Given a **multi-dimensional** array arr and a depth n, return a **flattened** version of that array.

A **multi-dimensional** array is a recursive data structure that contains integers or other **multi-dimensional** arrays.

A **flattened** array is a version of that array with some or all of the sub-arrays removed and replaced with the actual elements in that sub-array. This flattening operation should only be done if the current depth of nesting is less than n. The depth of the elements in the first array are considered to be 0.

Please solve it without the built-in Array.flat method.

**Example 1:**

Input  
arr = [1, 2, 3, [4, 5, 6], [7, 8, [9, 10, 11], 12], [13, 14, 15]]  
n = 0  
Output  
[1, 2, 3, [4, 5, 6], [7, 8, [9, 10, 11], 12], [13, 14, 15]]  
  
Explanation  
Passing a depth of n=0 will always result in the original array. This is because the smallest possible depth of a subarray (0) is not less than n=0. Thus, no subarray should be flattened.

**Example 2:**

Input  
arr = [1, 2, 3, [4, 5, 6], [7, 8, [9, 10, 11], 12], [13, 14, 15]]  
n = 1  
Output  
[1, 2, 3, 4, 5, 6, 7, 8, [9, 10, 11], 12, 13, 14, 15]  
  
Explanation  
The subarrays starting with 4, 7, and 13 are all flattened. This is because their depth of 0 is less than 1. However [9, 10, 11] remains unflattened because its depth is 1.

**Example 3:**

Input  
arr = [[1, 2, 3], [4, 5, 6], [7, 8, [9, 10, 11], 12], [13, 14, 15]]  
n = 2  
Output  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]  
  
Explanation  
The maximum depth of any subarray is 1. Thus, all of them are flattened.

**Constraints:**

* 0 <= count of numbers in arr <= 105
* 0 <= count of subarrays in arr <= 105
* maxDepth <= 1000
* -1000 <= each number <= 1000
* 0 <= n <= 1000