**Arrays:**

**Product of Array Except self:**

class Solution {

public int[] productExceptSelf(int[] nums) {

if(nums == null || nums.length == 0){

return new int[0];

}

int[] result = new int[nums.length];

//left

for(int i = 1; i< nums.length; i++){

result[0] = 1;

result[i] = result[i-1]\*nums[i-1];

}

//right

int right = 1;

for(int i = nums.length-1;i>=0;i--){

result[i] = right \* result[i];

right = right\*nums[i];

}

return result;

}

}

The time complexity is O(N).

The space complexity is O(1).

**Diagonal Traverse:**

class Solution {

public int[] findDiagonalOrder(int[][] mat) {

if(mat == null || mat.length == 0)

return new int[0];

int dir = 1;

int m = mat.length; int n = mat[0].length;

int row = 0, col = 0, i = 0;

int [] result = new int[m\*n];

while(i< m\*n){

result[i++]= mat[row][col];

if (dir == 1){

if(col == n-1){

row++;

dir = -1;

}

else if(row == 0){

col++;

dir = -1;

}

else{

row --;

col++;

}

}

else{

if(row == m-1){

dir = 1;

col++;

}

else if(col == 0){

dir = 1;

row++;

}

else{

row++;

col--;

}

}

}

return result;

}

}

The time complexity is O(MN)

The space complexity is O(1).

**Spiral Matrix:**

class Solution {

public List<Integer> spiralOrder(int[][] matrix) {

if(matrix == null || matrix.length == 0)

return new ArrayList<Integer>();

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

int m = matrix.length; int n = matrix[0].length;

int top = 0, bottom = m-1, left = 0, right = n-1;

while(top <= bottom && left <= right){

//left to right

for(int i = left; i<=right; i++){

result.add(matrix[top][i]);

}

top++;

//top to bottom

for(int i = top; i<= bottom; i++){

result.add(matrix[i][right]);

}

right--;

//right to left

if(top <= bottom){

for(int i = right; i >= left; i--){

result.add(matrix[bottom][i]);

}

}

bottom--;

//bottom to top

if(left <= right){

for(int i = bottom; i>=top; i--){

result.add(matrix[i][left]);

}

}

left++;

}

return result;

}

}

The time complexity is O(MN).