Three Number Sum

Input: array = [[1, 2, 3, 4],
[12, 13, 14, 5],
[11, 16, 15, 6],
[10, 9, 8, 7],]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

Input: An n x m 2D array (that can be square shaped when n == m)

Output: A 1D array of all the array's elements in spiral order

// Spiral order starts at the top left corner of the 2D array, goes to the right, and proceeds in a spiral pattern all the way until every element has been visited

```
// ------ITERATIVE SOLUTION------
// O(n) time | O(n) space
function spiralTraverse(array) {
  let startRow = 0;
  let endRow = array.length - 1;
  let startColumn = 0;
  let endColumn = array[0].length - 1;
  const result = [];

  while (startColumn <= endColumn && startRow <= endRow) {
    for (let col = startColumn; col <= endColumn; col++) {
      result.push(array[startRow][col]);
    }

    for (let row = startRow + 1; row <= endRow; row++) {
      result.push(array[row][endColumn]);
    }

    for (let col = endColumn - 1; col >= startColumn; col--)
      if (startRow === endRow) break;
      result.push(array[endRow][col]);
    }

    for (let row = endRow - 1; row >= startRow + 1; row--) {
      if (startColumn === endColumn) break;
      result.push(array[row][startColumn]);
    }

    startRow++;
    endRow--;
    startColumn++;
    endColumn--;
  }
  return result;
}
```

Time: O(n) (where n is the total # of elements in our n x m input array) since we traverse every element during the spiral

Space: O(n) since we are storing a new array with all n x m values

For the while loop, we go until <= since we still want the values when startX === endX but if either are > than, we exit out

The two if statements are for when we have one row or one column left.

```
// ------RECURSIVE SOLUTION------
// O(n) time | O(n) space
function spiralTraverse(array) {
  const result = [];
  spiralFill(array, 0, array[0].length - 1, 0, array.length - 1, result);
  return result;
}

function spiralFill(array, startColumn, endColumn, startRow, endRow, result) {
  if (startColumn > endColumn || startRow > endRow) return;

  for (let col = startColumn; col <= endColumn; col++) {
    result.push(array[startRow][col]);
  }

  for (let row = startRow + 1; row <= endRow; row++) {
    result.push(array[row][endColumn]);
  }

  for (let col = endColumn - 1; col >= startColumn; col--) {
    if (startRow === endRow) break;
    result.push(array[endRow][col]);
  }

  for (let row = endRow - 1; row >= startRow + 1; row--) {
    if (startColumn === endColumn) break;
    result.push(array[row][startColumn]);
  }

  spiralFill(array, startColumn + 1, endColumn - 1, startRow + 1, endRow - 1, result);
}
```

Similar to the iterative solution. Only difference is at each recursive call, we check if the startX has exceeded (>) the endX. Only then do we know we're at the end and we can return

☆ AND --> both have to be true for us to continue

so when do we stop ?

OR --> when either one becomes false

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

	SC			EC
SR	0,0 (1)	0,1 (2)	0,2 (3)	0,3 (4)
	1,0 (12)	1,1 (13)	1,2 (14)	1,3 (5)
	2,0 (11)	2,1 (16)	2,2 (15)	2,3 (6)
ER	3,0 (10)	3,1 (9)	3,2 (8)	3,3 (7)

[1, 2, 3, 14, 15, 8, 9, 10, 11, 12, 13, 16]

		EC	
	SC		
	0,0 (1)	0,1 (2)	0,2 (3)
SR	1,0 (12)	1,1 (13)	1,2 (14)
ER	2,0 (11)	2,1 (16)	2,2 (15)
	3,0 (10)	3,1 (9)	3,2 (8)

[1, 2, 3, 4, 5, 6, 15, 16, 11, 12, 13, 14]

		SC	EC	
	0,0 (1)	0,1 (2)	0,2 (3)	0,3 (4)
SR ER	1,0 (12)	1,1 (13)	1,2 (14)	1,3 (5)
	2,0 (11)	2,1 (16)	2,2 (15)	2,3 (6)