2.Medium\13.Spiral_traversal.cpp

```
/*
QUESTION:-

Given an m x n matrix, return all elements of the matrix in spiral order.

Example 1:
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]

Example 2:
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]

*/

/*
APPROACH:-

To traverse the matrix in a spiral order, we can use the following steps:

1. Initialize four variables: top, bottom, left, and right to keep track of the boundaries of the current spiral.
2. Create an empty vector called 'ans' to store the elements in spiral order.
3. While the top boundary is less than or equal to the bottom boundary and the left boundary is less than or equal to the right boundary:
   - Traverse the top row from left to right and add each element to 'ans'.
   - Increment the top boundary.
   - Traverse the right column from top to bottom and add each element to 'ans'.
   - Decrement the right boundary.
   - Check if the top boundary is still less than or equal to the bottom boundary:
     - Traverse the bottom row from right to left and add each element to 'ans'.
     - Decrement the bottom boundary.
   - Check if the left boundary is still less than or equal to the right boundary:
     - Traverse the left column from bottom to top and add each element to 'ans'.
     - Increment the left boundary.
4. Return the 'ans' vector containing all the elements in spiral order.

*/

// CODE:

vector<int> spiralOrder(vector<vector<int>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size();
    int top = 0, bottom = n - 1;
    int left = 0, right = m - 1;
    vector<int> ans;

    while (top <= bottom && left <= right) {
        // Traverse top row
        for (int i = left; i <= right; i++) {
            ans.push_back(matrix[top][i]);
```

```cpp
        }
        top++;

        // Traverse right column
        for (int i = top; i <= bottom; i++) {
            ans.push_back(matrix[i][right]);
        }
        right--;

        // Traverse bottom row
        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                ans.push_back(matrix[bottom][i]);
            }
            bottom--;
        }

        // Traverse left column
        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                ans.push_back(matrix[i][left]);
            }
            left++;
        }
    }

    return ans;
}

// TIME COMPLEXITY: O(N), where N is the total number of elements in the matrix.
// SPACE COMPLEXITY: O(1)

```