Daily Coding Problem #195

Problem

This problem was asked by Google.

Let A be an N by M matrix in which every row and every column is sorted.

Given i_1, j_1, i_2, and j_2, compute the number of elements of M smaller than $M[i_1, j_1]$ and larger than $M[i_2, j_2]$.

For example, given the following matrix:

```
[[1, 3, 7, 10, 15, 20],
 [2, 6, 9, 14, 22, 25],
 [3, 8, 10, 15, 25, 30],
 [10, 11, 12, 23, 30, 35],
 [20, 25, 30, 35, 40, 45]]
```

And $i_1 = 1$, $j_1 = 1$, $i_2 = 3$, $j_2 = 3$, return 15 as there are 15 numbers in the matrix smaller than 6 or greater than 23.

Solution

The naive solution here would be to simply iterate over all elements of the matrix and count all the numbers smaller than `matrix[i1][j1]` (we'll call this a) or greater than `m[i2][j2]` (we'll call this b).

```
def matrix_count_naive(matrix, i1, j1, i2, j2):
    return sum(sum(val < matrix[i1][j1] or val > matrix[i2][j2] for val in row)
for row in m)
```

This solution takes O(N * M) time and constant space. However, note that this doesn't depend on the matrix being both row- and column-sorted at all. We must take advantage of this property to improve the runtime.

Let's first consider searching for values smaller than a. Note that for a given column, if we know that a value in the column is greater than a, we can infer that all the values below it are also greater than a, so we don't need to examine them. Similarly, all the values below it and to the right are also greater than a.

The general idea is to start a pointer on the top right corner of the matrix and, moving down, find the number of values smaller than a for that column. Once we've found a number greater than a or reached the end, we can add the column length so far to our count and move the pointer directly left. We can do this since we know that all numbers below are greater than a, so we can safely ignore them. We do something similar for finding values greater than b but we'll add the values to the bottom of the matrix, since that is where the greater numbers are.

```
def matrix_count_edge(matrix, i1, j1, i2, j2):
    m, n = len(matrix), len(matrix[0])

    count = 0

    # Count numbers smaller than m[i1][j1]
    a = matrix[i1][j1]
    i, j = 0, m - 1
    for j in reversed(range(n)):
        while i < m and matrix[i][j] < a:
            i += 1
        count += i

    # Count numbers greater than m[i2][j2]
    b = matrix[i2][j2]
    i, j = 0, m - 1
    for j in reversed(range(n)):
        while i < m and matrix[i][j] < b:
            i += 1
        count += m - i

    return count
```

This algorithm now does, at worst, two passes over every row and column, which makes this O(M + N) time and constant space.