

Problem 59: Spiral Matrix II

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a positive integer

n

, generate an

$n \times n$

matrix

filled with elements from

1

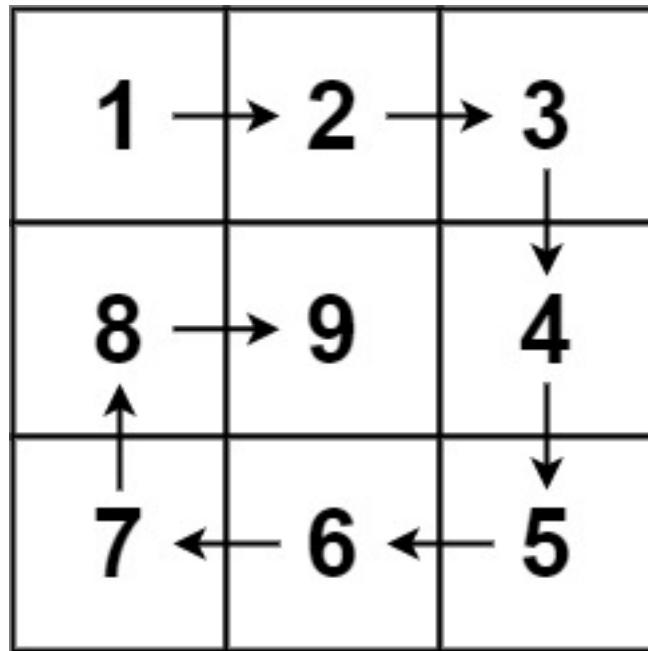
to

n

2

in spiral order.

Example 1:



Input:

$n = 3$

Output:

`[[1,2,3],[8,9,4],[7,6,5]]`

Example 2:

Input:

$n = 1$

Output:

`[[1]]`

Constraints:

$1 \leq n \leq 20$

Code Snippets

C++:

```
class Solution {  
public:  
vector<vector<int>> generateMatrix(int n) {  
  
}  
};
```

Java:

```
class Solution {  
public int[][] generateMatrix(int n) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function generateMatrix(n: number): number[][] {  
};
```

C#:

```
public class Solution {  
    public int[][] GenerateMatrix(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** generateMatrix(int n, int* returnSize, int** returnColumnSizes) {  
  
}
```

Go:

```
func generateMatrix(n int) [][]int {  
  
}
```

Kotlin:

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

Swift:

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

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn generate_matrix(n: i32) -> Vec<Vec<i32>> {
        let mut matrix = vec![vec![0; n]; n];
        for i in 0..n {
            for j in 0..n {
                if i < j {
                    matrix[i][j] = i * j;
                } else if i > j {
                    matrix[i][j] = (i * (i + 1)) / 2 + j;
                } else {
                    matrix[i][j] = i * (i + 1) / 2;
                }
            }
        }
        matrix
    }
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

    /**
     * @param Integer $n
     * @return Integer[][]
     */
    function generateMatrix($n) {
        $matrix = [];
        for ($i = 0; $i < $n; $i++) {
            $matrix[$i] = [];
            for ($j = 0; $j < $n; $j++) {
                if ($i < $j) {
                    $matrix[$i][$j] = $i * $j;
                } elseif ($i > $j) {
                    $matrix[$i][$j] = ($i * ($i + 1)) / 2 + $j;
                } else {
                    $matrix[$i][$j] = $i * ($i + 1) / 2;
                }
            }
        }
        return $matrix;
    }
}
```

Dart:

```
class Solution {
    List<List<int>> generateMatrix(int n) {
        List<List<int>> matrix = List.generate(n, (i) => List.generate(n, (j) => {
            if (i < j) {
                return i * j;
            } else if (i > j) {
                return ((i * (i + 1)) / 2) + j;
            } else {
                return (i * (i + 1)) / 2;
            }
        }));
        return matrix;
    }
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (generate-matrix n)  
  (-> exact-integer? (listof (listof exact-integer?)))  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Spiral Matrix II  
 * 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>> generateMatrix(int n) {  
  
}  
};
```

Java Solution:

```
/**  
* Problem: Spiral Matrix II  
* 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[][] generateMatrix(int n) {  
  
}  
}
```

Python3 Solution:

```
"""  
Problem: Spiral Matrix II  
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 generateMatrix(self, n: int) -> List[List[int]]:  
# TODO: Implement optimized solution  
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Spiral Matrix II
 * 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 generateMatrix = function(n) {

};
```

TypeScript Solution:

```
/**
 * Problem: Spiral Matrix II
 * 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 generateMatrix(n: number): number[][] {
```

```
};
```

C# Solution:

```
/*
 * Problem: Spiral Matrix II
 * 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[][] GenerateMatrix(int n) {
        ...
    }
}
```

C Solution:

```
/*
 * Problem: Spiral Matrix II
 * 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** generateMatrix(int n, int* returnSize, int** returnColumnSizes) {
```

```
}
```

Go Solution:

```
// Problem: Spiral Matrix II
// 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 generateMatrix(n int) [][]int {
}
```

Kotlin Solution:

```
class Solution {
    fun generateMatrix(n: Int): Array<IntArray> {
        return IntArray(n).map { IntArray(n) }.toTypedArray()
    }
}
```

Swift Solution:

```
class Solution {
    func generateMatrix(_ n: Int) -> [[Int]] {
        return IntArray(n).map { IntArray(n) }.toTypedArray()
    }
}
```

Rust Solution:

```
// Problem: Spiral Matrix II
// 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 generate_matrix(n: i32) -> Vec<Vec<i32>> {
        let mut matrix = vec![vec![0; n]; n];
        for i in 0..n {
            for j in 0..n {
                if i == j {
                    matrix[i][j] = i * (i + 1);
                } else if i < j {
                    matrix[i][j] = i * (i + 1) + j;
                } else {
                    matrix[i][j] = i * (i + 1) + n - j;
                }
            }
        }
        matrix
    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

```
class Solution {
    List<List<int>> generateMatrix(int n) {
        List<List<int>> matrix = List.generate(n, (_) => List.generate(n, (_) => 0));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i == j) {
                    matrix[i][j] = i * (i + 1);
                } else if (i < j) {
                    matrix[i][j] = i * (i + 1) + j;
                } else {
                    matrix[i][j] = i * (i + 1) + n - j;
                }
            }
        }
        return matrix;
    }
}
```

Scala Solution:

```
object Solution {
    def generateMatrix(n: Int): Array[Array[Int]] = {
        val matrix = Array.fill(n, Array.fill(n, 0))
        for (i <- 0 until n) {
            for (j <- 0 until n) {
                if (i == j) {
                    matrix(i)(j) = i * (i + 1)
                } else if (i < j) {
                    matrix(i)(j) = i * (i + 1) + j
                } else {
                    matrix(i)(j) = i * (i + 1) + n - j
                }
            }
        }
        matrix
    }
}
```

```
}
```

```
}
```

Elixir Solution:

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

    end
  end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (generate-matrix n)
  (-> exact-integer? (listof (listof exact-integer?)))
  )
```