fullName:_____

andrewID:_____

recitationLetter:_____

15-112 F22

Quiz9 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) 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!**

- **Note: There are three required problems in the following order: FR1, FR2, and CT1. Don't forget CT1!**

Free Response 1: powerSum(n, k) [38 points]

Write the function powerSum(n, k) that takes two non-negative integers, n and k, and returns the result of the following power-sum sequence:

`1**k + 2**k + 3**k + ... + n**k.`

For example, powerSum(4, 2) would return 30, because:
`1**2 + 2**2 + 3**2 + 4**2 == 1 + 4 + 9+ 16 == 30`

Note: You must solve this recursively (and without iteration) or you will not receive any credit.
You may not use loops or list comprehensions (or anything we haven't taught you in the notes, like generators). You may only use builtin functions or methods if they run in O(1.)

```
def testPowerSum():
    assert(powerSum(4, 2) == 30)
    assert(powerSum(4, 1) == 10)
    assert(powerSum(4, 0) == 4)
    assert(powerSum(5, 2) == 55)
    assert(powerSum(3, 3) == 36)
    assert(powerSum(0, 100) == 0)
    assert(powerSum(0, 0) == 0)
```

Begin your FR1 answer here or on the next page

You may begin or continue your FR1 answer here

Free Response 2: flatten(L) [50 points]

Write the recursive function flatten(L), which takes a list which may contain lists (which themselves may contain lists, and so on), and returns a single 1D list which contains each of the non-list items from the original list, in order. This is called flattening the list. The returned list contains no inner lists. For example, suppose we have the list

```
L = [[1, [2]], 3, [[[0]]]]
```

Flattening this list would return
`[1, 2, 3, 0]`

Note: You must solve this recursively (and without iteration) or you will not receive any credit.
You may not use loops or list comprehensions (or anything we haven't taught you in the notes, like generators). Slicing is allowed, and adding lists with '+' is allowed, but otherwise you may only use builtin functions or methods if they run in O(1).

```
def testFlatten():
    assert(flatten([[1, [2]], 3, [0]]) == [1, 2, 3, 0])
    assert(flatten([[[1]]]) == [1])
    assert(flatten([1, 2, 3, 4]) == [1, 2, 3, 4])
    assert(flatten(['F', [['L', 'A'], 'T'], ['T', 'E', ['N']]])
               == ['F','L','A','T','T','E','N'])
    assert(flatten([]) == [])
    assert(flatten([[],[[]],[[[]]]]) == [])
```

Begin your FR2 answer here or on the next page

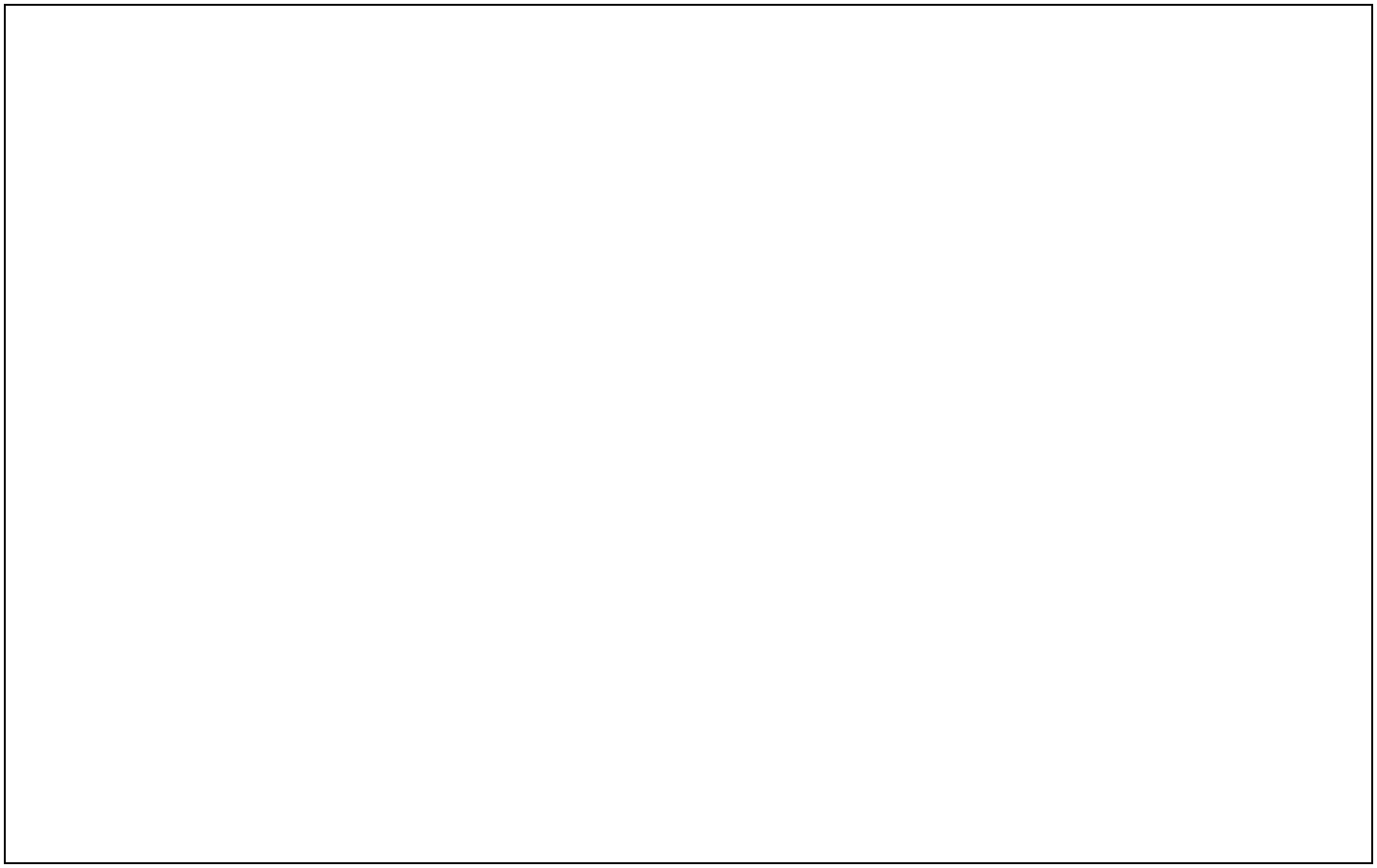
You may begin or continue your FR2 answer here

CT1: Code Tracing [12pts]

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

```
def ct1(L, depth = 0):
    if len(L) == 1:
        result = L[0]
        print(f'{depth}: {result}')
        return result
    if len(L) == 2:
        result = L[0] - L[1]
        print(f'{depth}: {result}')
        return result
    else:
        m = len(L)//2
        right = ct1(L[m:], depth+1)
        left = ct1(L[:m], depth+1)
        result = right + left
        print(f'{depth}: {result}')
        return result

print(ct1([6, 3, 2, 5, 4]))
```



You may use this page to show your work for CT1 (but we will only grade your answer in the box.)

bonusCT: Code Tracing [2pts bonus]

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

```
def bonusCt1(n):
    def f(x):
        return x and 2*x-1+f(x-1)
    def g(x):
        return x and 1+g(x//2)
    while f(g(n)) != n:
        n = f(g(n))
    return n
print(bonusCt1(31))
```