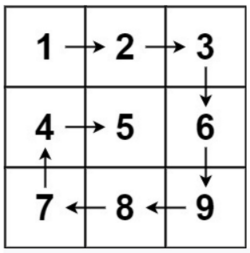
30. Spiral Traversal of a Matrix

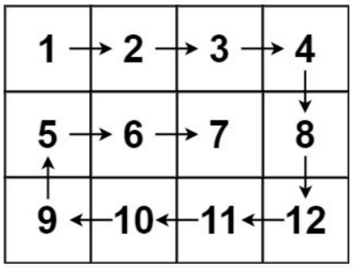
Given an `m x n matrix`, return *all elements of the* `matrix` *in spiral order.*

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:



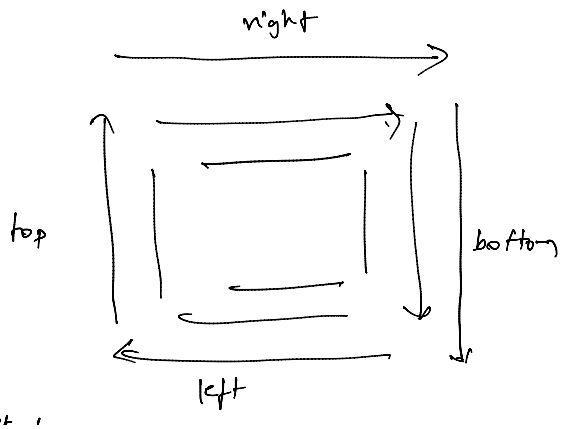

```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

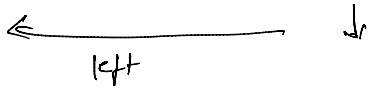
n x m ⇒ print it in spiral order.

1	2	3	4	5	6
20	21	22	23	24	7
19	32	33	34	25	8
18	31	36	35	26	9
17	30	29	28	27	10
16	15	14	13	12	11

o/p ⇒ 1 2 3 4 5 6 7 8 . . . 36

Optimal Solution :





1st layer

right → bottom → left → top

2nd layer

right → bottom → left → top

3rd layer

right → bottom → left

left left right right

	0	1	2	3	4	5	
0 top	1	2	3	4	5		⟹ to avoid repentation (for the top we move down to 1st index)
top	20	21	22	23	24	7	
2	19	32	33	34	25	8	
3	18	31	36	35	26	9	
4	17	30	29	28	27	10	
bottom 5	16	15	14	13	12		⟹ as bottom is done 'i' should be ignored so move right back

top = 0 bottom = 5

left = 0 right = 5

left ← right

1st layer printing

1st printing left → right

for (left → right) ⟹ right will print

a[top][i] ⟹ it stay constant

top++; ⟹ this will make sure right is performed if it's available.

2nd printing

top is gone down now we need to print bottom ↓

so we go top → bottom

for (i = top → bottom]

it stay constant a [top] [i]

top++

} bottom coll print

3rd printing right → left

for (i = right → left)

a [bottom] [i]

bottom--

} left is printed

4th printing top

bottom → top

for (i = bottom → top)

a [i] [left];

left++;

2nd layer printing

- if the left is smaller than right i.e there is column that has to be traversed (spiraler.
- similarly for top is lesser than bottom , " " " .

→ while (top <= bottom && left <= right)

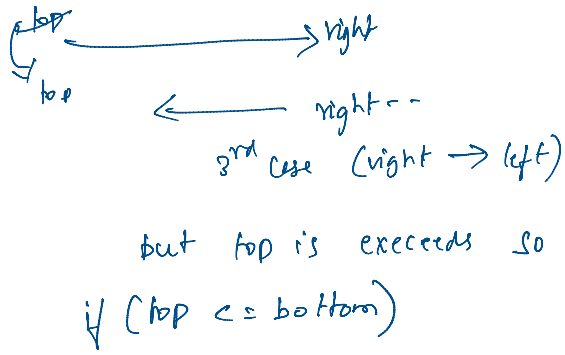
Now if we just single line it will perform only right

→

1st case will perform 1 right.

now top++

No need to do extra checks



but top is exceeds so check

if (top <= bottom)

```cpp
vector<int> spiralMatrix(vector<vector<int>> &matrix) {
  int n = matrix.size();
  int m = matrix[0].size();

  int left = 0, right = m - 1;
  int top = 0, bottom = n - 1;

  vector<int> ans;
  while (top <= bottom && left <= right) {
    // right
    for (int i = left; i <= right; i++) {
      ans.push_back(matrix[top][i]);
    }
    top++;

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

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

C++ Autocomplete

```cpp
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
      int n = matrix.size();
      int m = matrix[0].size();

      int left = 0, right = m - 1;
      int top = 0, bottom = n - 1;

      vector<int> ans;
      while (top <= bottom && left <= right) {
        // right
        for (int i = left; i <= right; i++) {
          ans.push_back(matrix[top][i]);
        }
        top++;

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

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

T.C => O(n×m)

S.C => O(n×m)