Problem 3537: Fill a Special Grid

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a non-negative integer

n

representing a

2

n

x 2

n

grid. You must fill the grid with integers from 0 to

2

2n

- 1

to make it

special

. A grid is

special

if it satisfies

all

the following conditions:

All numbers in the top-right quadrant are smaller than those in the bottom-right quadrant.

All numbers in the bottom-right quadrant are smaller than those in the bottom-left quadrant.

All numbers in the bottom-left quadrant are smaller than those in the top-left quadrant.

Each of its quadrants is also a special grid.

Return the

special

2

n

x 2

n

grid.

Note

: Any 1x1 grid is special.

Example 1:

Input:

n = 0

Output:

[[0]]

Explanation:

The only number that can be placed is 0, and there is only one possible position in the grid.

Example 2:

Input:

n = 1

Output:

[[3,0],[2,1]]

Explanation:

The numbers in each quadrant are:

Top-right: 0

Bottom-right: 1

Bottom-left: 2

Top-left: 3

Since

$0 < 1 < 2 < 3$

, this satisfies the given constraints.

Example 3:

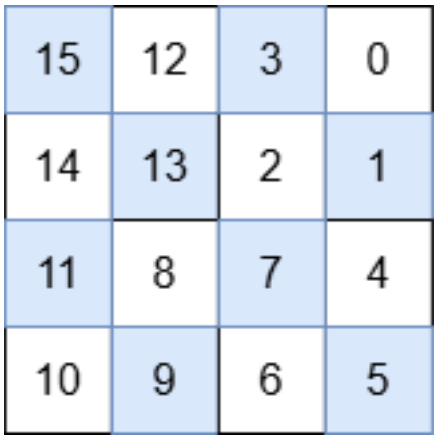
Input:

n = 2

Output:

[[15,12,3,0],[14,13,2,1],[11,8,7,4],[10,9,6,5]]

Explanation:



The numbers in each quadrant are:

Top-right: 3, 0, 2, 1

Bottom-right: 7, 4, 6, 5

Bottom-left: 11, 8, 10, 9

Top-left: 15, 12, 14, 13

max(3, 0, 2, 1) < min(7, 4, 6, 5)

max(7, 4, 6, 5) < min(11, 8, 10, 9)

max(11, 8, 10, 9) < min(15, 12, 14, 13)

This satisfies the first three requirements. Additionally, each quadrant is also a special grid. Thus, this is a special grid.

Constraints:

0 <= n <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> specialGrid(int n) {

}
};
```

Java:

```
class Solution {
public int[][] specialGrid(int n) {

}
}
```

Python3:

```
class Solution:
def specialGrid(self, n: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def specialGrid(self, n):
"""
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number[][]}
*/
var specialGrid = function(n) {

};
```

TypeScript:

```
function specialGrid(n: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SpecialGrid(int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** specialGrid(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func specialGrid(n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun specialGrid(n: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func specialGrid(_ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn special_grid(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[][]}
def special_grid(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function specialGrid($n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> specialGrid(int n) {

}
}
```

Scala:

```
object Solution {
def specialGrid(n: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec special_grid(n :: integer) :: [[integer]]
def special_grid(n) do

end
end
```

Erlang:

```
-spec special_grid(N :: integer()) -> [[integer()]].
special_grid(N) ->
.
```

Racket:

```
(define/contract (special-grid n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> specialGrid(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] specialGrid(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Fill a Special Grid
Difficulty: Medium
Tags: array
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def specialGrid(self, n: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def specialGrid(self, n):
"""
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {number[][]}
*/
var specialGrid = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function specialGrid(n: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[][] SpecialGrid(int n) {

}
}
```

C Solution:

```
/*
* Problem: Fill a Special Grid
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** specialGrid(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Fill a Special Grid
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func specialGrid(n int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun specialGrid(n: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func specialGrid(_ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Fill a Special Grid
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn special_grid(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[][]}
def special_grid(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function specialGrid($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> specialGrid(int n) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def specialGrid(n: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec special_grid(n :: integer) :: [[integer]]
def special_grid(n) do

end
end
```

Erlang Solution:

```
-spec special_grid(N :: integer()) -> [[integer()]].
special_grid(N) ->
.
```

Racket Solution:

```
(define/contract (special-grid n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```