Rotate Image LeetCode

Given a square matrix of integers, rotate the matrix 90 degrees clockwise.

Input:

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

Output:

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

Input:

```
matrix = [
  [11, 14, 17, 18, 19],
  [12, 33, 23, 19, 21],
  [65, 78, 99, 98, 54],
  [31, 32, 88, 91, 67],
  [66, 75, 55, 78, 89]
]
```

Output:

```
matrix = [
  [66, 31, 65, 12, 11],
  [75, 32, 78, 33, 14],
  [55, 88, 99, 23, 17],
  [78, 91, 98, 19, 18],
  [89, 67, 54, 21, 19]
]
```

1. **Approach 1: Function to rotate the matrix by creating a temporary matrix**

   - In this approach, we create a new temporary matrix to store the rotated elements of the original matrix. We then copy the elements from the original matrix to the temporary matrix in a rotated manner. After copying all the elements, we assign the temporary matrix back to the original matrix, effectively rotating the original matrix.

   - Here are the steps of the approach:
     1. Create a new temporary matrix of the same size as the original matrix. Let's call it temp.
     2. Iterate over each column of the original matrix (matrix) starting from the first column (index 0).
     3. For each column, create a new array arr and iterate over each row of the original matrix in reverse order, starting from the last row (index rows - 1). Push each element of the column into the arr array. This effectively takes the elements of the column in the reverse order.
     4. Once we have the arr array, representing the rotated column, push it into the temp matrix.
     5. Repeat steps 3-4 for all columns of the original matrix.
     6. After the iteration is complete, the temp matrix will contain the original matrix rotated by 90 degrees clockwise.
     7. Assign the temp matrix back to the original matrix to update the original matrix with the rotated elements.
   - **Time Complexity: The time complexity of this approach is O(N^2), where N is the number of rows (or columns) in the matrix.** This is because we need to iterate over each element of the matrix once to copy it to the temporary matrix.
   - **Space Complexity: The space complexity is also O(N^2), where N is the number of rows (or columns) in the matrix. This is because we create a new matrix (temp) of the same size as the original matrix to store the rotated elements.**

   - Let's take a small example to see how this approach works:

   - Example:
   - Input Matrix:

     1 2 3

     4 5 6

7 8 9

- Steps:

1. Create a temporary matrix (temp) of size 3x3.
2. Iterate over each column:
    - For column 0, reverse elements and store in **arr**: **arr = [7, 4, 1]**
    - Push **arr** into the temporary matrix.
    - For column 1, reverse elements and store in **arr**: **arr = [8, 5, 2]**
    - Push **arr** into the temporary matrix.
    - For column 2, reverse elements and store in **arr**: **arr = [9, 6, 3]**
    - Push **arr** into the temporary matrix.
3. The temporary matrix (temp) becomes:

    7 4 1

    8 5 2

    9 6 3
4. Assign the temporary matrix (temp) back to the original matrix.
5. The original matrix becomes:

    7 4 1

    8 5 2

    9 6 3

- The original matrix is now rotated by 90 degrees clockwise.

2. **Approach 2: Function to rotate the matrix in-place**

- In this approach, we rotate the matrix in-place without using any additional space. It involves two main steps: transposing the matrix and then reversing each row.
- Here are the steps of the approach:
- Transpose the matrix: Iterate over the upper triangle of the matrix (i.e., elements above the main diagonal), and for each pair of elements matrix[i][j] and matrix[j][i], swap them. This step effectively converts the rows into columns and columns into rows.

- Reverse each row: After the transposition step, each row of the matrix represents a column of the original matrix. Now, we simply reverse each row, effectively rotating the matrix 90 degrees clockwise.

- **Time Complexity: The time complexity of this approach is O(N^2), where N is the number of rows (or columns) in the matrix. The transposition step involves iterating over each element in the upper triangle of the matrix (N^2/2 iterations), and the row reversal step takes O(N) time for each row.**
- **Space Complexity: O(1) as we don't use any additional space.**
- **Example:** Input Matrix:

```
11  14  17  18  19
12  33  23  19  21
65  78  99  98  54
31  32  88  91  67
66  75  55  78  89
```

  Transpose the matrix:
- Swap matrix[0][1] and matrix[1][0]: Transpose elements (14, 12)
- Swap matrix[0][2] and matrix[2][0]: Transpose elements (17, 65)
- Swap matrix[0][3] and matrix[3][0]: Transpose elements (18, 31)
- Swap matrix[0][4] and matrix[4][0]: Transpose elements (19, 66)
- Swap matrix[1][2] and matrix[2][1]: Transpose elements (23, 78)
- ... and so on until the entire upper triangle is transposed.

  The matrix after transposition becomes:

```
11  12  65  31  66
14  33  78  32  75
17  23  99  88  55
18  19  98  91  78
19  21  54  67  89
```

- Reverse each row:
    - Reverse row 1: **66 31 65 12 11**
    - Reverse row 2: **75 32 78 33 14**
    - Reverse row 3: **55 88 99 23 17**
    - Reverse row 4: **78 91 98 19 18**
    - Reverse row 5: **89 67 54 21 19**
- The matrix after reversing each row becomes:

```
66  31  65  12  11
75  32  78  33  14
55  88  99  23  17
78  91  98  19  18
89  67  54  21  19
```

3. **Approach 3: Function to rotate the matrix in-place by using divide and conquer approach**

- In this approach, we rotate the matrix layer by layer from the outermost layer to the innermost layer. We divide the problem into smaller subproblems and rotate each layer individually.
- Here are the steps of the approach:

1. Perform rotations layer by layer: We start with the outermost layer of the matrix and perform rotations on each element in the four sides of the layer. We move from the top-left corner to the top-right corner, then to the bottom-right corner, and finally to the bottom-left corner.
2. For each layer, we perform four swaps to rotate the elements:
   - Save the top-left element in a temporary variable (top).
   - Move the element from the bottom to the top-left (top <- bottom).
   - Move the element from the right to the bottom (bottom <- right).
   - Move the element from the top to the right (right <- top).
   - Move the element from the temporary variable to the left (left <- top).
3. Continue this process for each layer until we reach the innermost layer of the matrix.

- **Time Complexity: O(N^2) where N is the number of rows (or columns) in the matrix.**
- **Space Complexity: O(1) as we don't use any additional space.**
- Example:
- Input Matrix:

```
13  14  23  33

39  44  51  57

58  62  67  79

89  93  98  99
```

- Output Matrix:

```
89  58  39  13
93  62  44  14
98  67  51  23
99  79  57  33
```

- Steps:

1. Perform rotations layer by layer:
   - Layer 1 (outermost layer):

- Start = 0, End = 3, Index = Start = 0 and Index < End
    - Save the top-left element (13).
    - Move the bottom-left element (89) to the top-left: 89 <- 13
    - Move the bottom-right element (99) to the bottom-left: 99 <- 89
    - Move the top-right element (33) to the bottom-right: 33 <- 99
    - Move the saved top-left element (13) to the top-right: 13 <- 33
- Start = 0, End = 3, Index = 1 and Index < End
    - Save the 2nd top-left element (14)
    - Move the current bottom-left element (58) to the top-left: 58 <- 14
    - Move the current bottom-right element (98) to the bottom-left: 98 <- 58
    - Move the top-right element (57) to the bottom-right: 57 <- 98
    - Move the saved top-left element (14) to the top-right: 14 <- 98
- Start = 0, End = 3, Index = 2 and Index < End
    - Save the 3rd top-left element (23)
    - Move the current bottom-left element (39) to the top-left: 39 <- 23
    - Move the current bottom-right element (79) to the bottom-left: 93 <- 39
    - Move the top-right element (57) to the bottom-right: 79 <- 93
    - Move the saved top-left element (23) to the top-right: 23 <- 79
- Now, Start = 0, End = 3, Index = 3, and Index < End (False)

  The matrix after the first layer rotation becomes:

  89 58 39 13

  93 44 51 14

  98 62 67 23

  99 79 57 33
- Layer 2 (Innermost Layer):
- Now Start = 1, End = 2, Index = Start = 1 and Index < End
    - Save the current top-left element (44)
    - Move the current bottom-left element (62) to the top-left: 62 <- 44

    - Move the current bottom-right element (67) to the bottom-left: 67 <- 62
    - Move the top-right element (51) to the bottom-right: 51 <- 67
    - Move the saved top-left element (44) to the top-right: 44 <- 51
- Start = 1, End = 2, Index = 2, Index < End (False), so coming out of the loop.
- **The final Matrix:**

  89 58 39 13

  93 62 44 14

  98 67 51 23

  99 79 57 33