fullName:_____

andrewID:_____

recitationLetter:_____

15-112 F22

Midterm2 version A

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. We have provided a box at the beginning of the exam where you can write any assumptions that you would like us to consider. (This is entirely optional and will not likely impact your grade. We expect that most students will leave this box empty.)
- You may not use any concepts (including builtin functions or modules) 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.
- **Write your answers entirely inside the boxes!**

If you are unsure how to interpret a problem, take your best guess.
We have provided this box at the beginning of the exam where you can write any assumptions about specific problems that you would like us to consider. **This is entirely optional** and will not likely impact your grade. We expect that most students will leave this box empty.

Please clearly indicate the number of the problem followed by the assumption you are making. (For example, "FR4: I assume x and y will be greater than 0.")

Multiple Choice [4pts total]

MC1. What is the worst-case big-O runtime of merge sort, where n is the length of the list? Select the best answer (fill in one circle).

- ○ O(1)
- ○ O(nlogn)
- ○ O(n**0.5)
- ○ O(n**2)
- ○ O(logn)

MC2. What is the worst-case big-O runtime of selection sort, where n is the length of the list? Select the best answer (fill in one circle).

- ○ O(1)
- ○ O(nlogn)
- ○ O(n**0.5)
- ○ O(n**2)
- ○ O(logn)

MC3. If `L == [0, 3, 4, 5, 11, 13, 15, 19]`, at most, how many indices do we need to check with binary search in order to conclude that a number is NOT in L?
Select the best answer (fill in one circle).

- ○ 1
- ○ 4
- ○ 5
- ○ 8
- ○ 10

MC4. If `L == [11, 13, 15, 20, 4, 5, 7, 8]`, at most, how many indices do we need to check with linear search in order to conclude that a number is NOT in L?
Select the best answer (fill in one circle).

- ○ 1
- ○ 4
- ○ 5
- ○ 8
- ○ 10

True/False [6pts total]

Mark each of the following statements as True or False.
Where applicable, assume that N is very large.

1.	○ True	○ False	We can use binary search on an unsorted list
2.	○ True	○ False	We can use linear search on an unsorted list
3.	○ True	○ False	We can use binary search on a list with duplicates
4.	○ True	○ False	We can use linear search on a list with negative numbers
5.	○ True	○ False	Checking for membership in a set is O(1)
6.	○ True	○ False	Dictionaries and sets both use hashing
7.	○ True	○ False	Sets are immutable
8.	○ True	○ False	Sets can be dictionary keys
9.	○ True	○ False	O(2**N) is faster than O(N**2)
10.	○ True	○ False	O(1000) is faster than O(N/1000)
11.	○ True	○ False	O(N**0.5) is faster than O(N)
12.	○ True	○ False	O(N) is faster than O(logN)

CT1: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Hint: This prints two lines

```
class A(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def f(self):
        return self.x + self.y
    def g(self):
        self.x += 2
        return self.f()/10

class B(A):
    def g(self):
        return self.f()*10

def ct1(x):
    a = A(x, 2*x)
    print(a.g())
    b = B(x, a.x)
    print(b.g())

ct1(4)
```

CT2: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Hint: This prints one line

```
def ct2(L):
    if (len(L) == 0):
        return [ ]
    else:
        return [max(L)] + ct2(L[1:-1]) + [min(L)]
print(ct2([1,5,9,5,1]))
```

CT3: Code Tracing [8pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Hint: This prints two lines

```
def ct3(L):
    d = dict()
    print(f(L, d))
    return d

def f(L, d):
    if len(d) >= 3:
        return L
    elif len(L) <= 1:
        return False
    else:
        d[L[0]] = L[1]
        return f(L[1:], d)

print(ct3([15, 1, 1, 2, 'mid', 'two']))
```

You may use this page for ungraded scrap work

Free Response 1: findNewValues(dA, dB) [16 points]

Write the non-destructive (ie. non-mutating) function `findNewValues(dA, dB)` that takes two dictionaries, `dA` and `dB` and returns a set of the values in `dB` that are not present in `dA`. You may assume that the keys and values in both dictionaries are all strings. Here is an example:

```
dA = {'item1': 'a',
      'item2': 'b',
      'item3': 'c'}

dB = {'thing1': 'c',
      'thing2': 'b',
      'thing3': 'cat',
      'thing4': 'dog'}
```

`findNewValues(dA, dB)` returns `{'cat', 'dog'}` because 'cat' and 'dog' are values in `dB`, but not in `dA`. `findNewValues(dB, dA)` returns `{'a'}` because 'a' is a value in `dA`, but not in `dB`.

Efficiency: **Your solution must have an efficiency of O(N) or better**, where n is the total number of values in the two dictionaries. Slower solutions will receive half credit at most, as will destructive/mutating functions. Assume that both dictionaries are approximately the same size.

```
def testFindNewValues():
    dA = {'item1': 'a',
          'item2': 'b',
          'item3': 'c'}

    dB = {'thing1': 'c',
          'thing2': 'b',
          'thing3': 'cat',
          'thing4': 'dog'}

    assert(findNewValues(dA, dB) == {'cat', 'dog'})
    assert(findNewValues(dB, dA) == {'a'})

    dA = {'item1': 'thing1', 'item2': 'thing2', 'item3': 'thing3'}
    dB = {'thing1': 'item3', 'thing2': 'item2', 'thing3': 'item1'}
    assert(findNewValues(dA, dB) == {'item1', 'item2', 'item3'})
    assert(findNewValues(dB, dA) == {'thing1', 'thing2', 'thing3'})

    dA = {'item1': 'thing1', 'item2': 'thing2', 'item3': 'thing3',
          'thing1': 'item3', 'thing2': 'item2', 'thing3': 'item1'}
    dB = dict()
    assert(findNewValues(dA, dB) == set())
    assert(findNewValues(dB, dA) == {'thing1', 'thing2', 'thing3',
                                     'item1', 'item2', 'item3'})
```

Begin your FR1 answer here

Continue your FR1 answer here

Free Response 2: gapSum(n) [18 points]

Write the recursive function `gapSum(n)` that takes a (possibly negative) integer n and returns its "gap sum" (a made-up term).

The gap sum is found by finding the absolute difference between each neighboring pair of digits and adding those together. For example:

`assert(gapSum(564) == 3)`

Because `abs(5-6) + abs(6-4) == 1 + 2 == 3`

Likewise:

`assert(gapSum(-47325) == 11)`

Because `abs(4-7) + abs(7-3) + abs(3-2) + abs(2-5) == 3 + 4 + 1 + 3 == 11`

See the test cases for more examples.

You must solve this function recursively in order to receive any points. You may not use loops. Also, you may not use strings, lists, tuples, sets, or dictionaries.

```
def testGapSum():
    assert(gapSum(564) == 3)
    assert(gapSum(-47325) == 11)
    assert(gapSum(1246) == 5)
    assert(gapSum(-1246) == 5)
    assert(gapSum(5555) == 0)
    assert(gapSum(5050) == 15)
    assert(gapSum(0) == 0)
    assert(gapSum(9) == 0)
    assert(gapSum(-6) == 0)
testGapSum()
```

Begin your FR2 answer on the following page

Begin your FR2 answer here

You may continue your FR2 answer here

Free Response 3: Chapter and Book classes [18 points]

Write the classes Chapter and Book so that the following test code passes. Do not hardcode the test cases, but you may assume that the parameters are always legal (so, for example, chapter indexes are always in bounds). You must use OOP properly. Do not add unneccessary methods to either class.

```
def testBookAndChapterClasses():
    chapterA = Chapter('I love CS!', 30) # chapter title, # of pages
    assert(chapterA.getTitle() == 'I love CS!')
    chapterB = Chapter('So do I!', 15)
    book1 = Book('CS is Fun!', [chapterA, chapterB]) # book title, chapters
    book2 = Book('The Short Book', [ Chapter('Quick Read!', 5) ])
    assert(book1.getChapterCount() == 2)
    assert(book1.getPageCount() == 45)
    assert(book2.getChapterCount() == 1)
    assert(book2.getPageCount() == 5)
    assert(book1.getChapter(0).getTitle() == 'I love CS!')
    assert(book1.getChapter(1).getTitle() == 'So do I!')
    assert(book2.getChapter(0).getTitle() == 'Quick Read!')
    # Move chapter 0 from book1 to the end of book2
    # so moveChapter always moves to the end of the target book.
    book1.moveChapter(0, book2)
    assert(book1.getChapterCount() == 1)
    assert(book1.getPageCount() == 15)
    assert(book1.getChapter(0).getTitle() == 'So do I!')
    assert(book2.getChapterCount() == 2)
    assert(book2.getPageCount() == 35)
    assert(book2.getChapter(0).getTitle() == 'Quick Read!')
    assert(book2.getChapter(1).getTitle() == 'I love CS!')
testBookAndChapterClasses()
```

Begin your FR3 answer on the following page

Begin your FR3 answer here

You may continue your FR3 answer here

You may continue your FR3 answer here

Free Response 4: makeLegalString(s) [18 points]

Write the function makeLegalString(s) that takes a string of lowercase letters and returns a new string that obeys the two rules below, or None if no such string can be made.

1. A legal string t is some reordering of the letters in s -- so `sorted(t) == sorted(s)`
2. Neighboring letters in a legal string cannot be next to each other in the alphabet. (For example, 'b' cannot be next to 'a' or 'c')

Please note:

- Identical letters **are** allowed to be next to each other. For example, 'aaa' is legal.
- If multiple valid strings exist, you only need to return one of them.

This is a recursive backtracking problem. Solutions that do not properly use recursive backtracking will only be eligible for up to half credit. In addition to containing all the proper elements of the backtracking algorithm, this means you may not simply generate all possible combinations of the letters in s, and should instead build up a legal solution one letter at a time, checking for legality as you go.

You may only use one loop in your solution. Each additional loop will incur a -5pt deduction.

```
def testMakeLegalString():
    # Your function only needs to return one of the strings in each of these sets
    assert(makeLegalString('abcd') in {'bdac', 'cadb'})
    assert(makeLegalString('cabs') in {'bsac', 'acsb', 'bsca', 'casb'})
    assert(makeLegalString('aah') in {'aah', 'aha', 'haa'})
    assert(makeLegalString('higgs') in {'hsigg', 'hsgig', 'gigsh',
                                        'ggish', 'iggsh', 'hsggi'})
    assert(makeLegalString('x') in {'x'})
    assert(makeLegalString('') in {''})

    # These tests have no valid result string and should return None
    assert(makeLegalString('abba') == None)
    assert(makeLegalString('xyzzy') == None)
```

Begin your FR4 answer on the following page

Begin your FR4 answer here

You may continue your FR4 answer here

You may continue your FR4 answer here

bonusCT1: Code Tracing [1pt bonus]

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

```
def bonusCt1(n=2):
    if h(n):
        return f(n)
    return bonusCt1(n+1)

def f(n):
    if n < 2:
        return n
    return 2*n-1+f(n-1)

def g(n):
    if n < 2:
        return n
    return 2*g(n-1)

def h(n):
    return g(n)/f(n) == round(g(n)/f(n))

print(bonusCt1())
```

bonusCT2: Code Tracing [1pt bonus]

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

```
def bonusCt2(z):
    b, bd = 0, 1
    for q in [z/1000 for z in range(1000)]:
        qd = abs(g(q) - z)
        if (qd < bd):
            b, bd = q, qd
    return 2**2 * b

def f(x):
    if (x == 0):
        return 1
    return x*f(x-1)

def g(x):
    q, s = 0, 1
    for i in range(1, 21, 2):
        q, s = q + s * x**i / f(i), -s
    return q

print(bonusCt2(2**0.5/2))
```