15-112 Fall 2021 Midterm 2D
80 minutes

Name: _____

Andrew ID: _____@andrew.cmu.edu

Section: _____

- You may not use any books, notes, or electronic devices during this exam.
- You **may not ask questions** about the exam except for language clarifications.
- Show your answers and work **on the exam** (not scratch paper) to receive credit.
- If you do use scratch paper, you must submit it with your andrew id on it, and we will ignore it.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.
- Do not use imports we have not covered in class (e.g.. NumPy)
- You may use almostEqual() and roundHalfUp() without writing them. You must write everything else.
- **Do not unstaple the exam**

Do not write below here

Question	**Points**	**Score**
1. Short Answers 1-6	10	
2. Code Tracing 1	5	
3. Code Tracing 2	6	
4. Code Tracing 3	6	
5. Code Tracing 4	6	
6. Code Tracing 5	6	
7. Code Tracing 6	6	
8. FR1: Matrix Class	25	
9. FR2: RemoveEvens(L)	20	
10. FR3: Arrange(L)	10	
11. BonusCT 1+2	(+2 bonus)	
TOTAL	100	

Short Answer 1 [1 pt]:

L == [0, 3, 4, 5, 7, 9, 10, 12, 15, 20]

At most, how many indices do we need to check with binary search in order to conclude that a number is NOT in L?

◯ a) 1
◯ b) 4
◯ c) 5
◯ d) 10

Short Answer 2 [1 pt]:

L == [10, 12, 15, 20, 0, 3, 4, 5, 7, 9]

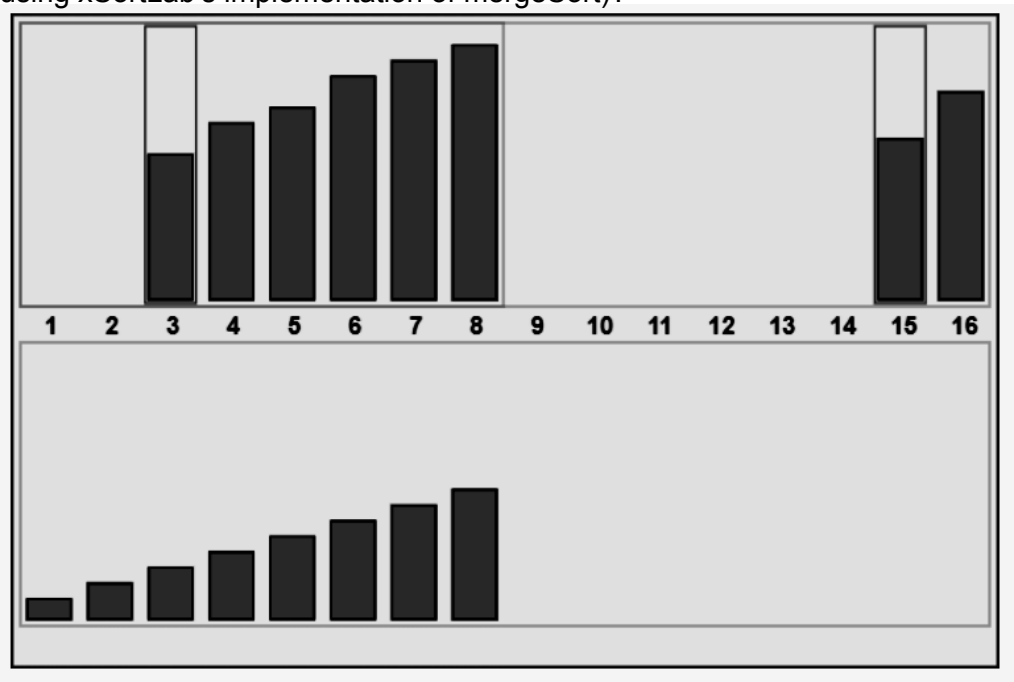
At most, how many indices do we need to check with linear search in order to conclude that a number is NOT in L?

◯ a) 1
◯ b) 4
◯ c) 5
◯ d) 10

Short Answer 3 [1 pts]:

For the xSortLab image below, using mergeSort, how many **passes** through the list have we **completed** so far (using xSortLab's implementation of mergeSort)?

◯ a) 0
◯ b) 1
◯ c) 2
◯ d) 3
◯ e) 5
◯ f) 6
◯ g) 8
◯ h) 16



Short Answer 4 [2 pts]:

Mark each of the following statements as True or False. Assume that N is very large.

☐**TRUE** ☐**FALSE** O(N) is faster than O(logN)

☐**TRUE** ☐**FALSE** O(N**0.5) is faster than O(N)

☐**TRUE** ☐**FALSE** O(2**N) is faster than O(N**2)

☐**TRUE** ☐**FALSE** O((N**2)*logN) is faster than O(N**2)

☐**TRUE** ☐**FALSE** O(100) is faster than O(N/100)

Short Answer 5 [2 pts]:

Mark each of the following statements as True or False

☐**TRUE** ☐**FALSE** Sets are immutable

☐**TRUE** ☐**FALSE** Checking for membership in a set is O(1)

☐**TRUE** ☐**FALSE** Sets are unordered

☐**TRUE** ☐**FALSE** Sets can be dictionary keys

☐**TRUE** ☐**FALSE** Dictionaries and sets both use hashing

Short Answer 6 [3 pts]:

Mark each of the following statements as True or False

☐**TRUE** ☐**FALSE** We can use binary search on a list with negative numbers

☐**TRUE** ☐**FALSE** We can use linear search on a list with negative numbers

☐**TRUE** ☐**FALSE** We can use binary search on a list with duplicates

☐**TRUE** ☐**FALSE** We can use linear search on a list with duplicates

☐**TRUE** ☐**FALSE** We can use binary search on an unsorted list

☐**TRUE** ☐**FALSE** We can use linear search on an unsorted list

Code Tracing 1 [5pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
# This prints one set of 5 values
def ct1(s, d):
    r = set()
    for k in d:
        if d[k] in s:
            r.add(k)
            r.add(d[k]**2)
    return r
print(ct1({2,3,5}, {1:2, 2:3, 3:2, 5:4}))
```

Answer: { _____ , _____ , _____ , _____ , _____ }

Code Tracing 2 [6pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
# This prints one dictionary
# containing two key/value pairs
def ct2(L):
    d = dict()
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if L[i] == L[j]:
                d[i+j] = (i,j)
    return d
print(ct2([1,2,3,2,3,4]))
```

Answer: { _____ : _____ ,

_____ : _____ }

Code Tracing 3 [6pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
#This prints one string in the form of:
#Foo<z=__, s='____'>
#...but with the blanks filled in
class Foo(object):
    def __init__(self, x, s):
        self.z = x+1
        self.s = s
    def __repr__(self):
        return f'Foo<z={self.z}, s={repr(self.s)}>'
    def bar(self, other):
        return Foo(self.z * other.z, self.s + other.s)
def ct3(k):
    return Foo(k,str(k)).bar(Foo(k+1,'wow'))
print(str(ct3(5)))
```

Answer: Foo<z=_____, s='_____'>

Code Tracing 4 [6pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
#This prints a single list with 6 integers.
def ct4(L):
    if L == [ ]:
        return [ ]
    elif len(L) % 2 == 0:
        return [L[0]] + ct4(L[1:])
    else:
        return ct4(L[1:]) + [L[0]]

print(ct4([1,2,3,4,5,6]))
```

Answer: [_____ , _____ , _____ , _____ , _____ , _____]

Code Tracing 5 [6pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
#This prints a single list with 6 integers.
def ct5(L, depth=0):
    if len(L) < 2:
        return [v * 10**depth for v in L]
    else:
        i = len(L)//2
        return ct5(L[:i], depth+1) + [L[i]] + ct5(L[i+1:], depth+1)

print(ct5([1,2,3,4,5,6]))
```

Answer: [_____ , _____ , _____ , _____ , _____ , _____]

Code Tracing 6 [6pts]: Indicate what the following code prints by completing the partial answer in the box below of the code. Do not place anything but your answer inside the box.

```
#This prints a single list with 6 integers.
def ct6(L):
    return ct6Helper(L, [ ])

def ct6Helper(L, M):
    if L == [ ]:
        return M
    else:
        M.append(sum(L))
        return ct6Helper(L[1:], M)

print(ct6([1,2,3,4,5,6]))
```

Answer: [_____ , _____ , _____ , _____ , _____ , _____]

Free Response 1: Matrix Class [25pts]

Write the class Matrix so that the following test code passes. Hardcoding will not receive any credit. Your code must work in general, not just for the specific values in the test cases. We have provided headers and space for each of the methods you should write. Add the proper arguments and code for each.

```
def testMatrixClass():
    print('Testing Matrix class...', end='')
    m1 = Matrix([[1,2,3],[4,5,6]])
    assert(str(m1) == '<2x3 Matrix: [[1, 2, 3], [4, 5, 6]]>')
    assert(m1.rows == 2)
    assert(m1.cols == 3)
    assert(m1.getRow(0) == [1,2,3])
    assert(m1.getCol(0) == [1,4])
    assert(m1.getRow(5) == m1.getCol(42) == None) # handle out-of-bounds indexes

    m2 = Matrix([[10,20,30],[40,50,60]]) # make another matrix
    assert(str(m2) == '<2x3 Matrix: [[10, 20, 30], [40, 50, 60]]>')

    m3 = m1.addMatrix(m2)        # create new Matrix instance where each
                                 # value in m1 is added to the corresponding
                                 # value in m2

    assert(str(m3) == '<2x3 Matrix: [[11, 22, 33], [44, 55, 66]]>')
    # Be sure the previous operation was non-destructive:
    assert(str(m1) == '<2x3 Matrix: [[1, 2, 3], [4, 5, 6]]>')

    m4 = Matrix([[1]])
    assert(str(m4) == '<1x1 Matrix: [[1]]>')
    assert(m1.addMatrix(m4) == None) # handle mismatched dimensions when adding
    print('Passed!')
```

Begin your answer on the next page!

(Answer FR1: Matrix Class on this page. Remember to add arguments in the parentheses!)

```
class Matrix:
    def __init__(                    ):



    def __repr__(                    ):
```

Continue your answer on the next page!

(Answer FR1: Matrix Class on this page. Remember to add arguments in the parentheses!)

```
    def getRow(                          ):
```

```
    def getCol(                          ):
```

Continue your answer on the next page!

(Answer FR1: Matrix Class on this page. Remember to add arguments in the parentheses!)

```
    def addMatrix(                        ):
```

Free Response 2: Recursive Functions [20pts]

Part 1: The following recursive function takes a list L of integers, and non-destructively returns a new list M which is the same as L but with all the even numbers removed.
For example, if L == [1, 2, 3, 4, 5], the function returns [1, 3, 5].

Fill in the blanks with the missing code (all blanks must be filled with a single line, statement or expression -- you may not add new lines to the code):

```
def removeEvens1(L):
    if (L == [ ]):
        return [ ]
    else:
        if L[0] % 2 == 0:

            return _____________________________________________________ # Blank 1
        else:

            return _____________________________________________________ # Blank 2
```

Part 2: The following function does the same thing, only in a different way.
Once again, fill in the blanks so that this function works properly:

```
def removeEvens2(L):
    return removeEvens2Helper(L, [ ])

def removeEvens2Helper(L, M):
    if (L == [ ]):
        return M
    else:

        if _____________________________________________________: # Blank 1


            _____________________________________________________ # Blank 2
        return removeEvens2Helper(L[1:], M)
```

Part 3: The following function does the same thing again, only this time it is destructive, so it returns None while directly removing the evens from L. Fill in the blanks so it works properly:

```
def removeEvens3(L, i=0):
    if i >= len(L):
        return
    else:
        if L[i] % 2 == 0:

            _____________________________________________________________ # Blank 1
        else:

            _____________________________________________________________ # Blank 2
        removeEvens3(L, i)
```

Free Response 3: arrange(L) [10pts]

Note: This is a backtracking problem. Loops are allowed. You are provided with a partial solution to this problem, with a few parts removed.
Fill in the blanks with the missing code (all blanks must be filled with a single line, statement or expression -- you may not add new lines to the code).

Using backtracking, complete the function arrange(L, d) that takes a list L of integers and an integer d, and returns a new list M composed of the same values in L but rearranged so that the absolute difference between each two consecutive values in M is d or smaller. Return None if no such arrangement exists.

For example, arrange([1,5,2,-1], 3) can return [1,-1,2,5] because

- `abs(1 - (-1)) == 2 and 2 <= 3`
- `abs((-1) - 2) == 3 and 3 <= 3`
- `abs(2 - 5) == 3 and 3 <= 3`

Note that other legal arrangements may exist. You may return any legal arrangement. Note that backtracking does not generate every possible arrangement of L.

Also, arrange([1,5,1,-1], 3) returns None, because there is no arrangement of values that works for this list.

Hint: We start with an empty list for M and keep trying to add one more value from L to M (and removing it from L if it is used in M), verifying that M still is a legal arrangement.
Our solution uses destructive methods to add and remove elements from remainingList and resultSoFar.

Write your answer on the next page!

(This is part of FR3 on the previous page. Fill in the blanks!)

```
def arrange(L, d):
    return solve([], copy.copy(L), d)

def solve(resultSoFar, remainingList, d):
    if remainingList == []:

        return _____________________________________________________ # Blank 1
    else:
        #For each remaining element
        for i in range(len(remainingList)):
            #Get element for next attempted move
            v = remainingList[i]
            if ((resultSoFar == []) or

                _____________________________________________________: # Blank 2
                remainingList.pop(i) # remove v from the remaining list
                resultSoFar.append(v) # and add it to the resultSoFar list

                _____________________________________________________ # Blank 3

                if _____________________________________________________: # Blank 4
                    return result     #Hint: You need to define result somewhere

                #Undo the move if no solution
                remainingList.insert(i, v) # replace v in the remaining list
                resultSoFar.pop() # and remove it from the resultSoFar list

        return None
```

Bonus/Optional Code Tracing 1 [1pts]

Indicate what this prints. Place your answer (and nothing else) in the box.

```
def bonusCt1(k):
    def f(n): return n%10*f(n//10) if n else 1
    def g(n): return g(n//2)+[str(n%2)] if n else []
    def h(n): return int(''.join(g(f(n))) or '0')
    def i(k): return h(2**k) or i(k+1)
    return i(k)
print(bonusCt1(10))
```

Bonus/Optional Code Tracing 2 [1pts]

Indicate what this prints. Place your answer (and nothing else) in the box.

```
def bonusCt2(L):
    # Hint: str([[1,2],[3,4]]) == '[[1, 2], [3, 4]]'
    if len(str(L)) > 128:
        return len(str(L))
    else:
        return bonusCt2([[L],[L]])
print(bonusCt2([1,2]))
```