fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

WS10+Quiz10 version A

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

- You may not unstaple any pages.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz 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 quiz, and we will not grade it.
- You may not ask questions during the quiz, 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 1-10.
- 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.

Writing Session 10 (10% of HW10)

WS1. oddSum(L)

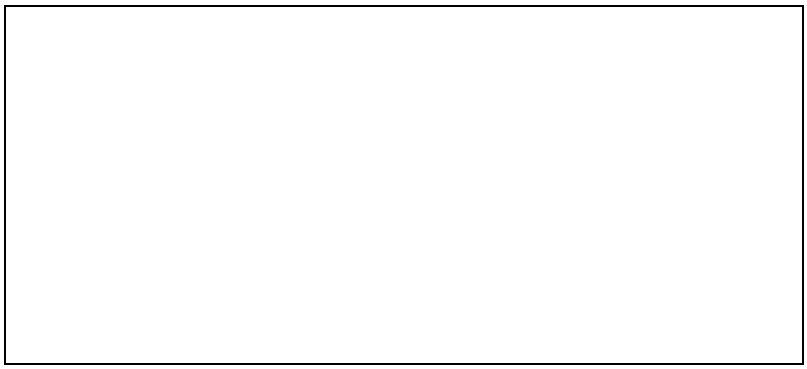
Write oddSum(L) from HW10. Your solution must use recursion properly, may not use loops, comprehensions, lists, strings, or anything else prohibited in that problem on HW10.

Quiz10

CT1: Code Tracing [10pts]

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

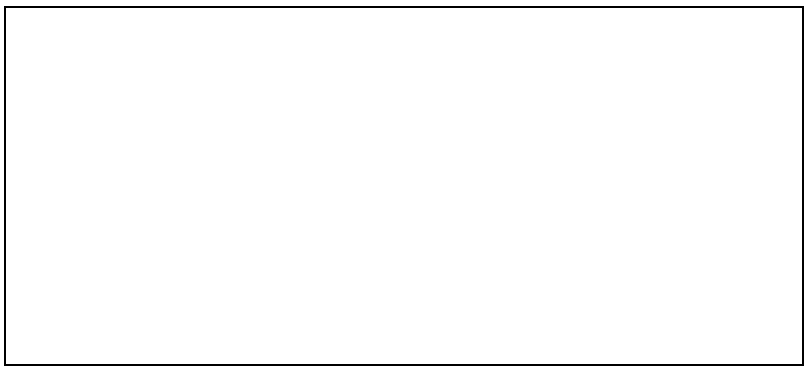
```
def ct1(L):
    if len(L) == 1: # note: 1 not 0
        return L[0]
    else:
        first, rest = L[0], L[1:]
        if first%2 == 0:
            print('A')
            return first * ct1(rest) # note: * not +
        else:
            print('B')
            return first + ct1(rest) # note: + not *
print(ct1([2,3,4,5]))
```



CT2: Code Tracing [10pts]

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

```
def ct2(M, d=0):
    if len(M) == 0:
        return [ ]
    else:
        i = len(M)//2
        L, v, R = M[:i], M[i], M[i+1:]
        return [(v, d)] + ct2(L, d+1) + ct2(R, d+1)
print(ct2([3,4,5,6]))
```



Short Answers [10pts total]

SA1. The following function is from the notes on Towers of Hanoi. It returns a list of moves to move n discs from the source tower to the target tower. In the box provided, provide the missing line of code:

```
def move(n, source, target, temp):
    if n == 1:
        return [(source, target)]
    else:
        return (move(n-1, source, temp,   target) +



                _____________________________________________ # <-- PROVIDE THIS LINE
                move(n-1, temp,   target, source))
```

SA2. The following function is from the notes on floodFill. It replaces a connected region in the oldColor with the newColor. In the box provided, provide the missing line of code:

```
def floodFillHelper(board, row, col, oldColor, newColor):
    rows, cols = len(board), len(board[0])
    if ((row < 0) or (row >= rows) or
        (col < 0) or (col >= cols) or



        _____________________________________________________) # <-- PROVIDE THIS LINE
        return
    else:
        board[row][col] = newColor
        floodFillHelper(board, row-1, col, oldColor, newColor) # up
        floodFillHelper(board, row+1, col, oldColor, newColor) # down
        floodFillHelper(board, row, col-1, oldColor, newColor) # left
        floodFillHelper(board, row, col+1, oldColor, newColor) # right
```

Free Response 1: smallestOdd(L) [35pts]

Notes for this problem:

1. you must use recursion properly.
2. do not use 'while' or 'for' loops.
3. do not use list comprehensions.
4. do not use any helper functions (but you are free to add default parameters if you wish, although there is a good solution that does not use them).
5. do not sort any lists.
6. do not create any new lists (except that you may slice a list).

With that in mind, write the function `smallestOdd(L)` (again, with any additional default parameters that you may wish to use) that takes a list L of integers and returns the smallest odd value in L, or None if L contains no odd integers.
Here are some tests for you:

```
assert(smallestOdd([2,5,4,3,6]) == 3)
assert(smallestOdd([2,5,-3,-3,5]) == -3)
assert(smallestOdd([2,4,6,8]) == None)
assert(smallestOdd([43]) == 43)
assert(smallestOdd([]) == None)
```

Begin your FR1 answer here or on the following page

Begin or continue your FR1 answer here:

Free Response 2: indexSetMap(s) [35pts]

Notes for this problem:

1. you must use recursion properly.
2. do not use 'while' or 'for' loops.
3. do not use list or dict comprehensions.
4. do not use any methods that will run in O(N) here (such as s.find(v))
5. do not use any default parameters (but you are free to use helper functions if you wish).

With that in mind, write the function `indexSetMap(s)` that takes a string s and returns a dict mapping each letter in s to a set of the indexes in s where that letter appears.

Notes:

1. This is case-insensitive and keys should be uppercase, so `indexSetMap('aA')` returns `{'A': {0,1}}`.
2. Ignore non-letters in s.

Here are some test cases for you:

```
assert(indexSetMap('aA') == {'A':{0,1}})
assert(indexSetMap('abA') == {'A':{0,2}, 'B':{1}})
assert(indexSetMap('d-cC?') == {'C':{2, 3}, 'D':{0}})
assert(indexSetMap('     ') == dict())
```

Begin your FR2 answer here or on the following page.

Begin or continue your FR2 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, b=2):
    def f(n): return '' if n==0 else f(n//b)+str(n%b)
    s = f(n)
    return b if  s == '1'*len(s) else bonusCt1(n, b+1)
print(bonusCt1(43))
```

bonusCt2 [optional, 1pt]

```
def bonusCt2(L, m=0):
    def f(L):
        if L == [ ]:
            return [ ]
        else:
            z = [L[0]-1] if L[0] else []
            return z + f(L[1:])
    if L == [ ]:
        return m
    else:
        L = f(L)
        return bonusCt2(L,m+1)
print(bonusCt2([1,2,3,4,3,2,1]*100))
```