worst-case big-o runtime of each of the following, assuming L is a list containing N integers? Write your answer on each line. (1pt ea.)

3a. _____ `len(L)`

3b. _____ `set(sorted(L))`

3c. _____ `L.append(1)`

3d. _____ `L.insert(0, 1)`

3e. _____ `[L+[i] for i in range(len(L))]`

Multiple Choice (4pts total, 2pts ea.)

MC1. Which of the following best describes how we stored the position of the snake in the Snake case study? (fill in one circle).

○ a) As a string containing the position of each snake piece using (row, col)

○ b) As a list of tuples, where each tuple contains the position of a snake piece using (row, col) indices.

○ c) As a 2d rectangular list of booleans, where True indicates that a snake piece exists in that cell.

○ d) As a 1d list of integers, where each integer encodes the row and col indices of a snake piece using % and //

MC2. Which one (and only one) of the following statements is True? (fill in one circle).

○ a) Sets can contain sets

○ b) Sets can contain lists

○ c) Sets and dictionaries are mutable

○ d) Keys in a dictionary must be mutable

○ e) Values in a dictionary must be mutable

Free Response 1: Efficiency [16pts]

Given the function f(L) below which takes a list L containing N integers, write the function g(L) such that:

- g(L) works identically to f(L) for any list L with N integers
- g(L) runs in **O(N)** time.

```
def f(L):
    if L == [ ]: return None
    M = [ ]
    for v in L:
        if v not in M:
            M.append(v)
    N = sorted([L.count(v) for v in M])
    return N[-1]
```

Begin your FR1 answer here or on the following page

You may begin or continue your FR1 answer here:

Free Response 2: Sum class [16pts]

Write the Sum class so that the following test function passes. Do not hardcode to the test cases, and use OOP properly.

```
def testSumClass():
    print('Testing Sum class...', end='')
    # start with a sum of 0:
    s1 = Sum()
    assert(str(s1) == 'Sum(0)')
    assert(str([s1]) == '[Sum(0)]')

    # add values to the sum:
    s1.add(1)
    assert(str(s1) == 'Sum(1)')
    s1.add(2)
    assert(str(s1) == 'Sum(3)') # 1+2

    # we can add a whole list at once:
    s1.add([3,4])
    assert(str(s1) == 'Sum(10)') # 1+2+3+4

    s2 = Sum()
    assert(s1 != s2)
    s2.add(10)
    assert(s1 == s2)
    assert(s1 != 'do not crash here')

    s = set()
    assert(s1 not in s)
    s.add(s1)
    assert(s2 in s) # note that s1 == s2

    s3 = s1.combine(s2)
    assert(str(s1) == 'Sum(10)')
    assert(str(s2) == 'Sum(10)')
    assert(str(s3) == 'Sum(20)')
```

Begin your FR2 answer on the following page

Begin your FR2 answer here:

You may continue your FR2 answer here, if needed:

Free Response 3: Recursion [16pts]

Without using for or while loops, and without using list comprehensions, and using recursion properly, write the function f(x, y) that takes integers x and y and returns a list of the multiples of 10 that lie between x and y inclusively. Note that x can be less than, equal to, or greater than y. For example:

```
assert(f(5, 21) == [10, 20])
assert(f(21, 5) == [10, 20])
assert(f(20, 5) == [10, 20])
assert(f(19, 5) == [10])
assert(f(10, 10) == [10])
assert(f(1, 9) == [ ])
```

Note that you may use a wrapper function or default parameters in your solution if you wish, as long as f(x, y) still works as required.

Begin your FR3 answer here or on the following page

You may begin or continue your FR3 answer here:

You may continue your FR3 answer here, if needed:

Free Response 4: solveSumOrProductPuzzle(L) [17pts]

Background: in the "Sum or Product Puzzle" (a coined term), you are given a list of integers L, containing at least 3 values, and you solve the puzzle by arranging those values so that each value (except the first two) is either the sum or the product of the previous two values.

For example, consider this list:
`L = [ -15, -10, -5, 2 ]`
We can rearrange the values like so:
`M = [ 2, -5, -10, -15 ]`
The list M is a solution to the puzzle because:
`-10 == 2 * -5`, and
`-15 == -5 + -10`
With this in mind, write the function solveSumOrProductPuzzle(L) that takes a list L with at least 3 integers, and (without modifying L) returns a list M that arranges the values in L to solve the Sum or Product Puzzle, or None if there is no solution.

Note: this is a backtracking problem. To receive full credit, you must use backtracking properly (even if there is another way to solve the problem). Also, in particular, you cannot generate every possible permutation of L and then test to see if each permutation solves the problem. Also, the elements of L are not guaranted to be unique.

Also, be sure to not ever mutate L.

```
def testSolveSumOrProductPuzzle():
    print('Testing solveSumOrProductPuzzle()...', end='')
    L = [ -15, -10, -5, 2 ]
    M = [ 2, -5, -10, -15 ]
    L0 = copy.copy(L)
    assert(solveSumOrProductPuzzle(L) == M)
    assert(L == L0) # check for mutation

    L = [ -15, -10, -5, 2, 22 ]
    L0 = copy.copy(L)
    assert(solveSumOrProductPuzzle(L) == None)
    assert(L == L0) # check for mutation

    L = [-8, -5, -3, 16, 24]
    M = [-5, -3, -8, 24, 16]
    L0 = copy.copy(L)
    assert(solveSumOrProductPuzzle(L) == M)
    assert(L == L0) # check for mutation
```

Begin your FR4 answer on the following page:

Begin your FR4 answer here:

Continue your FR4 answer here:

Continue your FR4 answer here:

The problems below are not required. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below.

bonusCt1 [optional, 1pt]

```
def bonusCt1(n):
    def f(n): return 0 if n==0 else f(n-1) + 2*n - 1
    def g(n): return 0 if n==0 else g(n-1) + 3*f(n) - 3*n + 1
    return (g(n) - f(n))//f(n)
print(bonusCt1(123))
```

bonusCt2 [optional, 1pt]

```
def bonusCt2():
    L = [ ]
    def g(x, y): return g(y,x%y) if y else x==1
    def h(x):
        for v in L:
            if not g(x, v): return
        L.append(x)
    for x in range(2,12345):
        h(x)
        if sum(L[-2:]) > 42:
            return L[-1]
print(bonusCt2())
```