# 题目

给定一个n×n的二维矩阵matrix表示一个图像。请你将图像顺时针旋转90度。

你必须在原地旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。

示例 1：



输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]

输出：[[7,4,1],[8,5,2],[9,6,3]]

示例 2：



输入：matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

输出：[[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

示例 3：

输入：matrix = [[1]]

输出：[[1]]

示例 4：

输入：matrix = [[1,2],[3,4]]

输出：[[3,1],[4,2]]

提示：

matrix.length == n

matrix[i].length == n

1 <= n <= 20

-1000 <= matrix[i][j] <= 1000

# 分析

## 方法一：转置+排序（推荐）

思路：

**矩阵转置**：将矩阵的行和列交换，即 matrix[i][j] 与 matrix[j][i] 交换。

**反转每一行**：将每一行的元素从左到右反转。

这样可以在原地完成旋转操作。

代码：

class Solution {

public:

void rotate(vector<vector<int>>& matrix) {

int n = matrix.size();

// 1. 转置矩阵

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

for (int j = i + 1; j < n; ++j) {

swap(matrix[i][j], matrix[j][i]);

}

}

// 2. 反转每一行

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

reverse(matrix[i].begin(), matrix[i].end());

}

}

};

时间复杂度：

转置矩阵的时间复杂度为O(n2)，因为需要遍历上三角部分。

每行反转的时间复杂度为O(n)，总共n行，整体时间复杂度为O(n2)。

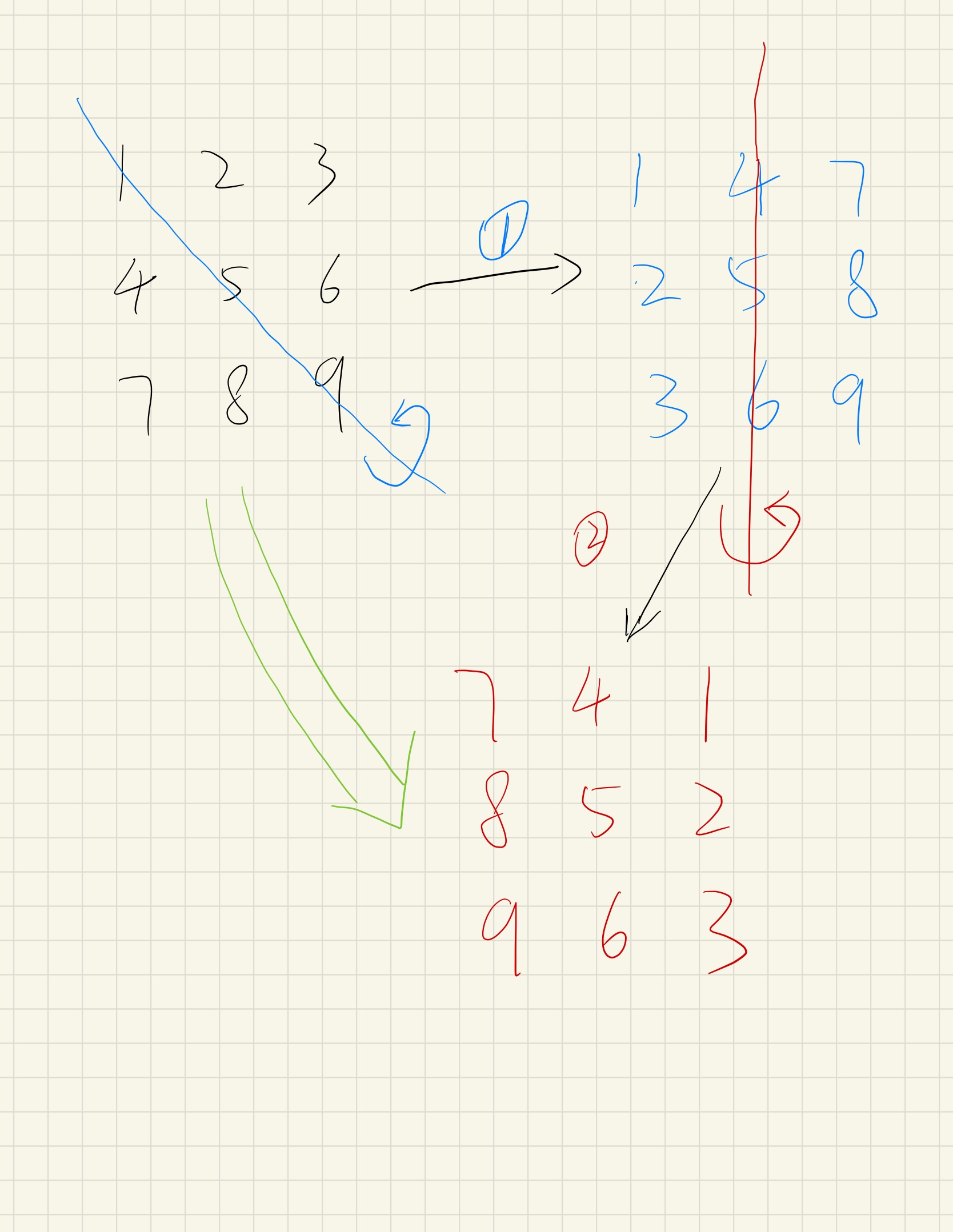
空间复杂度：

原地操作，无需额外空间，空间复杂度为O(1)。

## 方法二：对角线+倒排

**思路：**

先对矩阵做沿主对角线的翻转（其实就是矩阵的转置），原地完成；接着再对操作后的矩阵进行根据列序号倒排。



从线性代数的思路分析，实现一个算法将二维矩阵顺时针旋转90度，可以通过以下两步完成：

**代码：**

class Solution {

public:

//先按照主对角线翻转，再按列号倒排

void rotate(vector<vector<int>>& matrix) {

int n = matrix.size();

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

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

swap(matrix[i][j], matrix[j][i]);

}

}

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

for(int j=0 ; j<n/2 ; ++j){

swap(matrix[i][j], matrix[i][n-1-j]);

}

}

}

};

## 方法三：原地交换

思路：

原地交换的实现思路是，对于矩阵中的每一个元素 (i, j)，将其与对应的旋转后位置 (j, n-1-i) 的元素进行交换，其中 n 是矩阵的边长。

下面是具体的代码实现：

class Solution {

public:

void rotate(vector<vector<int>>& matrix) {

int n = matrix.size();

// 首先沿着副对角线交换元素

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

for (int j = 0; j < n - i; j++) {

swap(matrix[i][j], matrix[n - 1 - j][n - 1 - i]);

}

}

// 然后沿着水平中线交换元素

for (int i = 0; i < n / 2; i++) {

for (int j = 0; j < n; j++) {

swap(matrix[i][j], matrix[n - 1 - i][j]);

}

}

}

};

这个算法也是原地修改矩阵的，时间复杂度也是 O(n^2)。