ARRAYS-2

Question: Given an array with n objects colored 0, 1 or 2, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

YOU HAVE 10 MINUTES

EXAMPLE:

INPUT: [2,0,2,1,1,0]

OUTPUT: [0,0,1,1,2,2]

Link: https://leetcode.com/problems/sort-colors/

We can start by maintaining 2 pointers. One pointer for the rightmost 0 in the array and one pointer for the leftmost 2.

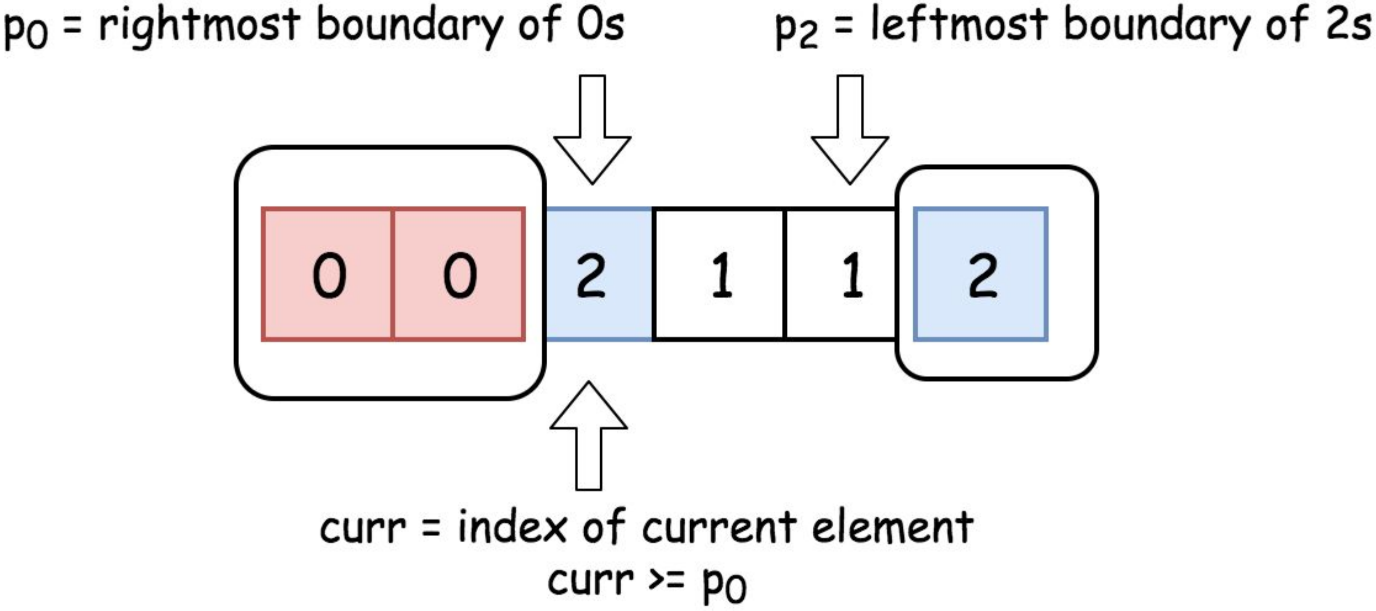
Additionally, we can have a pointer called current or iterator, that start from 0th index and moves left -> right.

Whenever nums[iterator] = 0, we swap nums[iterator] with nums[rightMost0] and increment the pointer to the right most 0. When nums[iterator] = 2 we swap nums[iterator] with nums[leftMost2] and decrement the pointer to the left most 2.

When nums[iterator] = 1, we do nothing and just move right.

We do this till current/iterator is less than leftMost2.

We will partition the array in 3 groups when we are done.



p_0 = rightmost boundary of 0s

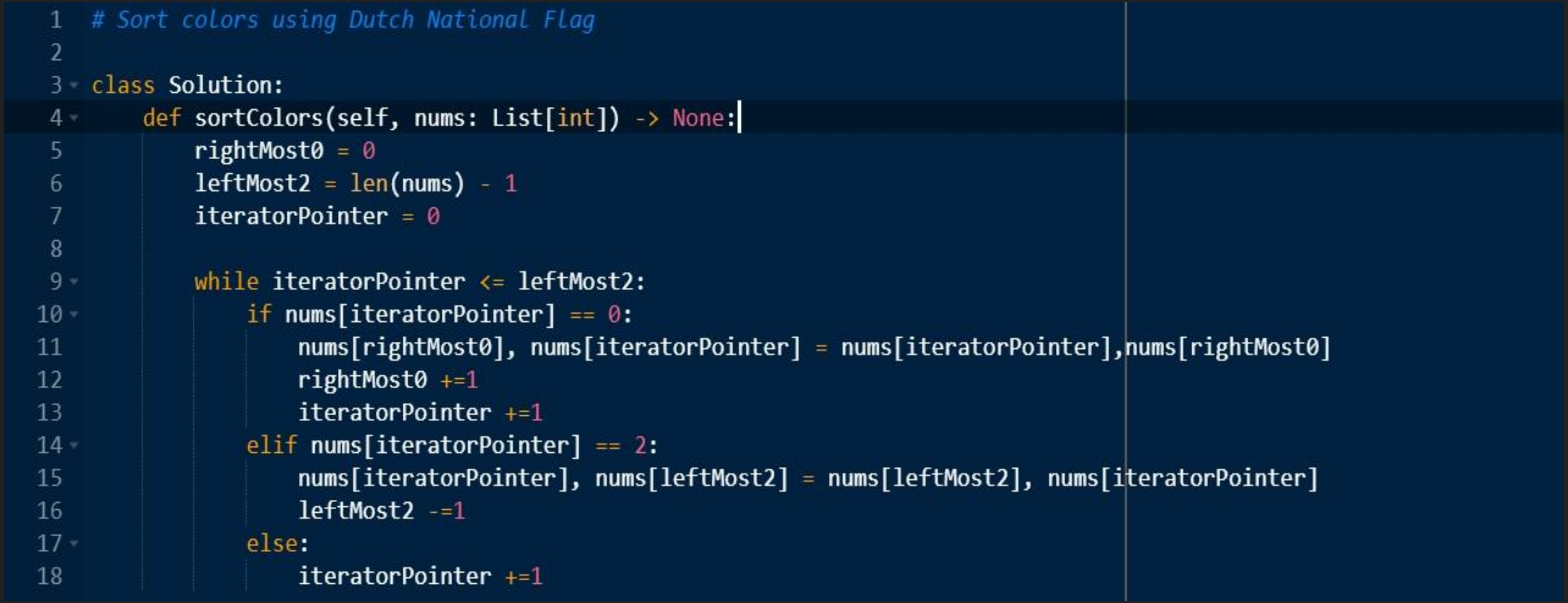
p_2 = leftmost boundary of 2s

0	0	2	1	1	2

curr = index of current element

curr >= p_0

SOLUTION (SINGLE PASS)



```python
# Sort colors using Dutch National Flag

class Solution:
    def sortColors(self, nums: List[int]) -> None:
        rightMost0 = 0
        leftMost2 = len(nums) - 1
        iteratorPointer = 0

        while iteratorPointer <= leftMost2:
            if nums[iteratorPointer] == 0:
                nums[rightMost0], nums[iteratorPointer] = nums[iteratorPointer],nums[rightMost0]
                rightMost0 +=1
                iteratorPointer +=1
            elif nums[iteratorPointer] == 2:
                nums[iteratorPointer], nums[leftMost2] = nums[leftMost2], nums[iteratorPointer]
                leftMost2 -=1
            else:
                iteratorPointer +=1
```

TIME COMPLEXITY - O(N) and SPACE COMPLEXITY - O(1). Constant space

Question: Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. You are given a target value to search. If found in the array return its index, otherwise return -1. YOU HAVE 15 MINUTES

EXAMPLE:

INPUT: [4,5,6,7,0,1,2], target = 0

OUTPUT: 4

Link:
https://leetcode.com/problems/search-in-rotated-sorted-array/

PLEASE LOOK UP BINARY SEARCH IN THE FOLLOWING ARTICLE:

https://www.geeksforgeeks.org/binary-search/

Essentially a sorted and rotated array can be thought as 2 sorted arrays appended.

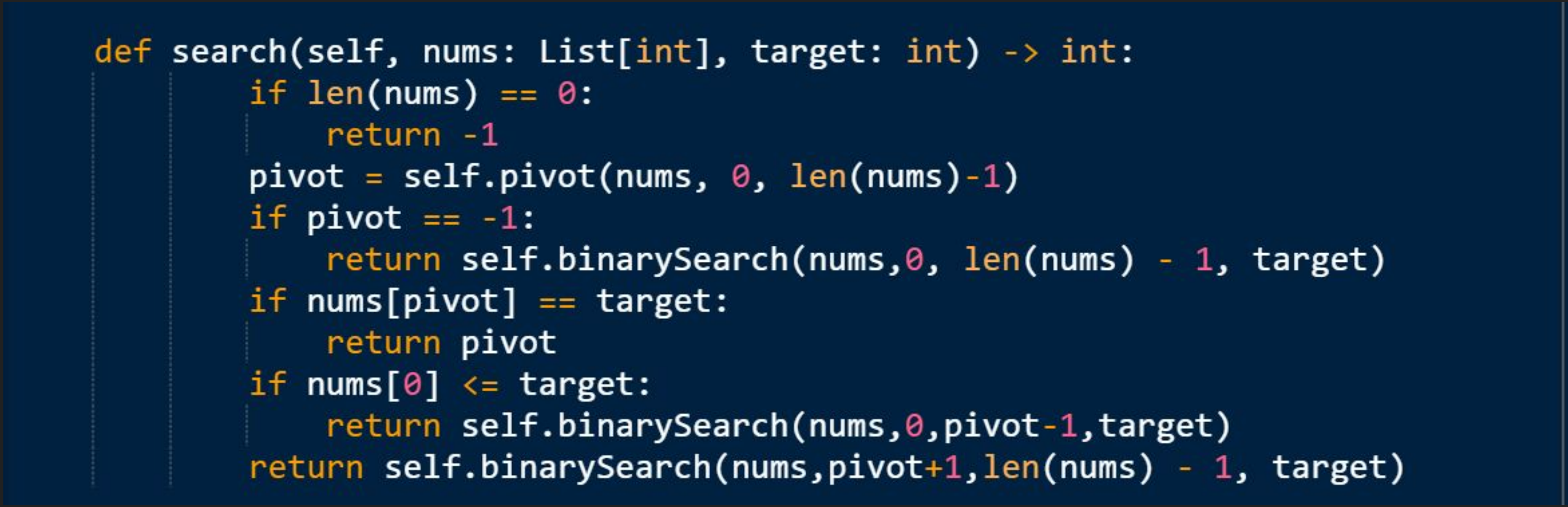
We can apply binary search individually on these two sub arrays.

If we can find the pivot, we can simply do binary search between (0, nums[pivot-1]) and (nums[pivot +1], nums[nums.size-1]). Assuming of course nums[pivot] != target. Else we just return pivot.

But we need to find the pivot, in O(LogN)! (modify binary search ?)

Essentially the final answer must search using findPivot and regular binarySearch as subroutines/utility functions.

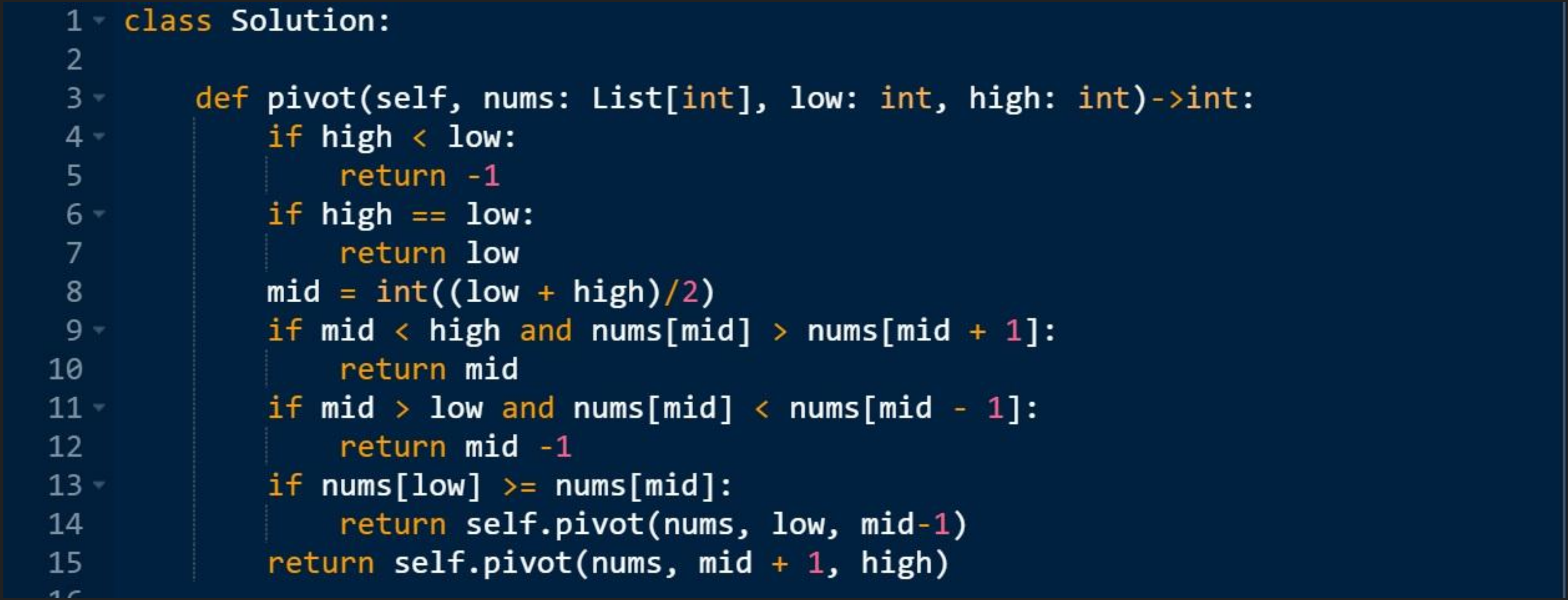
SOLUTION (pivoted-BINARY SEARCH)



```python
def search(self, nums: List[int], target: int) -> int:
        if len(nums) == 0:
            return -1
        pivot = self.pivot(nums, 0, len(nums)-1)
        if pivot == -1:
            return self.binarySearch(nums,0, len(nums) - 1, target)
        if nums[pivot] == target:
            return pivot
        if nums[0] <= target:
            return self.binarySearch(nums,0,pivot-1,target)
        return self.binarySearch(nums,pivot+1,len(nums) - 1, target)
```

TIME COMPLEXITY - O(LOG(N))

SOLUTION (find PIVOT)



```
class Solution:

    def pivot(self, nums: List[int], low: int, high: int)->int:
        if high < low:
            return -1
        if high == low:
            return low
        mid = int((low + high)/2)
        if mid < high and nums[mid] > nums[mid + 1]:
            return mid
        if mid > low and nums[mid] < nums[mid - 1]:
            return mid -1
        if nums[low] >= nums[mid]:
            return self.pivot(nums, low, mid-1)
        return self.pivot(nums, mid + 1, high)
```

TIME COMPLEXITY - O(LOG(N))

Question: Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

1. Integers in each row are sorted in ascending from left to right.
2. Integers in each column are sorted in ascending from top to bottom.

Constraint -> IN LINEAR TIME

YOU HAVE 15 MINUTES

Link: https://leetcode.com/problems/search-a-2d-matrix-ii/

FIND target = 5

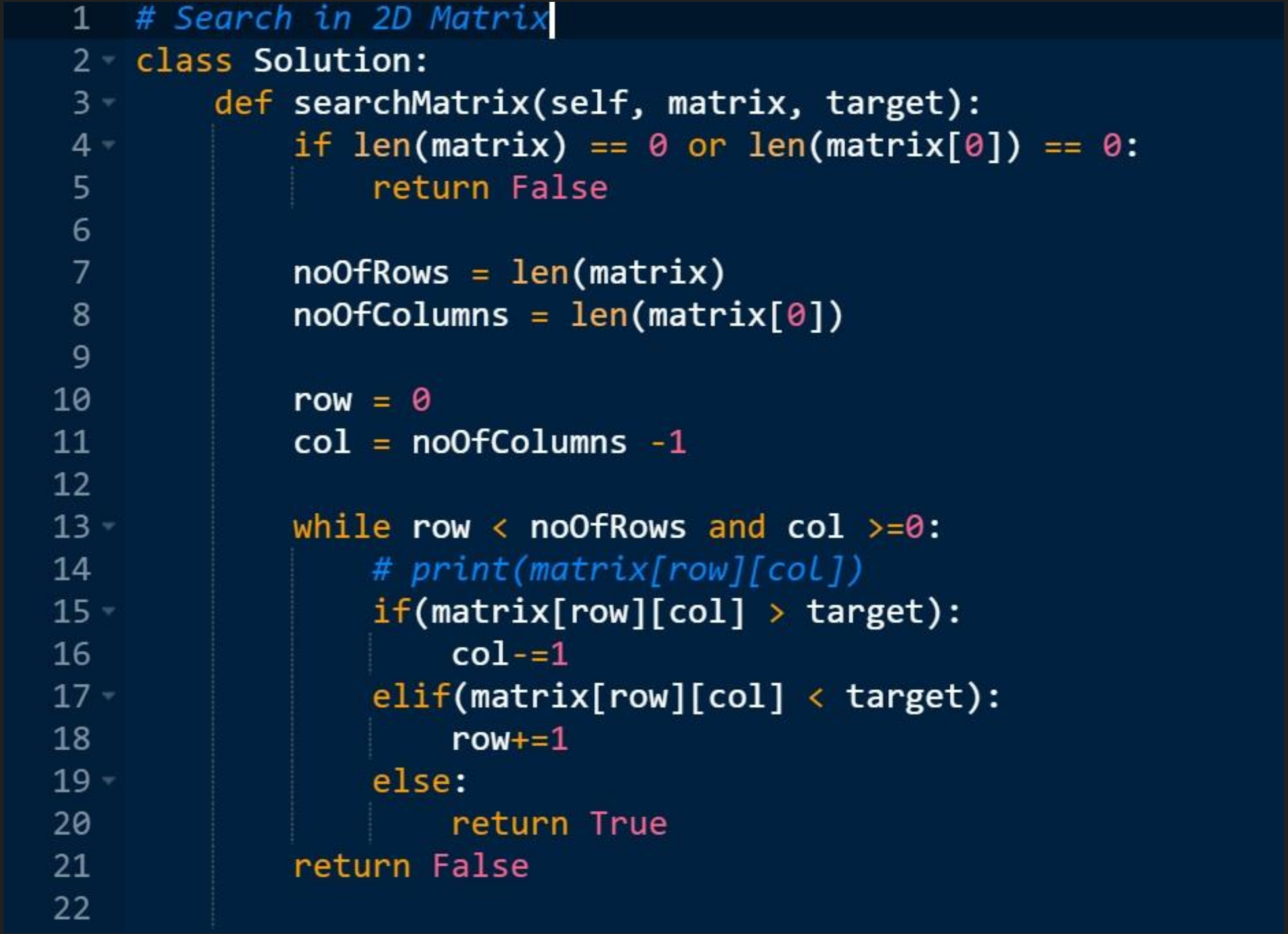
1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

FIND target = 5

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

FIND target = 13

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30



```
# Search in 2D Matrix
class Solution:
    def searchMatrix(self, matrix, target):
        if len(matrix) == 0 or len(matrix[0]) == 0:
            return False

        noOfRows = len(matrix)
        noOfColumns = len(matrix[0])

        row = 0
        col = noOfColumns -1

        while row < noOfRows and col >=0:
            # print(matrix[row][col])
            if(matrix[row][col] > target):
                col-=1
            elif(matrix[row][col] < target):
                row+=1
            else:
                return True
        return False

```

TIME COMPLEXITY - O(M + N). For a MxN matrix

Question: You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (anti - clockwise).

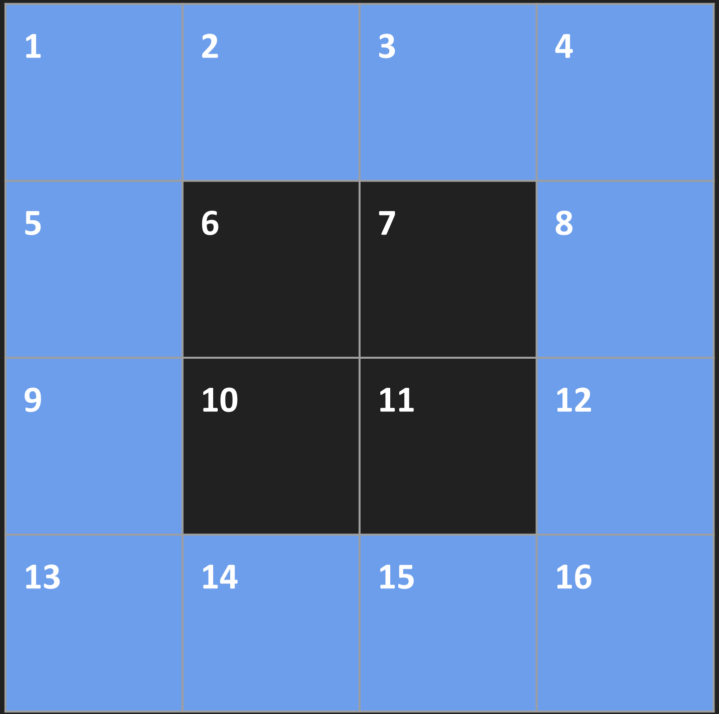
Let’s just discuss first.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

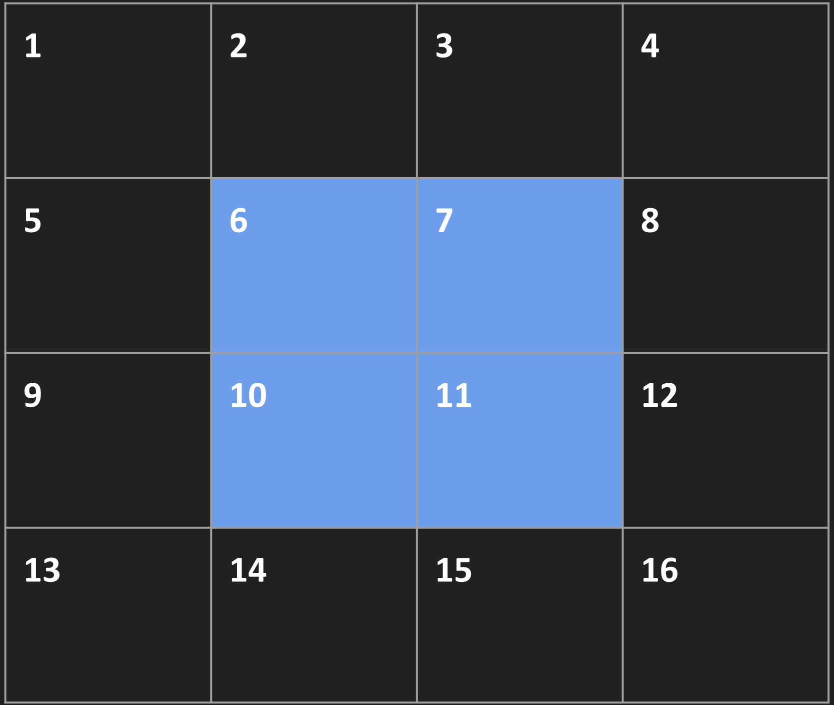
→

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

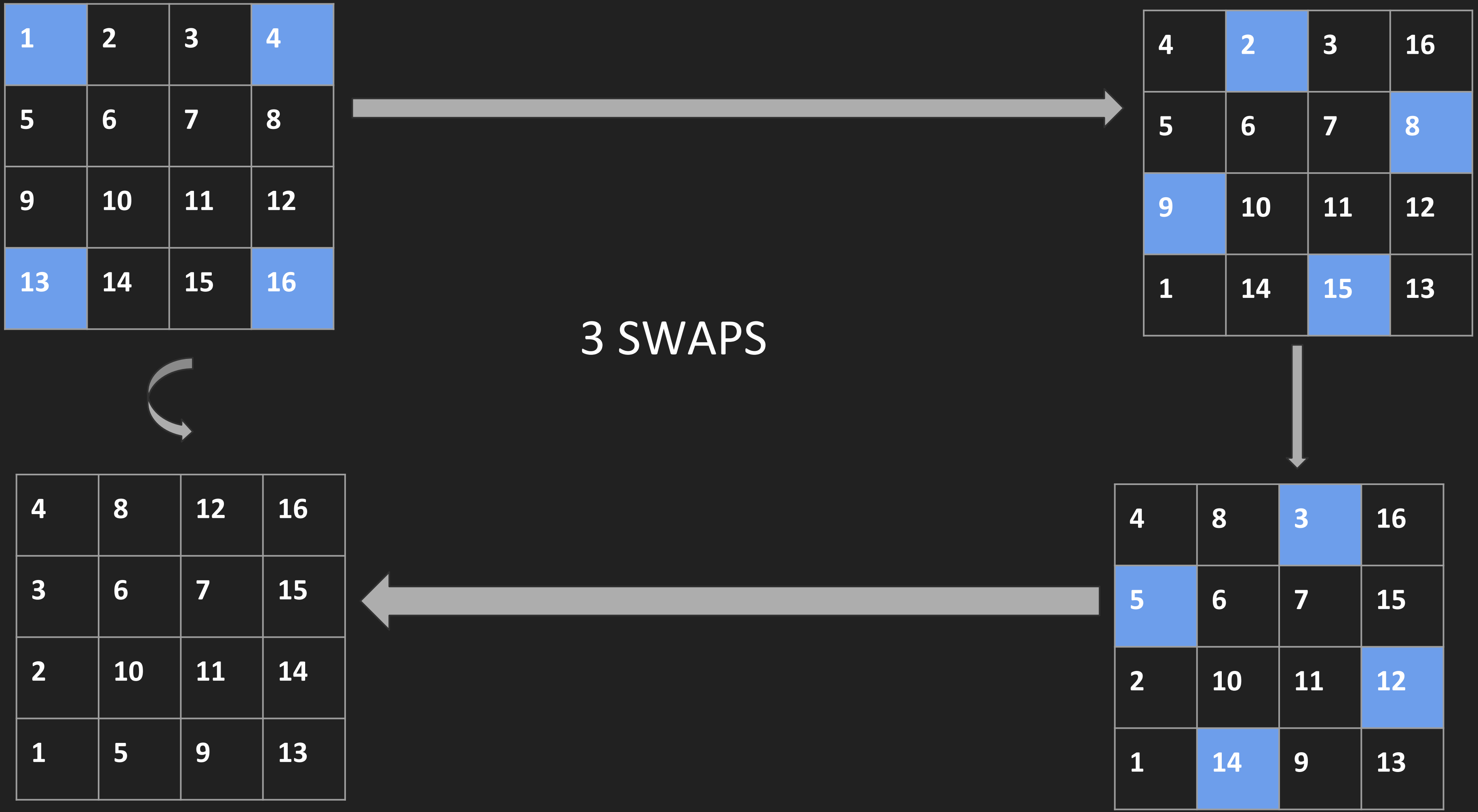
Rotate boundary elements and then move inwards ?



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

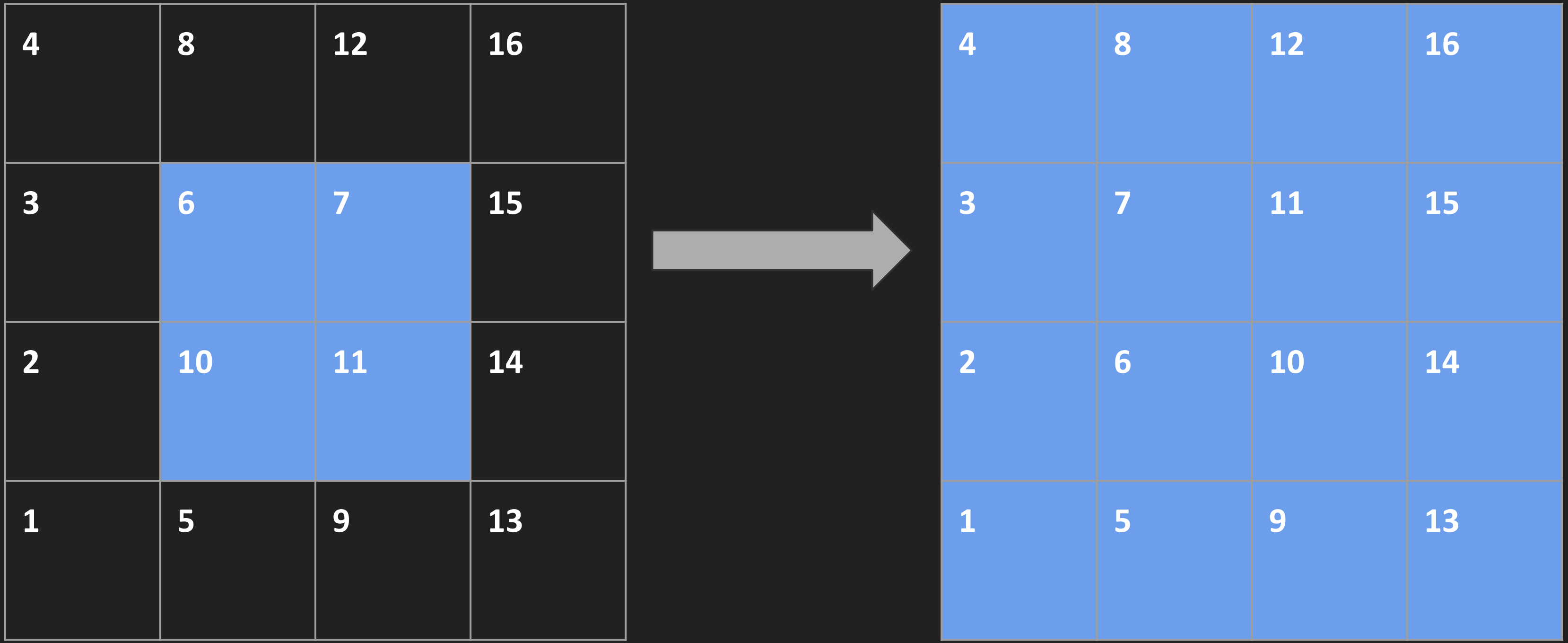
4	2	3	16
5	6	7	8
9	10	11	12
1	14	15	13

4	8	3	16
5	6	7	15
2	10	11	12
1	14	9	13

4	8	12	16
3	6	7	15
2	10	11	14
1	5	9	13

3 SWAPS

1 SWAP



4	8	12	16
3	6	7	15
2	10	11	14
1	5	9	13

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

For N = 2, there is only 1 boundary. You will have to swap once.

For N= 3, there are 2 boundaries. 2 swaps for outer boundary, 0 swap for inner boundary.

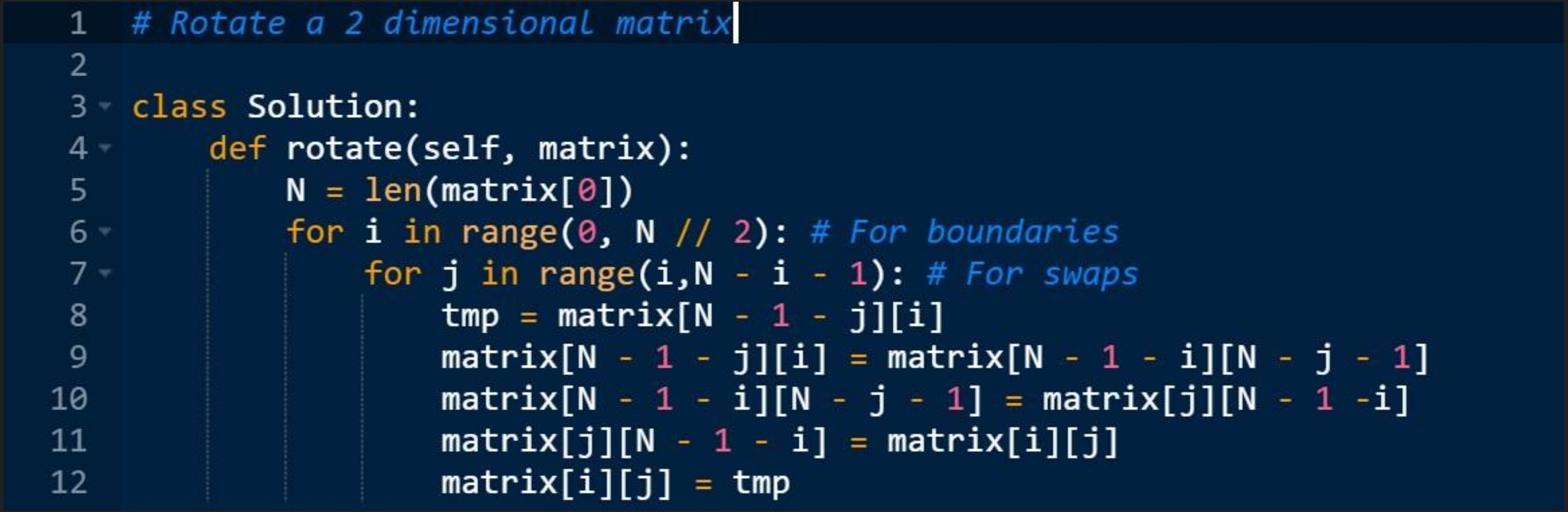
For N = 4, there are 2 boundaries. 3 swaps for outer. 1 swap for inner boundary.

For N =5, there 3 boundaries. 4 swaps for outermost. 3 for next boundary. 0 for innermost boundary.

Question: You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

YOU HAVE 15 MINUTES

Link: https://leetcode.com/problems/rotate-image/



```python
# Rotate a 2 dimensional matrix

class Solution:
    def rotate(self, matrix):
        N = len(matrix[0])
        for i in range(0, N // 2): # For boundaries
            for j in range(i,N - i - 1): # For swaps
                tmp = matrix[N - 1 - j][i]
                matrix[N - 1 - j][i] = matrix[N - 1 - i][N - j - 1]
                matrix[N - 1 - i][N - j - 1] = matrix[j][N - 1 -i]
                matrix[j][N - 1 - i] = matrix[i][j]
                matrix[i][j] = tmp
```

TIME COMPLEXITY : O(MN), for MxN matrix