15-112
Spring 2024 Exam 2
March 24, 2024

Name:

Andrew ID:

- 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 work on the exam to receive credit.
- Write your answers in the specified places. If you run out of space for an answer, you may write on the backs of pages, but make sure to write a note telling the grader where to look for the rest of your answer.
- All code samples run without crashing. Assume any imports are already included as required.
- You may assume that math, string, and copy are imported; do not import any other modules.
- Do not use these post-midterm 2 topics/constructs: try/except, classes/OOP.

Don't write anything in the table below.

Question	Points	Score
1	21	
2	15	
3	12	
4	16	
5	20	
6	16	
7	0	
Total:	100	

1. **Code Tracing**: Indicate what the following programs print. Place your answers (and nothing else) in the boxes below the code.

   (a) (7 points) CT1

```
def ctList(a):
    b = a
    c = copy.copy(a)
    c[0] = 20
    b[0] = 90
    a[0] = "cat"
    print(a[0], b[0], c[0])
    a = c
    a[1] = "dog"
    b[1] = 2
    c[1] = 73
    print(a[1], b[1], c[1])
    c = b
    a[2] = 7
    b[2] = 8
    c[2] = 9
    print(a[2], b[2], c[2])
    b += [16]
    b[1] += 4
    print(a)
    print(b)
    print(c)

myList = [10, 11, 12]
ctList(myList)
print(myList)
```

(b) (6 points) CT2

```
def ct2(L):
    d = dict()
    for i in range(len(L)):
        s = set(L[i:])
        d[L[i]] = sum(s)
    print("1:", d)
    s = set(L)
    for e in L:
        if d.get(e, 0) % 2 == 1:
            del d[e]
    return d

print("2:", ct2([4, 2, 1, 2, 3, 4]))
```

(c) (2 points) CT3

```
def ct3(L):
    if len(L) == 1:
        return L[0]
    else:
        x = ct3(L[1:])
        return L[0] if L[0] > x else x

print(ct3([10, 18, 19, 7, 9, 20, 12, 4, 15, 11, 17, 8, 3, 6, 5]))
```

Hint: Don't manually trace the execution. Instead, figure out what the function does.

(d) (6 points) CT4

```
def ctHelper(L, r, c):
    print(L[r][c])
    lim = len(L)
    if r + 1 < lim and L[r + 1][c] % 2 == 1:
        ctHelper(L, r + 1, c)
    if c + 1 < lim and L[r][c + 1] % 2 == 1:
        ctHelper(L, r, c + 1)

def ct4(L):
    ctHelper(L, 0, 0)

L = [
        [3, 2, 4],
        [9, 7, 5],
        [1, 6, 11]
    ]
ct4(L)
```

2. (15 points) **Big Oh**

For each function shown below, write next to each line of the function either the Big-O runtime of the line or the number of times the line loops. Then write the total Big-O runtime of the function in terms of N in the box to the right of the code. **All answers must be simplified- do not include lower-order terms! For full credit, you must include line-by-line Big-O.**

```
def f1(s):  # s contains N characters          # Big-O
    i = 0                                      #______
    sum = 0                                    #______
    while i < len(s):                          #______
        sum += ord(s[i])                       #______
        i += 1                                 #______
    v = sum % 10                               #______
    return v                                   #______
```

```
def f2(L):  # L contains N iterms              # Big-O
    s = set()                                  #______
    for i in range(len(L) - 1, -1, -1):        #______
        s.add(L[i])                            #______
        L.pop()                                #______
        # update i                             #______
    for e in s:                                #______
        if e in L and e % 2 == 1:              #______
            print(e)                           #______
        # update e                             #______
```

Note: `pop()` removes an item from the end of the list and returns it. It does not traverse the list; instead it removes and returns the last item directly.

```
def f3(d):  # d contains N items               # Big-O
    L = list(d.items())                        #______
    L.sort()                                   #______
    s = set()                                  #______
    for t in L:                                #______
        for j in range(10):                    #______
            s.add(t[0] + t[1] - j)             #______
            # update j                         #______
        # update t                             #______
    return s                                   #______
```

---

3. **Free Response**: List Rotator

   In this problem you will write the same function two ways. `listRotator(lst, k)` takes a list `lst` and an integer `k` as inputs and rotates the list k positions to the right. The function returns the rotated list.

   Consider the following test cases:

```
assert(listRotator(['CMU', '112', 'Exam'], 2)==['112', 'Exam', 'CMU'])
assert(listRotator([1, 2, 3, 4, 5], 3)==[3, 4, 5, 1, 2])
assert(listRotator([], 5)==[])
```

   (a) (6 points) Write the function using a loop where, during each iteration of the loop, the list is rotated once.

   (b) (6 points) Write the function without using any loops, instead relying on list slicing. (This is not a recursion problem, so don't do it recursively, either.)

4. (16 points) **Recursive Free Response**: Largest Consecutive Pair Sum

   A consecutive pair sum (coined term) is the sum of two adjacent numbers in a list. In the list `[1, 4, 12]` there are two different consecutive pair sums: $1 + 4 = 5$ and $4 + 12 = 16$.

   Write the recursive function `largestConsecPairSum(L)` that takes a list of integers and returns the largest consecutive pair sum in the list. For example, `largestConsecPairSum([1,17,-4,0,-15,13,15,-2,20,-19])` returns 28 because the largest consecutive pair sum is $13 + 15 = 28$.

   Consider the following test cases:

```
assert largestConsecPairSum([]) == None
assert largestConsecPairSum([5]) == None
assert largestConsecPairSum([0, 5, 13, 16, -13, -5, 3, -7, 4, -2]) == 29
assert largestConsecPairSum([1, 17, -4, 0, -15, 13, 15, -2, 20, -19]) == 28
assert largestConsecPairSum([12, 9, -18, -13, -7, 3, 13]) == 21
assert largestConsecPairSum([-3, -6, -16, -4, -7, -20]) == -9
```

   Your solution must be entirely recursive. No loops or iterative functions are allowed; using them will result in a zero score for this problem.

5. (20 points) **Free Response**: List Commonality

   Write a function `commonalityAnalyzer(L1, L2)` that takes two lists of equal size, finds the common elements in these lists, and returns a dictionary that has the total count of the common elements in the two lists. Your solution should do better than $O(N^2)$.

   Consider the following test cases:

```
assert(commonalityAnalyzer( [1, 2, 3, 4, 5], [4, 5, 6, 7, 8]) == {4: 2, 5: 2})
assert(commonalityAnalyzer(['a','b', 'c', 'a'],  ['b', 'a', 'e', 'f']) ==  {'a': 3, 'b': 2})
assert(commonalityAnalyzer([1, 2, 3, 4, 5],[])=={})
assert(commonalityAnalyzer([],[])=={})
```

6. (16 points) **Free Response**: Color Sequences

Imagine a game that uses a 2D board of colored squares, where each square is either red, green, or blue. Whenever there are three or more squares of the same color *in a horizontal line*, that sequence of colored squares is eliminated from the board.

Write the *destructive* function `blankHorizontalSequences(board)` which, given a board, finds any horizontal sequences of length three or more and replaces all of the color codes with a space. The function is destructive, so it does not return anything.

Consider the following example:

```
gameBoard = [
    ["R", "R", "B", "G", "B", "G", "R", "G", "G", "B"],
    ["B", "R", "G", "B", "B", "R", "B", "R", "R", "B"],
    ["R", "R", "R", "R", "B", "G", "B", "R", "B", "B"],
    ["R", "G", "G", "G", "B", "R", "R", "B", "B", "B"],
    ["G", "R", "R", "G", "G", "G", "G", "G", "B", "G"],
]
blankHorizontalSequences(gameBoard)
```

After this code runs, `gameBoard` contains:

```
[
    ["R", "R", "B", "G", "B", "G", "R", "G", "G", "B"],
    ["B", "R", "G", "B", "B", "R", "B", "R", "R", "B"],
    [" ", " ", " ", " ", "B", "G", "B", "R", "B", "B"],
    ["R", " ", " ", " ", "B", "R", "R", " ", " ", " "],
    ["G", "R", "R", " ", " ", " ", " ", " ", "B", "G"],
]
```

7. (5 points (bonus)) **Free Response**: Largest Pair Sum

   This is a bonus problem, and so does not have partial credit: It is all or nothing. You should probably not attempt it unless you are finished with the rest of the exam.

   Write the recursive function `largestPairSum(L)` that takes a list of integers and returns the largest sum of *any pair* of numbers from the list. For example, `largestPairSum([1,17,-4,0,-15,13,15,-2,20,-19])` returns 37 because the largest pair sum is $20 + 17 = 37$.

   Consider the following test cases:

```
assert largestPairSum([]) == None
assert largestPairSum([5]) == None
assert largestPairSum([0, 5, 13, 16, -13, -5, 3, -7, 4, -2]) == 29
assert largestPairSum([1, 17, -4, 0, -15, 13, 15, -2, 20, -19]) == 37
assert largestPairSum([12, 9, -18, -13, -7, 3, 13]) == 25
assert largestPairSum([-3, -6, -16, -4, -7, -20]) == -7
```

   Your solution must be entirely recursive. No loops or iterative functions are allowed; using them will result in a zero score for this problem.