Count Sorted Rows

Basic Accuracy: 32.75% Submissions: 7K+ Points: 1

Given two integers N and M and a matrix of dimensions N*M. Count all the rows in a matrix that are sorted either in strictly increasing order or in strictly decreasing order.

Example 1:

Input:
N=3,M=3
Mat=[[1,2,3],[6,5,4],[7,9,8]]
Output:
2
Explanation:
The first row is sorted in strictly increasing order while the second row is sorted in strictly decreasing order.

Example 2:

Input:
N=3,M=3
Mat=[[1,2,3],[4,5,6],[7,8,9]]
Output:
3
Explanation:
All the rows are sorted in strictly increasing order.

Your Task:
You don't need to read input or print anything.Your task is to complete the function **sortedCount()** which takes the two integers N,M and the matrix Mat as input parameters and returns the number of rows which are sorted in either strictly increasing order or strictly decreasing order.

Expected Time Complexity:O(N*M)
Expected Auxillary Space:O(1)

Constraints:
1<=N,M,Mat[i][j]<=1000

```cpp
// https://www.geeksforgeeks.org/problems/count-sorted-rows2702/1
// Count Sorted Rows

class Solution {
  public:
    bool isIncreasing(vector<int>&v){
        for(int i=1;i<v.size();i++){
            if(v[i]<=v[i-1]) return false;
        }
        return true;
    }
    bool isDecreasing(vector<int>&v){
        for(int i=1;i<v.size();i++){
            if(v[i]>=v[i-1]) return false;
        }
        return true;
    }
    int sortedCount(int N, int M, vector<vector<int>> Mat) {
        int ans=0;
        for(int i=0;i<N;i++){
            if(isIncreasing(Mat[i]) || isDecreasing(Mat[i])) ans++;
        }
        return ans;
    }
};
```

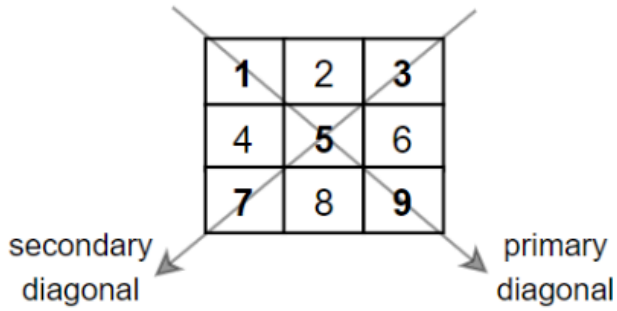
1572. Matrix Diagonal Sum

Easy | Topics | Companies | Hint

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.
```

Example 2:

```
Input: mat = [[1,1,1,1],
              [1,1,1,1],
              [1,1,1,1],
              [1,1,1,1]]
Output: 8
```

Example 3:

```
Input: mat = [[5]]
Output: 5
```

Constraints:

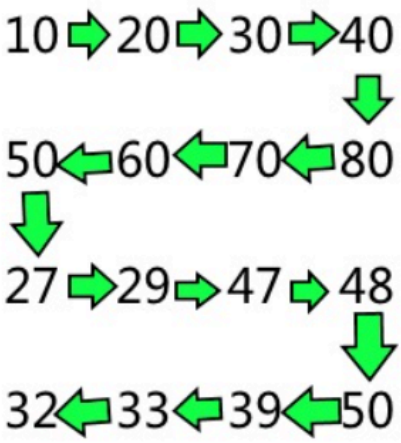
- `n == mat.length == mat[i].length`
- `1 <= n <= 100`
- `1 <= mat[i][j] <= 100`

```cpp
// https://leetcode.com/problems/matrix-diagonal-sum/description/
// Matrix Diagonal Sum

class Solution {
public:
    int diagonalSum(vector<vector<int>>& mat) {
        int ans=0;
        for(int i=0;i<mat.size();i++){
            for(int j=0;j<mat[0].size();j++){
                if(i==j || i+j==mat.size()-1){
                    ans+=mat[i][j];
                }
            }
        }
        return ans;
    }
};
```

Print Matrix in snake Pattern

Easy Accuracy: 68.08% Submissions: 66K+ Points: 2

Given a **matrix** of size **N x N**. Print the elements of the matrix in the snake like pattern depicted below.



Output: 10 20 30 40 80 70 60 50 27 29 47 48 50 39 33 32

Example 1:

Input:
N = 3
matrix[][] = {{45, 48, 54},
{21, 89, 87}
{70, 78, 15}}

Output:
45 48 54 87 89 21 70 78 15

Explanation:
Matrix is as below:
45 48 54
21 89 87
70 78 15
Printing it in snake pattern will lead to
the output as 45 48 54 87 89 21 70 78 15.

Example 2:

Input:
N = 2
matrix[][] = {{1, 2},
{3, 4}}
Output:
1 2 4 3
Explanation:
Matrix is as below:
1 2
3 4
Printing it in snake pattern will
give output as 1 2 4 3.

Your Task:
You dont need to read input or print anything. Complete the function **snakePattern()** that takes matrix as input parameter and returns a list of integers in order of the values visited in the snake pattern.

Expected Time Complexity: O(N * N)
Expected Auxiliary Space: O(N * N) for the resultant list only.

Constraints:
$1 <= N <= 10^3$
$1 <= mat[i][j] <= 10^9$

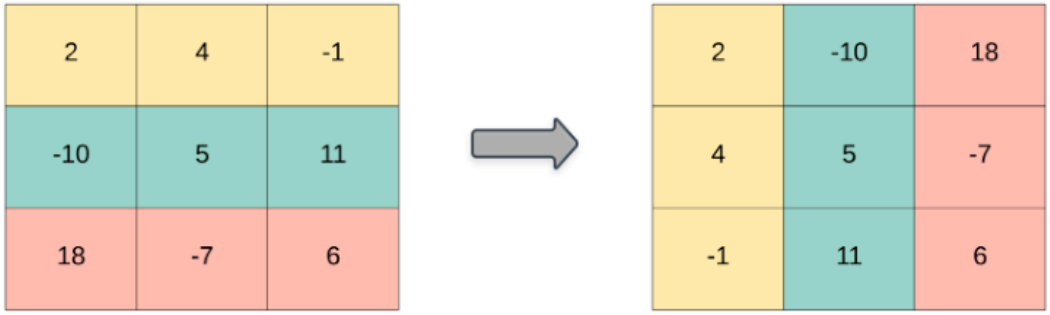
```cpp
// https://www.geeksforgeeks.org/problems/print-matrix-in-snake-pattern-1587115621/1
// Print Matrix in snake Pattern

class Solution
{
    public:
    //Function to return list of integers visited in snake pattern in matrix.
    vector<int> snakePattern(vector<vector<int> > matrix)
    {
        vector<int> ans;
        int n=matrix.size();
        int m=matrix[0].size();
        for(int i=0;i<n;i++){
            if(i%2==0){
                for(int j=0;j<m;j++){
                    ans.push_back(matrix[i][j]);
                }
            }
            else{
                for(int j=m-1;j>=0;j--){
                    ans.push_back(matrix[i][j]);
                }
            }
        }
        return ans;
    }
};
```

867. Transpose Matrix

Easy | Topics | Companies | Hint

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.



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]]
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 1000`
- `1 <= m * n <= 10^5`
- `-10^9 <= matrix[i][j] <= 10^9`

```cpp
// https://leetcode.com/problems/transpose-matrix/description/
// Transpose Matrix

class Solution {
public:
    vector<vector<int>> transpose(vector<vector<int>> A) {
        int M = A.size();
        int N = A[0].size();
        vector<vector<int>> B(N, vector<int>(M, 0));
        for (int j = 0; j < N; j++){
            for (int i = 0; i < M; i++){
                B[j][i] = A[i][j];
            }
        }
        return B;
    }
};
```

1380. Lucky Numbers in a Matrix

Easy | Topics | Companies | Hint

Given an `m x n` matrix of **distinct** numbers, return *all **lucky numbers** in the matrix in **any** order*.

A **lucky number** is an element of the matrix such that it is the minimum element in its row and maximum in its column.

Example 1:

```
Input: matrix = [[3,7,8],[9,11,13],[15,16,17]]
Output: [15]
Explanation: 15 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Example 2:

```
Input: matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]]
Output: [12]
Explanation: 12 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Example 3:

```
Input: matrix = [[7,8],[1,2]]
Output: [7]
Explanation: 7 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= n, m <= 50`
- `1 <= matrix[i][j] <= 10^5`.
- All elements in the matrix are distinct.

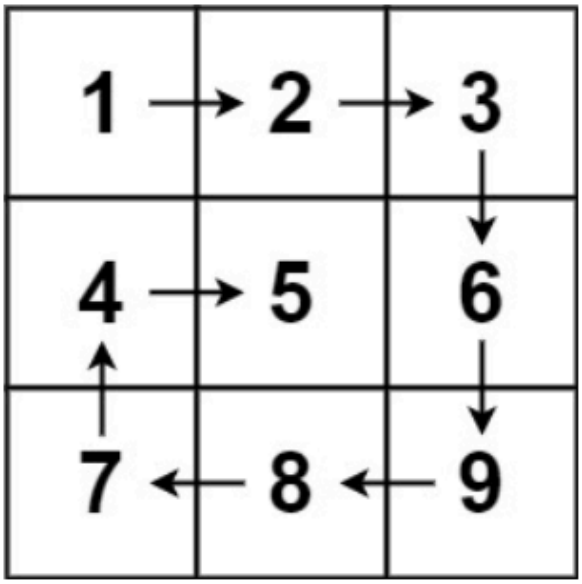
```cpp
// https://leetcode.com/problems/lucky-numbers-in-a-matrix/description/
// Lucky Numbers in a Matrix

class Solution {
public:
    vector<int> luckyNumbers (vector<vector<int>>& matrix) {
        int small, big;
        vector<int> ans;
        for(int i = 0; i < matrix.size(); i++){
            int k = 0;
            small = 999999;
            for(int j = 0; j < matrix[0].size(); j++){
                if(small > matrix[i][j]){
                    small = matrix[i][j];
                    k = j;
                }
            }
            big = small;
            for(int j = 0; j < matrix.size(); j++){
                big = max(big, matrix[j][k]);
            }
            if(small == big) {
                ans.push_back(small);
            }
        }
        return ans;
    }
};
```

54. Spiral Matrix

Medium | Topics | Companies | Hint

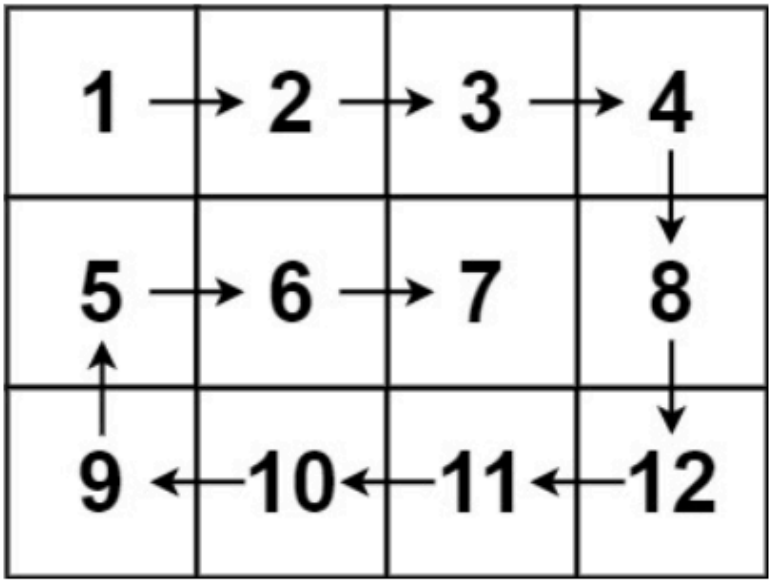
Given an `m x n` `matrix`, return *all elements of the* `matrix` *in spiral order*.

Example 1:




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

Example 2:




```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 10`
- `-100 <= matrix[i][j] <= 100`

```
// https://leetcode.com/problems/spiral-matrix/description/
// leetcode: Spiral Matrix

/*
Initialize variables: m and n store the dimensions of the matrix,top, bot,
    left, and rigt represent the boundaries of the current spiral.

The while loop continues as long as the top boundary is less than or equal
    to the bottom boundary and the left boundary is less than or equal to
    the right boundary. This condition ensures that there are still elements
    to traverse in the matrix.

Traverse the top row from left to right. Append each element to the ans
    vector.

Increment the top boundary since the top row has been traversed.

Traverse the right column from top to bottom. Append each element to the ans
    vector.

Decrement the right boundary since the right column has been traversed.

Check if the top boundary is still less than or equal to the bottom boundary
    (to avoid duplicate traversal).

If the condition is true, traverse the bottom row from right to left. Append
    each element to the ans vector.

Decrement the bottom boundary since the bottom row has been traversed.

Check if the left boundary is still less than or equal to the right boundary
    (to avoid duplicate traversal).

If the condition is true, traverse the left column from bottom to top.
    Append each element to the ans vector.

Increment the left boundary since the left column has been traversed.

Repeat stepsuntil the entire matrix is traversed.

Return the resulting ans vector containing the elements in spiral order.
*/
```

```cpp
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        int m = matrix.size();
        int n = matrix[0].size();
        vector<int>ans;
        int top=0,bot=m-1,left=0,rigt=n-1;

        while(top<=bot && left<=rigt){

           // For moving left to right
            for(int i=left;i<=rigt;i++){
                ans.push_back(matrix[top][i]);
            }
            top++;

             // For moving top to bottom.
            for(int i=top;i<=bot;i++){
                ans.push_back(matrix[i][rigt]);
            }
            rigt--;

            // For moving right to left.
            if(top<=bot){
                for(int i=rigt;i>=left;i--){
                    ans.push_back(matrix[bot][i]);
                }
                bot--;
            }

             // For moving bottom to top.
             if(left<=rigt){
                 for(int i=bot;i>=top;i--){
                     ans.push_back(matrix[i][left]);
                 }
                 left++;
             }
        }
        return ans;
    }
};
```

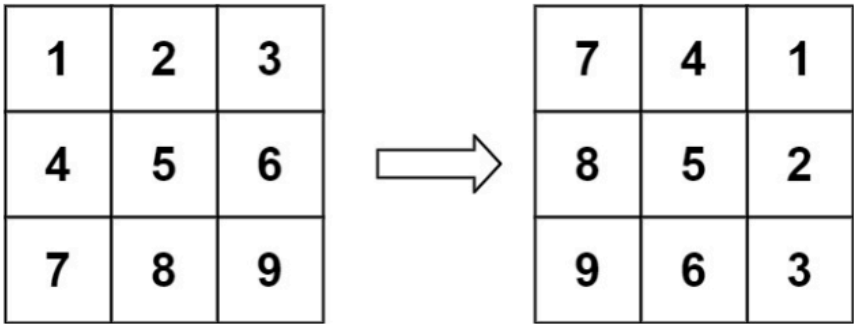
48. Rotate Image

Medium | Topics | Companies

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

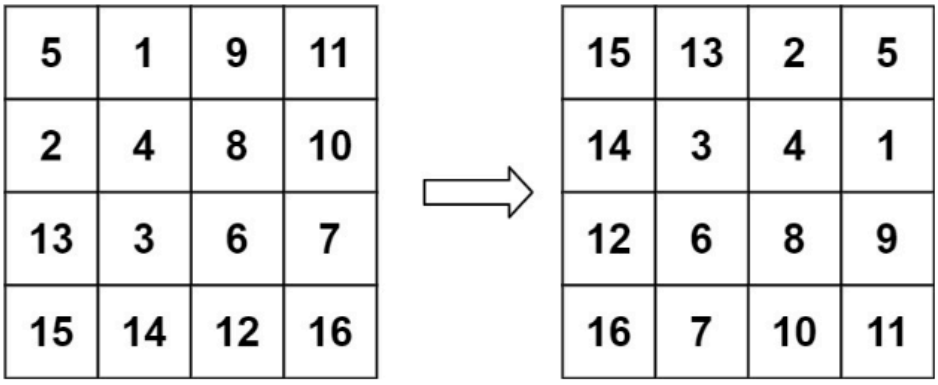
You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

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:




```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

Constraints:

- `n == matrix.length == matrix[i].length`
- `1 <= n <= 20`
- `-1000 <= matrix[i][j] <= 1000`

```cpp
// https://leetcode.com/problems/rotate-image/description/
// Rotate Image

// First transpose the matrix and then reverse the columns

class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n=matrix.size();
        for(int i=0;i<n;i++){
            for(int j=0;j<=i;j++){
                swap(matrix[i][j],matrix[j][i]);
            }
        }
        for(int i=0;i<n;i++){
            reverse(matrix[i].begin(),matrix[i].end());
        }
    }
};
```

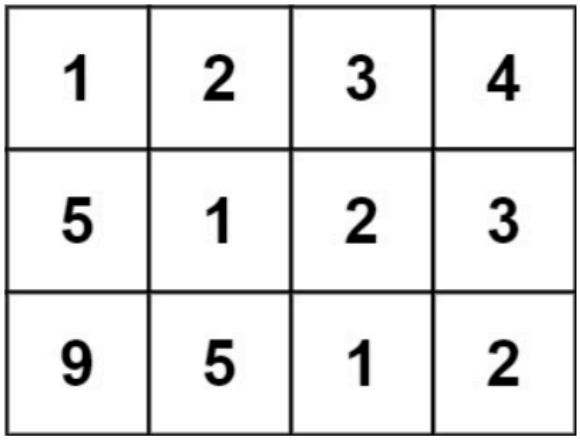
766. Toeplitz Matrix

Easy | Topics | Companies | Hint

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

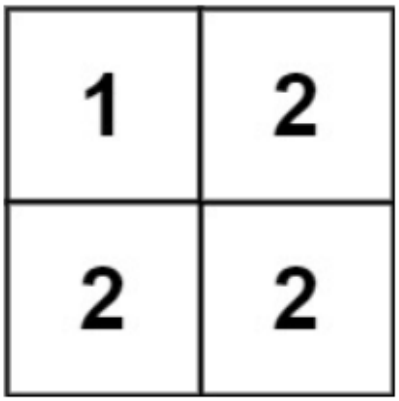
A matrix is **Toeplitz** if every diagonal from top-left to bottom-right has the same elements.

Example 1:




```
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.
```

Example 2:




```
Input: matrix = [[1,2],[2,2]]
Output: false
Explanation:
The diagonal "[1, 2]" has different elements.
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 20`
- `0 <= matrix[i][j] <= 99`

Follow up:

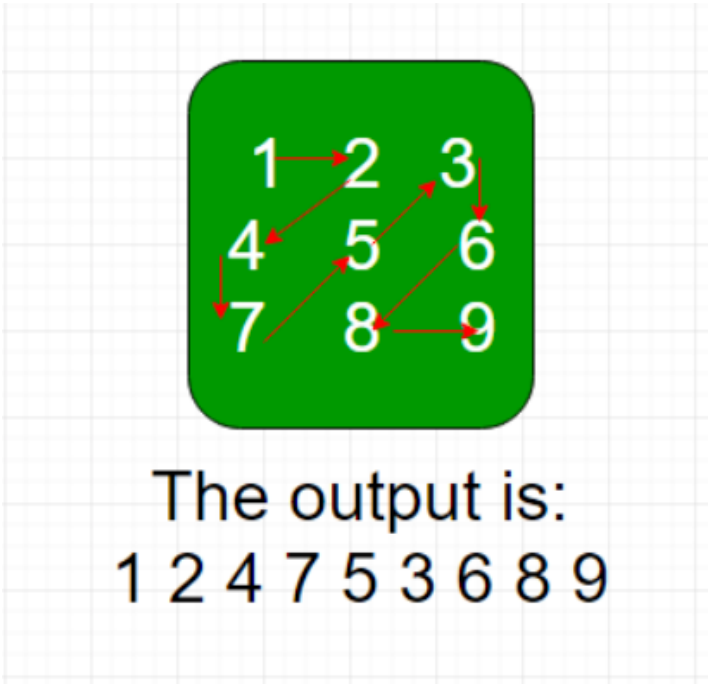
- What if the `matrix` is stored on disk, and the memory is limited such that you can only load at most one row of the matrix into the memory at once?
- What if the `matrix` is so large that you can only load up a partial row into the memory at once?

```cpp
// https://leetcode.com/problems/toeplitz-matrix/description/
// Toeplitz Matrix

class Solution {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        int m = matrix.size(), n = matrix[0].size();
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                if (matrix[i][j] != matrix[i - 1][j - 1])
                    return false;
        return true;
    }
};
```

Print matrix in diagonal pattern

Easy Accuracy: 59.54% Submissions: 5K+ Points: 2

Given a matrix M of n*n size, the task is to complete the function which prints its elements in a diagonal pattern as depicted below.



Example 1:

Input:
N = 3
mat[][] = {{1 2 3},{4 5 6},{7 8 9}}
Output: 1 2 4 7 5 3 6 8 9

Example 2:

Input:
N = 2
mat[][] = {{1 2},{3 4}}
Output: 1 2 3 4

Your Task:

You only need to implement the given function **matrixDiagonally()** which returns a list containing the matrix diagonally. Do not read input, instead use the arguments given in the function. Print the elements in Matrix in a diagonal pattern.

Expected Time Complexity: O(N*M)

Expected Auxiliary Space: O(1)

Constraints:

1<=N<=100

```cpp
// https://www.geeksforgeeks.org/problems/print-matrix-in-diagonal-pattern/1
// Print matrix in diagonal pattern

class Solution{
  public:
    vector<int> matrixDiagonally(vector<vector<int>>&mat)
    {
        vector<int> ans;
        int n=mat.size();
        int m=mat[0].size();
        bool up=true;
        //iterate using first row
        for(int i=0;i<m;i++){
            int ii=0,jj=i;
            vector<int> temp;
            while(ii<n && jj>=0){
                temp.push_back(mat[ii][jj]);
                ii++;
                jj--;
            }
            if(up){
                for(int k=temp.size()-1;k>=0;k--){
                    ans.push_back(temp[k]);
                }
            }
            else{
                for(int k=0;k<temp.size();k++){
                    ans.push_back(temp[k]);
                }
            }
            up=!up;
        }
```

```
33          if(n&1) up=false;
34          else up=true;
35          //iterate using last column
36          for(int i=1;i<n;i++){
37              int ii=i,jj=m-1;
38              vector<int> temp;
39              while(ii<n && jj>=0){
40                  temp.push_back(mat[ii][jj]);
41                  ii++;
42                  jj--;
43              }
44              if(up){
45                  for(int k=temp.size()-1;k>=0;k--){
46                      ans.push_back(temp[k]);
47                  }
48              }
49              else{
50                  for(int k=0;k<temp.size();k++){
51                      ans.push_back(temp[k]);
52                  }
53              }
54              up=!up;
55          }
56          return ans;
57      }
58  };
```