Description | Editorial | Solutions (6.1K) | Submissions

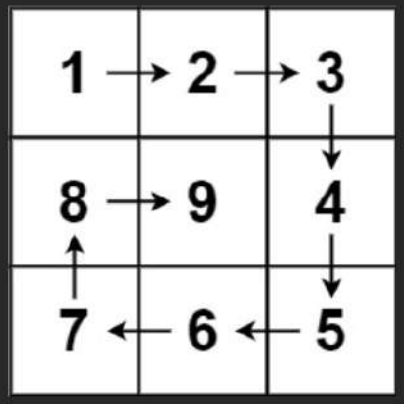
59. Spiral Matrix II

Medium 5.9K 238

Companies

Given a positive integer `n`, generate an `n x n` `matrix` filled with elements from `1` to `n²` 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]]
```

Java | Auto

```java
class Solution {
    public int[][] generateMatrix(int n) {
        int[][] ans = new int[n][n];
        //right -> bottom -> left -> top
        int top = 0, left = 0;
        int bottom = n-1, right = n-1;
        int k = 1;
        while(top<=bottom && left<=right){
            //right
            for(int i=left;i<=right;i++){
                ans[top][i] = k++ ;
            }
            top++;
            //bottom
            for(int i=top;i<=bottom;i++){
                ans[i][right] = k++;
            }
            right--;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

n =
3

Output

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

Console | Run | Submit

Description | Editorial | Solutions (10.8K) | Submissions

33. Search in Rotated Sorted Array

Medium | 23.3K | 1.4K

Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Example 3:

Java | Auto

```java
class Solution {
    public int search(int[] nums, int target) {
        int start = 0, end = nums.length - 1;
        int mid = (start + end) / 2;
        while (start <= end) {
            mid = (start + end) / 2;
            if (target == nums[mid]) {
                return mid;
            }
            if (nums[start] <= nums[mid]) {
                if (nums[start] <= target && nums[mid] >= target) {
                    end = mid - 1;
                } else {
                    start = mid + 1;
                }
            } else {
                if (nums[end] >= target && nums[mid] <= target) {
                    start = mid + 1;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

nums =
`[4,5,6,7,0,1,2]`

target =
`0`

Output

Console | Run | Submit