Overview

Both solutions recursively traverse a nested array and produce a flattened string of numbers separated by commas. Recursion handles arrays within arrays in each case, but the two approaches differ in how they iterate and how they format the output.

My Solution

Js Copy

```js
(function () {
  let arr = [[1, 2, 3, 4], 5, 6, [7, 8, 9], 10, 11, [12, 13, 14, 15]];
  let data = "";

  function printArr(arr) {
    for (let i = 0; i < arr.length; i++) {
      if (Array.isArray(arr[i])) {
        printArr(arr[i]);
      } else {
        // Add a comma only if data already has content
        data += `${data.length === 0 ? "" : ","} ${arr[i]}`;
      }
    }
  }

  printArr(arr);
  console.log(data);
})();
```

Highlights:

- **Controlled Formatting:**
  Checks if `data` is empty before adding a comma, avoiding a leading comma and producing clean output:
  `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15`.
- **Explicit Iteration:**
  Uses a `for` loop for clear, step-by-step control over traversal.
- **Readability:**
  The comma-handling logic is inline with value addition, making the formatting intent obvious.

Metwally’s Solution

Js Copy

```js
(function () {
  let arr = [[1, 2, 3, 4], 5, 6, [7, 8, 9], 10, 11, [12, 13, 14, 15]];
  let data = "";

  function process_array(ar) {
    if (Array.isArray(ar)) {
      ar.forEach((e) => process_array(e));
    } else {
      data = data + ", " + ar;
    }
  }

  process_array(arr);
  console.log(data);
})();
```

Highlights:

- **Functional Style:**
  Uses `forEach` for concise, modern iteration with recursion embedded directly.
- **Formatting Issue:**
  Always prepends `", "` when adding a number, resulting in a leading comma: `, 1, 2, 3, 4, 5, …`.
- **Simplicity vs. Control:**
  Less boilerplate, but no handling for the leading-comma edge case.

Comparison

Aspect	My Solution	Metwally’s Solution
Iteration	`for` loop	`forEach` with recursion
Formatting	Conditional comma to avoid leading separator	Always adds comma, causing a leac
Readability	Very explicit, step-by-step	Concise, but hides formatting flaw
Flexibility	Easy to adapt formatting rules	Cleaner recursion style, but less out

Verdict

My Solution is more robust for production because it handles formatting cleanly without post-processing. **Metwally’s Solution** is shorter and stylistically modern, but the leading comma detracts from output quality when strict formatting is required.

Further Thoughts

Both run in O(n) time. For cleaner separation of concerns, you could collect values in an array and join them at the end:

Js Copy

```js
(function () {
  let arr = [[1, 2, 3, 4], 5, 6, [7, 8, 9], 10, 11, [12, 13, 14, 15]];
  let result = [];

  function flatten(ar) {
    if (Array.isArray(ar)) {
      ar.forEach(flatten);
    } else {
      result.push(ar);
    }
  }

  flatten(arr);
  console.log(result.join(", "));
})();
```

Or, in modern environments:

Js Copy

```js
(function () {
  let arr = [[1, 2, 3, 4], 5, 6, [7, 8, 9], 10, 11, [12, 13, 14, 15]];
  console.log(arr.flat(Infinity).join(", "));
})();
```

These alternatives show that recursion works well, but built-in methods can be both shorter and correct.