fullName:_____andrewID:_____
recitationLetter:_____ Lecture (1, 2, or 3)_____

Midterm2 version A

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

- You may not unstaple any pages.
- 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 this in with your paper exam, and we will not grade it.
- 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 not use any concepts we have not covered in the notes this semester. We may test your code using additional test cases. Do not hardcode. Assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you. You must write all other helper functions you wish to use.
- The entire exam is to be taken on paper. No computers, calculators, or additional resources are allowed.

Exam Outline:

- 4 MCs [4pts total]
- 8 SAs [16pts total]
- 4 CTs [20pts total]
- 3 FRs [60pts total]

Do not open this exam until instructed. Doing so will be considered cheating.

Multiple Choice [1pt ea, 4pts total]

Clearly place an X in the blank next to the letter of each correct answer for the following 4 questions.

MC1: Which of the following is the worst-case big-oh runtime of linear search?
____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC2: Which of the following is the worst-case big-oh runtime of binary search?
____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC3: Which of the following is the worst-case big-oh runtime of selection sort?
____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC4: Which of the following is the worst-case big-oh runtime of merge sort?
____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

Short Answer [2pts ea, 16pts total]

Clearly write your answer in the blank for the following 8 questions. For simplicity, assume that 2**10 == 1000. Your answers should be numbers rather than mathematical expressions (e.g. if an answer is approximately equal to log(1000), write 10 instead of log(1000). Do not write log as part of your answers.)

SA1: What is the maximum number of values linear search must check to find a value in a list with 4 million values?

_____

SA2: What is the maximum number of values binary search must check to find a value in a list with 4 million values?

_____

SA3: How many passes will selection sort require to sort a list of 2000 values?

_____

SA4: How many passes will merge sort require to sort a list of 2000 values?

_____

SA5: If on a given computer, selection sort takes 2 seconds to a sort a list with N values, **roughly** how long will it take on the same computer to sort a list with 8N values?

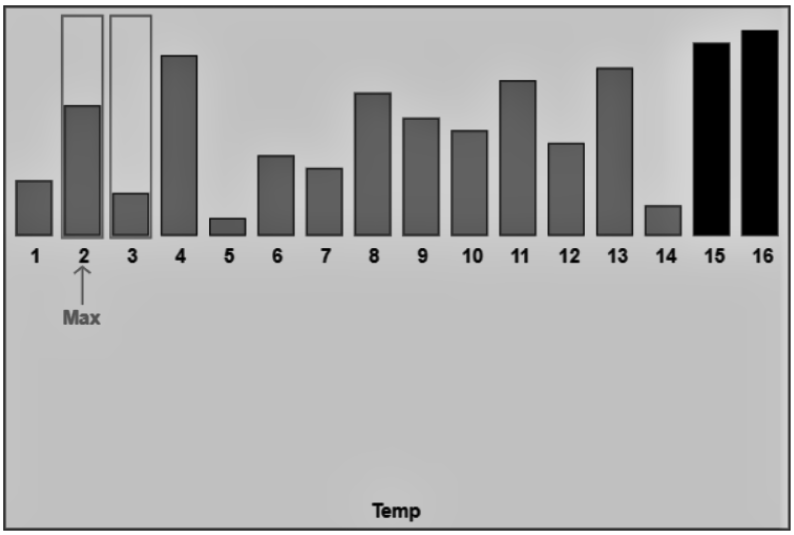
_____

SA6: If on a given computer, merge sort takes 2 seconds to a sort a list with N values, **roughly** how long will it take on the same computer to sort a list with 8N values?

_____

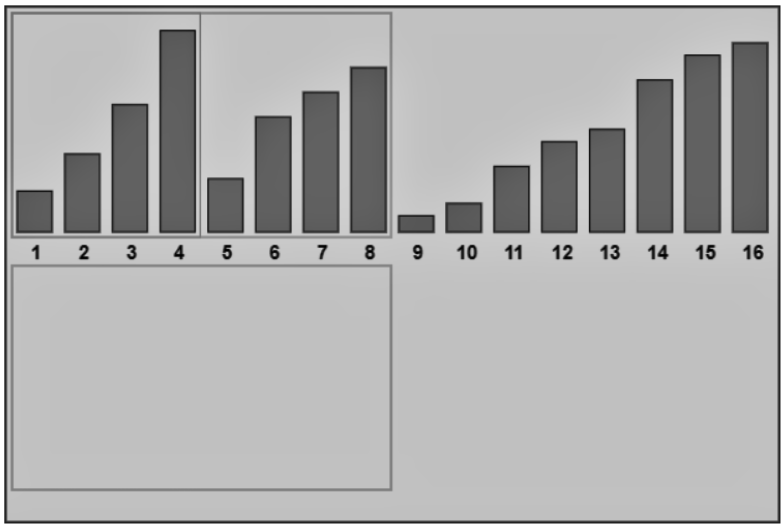
SA7: In the picture below of xSortLab running selection sort, what are the indexes of the next two values to be swapped? (Use xSortLab's indices, which begin at 1 rather than 0)

_____



SA8:In this picture below of xSortLab running merge sort, what is the index of the next value to be copied? (Use xSortLab's indices, which begin at 1 rather than 0)

_____



CT1: Code Tracing [5pts]

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

```
#This prints a list of length 4
def ct1(L):
    d = dict()
    for v in sorted(set(L)):
        k = (v**2) % 10
        d[v] = k
        d[k] = v**2
    return sorted([d[v] for v in d])
print(ct1([2,4,8,4]))
```

[_____, _____, _____, _____]

CT2: Code Tracing [5pts]

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

```
#This prints a list of length 3
class C:
    z = 1
    def __init__(self, x):
        self.x = x + C.z
        C.z += 1
    def f(self, other):
        if other.x >= self.x:
            self.x = other.x
        else:
            self.x = -self.x

def ct2(L):
    L = [C(x) for x in L]
    for u in L:
        for v in L:
            u.f(v)
    return [v.x for v in L]

print(ct2([2,5,1]))
```

[_____, _____, _____]

CT3: Code Tracing [5pts]

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

```
#This prints a list of length 3
def ct3(s):
    if (s == ''):
        return [ ]
    else:
        return [int(s[0]+s[-1])] + ct3(s[1:-1])
print(ct3('12345'))
```

[_____, _____, _____]

CT4: Code Tracing [5pts]

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

```
#This prints a list of length 5
def ct4(L, depth=0):
    if (L == [ ]):
        return [(42, depth)]
    if (len(L) < 2):
        return [(L[0], depth)]
    else:
        i = len(L)//2
        return [(L[i], depth)] + ct4(L[:i], depth+1) + ct4(L[i+1:], depth+1)
print(ct4([2,4,6,8]))
```

[_____, _____, _____, _____, _____]

Free Response 1: repeats [20pts]

Write the function repeats(d) that takes a dictionary d and returns a set of all the values that occur more than once, whether as a key or as a value in d.

You may assume that all the keys and values in d are all integers.

```
def testRepeats():
    assert(repeats({1:2, 2:3, 4:5, 6:7, 7:3, 8:8}) == {2, 3, 7, 8})
    assert(repeats({}) == set())
```

Free Response 2: DogNames class [20pts]

Write the DogNames class so that the following test function passes.

To receive credit, your code must not hardcode the test cases but must work in general, and must follow standard OOP practices.

```
def testDogNamesClass():
    print('Testing DogNames class...', end='')
    dn1 = DogNames('Spot,Rex,Ace') # a string, not a list
    assert(dn1.names == ['Spot', 'Rex', 'Ace']) # a list, not a string
    assert(str(dn1) == "DogNames(names=Ace and Rex and Spot)") # sorted names!
    assert(dn1.names == ['Spot', 'Rex', 'Ace']) # still not sorted

    dn2 = DogNames('Fido,Ace')
    assert(str(dn2) == f"DogNames(names=Ace and Fido)")
    assert(dn2.names == ['Fido', 'Ace'])

    dn3 = DogNames('Spot')
    assert(dn3.names == ['Spot'])
    assert(str(dn3) == f'DogNames(names=Spot)')

    assert(dn1.sharesSomeName(dn2) == True)  # Ace is in both
    assert(dn1.sharesSomeName(dn3) == True)  # Spot is in both
    assert(dn2.sharesSomeName(dn3) == False) # no name is in both

    # To merge names, use all the names in the first DogNames instance (self),
    # then add all the names in the other DogNames instance that are not
    # in the first list. Hint: .join() may be useful here:
    dn4 = dn1.merge(dn2)
    assert(isinstance(dn4, DogNames))
    assert(dn4.names == ['Spot', 'Rex', 'Ace', 'Fido'])

    print('Passed!')

testDogNamesClass()
```

(You may write your answer to FR2 on this page)

(You may write your answer to FR2 on this page)

(You may write your answer to FR2 on this page)

Free Response 3: makeSilly(L) [20pts]

Note: for this problem, you may use loops, but you must use backtracking properly (even if this could be solved some other way).
In particular, you must check the problem's constraints as you go and not just at the end of the recursion.

Background: given a list L of non-empty strings, we will say that L is "silly" (a coined term) if each consecutive pair of values in L do not share any common characters.

For example, ['ab', 'cd', 'ef', 'ac'] is silly, but ['ab', 'cd', 'ad', 'ef'] is not silly (since 'cd' and 'ad' share the letter 'd').

With this in mind, write the function makeSilly(L) that takes a list L of non-empty strings, and returns another list M with all the values in L only where M is silly, or returns None if no such list exists.

For example:
makeSilly(['ab', 'ac', 'de']) could return either ['ab', 'de', 'ac'] or ['ac', 'de', 'ab']

And makeSilly(['ab', 'ac', 'bc']) returns None.

(You may write your answer to FR3 on this page)

(You may write your answer to FR3 on this page)

bonusCT1 [2pts]

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

```
def bonusCt1(L):
    if 0 not in L:
        return L[-1]
    else:
        L.remove(0)
        return bonusCt1(L + [sum(L)])
print(bonusCt1(list(range(3))*5))
```

bonusCT2 [2pts]

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

```
def bonusCt2(m):
    def f(m): return 0 if m==0 else 2 + f(m//2)
    return m if m<10  else 1+bonusCt2(f(m))
print(bonusCt2(1024))
```