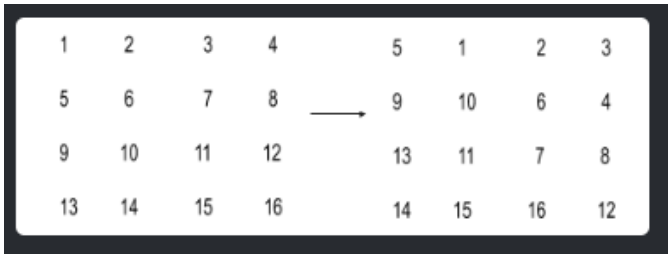
Rotate matrix

Rotate matrix in clockwise direction



The matrix is rotated in clockwise direction such that 5 ocuupies 1’s place 1 ocuupies 2’s place and so if we can keep a track of previous number and current number the problem can be solved to iterate clockwise 4 pointers will be needed i.e rowstart,rowend,colstart,colend.

Setting current element to previous and then storing current number in previous. doing this for all four direction left to right,top to down,right to left and bottom to up.

```
#include <bits/stdc++.h>

void rotateMatrixHelper(vector<vector<int>> &mat,int rowstart,int rowend,int colstart,int colend)
{
    if(rowstart>=rowend-1 || colstart>=colend-1)
    {
        return;
    }
    int current,previous=mat[rowstart+1][colstart];
    for(int i=colstart;i<colend;i++)
    {
        current=mat[rowstart][i];
        mat[rowstart][i]=previous;
        previous=current;
    }
    rowstart++;
    for(int i=rowstart;i<rowend;i++)
    {
        current=mat[i][colend-1];
        mat[i][colend-1]=previous;
        previous=current;
    }
    colend--;

    for (int i = colend - 1; i >= colstart; i--) {
    current = mat[rowend - 1][i];
    mat[rowend - 1][i] = previous;
    previous = current;
    }
    rowend--;

    for(int i=rowend-1;i>=rowstart;i--)
    {
        current=mat[i][colstart];
```

```
        mat[i][colstart]=previous;
        previous=current;
    }
    colstart++;


    rotateMatrixHelper(mat,rowstart,rowend,colstart,colend);
}
void rotateMatrix(vector<vector<int>> &mat, int n, int m)
{
    // Write your code here
rotateMatrixHelper(mat,0,n,0,m);
}
```

- **Time Complexity**

O(N * M), where N is the number of rows and M is the number of columns in the matrix.

Since we are traversing each element of the matrix just once, the time complexity of the above algorithm is O(N * M).

- **Space Complexity**

O(max(N, M)), where N is the number of rows and M is the number of columns in the matrix.

We are making max(N/2, M/2) recursive calls and thus, stack space will be used. Hence, the space complexity is O(max(N, M)).

Iterative version

```
/*

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

    Where N, M are the number of rowStarts and the number of colStartumns of the matrix,
    respectively.

*/


void rotateMatrix(vector<vector<int>> &mat, int n, int m)
{
    // Index of starting row and column
    int rowStart = 0, colStart = 0;

    int previous, current;

    while (rowStart < n && colStart < m)
    {
        // If we have rotated the whole matrix
        if (rowStart == n - 1 || colStart == m - 1)
        {
            break;
        }
        // Store the first element of next rown as this element will replace the first element of current row
```

```
        previous = mat[rowStart + 1][colStart];

        // Move elements of first row from the remaining rows
        for (int i = colStart; i < m; i++)
        {
            current = mat[rowStart][i];
            mat[rowStart][i] = previous;
            previous = current;
        }
        rowStart++;

        // Move elements of last column from the remaining columns
        for (int i = rowStart; i < n; i++)
        {
            current = mat[i][m-1];
            mat[i][m-1] = previous;
            previous = current;
        }
        m--;

        // Move elements of last rowStart from the remaining rows

        for (int i = m-1; i >= colStart; i--)
        {
            current = mat[n-1][i];
            mat[n-1][i] = previous;
            previous = current;
        }

        n--;

        // Move elements of first column from the remaining rows

        for (int i = n-1; i >= rowStart; i--)
        {
            current = mat[i][colStart];
            mat[i][colStart] = previous;
            previous = current;
        }

        colStart++;
    }
}
```