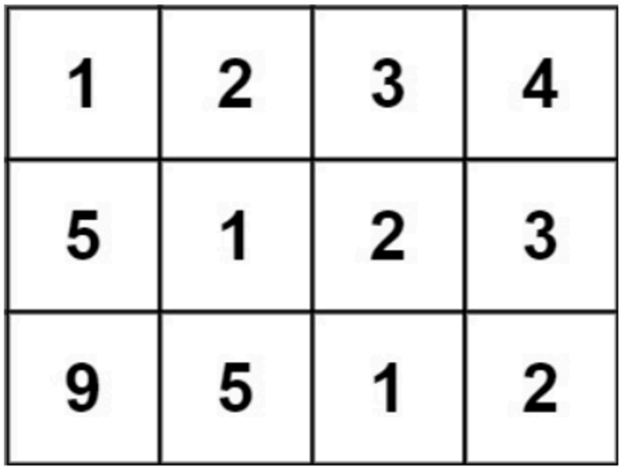
Class Notes 5

Python

<aside> 💡 **Question 1**

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.

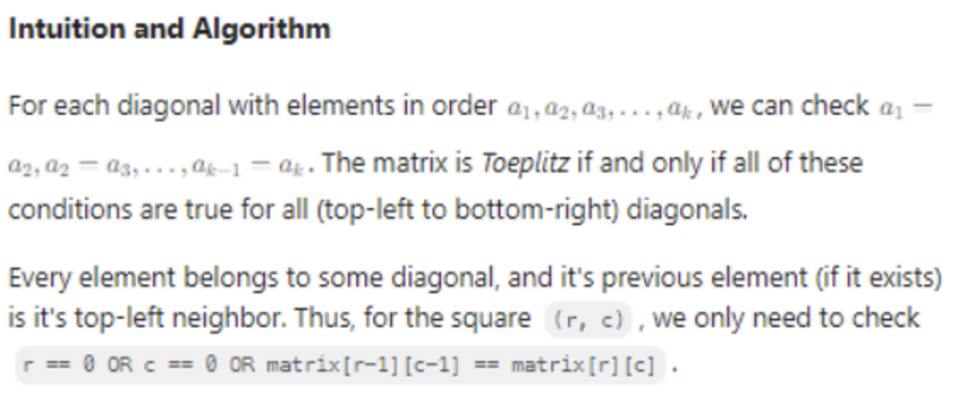


1	2	3	4
5	1	2	3
9	5	1	2

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.

Solution:



Intuition and Algorithm

For each diagonal with elements in order $a_1, a_2, a_3, \dots, a_k$, we can check $a_1 = a_2, a_2 = a_3, \dots, a_{k-1} = a_k$. The matrix is *Toeplitz* if and only if all of these conditions are true for all (top-left to bottom-right) diagonals.

Every element belongs to some diagonal, and it's previous element (if it exists) is it's top-left neighbor. Thus, for the square `(r, c)`, we only need to check `r == 0 OR c == 0 OR matrix[r-1][c-1] == matrix[r][c]`.

Time Complexity: *O*(*M*∗*N*), as defined in the problem statement.

Space Complexity: *O*(1).

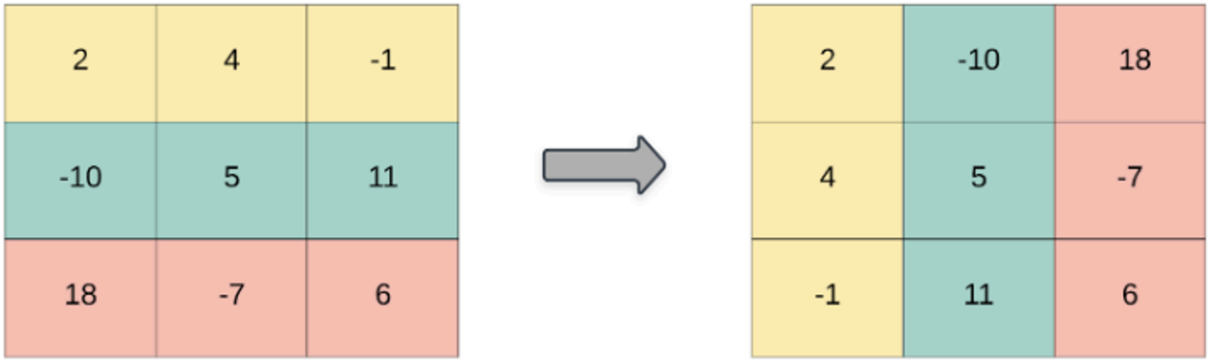
```
def isToeplitzMatrix(matrix):
   for r in range(len(matrix)):
       for c in range(len(matrix[0])):
           if r > 0 and c > 0 and matrix[r-1][c-1] != matrix[r][c]:
               return False
   return True
```

</aside>

<aside> 💡 **Question 2**

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

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

Example 1:

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

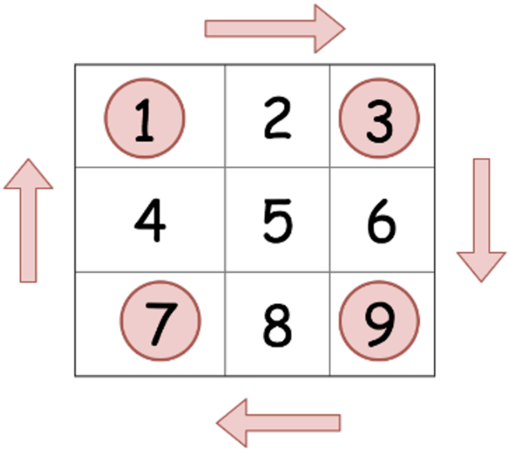
Output:

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

Solution:

Intuition

Observe how the cells move in groups when we rotate the image.



1	2	3
4	5	6
7	8	9

We can iterate over each group of four cells and rotate them.

Complexity Analysis

Let *M* be the number of cells in the matrix.

Time complexity: O(*M*), as each cell is getting read once and written once.

Space complexity: O(1) because we do not use any other additional data structures.

```
def transpose(A):
  R = len(A)
  C = len(A[0])
  ans = [[0 for _ in range(R)] for _ in range(C)]
  for r in range(R):
     for c in range(C):
        ans[c][r] = A[r][c]
  return ans
```

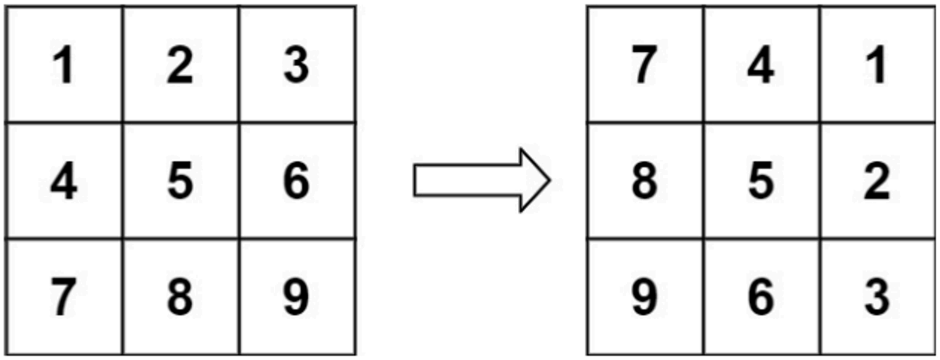
</aside>

<aside> 💡 **Question 3**

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]]

Solution:

Intuition and Algorithm

The transpose of a matrix A with dimensions R x C is a matrix *ans* with dimensions C x R for which ans[c][r] = A[r][c].

We initialize a new matrix *ans* representing the answer. Then, we'll copy each entry of the matrix as appropriate.

Complexity Analysis

Time Complexity: $O(R*C)$, where R and C are the number of rows and columns in the given matrix A.

Space Complexity: $O(R*C)$, the space used by the answer.

```
def rotate(matrix):
    n = len(matrix)
    for i in range((n + 1) // 2):
        for j in range(n // 2):
            temp = matrix[n - 1 - j][i]
            matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1]
            matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 - i]
            matrix[j][n - 1 - i] = matrix[i][j]
            matrix[i][j] = temp
```

- </aside>

  <aside> 💡 **Question 4**

  Given a non-empty array of non-negative integers nums, the **degree** of this array is defined as the maximum frequency of any one of its elements.

  Your task is to find the smallest possible length of a (contiguous) subarray of nums, that has the same degree as nums.

  Example 1:

  Input: nums = [1,2,2,3,1]

  Output: 2

  Explanation:

  The input array has a degree of 2 because both elements 1 and 2 appear twice.

  Of the subarrays that have the same degree:

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

  The shortest length is 2. So return 2.

  Complexity Analysis

    - Time Complexity: O(m*n)
    - Space Complexity: O(1)

Solution:

```
def maximumWealth(accounts):
    # Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
    maxWealthSoFar = 0

    # Iterate over accounts
    for account in accounts:
        # For each account, initialize the sum to 0
        currCustomerWealth = 0
        # Add the money in each bank
        for money in account:
            currCustomerWealth += money
        # Update the maximum wealth seen so far if the current wealth is greater
        maxWealthSoFar = max(maxWealthSoFar, currCustomerWealth)

    # Return the maximum wealth
    return maxWealthSoFar
```

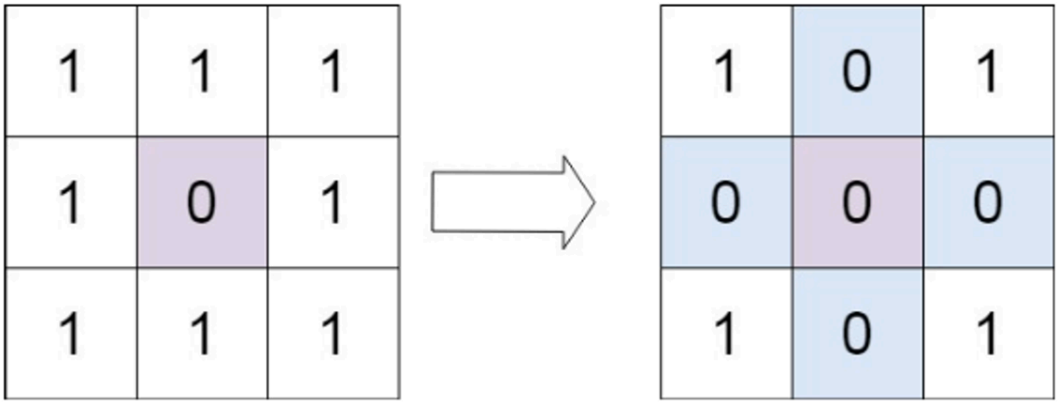
- </aside>

  <aside> 💡 **Question 5**

  Given an m x n integer matrix, if an element is 0, set its entire row and column to 0's.

  You must do it in place.

  Example 1:



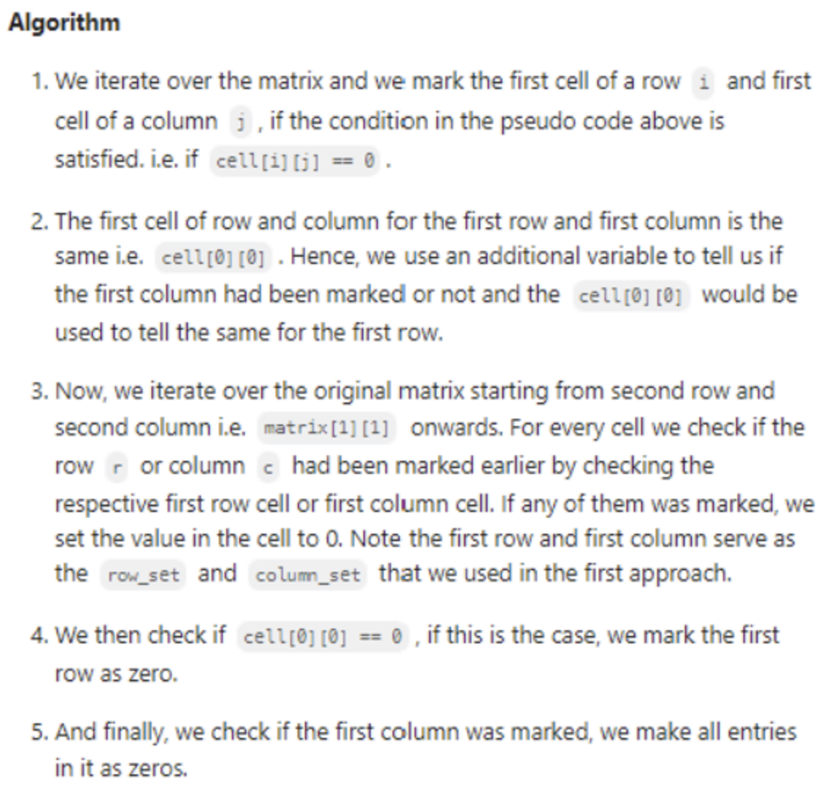
1	1	1
1	0	1
1	1	1

1	0	1
0	0	0
1	0	1

  Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]

  Output: [[1,0,1],[0,0,0],[1,0,1]]

  Solution:



Algorithm

1. We iterate over the matrix and we mark the first cell of a row `i` and first cell of a column `j`, if the condition in the pseudo code above is satisfied. i.e. if `cell[i][j] == 0`.
2. The first cell of row and column for the first row and first column is the same i.e. `cell[0][0]`. Hence, we use an additional variable to tell us if the first column had been marked or not and the `cell[0][0]` would be used to tell the same for the first row.
3. Now, we iterate over the original matrix starting from second row and second column i.e. `matrix[1][1]` onwards. For every cell we check if the row `r` or column `c` had been marked earlier by checking the respective first row cell or first column cell. If any of them was marked, we set the value in the cell to 0. Note the first row and first column serve as the `row_set` and `colum_set` that we used in the first approach.
4. We then check if `cell[0][0] == 0`, if this is the case, we mark the first row as zero.
5. And finally, we check if the first column was marked, we make all entries in it as zeros.

Complexity Analysis

- **Time Complexity:** $O(M×N)$
- **Space Complexity:** $O(1)$

```
def setZeroes(matrix):
    isCol = False
    R = len(matrix)
    C = len(matrix[0])

    for i in range(R):
        if matrix[i][0] == 0:
            isCol = True

        for j in range(1, C):
            if matrix[i][j] == 0:
                matrix[0][j] = 0
                matrix[i][0] = 0

    for i in range(1, R):
        for j in range(1, C):
            if matrix[i][0] == 0 or matrix[0][j] == 0:
```

```
            matrix[i][j] = 0

if matrix[0][0] == 0:
    for j in range(C):
        matrix[0][j] = 0

if isCol:
    for i in range(R):
        matrix[i][0] = 0
```

- </aside>

  <aside> 💡 **Question 6**

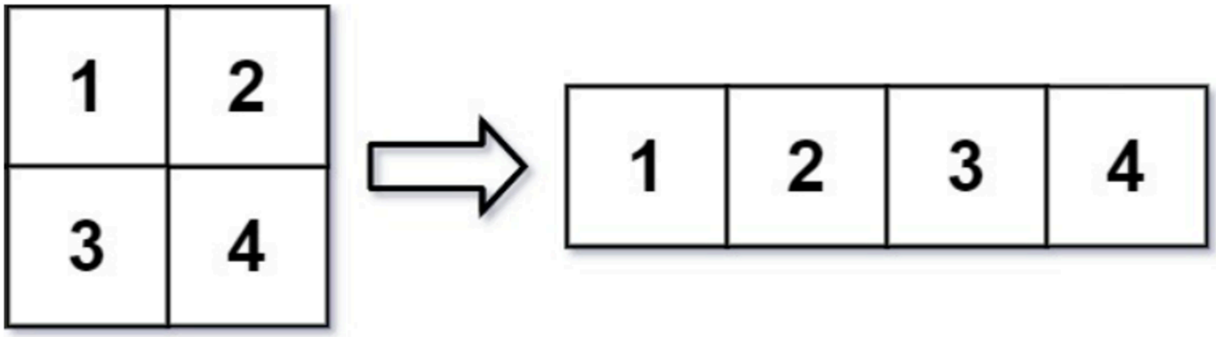
  In MATLAB, there is a handy function called reshape which can reshape an m x n matrix into a new one with a different size r x c keeping its original data.

  You are given an m x n matrix mat and two integers r and c representing the number of rows and the number of columns of the wanted reshaped matrix.

  The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

  If the reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

  Example 1:



  Input: mat = [[1,2],[3,4]], r = 1, c = 4

  Output:

  [[1,2,3,4]]

Solution:

The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again.

The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

Complexity Analysis

- **Time complexity:** $O(m \cdot n)$. We traverse over $m \cdot n$ elements twice. Here, m and n refer to the number of rows and columns of the given matrix respectively.
- **Space complexity:** $O(m \cdot n)$. The queue formed will be of size $m \cdot n$.

```
from collections import deque

def matrixReshape(nums, r, c):
    res = [[0] * c for _ in range(r)]
    if len(nums) == 0 or r * c != len(nums) * len(nums[0]):
        return nums
    queue = deque()
    for i in range(len(nums)):
        for j in range(len(nums[0])):
            queue.append(nums[i][j])
    for i in range(r):
        for j in range(c):
            res[i][j] = queue.popleft()
    return res
```

- </aside>

  <aside> 💡 **Question 7**

  Given an n x n binary matrix image, flip the image **horizontally**, then invert it, and return *the resulting image*.

  To flip an image horizontally means that each row of the image is reversed.

  - For example, flipping [1,1,0] horizontally results in [0,1,1].
- To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0.

- For example, inverting [0,1,1] results in [1,0,0].

Example 1:

Input: image = [[1,1,0],[1,0,1],[0,0,0]]

Output: [[1,0,0],[0,1,0],[1,1,1]]

Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].

Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Solution:

Intuition and Algorithm

We can do this in place. In each row, the ith value from the left is equal to the inverse of the ith value from the right.

We use (C+1) / 2 (with floor division) to iterate over all indexes i in the first half of the row, including the center.

Complexity Analysis

Time Complexity: *O*(*N*), where N is the total number of elements in A.

Space Complexity: *O*(1) in *additional* space complexity.

```
def flipAndInvertImage(A):
    C = len(A[0])
    for row in A:
        for i in range((C + 1) // 2):
            tmp = row[i] ^ 1
            row[i] = row[C - 1 - i] ^ 1
            row[C - 1 - i] = tmp
    return A
```

- </aside>

JavaScript

<aside> 💡 **Question 1**

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.

Solution:

Time Complexity: *O*(*M*∗*N*), as defined in the problem statement.

Space Complexity: *O*(1).

```
function isToeplitzMatrix(matrix) {
  for (let r = 0; r < matrix.length; ++r) {
    for (let c = 0; c < matrix[0].length; ++c) {
      if (r > 0 && c > 0 && matrix[r-1][c-1] !== matrix[r][c]) {
        return false;
      }
    }
  }
  return true;
}
```

</aside>

<aside> 💡 **Question 2**

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]]

Solution:

Intuition

Observe how the cells move in groups when we rotate the image.

We can iterate over each group of four cells and rotate them.

Complexity Analysis

Let *M* be the number of cells in the matrix.

Time complexity: O(*M*), as each cell is getting read once and written once.

Space complexity: O(1) because we do not use any other additional data structures.

```
function transpose(A) {
  const R = A.length;
  const C = A[0].length;
  const ans = new Array(C).fill(0).map(() => new Array(R));
  for (let r = 0; r < R; ++r) {
      for (let c = 0; c < C; ++c) {
          ans[c][r] = A[r][c];
      }
  }
  return ans;
}
```

</aside>

<aside> 💡 **Question 3**

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]]

Solution:

Intuition and Algorithm

The transpose of a matrix A with dimensions R x C is a matrix *ans* with dimensions C x R for which ans[c][r] = A[r][c].

We initialize a new matrix *ans* representing the answer. Then, we'll copy each entry of the matrix as appropriate.

Complexity Analysis

Time Complexity: $O(R*C)$, where R and C are the number of rows and columns in the given matrix A.

Space Complexity: $O(R*C)$, the space used by the answer.

```
function rotate(matrix) {
  const n = matrix.length;
  for (let i = 0; i < Math.floor((n + 1) / 2); i++) {
     for (let j = 0; j < Math.floor(n / 2); j++) {
        const temp = matrix[n - 1 - j][i];
```

```
            matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
            matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 - i];
            matrix[j][n - 1 - i] = matrix[i][j];
            matrix[i][j] = temp;
        }
    }
}
```

- </aside>

  <aside> 💡 **Question 4**

  Given a non-empty array of non-negative integers nums, the **degree** of this array is defined as the maximum frequency of any one of its elements.

  Your task is to find the smallest possible length of a (contiguous) subarray of nums, that has the same degree as nums.

  Example 1:

  Input: nums = [1,2,2,3,1]

  Output: 2

  Explanation:

  The input array has a degree of 2 because both elements 1 and 2 appear twice.

  Of the subarrays that have the same degree:

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

  The shortest length is 2. So return 2.

  Complexity Analysis

  - Time Complexity: O(m*n)
  - Space Complexity: O(1)

Solution:

```
 function maximumWealth(accounts) {
   // Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
   let maxWealthSoFar = 0;
```

```
  // Iterate over accounts
  for (let i = 0; i < accounts.length; i++) {
    const account = accounts[i];
    // For each account, initialize the sum to 0
    let currCustomerWealth = 0;
    // Add the money in each bank
    for (let j = 0; j < account.length; j++) {
      const money = account[j];
      currCustomerWealth += money;
    }
    // Update the maximum wealth seen so far if the current wealth is greater
    maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWealth);
  }

  // Return the maximum wealth
  return maxWealthSoFar;
}
```

- </aside>

  <aside> 💡 **Question 5**

  Given an m x n integer matrix, if an element is 0, set its entire row and column to 0's.

  You must do it in place.

  Example 1:

  Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]

  Output: [[1,0,1],[0,0,0],[1,0,1]]

  Solution:

Complexity Analysis

- **Time Complexity:** $O(M \times N)$
- **Space Complexity:** $O(1)$

```
function setZeroes(matrix) {
   let isCol = false;
   const R = matrix.length;
   const C = matrix[0].length;

   for (let i = 0; i < R; i++) {
      if (matrix[i][0] === 0) {
         isCol = true;
      }

      for (let j = 1; j < C; j++) {
         if (matrix[i][j] === 0) {
            matrix[0][j] = 0;
            matrix[i][0] = 0;
         }
      }
   }

   for (let i = 1; i < R; i++) {
      for (let j = 1; j < C; j++) {
         if (matrix[i][0] === 0 || matrix[0][j] === 0) {
            matrix[i][j] = 0;
         }
      }
   }

   if (matrix[0][0] === 0) {
      for (let j = 0; j < C; j++) {
         matrix[0][j] = 0;
      }
   }

   if (isCol) {
      for (let i = 0; i < R; i++) {
         matrix[i][0] = 0;
      }
   }
}
```

- </aside>

  <aside> 💡 **Question 6**

  In MATLAB, there is a handy function called reshape which can reshape an m x n matrix into a new one with a different size r x c keeping its original data.

  You are given an m x n matrix mat and two integers r and c representing the number of rows and the number of columns of the wanted reshaped matrix.

  The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

  If the reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

  Example 1:

  Input: mat = [[1,2],[3,4]], r = 1, c = 4

  Output:

  [[1,2,3,4]]

  Solution:

  The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again.

  The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

  Complexity Analysis

  - **Time complexity:** $O(m \cdot n)$. We traverse over $m \cdot n$ elements twice. Here, m and n refer to the number of rows and columns of the given matrix respectively.

- **Space complexity:** $O(m \cdot n)$. The queue formed will be of size $m \cdot n$.

```
function matrixReshape(nums, r, c) {
    const res = new Array(r).fill(0).map(() => new Array(c).fill(0));
    if (nums.length === 0 || r * c !== nums.length * nums[0].length)
        return nums;
    const queue = [];
    for (let i = 0; i < nums.length; i++) {
        for (let j = 0; j < nums[0].length; j++) {
            queue.push(nums[i][j]);
        }
    }
    for (let i = 0; i < r; i++) {
        for (let j = 0; j < c; j++) {
            res[i][j] = queue.shift();
        }
    }
    return res;
}
```

- </aside>

  <aside> 💡 **Question 7**

  Given an n x n binary matrix image, flip the image **horizontally**, then invert it, and return *the resulting image*.

  To flip an image horizontally means that each row of the image is reversed.

  - For example, flipping [1,1,0] horizontally results in [0,1,1].
- To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0.

  - For example, inverting [0,1,1] results in [1,0,0].

Example 1:

Input: image = [[1,1,0],[1,0,1],[0,0,0]]

Output: [[1,0,0],[0,1,0],[1,1,1]]

Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].

Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Solution:

Intuition and Algorithm

We can do this in place. In each row, the ith value from the left is equal to the inverse of the ith value from the right.

We use (C+1) / 2 (with floor division) to iterate over all indexes i in the first half of the row, including the center.

Complexity Analysis

Time Complexity: *O*(*N*), where N is the total number of elements in A.

Space Complexity: *O*(1) in *additional* space complexity.

```
 function flipAndInvertImage(A) {
    const C = A[0].length;
    for (let row of A) {
        for (let i = 0; i < Math.floor((C + 1) / 2); ++i) {
            const tmp = row[i] ^ 1;
            row[i] = row[C - 1 - i] ^ 1;
            row[C - 1 - i] = tmp;
        }
    }
    return A;
}
```

- </aside>

Java

<aside> 💡 **Question 1**

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.

Solution:

Time Complexity: *O*(*M*∗*N*), as defined in the problem statement.

Space Complexity: *O*(1).

```
 class Solution {
    public boolean isToeplitzMatrix(int[][] matrix) {
        for (int r = 0; r < matrix.length; ++r)
            for (int c = 0; c < matrix[0].length; ++c)
                if (r > 0 && c > 0 && matrix[r-1][c-1] != matrix[r][c])
                    return false;
        return true;
    }
}
```

</aside>

<aside> 💡 **Question 2**

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]]

Solution:

Intuition

Observe how the cells move in groups when we rotate the image.

We can iterate over each group of four cells and rotate them.

Complexity Analysis

Let *M* be the number of cells in the matrix.

Time complexity: O(*M*), as each cell is getting read once and written once.

Space complexity: O(1) because we do not use any other additional data structures.

```
class Solution {
  public static int[][] transpose(int[][] A) {
  int R = A.length;
  int C = A[0].length;
  int[][] ans = new int[C][R];

  for (int r = 0; r < R; ++r) {
    for (int c = 0; c < C; ++c) {
      ans[c][r] = A[r][c];
    }
  }
  return ans;
      }
}
```

</aside>

<aside> 💡 **Question 3**

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]]

Solution:

Intuition and Algorithm

The transpose of a matrix A with dimensions R x C is a matrix *ans* with dimensions C x R for which ans[c][r] = A[r][c].

We initialize a new matrix *ans* representing the answer. Then, we'll copy each entry of the matrix as appropriate.

Complexity Analysis

Time Complexity: $O(R*C)$, where R and C are the number of rows and columns in the given matrix A.

Space Complexity: $O(R*C)$, the space used by the answer.

```
class Solution {
   public void rotate(int[][] matrix) {
       int n = matrix.length;
       for (int i = 0; i < (n + 1) / 2; i ++) {
           for (int j = 0; j < n / 2; j++) {
               int temp = matrix[n - 1 - j][i];
               matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
               matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 -i];
               matrix[j][n - 1 - i] = matrix[i][j];
               matrix[i][j] = temp;
           }
       }
   }
}
```

- </aside>

  <aside> 💡 **Question 4**

  Given a non-empty array of non-negative integers nums, the **degree** of this array is defined as the maximum frequency of any one of its elements.

  Your task is to find the smallest possible length of a (contiguous) subarray of nums, that has the same degree as nums.

  Example 1:

  Input: nums = [1,2,2,3,1]

  Output: 2

  Explanation:

  The input array has a degree of 2 because both elements 1 and 2 appear twice.

  Of the subarrays that have the same degree:

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

  The shortest length is 2. So return 2.

  Complexity Analysis

  - Time Complexity: O(m*n)
  - Space Complexity: O(1)

Solution:

```
class Solution {
  public int maximumWealth(int[][] accounts) {
     // Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
     int maxWealthSoFar = 0;

     // Iterate over accounts
     for (int[] account : accounts) {
        // For each account, initialize the sum to 0
        int currCustomerWealth = 0;
        // Add the money in each bank
        for (int money : account) {
                currCustomerWealth += money;
            }
            // Update the maximum wealth seen so far if the current wealth is greater
            // If it is less than the current sum
            maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWealth);
        }

        // Return the maximum wealth
        return maxWealthSoFar;
    }
}
```

- </aside>

  <aside> 💡 **Question 5**

  Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

  You must do it in place.

  Example 1:

  Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]

  Output: [[1,0,1],[0,0,0],[1,0,1]]

  Solution:

  Complexity Analysis

  - **Time Complexity:** $O(M×N)$
  - **Space Complexity:** $O(1)$

```
class Solution {
  public void setZeroes(int[][] matrix) {
    Boolean isCol = false;
    int R = matrix.length;
    int C = matrix[0].length;

    for (int i = 0; i < R; i++) {

      // Since first cell for both first row and first column is the same i.e. matrix[0][0]
      // We can use an additional variable for either the first row/column.
      // For this solution we are using an additional variable for the first column
      // and using matrix[0][0] for the first row.
      if (matrix[i][0] == 0) {
        isCol = true;
      }

      for (int j = 1; j < C; j++) {
        // If an element is zero, we set the first element of the corresponding row and column to 0
        if (matrix[i][j] == 0) {
          matrix[0][j] = 0;
          matrix[i][0] = 0;
        }
      }
    }

    // Iterate over the array once again and using the first row and first column, update the
elements.
    for (int i = 1; i < R; i++) {
      for (int j = 1; j < C; j++) {
        if (matrix[i][0] == 0 || matrix[0][j] == 0) {
          matrix[i][j] = 0;
        }
      }
    }

    // See if the first row needs to be set to zero as well
    if (matrix[0][0] == 0) {
      for (int j = 0; j < C; j++) {
        matrix[0][j] = 0;
      }
    }

    // See if the first column needs to be set to zero as well
    if (isCol) {
      for (int i = 0; i < R; i++) {
        matrix[i][0] = 0;
      }
    }
  }
}
```

- </aside>

  <aside> 💡 **Question 6**

  In MATLAB, there is a handy function called reshape which can reshape an m x n matrix into a new one with a different size r x c keeping its original data.

  You are given an m x n matrix mat and two integers r and c representing the number of rows and the number of columns of the wanted reshaped matrix.

  The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

  If the reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

  Example 1:

  Input: mat = [[1,2],[3,4]], r = 1, c = 4

  Output:

  [[1,2,3,4]]

  Solution:

  The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again.

  The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

Complexity Analysis

- **Time complexity:** $O(m \cdot n)$. We traverse over $m \cdot n$ elements twice. Here, m and n refer to the number of rows and columns of the given matrix respectively.
- **Space complexity:** $O(m \cdot n)$. The queue formed will be of size $m \cdot n$.

```
class Solution {
    public int[][] matrixReshape(int[][] nums, int r, int c) {
        int[][] res = new int[r][c];
        if (nums.length == 0 || r * c != nums.length * nums[0].length)
            return nums;
        Queue<Integer> queue = new LinkedList();
        for (int i = 0; i < nums.length; i++) {
            for (int j = 0; j < nums[0].length; j++) {
                queue.add(nums[i][j]);
            }
        }
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                res[i][j] = queue.remove();
            }
        }
        return res;
    }
}
```

- </aside>

  <aside> 💡 **Question 7**

  Given an n x n binary matrix image, flip the image **horizontally**, then invert it, and return *the resulting image*.

  To flip an image horizontally means that each row of the image is reversed.

  - For example, flipping [1,1,0] horizontally results in [0,1,1].
- To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0.
  - For example, inverting [0,1,1] results in [1,0,0].

Example 1:

Input: image = [[1,1,0],[1,0,1],[0,0,0]]

Output: [[1,0,0],[0,1,0],[1,1,1]]

Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].

Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Solution:

Intuition and Algorithm

We can do this in place. In each row, the ith value from the left is equal to the inverse of the ith value from the right.

We use (C+1) / 2 (with floor division) to iterate over all indexes i in the first half of the row, including the center.

Complexity Analysis

Time Complexity: *O*(*N*), where N is the total number of elements in A.

Space Complexity: *O*(1) in *additional* space complexity.

```
class Solution {
  public int[][] flipAndInvertImage(int[][] A) {
      int C = A[0].length;
      for (int[] row: A)
          for (int i = 0; i < (C + 1) / 2; ++i) {
              int tmp = row[i] ^ 1;
              row[i] = row[C - 1 - i] ^ 1;
              row[C - 1 - i] = tmp;
          }

      return A;
  }
}
```

- </aside>

C++

<aside> 💡 **Question 1**

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.

Solution:

Time Complexity: *O*(*M*∗*N*), as defined in the problem statement.

Space Complexity: *O*(1).

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

</aside>

<aside> 💡 **Question 2**

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]]

Solution:

Intuition

Observe how the cells move in groups when we rotate the image.

We can iterate over each group of four cells and rotate them.

Complexity Analysis

Let *M* be the number of cells in the matrix.

Time complexity: O(*M*), as each cell is getting read once and written once.

Space complexity: O(1) because we do not use any other additional data structures.

```
 class Solution {
public:
    vector<vector<int>> transpose(vector<vector<int>>& matrix) {
        int R = matrix.size();
        int C = matrix[0].size();
        vector<vector<int>> ans(C, vector<int>(R));

        for (int r = 0; r < R; ++r) {
            for (int c = 0; c < C; ++c) {
                ans[c][r] = matrix[r][c];
            }
        }
        return ans;
    }
};
```

</aside>

<aside> 💡 **Question 3**

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](https://en.wikipedia.org/wiki/In-place_algorithm), 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]]

Solution:

Intuition and Algorithm

The transpose of a matrix A with dimensions R x C is a matrix *ans* with dimensions C x R for which ans[c][r] = A[r][c].

We initialize a new matrix *ans* representing the answer. Then, we'll copy each entry of the matrix as appropriate.

Complexity Analysis

Time Complexity: $O(R*C)$, where R and C are the number of rows and columns in the given matrix A.

Space Complexity: $O(R*C)$, the space used by the answer.

```
 class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
        for (int i = 0; i < (n + 1) / 2; i++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[n - 1 - j][i];
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 - i];
                matrix[j][n - 1 - i] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }
};
```

- </aside>

  <aside> 💡 **Question 4**

  Given a non-empty array of non-negative integers nums, the **degree** of this array is defined as the maximum frequency of any one of its elements.

  Your task is to find the smallest possible length of a (contiguous) subarray of nums, that has the same degree as nums.

  Example 1:

  Input: nums = [1,2,2,3,1]

  Output: 2

  Explanation:

  The input array has a degree of 2 because both elements 1 and 2 appear twice.

  Of the subarrays that have the same degree:

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

  The shortest length is 2. So return 2.

Complexity Analysis

- Time Complexity: O(m*n)
- Space Complexity: O(1)

Solution:

```
 class Solution {
public:
    int maximumWealth(vector<vector<int>>& accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimum wealth possible)
        int maxWealthSoFar = 0;

        // Iterate over accounts
        for (const vector<int>& account : accounts) {
            // For each account, initialize the sum to 0
            int currCustomerWealth = 0;
            // Add the money in each bank
            for (int money : account) {
                currCustomerWealth += money;
            }
            // Update the maximum wealth seen so far if the current wealth is greater
            // than the current sum
            maxWealthSoFar = max(maxWealthSoFar, currCustomerWealth);
        }

        // Return the maximum wealth
        return maxWealthSoFar;
    }
};
```

- </aside>

  <aside> 💡 **Question 5**

  Given an m x n integer matrix, if an element is 0, set its entire row and column to 0's.

  You must do it in place.

  Example 1:

Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]

Output: [[1,0,1],[0,0,0],[1,0,1]]

Solution:

Complexity Analysis

- **Time Complexity:** *O*(*M*×*N*)
- **Space Complexity:** *O*(1)

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        bool isCol = false;
        int R = matrix.size();
        int C = matrix[0].size();

        for (int i = 0; i < R; i++) {
            if (matrix[i][0] == 0) {
                isCol = true;
            }

            for (int j = 1; j < C; j++) {
                if (matrix[i][j] == 0) {
                    matrix[0][j] = 0;
                    matrix[i][0] = 0;
                }
            }
        }

        for (int i = 1; i < R; i++) {
            for (int j = 1; j < C; j++) {
                if (matrix[i][0] == 0 || matrix[0][j] == 0) {
                    matrix[i][j] = 0;
                }
            }
        }
        if (matrix[0][0] == 0) {
            for (int j = 0; j < C; j++) {
                matrix[0][j] = 0;
            }
        }

        if (isCol) {
            for (int i = 0; i < R; i++) {
                matrix[i][0] = 0;
            }
        }
    }
};
```

- </aside>

  <aside> 💡 **Question 6**

  In MATLAB, there is a handy function called reshape which can reshape an m x n matrix into a new one with a different size r x c keeping its original data.

  You are given an m x n matrix mat and two integers r and c representing the number of rows and the number of columns of the wanted reshaped matrix.

  The reshaped matrix should be filled with all the elements of the original matrix in the same row-traversing order as they were.

  If the reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

  Example 1:

  Input: mat = [[1,2],[3,4]], r = 1, c = 4

  Output:

  [[1,2,3,4]]

  Solution:

The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again.

The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

Complexity Analysis

- **Time complexity:** $O(m \cdot n)$. We traverse over $m \cdot n$ elements twice. Here, m and n refer to the number of rows and columns of the given matrix respectively.
- **Space complexity:** $O(m \cdot n)$. The queue formed will be of size $m \cdot n$.

```
class Solution {
public:
    vector<vector<int>> matrixReshape(vector<vector<int>>& nums, int r, int c) {
        vector<vector<int>> res(r, vector<int>(c));
        if (nums.size() == 0 || r * c != nums.size() * nums[0].size())
            return nums;

        queue<int> queue;
        for (int i = 0; i < nums.size(); i++) {
            for (int j = 0; j < nums[0].size(); j++) {
                queue.push(nums[i][j]);
            }
        }

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                res[i][j] = queue.front();
                queue.pop();
            }
        }
        return res;
    }
};
```

- </aside>

  <aside> 💡 **Question 7**

  Given an n x n binary matrix image, flip the image **horizontally**, then invert it, and return *the resulting image*.

To flip an image horizontally means that each row of the image is reversed.

  - For example, flipping [1,1,0] horizontally results in [0,1,1].
- To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0.
  - For example, inverting [0,1,1] results in [1,0,0].

Example 1:

Input: image = [[1,1,0],[1,0,1],[0,0,0]]

Output: [[1,0,0],[0,1,0],[1,1,1]]

Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].

Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Solution:

Intuition and Algorithm

We can do this in place. In each row, the ith value from the left is equal to the inverse of the ith value from the right.

We use (C+1) / 2 (with floor division) to iterate over all indexes i in the first half of the row, including the center.

Complexity Analysis

Time Complexity: *O*(*N*), where N is the total number of elements in A.

Space Complexity: *O*(1) in *additional* space complexity.

```
class Solution {
public:
    vector<vector<int>> flipAndInvertImage(vector<vector<int>>& A) {
        int C = A[0].size();
        for (vector<int>& row : A) {
            for (int i = 0; i < (C + 1) / 2; ++i) {
                int tmp = row[i] ^ 1;
                row[i] = row[C - 1 - i] ^ 1;
                row[C - 1 - i] = tmp;
            }
        }

        return A;
    }
};
```

- </aside>