Sohan Chatterjee

Lab 11 Complexity Analysis

SSW 315

14 November 2022

Complexity Analysis #2 - Image Helper

Method	**Worst Case**	**Storage Complexity**	**Justification for non-constant complexities**
validMatrix(int[][] matrix)	n=matrix.length	O(n)	The method runs on a single for-loop that depends on the length of the input matrix, running as many times as the size of the matrix.
flipVertical(int[][] matrix)	n=matrix.length	O(n^2)	The method runs on a nested for-loop which both depend on the length of the input matrix. The first runs as many times as the size of the matrix, and the second runs as many times as half of a matrix row's size.
flipHorizontal(int[][] matrix)	n=matrix.length	O(n^2)	The method runs on a previously created method which has a complexity of n, and the rest of the code runs on a for-loop that depends on the size of the matrix.
rotateClockwise(int[][] matrix)	n=matrix.length	O(n^3)	The method starts with a previously created method which has a complexity of n, and then runs two more methods from the class which have complexities of n^2 each. This gives an overall complexity of O(n)*O(n^2+n^2) which simplifies to O(n^3).
rotateCounterClockwise(int[][] matrix)	n=matrix.length	O(n^3)	The method starts with a previously created method which has a complexity of n, and then runs two more methods from the class which have complexities of n^2 each. This gives an overall complexity of O(n)*O(n^2+n^2) which simplifies to O(n^3).
transpose(int[][]	n=matrix.leng	O(n^2)	The method runs on a nested for-loop

matrix)	th		which both depend on the length of the input matrix. The first runs as many times as the size of the matrix as does the second.
reflect(int[][] matrix)	n=matrix.length	$O(n^2)$	The method runs on a nested for-loop which both depend on the length of the input matrix. The first runs as many times as the size of the matrix, and the second runs as many times as half of the matrix's size.

```
//Sohan Chatterjee
//SSW 315 Image Helper
//October 5, 2022
public class ImageHelper {
    static boolean validMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            if (matrix.length != matrix[i].length)
                return false;
        }
        return true;
    }

    static void flipVertical(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length / 2; j++) {
                int temp = matrix[i][matrix[i].length - j - 1];
                matrix[i][matrix[i].length - j - 1] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }

    static void flipHorizontal(int[][] matrix) {
        if (validMatrix(matrix)) {
            int size = matrix.length - 1;
            for (int i = size / 2; i >= 0; i--) {
                int[] temp = matrix[i];
                matrix[i] = matrix[size - i];
                matrix[size - i] = temp;
            }
        }
    }

    static void rotateClockwise(int[][] matrix) {
        if (validMatrix(matrix)) {
            transpose(matrix);
            reflect(matrix);
        }
    }

    static void rotateCounterClockwise(int[][] matrix) {
        if (validMatrix(matrix)) {
            reflect(matrix);
            transpose(matrix);
        }
    }

    static void transpose(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = i + 1; j < matrix.length; j++) {
                int temp = matrix[j][i];
                matrix[j][i] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }

    static void reflect(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length / 2; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[i][matrix.length - j - 1];
                matrix[i][matrix.length - j - 1] = temp;
            }
        }
    }
}
```