## Spiral Matrix

Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

For example,
Given the following matrix:

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

You should return `[1,2,3,6,9,8,7,4,5]`.

## Solution 1

This is a very simple and easy to understand solution. I traverse right and increment rowBegin, then traverse down and decrement colEnd, then I traverse left and decrement rowEnd, and finally I traverse up and increment colBegin.

The only tricky part is that when I traverse left or up I have to check whether the row or col still exists to prevent duplicates. If anyone can do the same thing without that check, please let me know!

Any comments greatly appreciated.

```java
public class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> res = new ArrayList<Integer>();

        if (matrix.length == 0) {
            return res;
        }

        int rowBegin = 0;
        int rowEnd = matrix.length-1;
        int colBegin = 0;
        int colEnd = matrix[0].length - 1;

        while (rowBegin <= rowEnd && colBegin <= colEnd) {
            // Traverse Right
            for (int j = colBegin; j <= colEnd; j ++) {
                res.add(matrix[rowBegin][j]);
            }
            rowBegin++;

            // Traverse Down
            for (int j = rowBegin; j <= rowEnd; j ++) {
                res.add(matrix[j][colEnd]);
            }
            colEnd--;

            if (rowBegin <= rowEnd) {
                // Traverse Left
                for (int j = colEnd; j >= colBegin; j --) {
                    res.add(matrix[rowEnd][j]);
                }
            }
            rowEnd--;

            if (colBegin <= colEnd) {
                // Traver Up
                for (int j = rowEnd; j >= rowBegin; j --) {
                    res.add(matrix[j][colBegin]);
                }
            }
            colBegin ++;
        }

        return res;
    }
}
```

written by qwl5004 original link here

## Solution 2

When traversing the matrix in the spiral order, at any time we follow one out of the following four directions: RIGHT DOWN LEFT UP. Suppose we are working on a 5 x 3 matrix as such:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Imagine a cursor starts off at (0, -1), i.e. the position at '0', then we can achieve the spiral order by doing the following:

1. Go right 5 times
2. Go down 2 times
3. Go left 4 times
4. Go up 1 times.
5. Go right 3 times
6. Go down 0 times -> quit

Notice that the directions we choose always follow the order 'right->down->left->up', and for horizontal movements, the number of shifts follows:{5, 4, 3}, and vertical movements follows {2, 1, 0}.

Thus, we can make use of a direction matrix that records the offset for all directions, then an array of two elements that stores the number of shifts for horizontal and vertical movements, respectively. This way, we really just need one for loop instead of four.

Another good thing about this implementation is that: If later we decided to do spiral traversal on a different direction (e.g. Counterclockwise), then we only need to change the Direction matrix; the main loop does not need to be touched.

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    vector<vector<int> > dirs{{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    vector<int> res;
    int nr = matrix.size();     if (nr == 0) return res;
    int nc = matrix[0].size();  if (nc == 0) return res;

    vector<int> nSteps{nc, nr-1};

    int iDir = 0;   // index of direction.
    int ir = 0, ic = -1;    // initial position
    while (nSteps[iDir%2]) {
        for (int i = 0; i < nSteps[iDir%2]; ++i) {
            ir += dirs[iDir][0]; ic += dirs[iDir][1];
            res.push_back(matrix[ir][ic]);
        }
        nSteps[iDir%2]--;
        iDir = (iDir + 1) % 4;
    }
    return res;
}
```

written by stellari original link here

## Solution 3

```
    public List<Integer> spiralOrder(int[][] matrix) {
    List<Integer> spiralList = new ArrayList<>();
    if(matrix == null || matrix.length == 0) return spiralList;

    // declare indices
    int top = 0;
    int bottom = matrix.length - 1;
    int left = 0;
    int right = matrix[0].length - 1;

    while(true){
        // 1. print top row
        for(int j=left; j <=right;j++){
            spiralList.add(matrix[top][j]);
        }
        top++;
        if(boundriesCrossed(left,right,bottom,top))
            break;

        // 2. print rightmost column
        for(int i=top; i <= bottom; i++){
            spiralList.add(matrix[i][right]);
        }
        right--;
        if(boundriesCrossed(left,right,bottom,top))
            break;

        // 3. print bottom row
        for(int j=right; j >=left; j--){
            spiralList.add(matrix[bottom][j]);
        }
        bottom--;
        if(boundriesCrossed(left,right,bottom,top))
            break;

        // 4. print leftmost column
        for(int i=bottom; i >= top; i--){
            spiralList.add(matrix[i][left]);
        }
        left++;
        if(boundriesCrossed(left,right,bottom,top))
            break;
    }// end while true

    return spiralList;
}

private boolean boundriesCrossed(int left,int right,int bottom,int top){
    if(left>right || bottom<top)
        return true;
    else
        return false;
}
```

written by ishanpande original link here

From Leetcoder.