Work with your neighbor. (This will be graded for participation only.)

1. We have written the recursive function version of `sumlist(L)` that returns the sum of the elements in `L`. Re-write the recursive part in two different ways:
   a. Recurse on the first through second to the last elements and add the last element

   ANS:

```
def sumlist(L):
    if L == []:
        return 0
    else:
        return sumlist(L[:-1]) + L[-1]
```

   b. Recurse on each half and add them together

   ANS:

```
def sumlist(L):
    print(L)   # run this with a print to see the breakdown
    if L == []:
        return 0
    elif len(L) == 1:
        return L[0]
    else:
        mid = len(L) // 2

        return sumlist(L[:mid]) + sumlist(L[mid:])
```

2. Write a recursive function `bin_search(alist, item)` that searches for `item` in `alist` and returns `True` if found and `False` otherwise.

   ANS:

```
def bin_search(alist, item):

    if alist == []:
        return False
    mid = len(alist)//2
    if alist[mid] == item:
        return True
```

```
    else:
        if item < alist[mid]:
            return bin_search(alist[:mid],item)
        else:

            return bin_search(alist[mid+1:], item)
```

3. On the last ICA, you wrote `sum_cols(grid, n)` that sums column `n` in a grid:

```
def sum_cols(grid, n):
    if len(grid) == 0:
        return 0
    else:
        return grid[0][n] + sum_cols(grid[1:], n)
```

Now consider summing along the diagonal. Write a recursive function `sum_diag(grid)` that returns the sum of the diagonal from upper left to bottom right in a grid, i.e., it sums `grid[0][0]`, `grid[1][1]`, and so on. You may assume the grid is square.

Question: You can slice the grid (list of lists) in each round of recursion as usual. That means that `grid[0]` is the next row in each recursive call. But how will you know which column you need to index into for each recursive step?

Hint: Have `sum_diag(grid)` call a "helper" function called `sum_diag_helper` that is recursive. It has a **new** argument, `col`, that keeps track of the current column: `sum_diag_helper(grid, col)`. Call the helper function with 0 as the column number to start with.

```
def sum_diag(grid):

    return sum_diag_helper(grid, 0) # call the helper function

# sum_diag: a helper function
# the helper function has an additional argument, col
# col will keep track of the current column
# in the diagonal
def sum_diag_helper(grid, col):

    # your code goes here
```

ANS:

```
def sum_diag_helper(grid, col):

    # your code goes here

    if grid == []:
```

```
        return 0
    else:
        return grid[0][col] + sum_diag_helper(grid[1:], col + 1)
```

NOTE: I combined problems 4 and 5 and moved them to ICA-22.