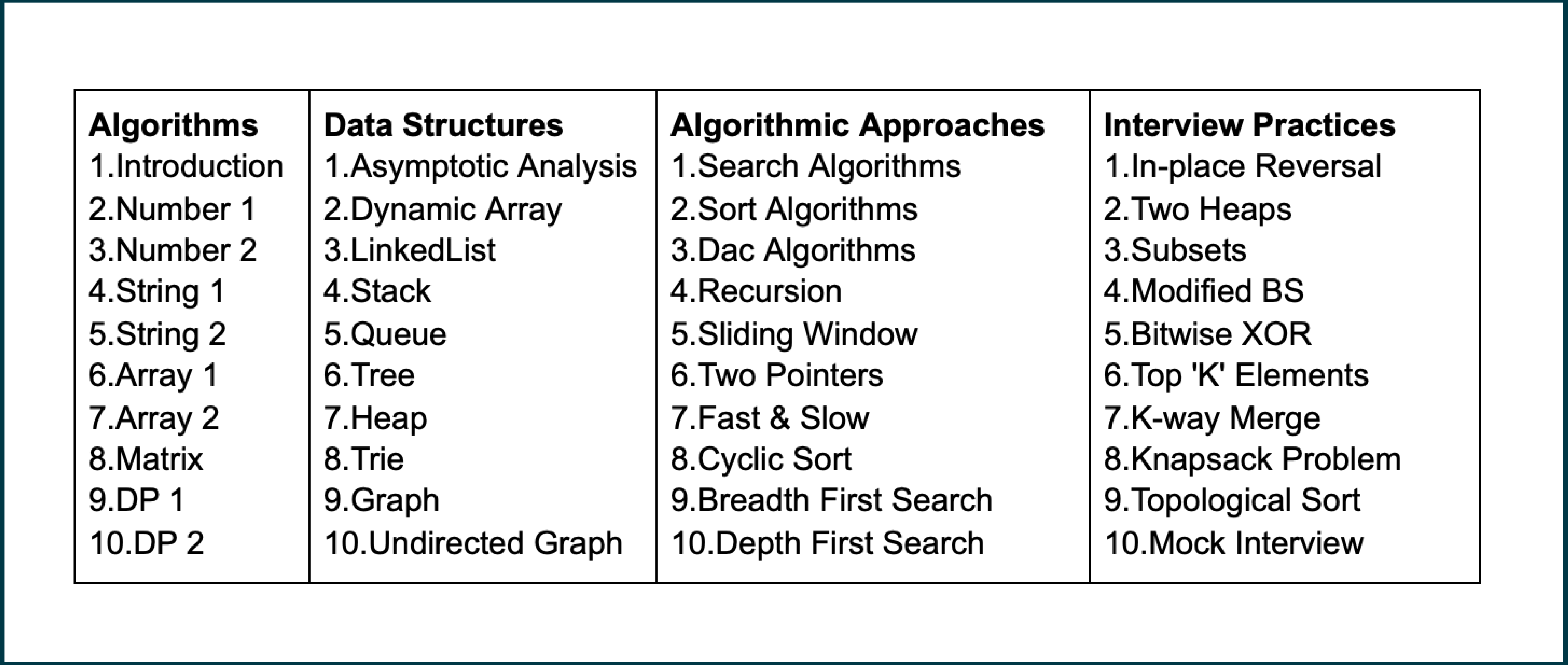
PDP ONLINE

DSA - Algorithms Matrix

Course Planning

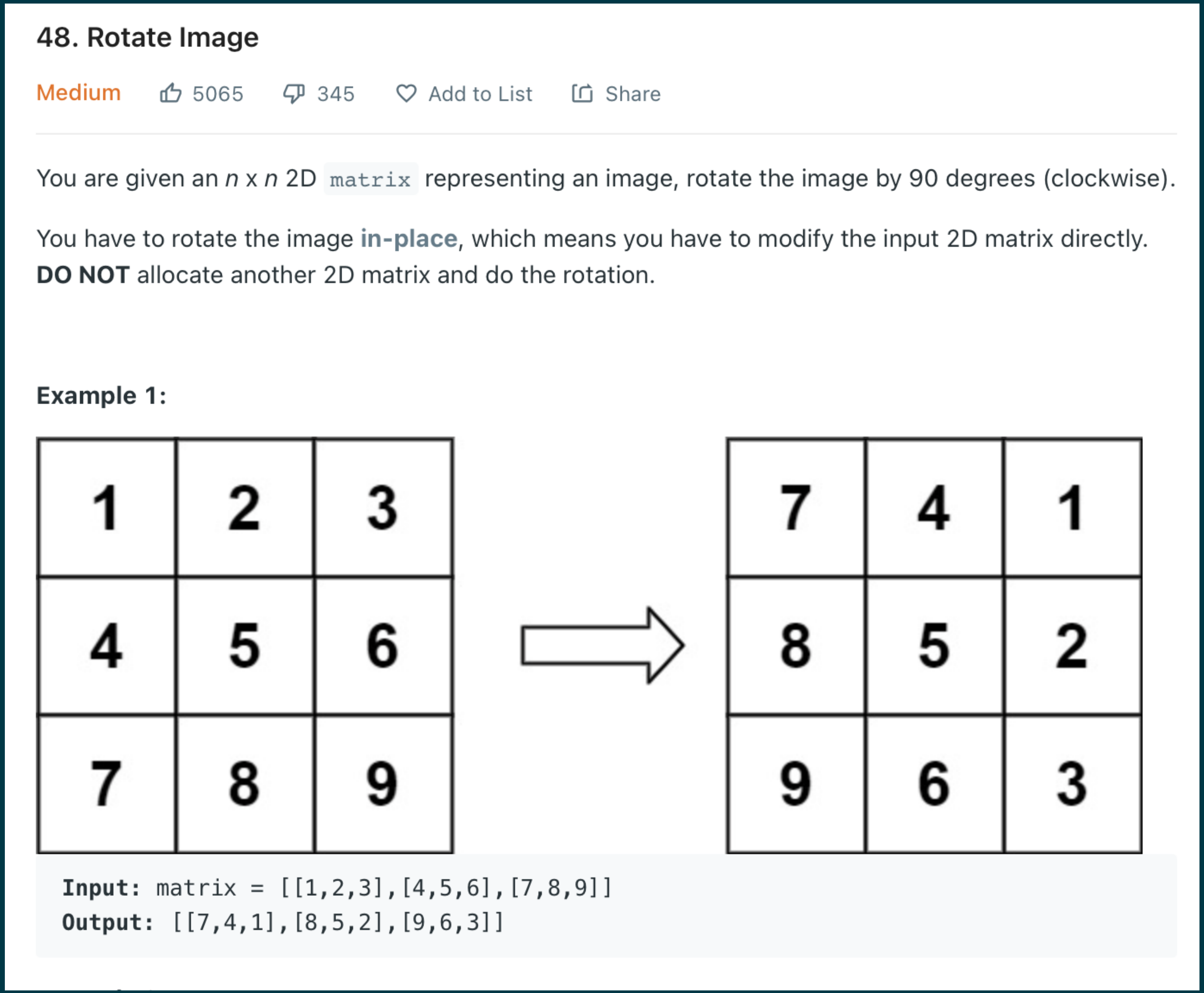


Algorithms	**Data Structures**	**Algorithmic Approaches**	**Interview Practices**
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.Tree	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Heap	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

Asked by Facebook

Explanation



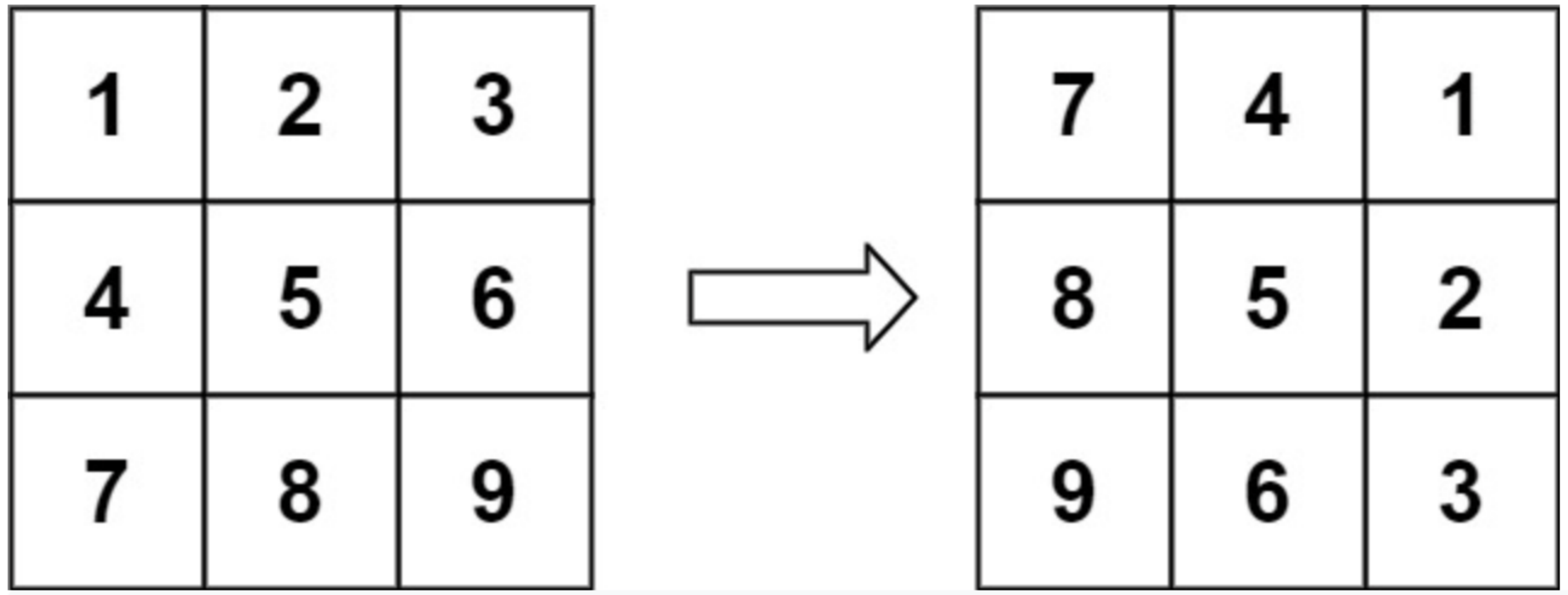
48. Rotate Image

Medium 5065 345 Add to List Share

You are given an *n* x *n* 2D `matrix` representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

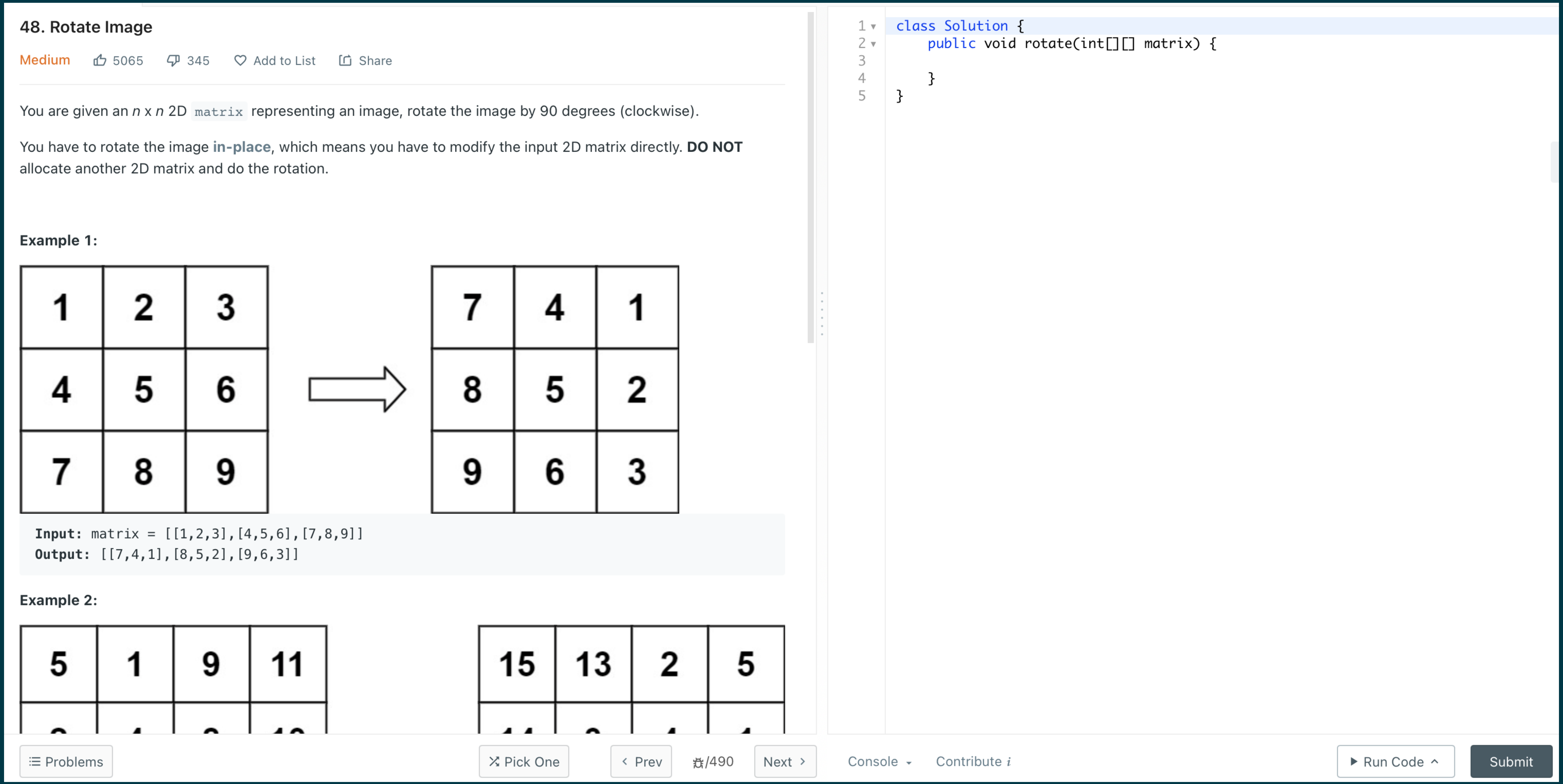


1	2	3
4	5	6
7	8	9

7	4	1
8	5	2
9	6	3

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Rotate Image



48. Rotate Image

Medium 5065 345 Add to List Share

You are given an *n* x *n* 2D `matrix` representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

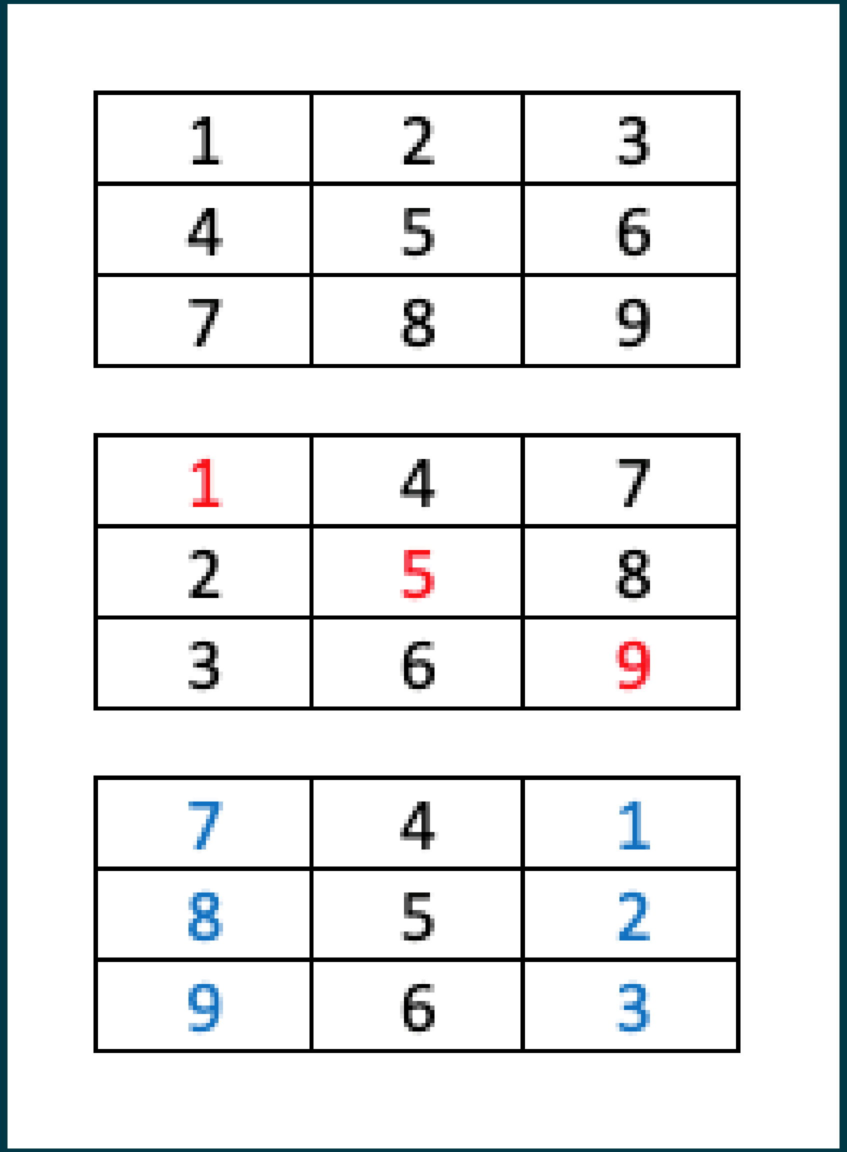
```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:

```java
class Solution {
    public void rotate(int[][] matrix) {

    }
}
```

https://leetcode.com/problems/rotate-image/

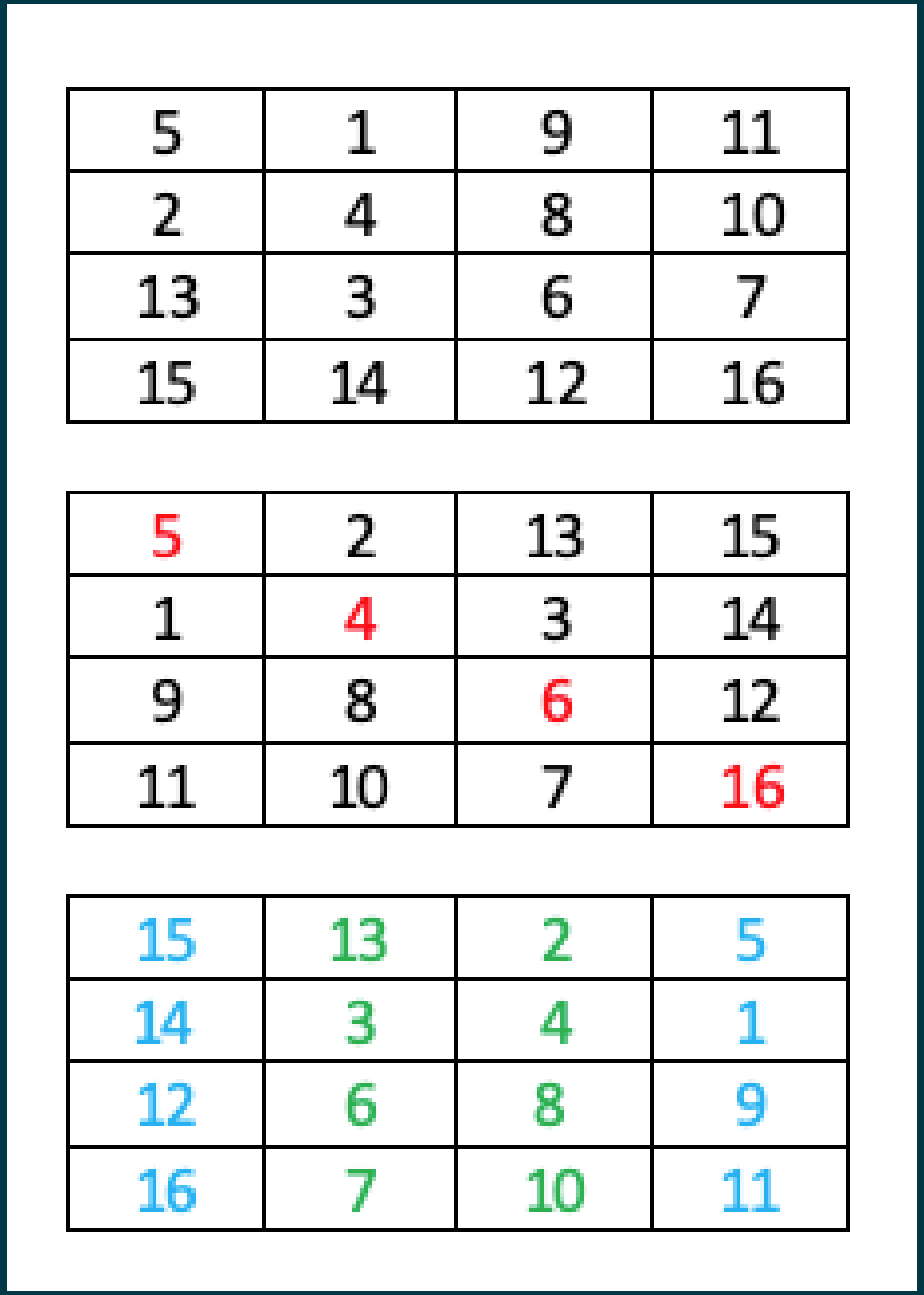
First Theory



1	2	3
4	5	6
7	8	9

1	4	7
2	5	8
3	6	9

7	4	1
8	5	2
9	6	3

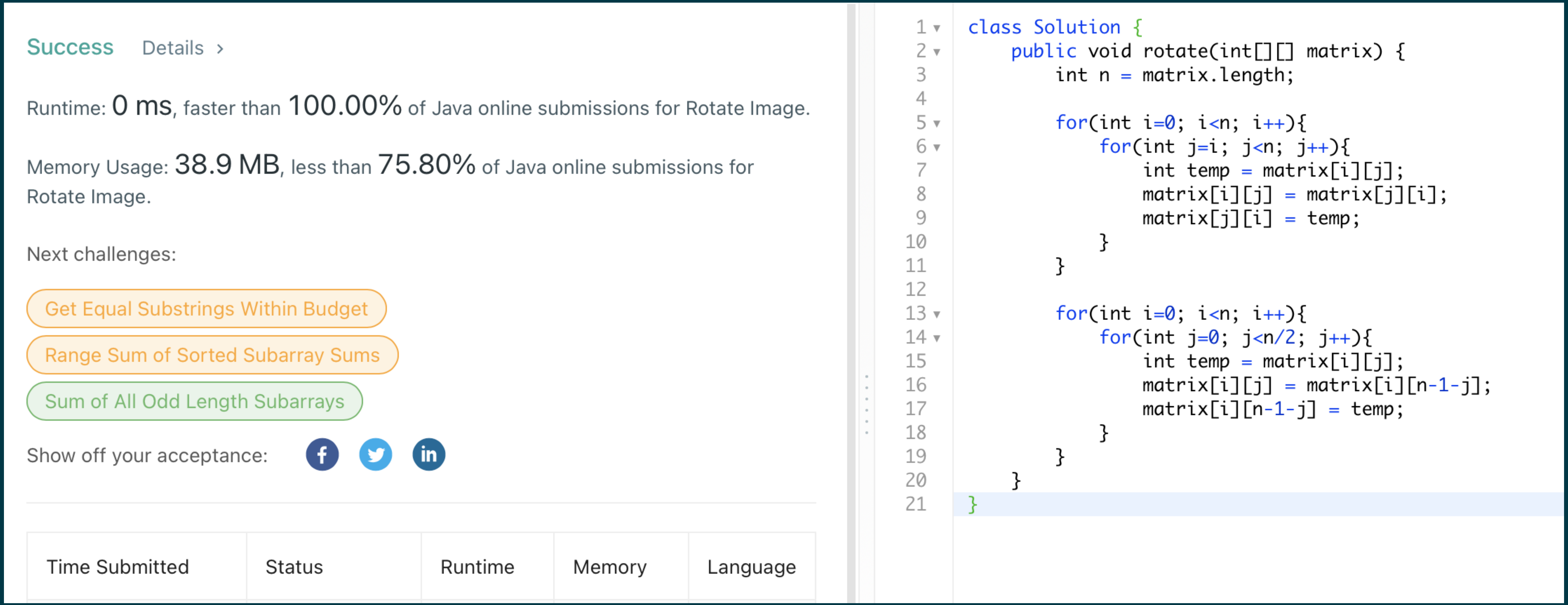


5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16

5	2	13	15
1	4	3	14
9	8	6	12
11	10	7	16

15	13	2	5
14	3	4	1
12	6	8	9
16	7	10	11

First Solution



Success Details ›

Runtime: 0 ms, faster than 100.00% of Java online submissions for Rotate Image.

Memory Usage: 38.9 MB, less than 75.80% of Java online submissions for Rotate Image.

Next challenges:

Get Equal Substrings Within Budget

Range Sum of Sorted Subarray Sums

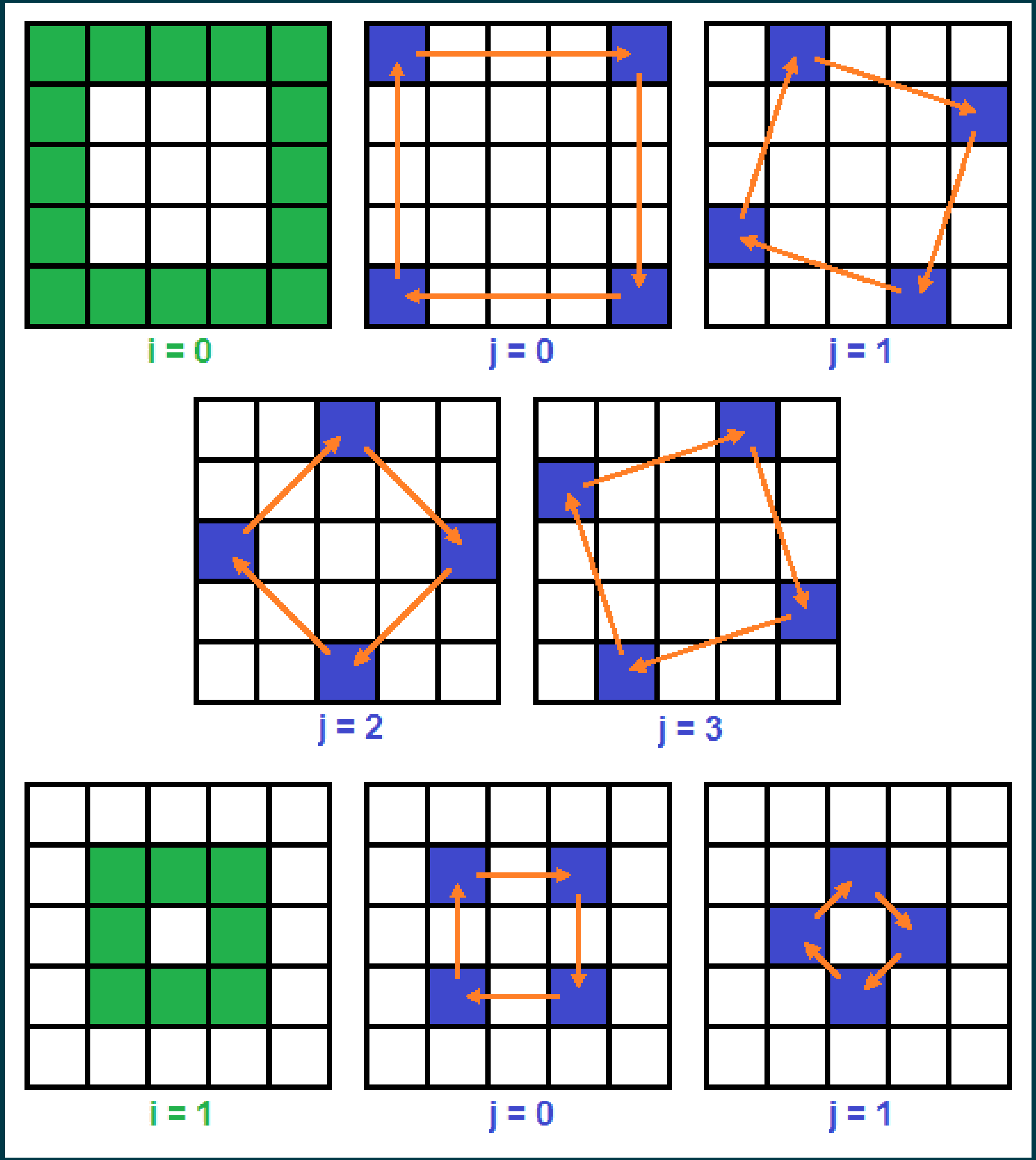
Sum of All Odd Length Subarrays

Show off your acceptance:

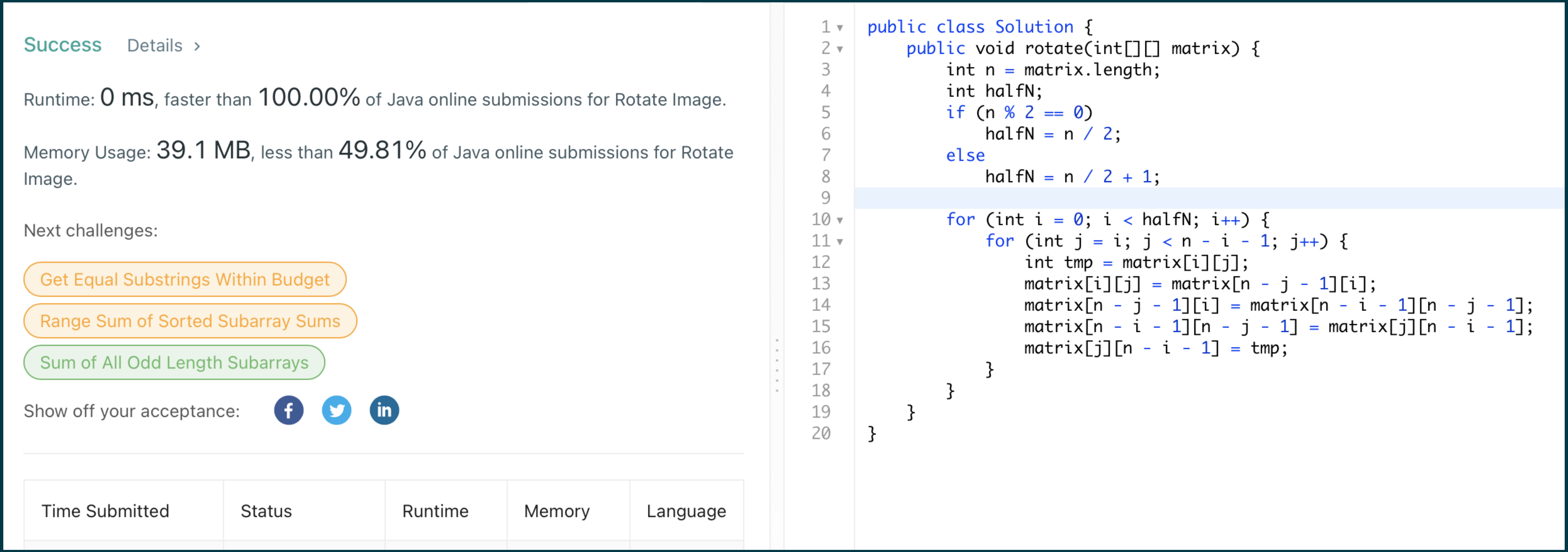
Time Submitted	Status	Runtime	Memory	Language

```java
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;

        for(int i=0; i<n; i++){
            for(int j=i; j<n; j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }

        for(int i=0; i<n; i++){
            for(int j=0; j<n/2; j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[i][n-1-j];
                matrix[i][n-1-j] = temp;
            }
        }
    }
}
```

Second Theory



Second Solution



Success Details ›

Runtime: 0 ms, faster than 100.00% of Java online submissions for Rotate Image.

Memory Usage: 39.1 MB, less than 49.81% of Java online submissions for Rotate Image.

Next challenges:

Get Equal Substrings Within Budget

Range Sum of Sorted Subarray Sums

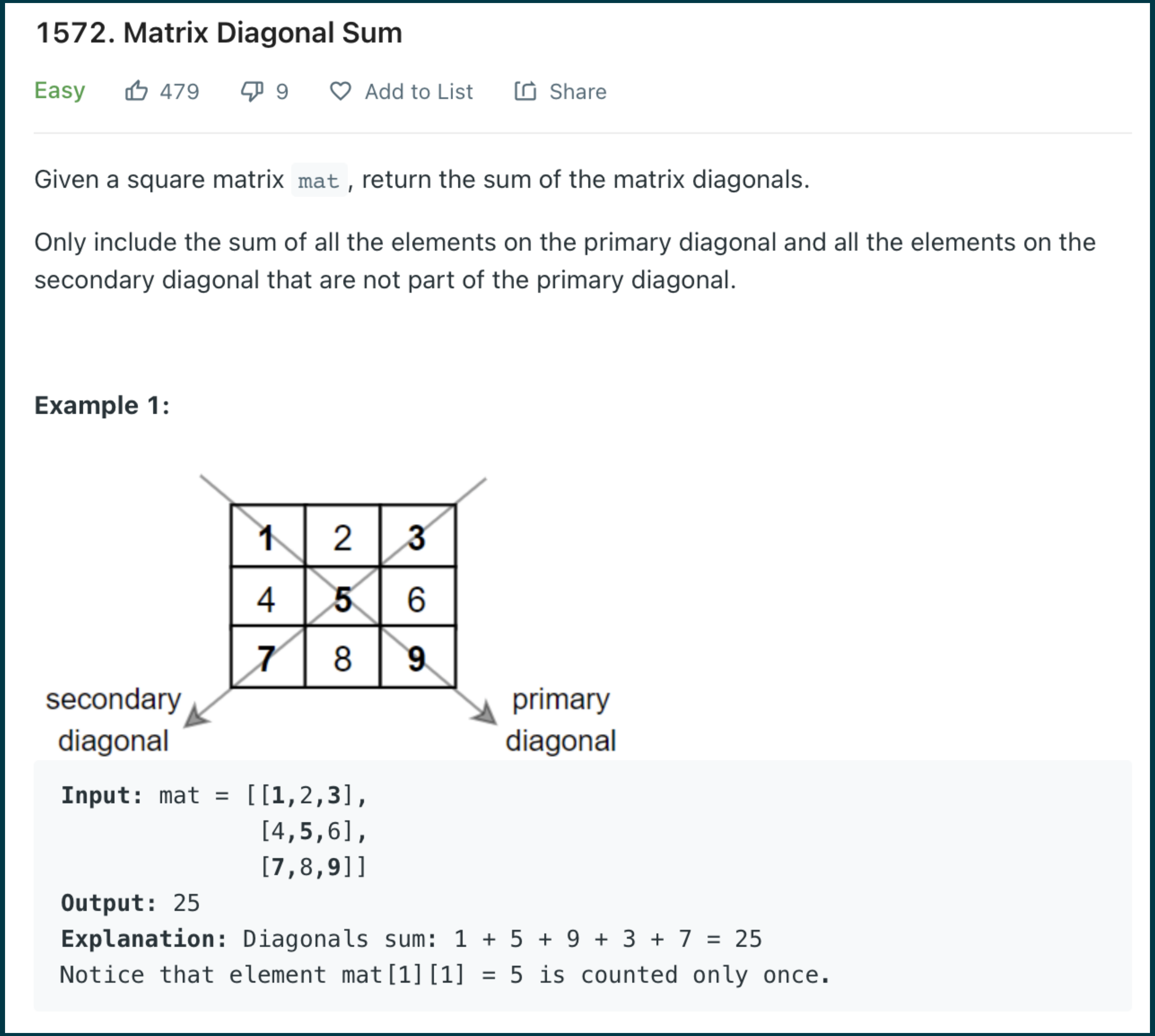
Sum of All Odd Length Subarrays

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language

```java
public class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        int halfN;
        if (n % 2 == 0)
            halfN = n / 2;
        else
            halfN = n / 2 + 1;

        for (int i = 0; i < halfN; i++) {
            for (int j = i; j < n - i - 1; j++) {
                int tmp = matrix[i][j];
                matrix[i][j] = matrix[n - j - 1][i];
                matrix[n - j - 1][i] = matrix[n - i - 1][n - j - 1];
                matrix[n - i - 1][n - j - 1] = matrix[j][n - i - 1];
                matrix[j][n - i - 1] = tmp;
            }
        }
    }
}
```

Task 1 – Matrix Diagonal Sum

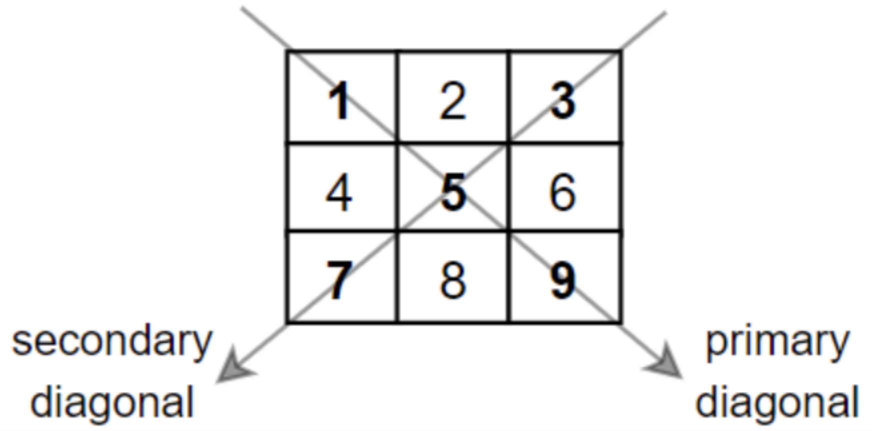


1572. Matrix Diagonal Sum

Given a square matrix `mat`, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

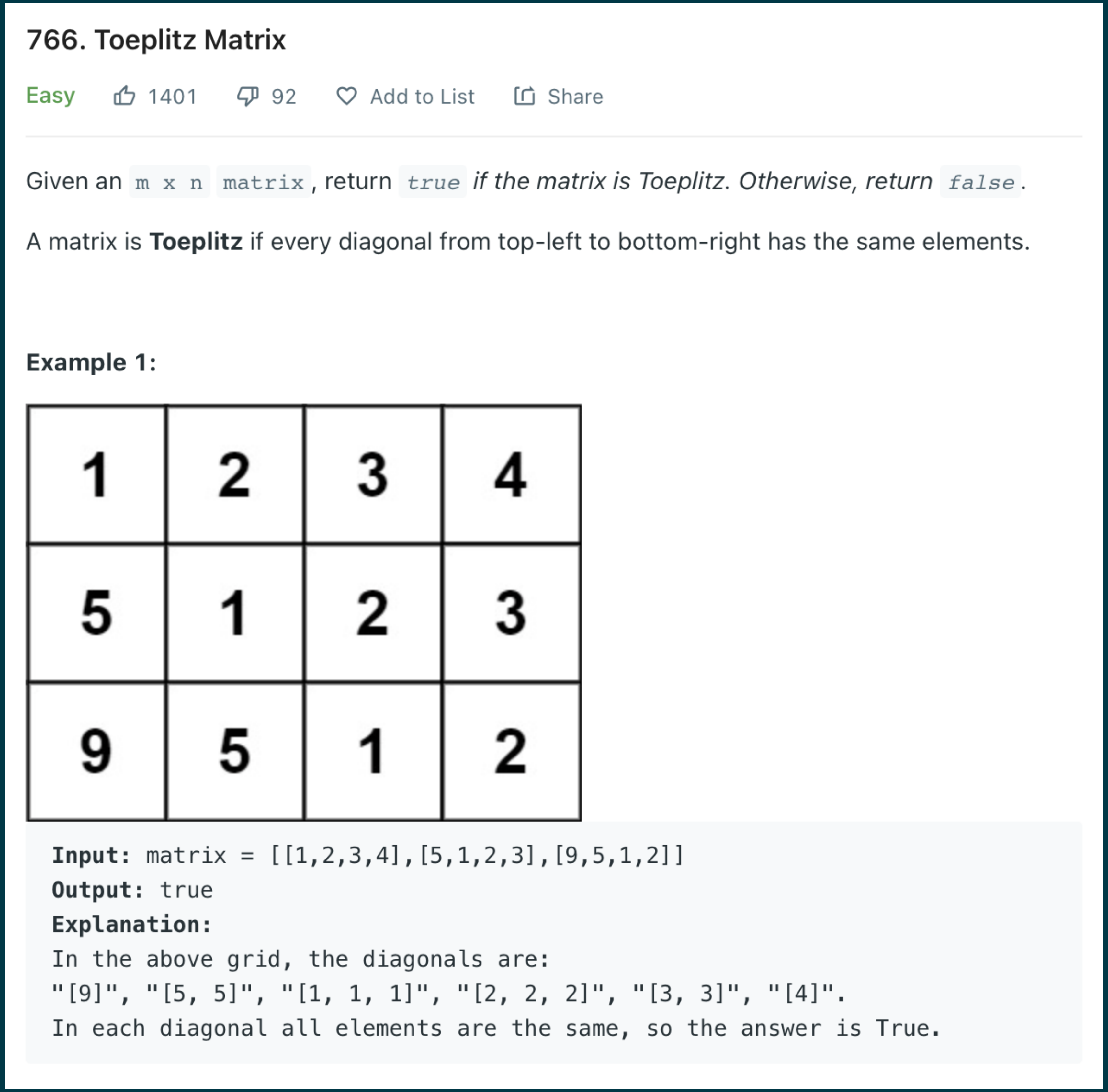
Example 1:



```
Input: mat = [[1,2,3],
              [4,5,6],
              [7,8,9]]
Output: 25
Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25
Notice that element mat[1][1] = 5 is counted only once.
```

https://leetcode.com/problems/matrix-diagonal-sum/

Task 2 – Toeplitz Matrix



766. Toeplitz Matrix

Easy 1401 92 Add to List Share

Given an `m x n` `matrix`, return `true` *if the matrix is Toeplitz. Otherwise, return* `false`.

A matrix is **Toeplitz** if every diagonal from top-left to bottom-right has the same elements.

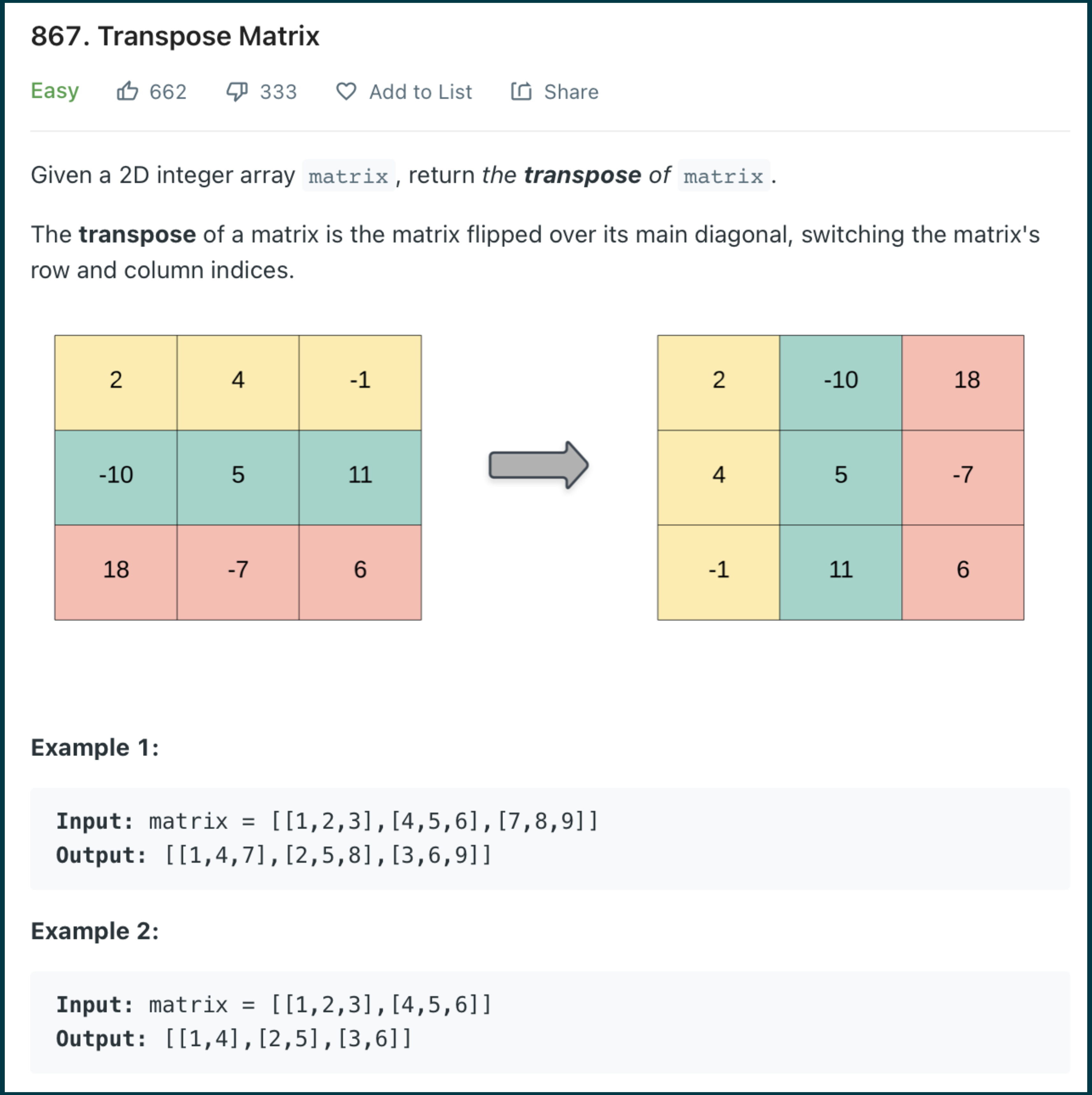
Example 1:

1	2	3	4
5	1	2	3
9	5	1	2

```
Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
Output: true
Explanation:
In the above grid, the diagonals are:
"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".
In each diagonal all elements are the same, so the answer is True.
```

https://leetcode.com/problems/toeplitz-matrix/

Task 3 – Transpose Matrix



867. Transpose Matrix

Easy 662 333 Add to List Share

Given a 2D integer array `matrix`, return *the **transpose** of* `matrix`.

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

2	4	-1
-10	5	11
18	-7	6

$\Rightarrow$

2	-10	18
4	5	-7
-1	11	6

Example 1:

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[1,4,7],[2,5,8],[3,6,9]]
```

Example 2:

```
Input: matrix = [[1,2,3],[4,5,6]]
Output: [[1,4],[2,5],[3,6]]
```

https://leetcode.com/problems/transpose-matrix/