<https://leetcode.com/problems/minimum-domino-rotations-for-equal-row/description/>

In a row of dominoes, tops[i] and bottoms[i] represent the top and bottom halves of the ith domino. (A domino is a tile with two numbers from 1 to 6 - one on each half of the tile.)

We may rotate the ith domino, so that tops[i] and bottoms[i] swap values.

Return the minimum number of rotations so that all the values in tops are the same, or all the values in bottoms are the same.

If it cannot be done, return -1.

**Example 1:**



**Input:** tops = [2,1,2,4,2,2], bottoms = [5,2,6,2,3,2]

**Output:** 2

**Explanation:** The first figure represents the dominoes as given by tops and bottoms: before we do any rotations.

If we rotate the second and fourth dominoes, we can make every value in the top row equal to 2, as indicated by the second figure.

**Example 2:**

**Input:** tops = [3,5,1,2,3], bottoms = [3,6,3,3,4]

**Output:** -1

**Explanation:** In this case, it is not possible to rotate the dominoes to make one row of values equal.

**Constraints:**

2 <= tops.length <= 2 \* 104

bottoms.length == tops.length

1 <= tops[i], bottoms[i] <= 6

**Attempt 1: 2025-06-28**

**Solution 1: Math (10 min)**

class Solution {

    public int minDominoRotations(int[] tops, int[] bottoms) {

        // Frequency of each number (1-6) in the top row

        int[] topFreq = new int[7];

        // Frequency of each number (1-6) in the bottom row

        int[] bottomFreq = new int[7];

        // Frequency where both top and bottom have the same number at same position

        int[] same = new int[7];

        for(int i = 0; i < tops.length; i++) {

            topFreq[tops[i]]++;

            bottomFreq[bottoms[i]]++;

            if(tops[i] == bottoms[i]) {

                same[tops[i]]++;

            }

        }

        for(int num = 1; num <= 6; num++) {

            // This means the number appears enough times (considering overlaps) to fill a row

            if(topFreq[num] + bottomFreq[num] - same[num] >= tops.length) {

                return Math.min(topFreq[num], bottomFreq[num]) - same[num];

            }

        }

        return -1;

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to Deepseek**

This problem requires finding the minimum number of rotations needed to make all values in either the top or bottom row of dominoes the same. Here's an efficient Java solution with explanation:

**Solution Code**

class Solution {

public int minDominoRotations(int[] tops, int[] bottoms) {

int n = tops.length;

int[] countTop = new int[7]; // Count of each number in tops

int[] countBottom = new int[7]; // Count of each number in bottoms

int[] same = new int[7]; // Count where both have same number

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

countTop[tops[i]]++;

countBottom[bottoms[i]]++;

if (tops[i] == bottoms[i]) {

same[tops[i]]++;

}

}

for (int num = 1; num <= 6; num++) {

if (countTop[num] + countBottom[num] - same[num] >= n) {

return Math.min(countTop[num], countBottom[num]) - same[num];

}

}

return -1;

}

}

**Explanation**

**Approach**

**Count Occurrences**:

We maintain three count arrays:

countTop: Frequency of each number (1-6) in the top row

countBottom: Frequency of each number (1-6) in the bottom row

same: Frequency where both top and bottom have the same number

**Check Feasibility**:

For each number (1-6), check if it can form a complete row by verifying:countTop[num] + countBottom[num] - same[num] >= n

This means the number appears enough times (considering overlaps) to fill a row

**Calculate Minimum Rotations**:

If feasible, the minimum rotations needed is:Math.min(countTop[num], countBottom[num]) - same[num]

This represents the lesser of:

Rotating bottom dominoes to match top (countTop[num] - same[num])

Rotating top dominoes to match bottom (countBottom[num] - same[num])

**Time Complexity**

O(n) where n is the number of dominoes

We make one pass through the arrays to count occurrences

Then a constant time check (6 iterations) to find the solution

**Space Complexity**

O(1) - We use fixed-size arrays (size 7) regardless of input size

**Example Walkthrough**

For input:

tops = [2,1,2,4,2,2]

bottoms = [5,2,6,2,3,2]

Counts for number 2:

countTop[2] = 4 (appears in top 4 times)

countBottom[2] = 3 (appears in bottom 3 times)

same[2] = 1 (position where both have 2)

Verification: 4 + 3 - 1 = 6 ≥ 6 (n) → feasible

Minimum rotations: min(4, 3) - 1 = 2

This solution efficiently checks all possible candidate numbers and calculates the minimum rotations needed while handling edge cases where numbers appear in both rows.