To rotate a given n x n 2D matrix by 90 degrees clockwise **in-place**, the solution follows a two-step approach:

Step 1: Transpose the Matrix

The first step is to transpose the matrix. Transposing means converting all rows into columns (or vice versa). Specifically, for each element at position `(i, j)` in the matrix, it swaps with the element at position `(j, i)`. This is done using the nested loops:

```
for(int i = 0; i < s; i++) {
    for(int j = i + 1; j < s; j++) {
        swap(matrix[i][j], matrix[j][i]);
    }
}
```

- The outer loop iterates over the rows (`i`), and the inner loop iterates over the columns starting from `i+1` (`j`).
- The `swap` function exchanges the elements at positions `(i, j)` and `(j, i)`.
- After this step, the matrix is transposed.

For example: Original matrix:

```
1 2 3
4 5 6
7 8 9
```

After transpose:

```
1 4 7
2 5 8
3 6 9
```

Step 2: Reverse Each Row

The second step is to reverse each row of the transposed matrix. Reversing a row flips the order of its elements. This is done using the `reverse` function:

```
for(int i = 0; i < s; i++) {
    reverse(matrix[i].begin(), matrix[i].end());
}
```

- The outer loop iterates over each row (`i`).
- The `reverse` function reverses the elements of the row from the beginning to the end.

For example: Transposed matrix:

```
1 4 7
2 5 8
3 6 9
```

After reversing each row:

```
7 4 1
8 5 2
9 6 3
```

Why Does This Work?

Rotating a matrix by 90 degrees clockwise can be broken down into these two steps:

1. **Transpose**: Converts rows into columns.
2. **Reverse Rows**: Flips the order of elements in each row to achieve the final rotation.

By combining these two operations, the matrix is effectively rotated in-place without requiring additional memory for another matrix.

Final Output

For the input matrix:

```
1 2 3
4 5 6
7 8 9
```

The output after applying the solution will be:

```
7 4 1
8 5 2
9 6 3
```

This matches the expected result of a 90-degree clockwise rotation.

Key Points

1. The solution modifies the matrix **in-place**, ensuring no extra space is used for another matrix.
2. The time complexity is **O(n^2)** because we iterate through all elements of the matrix twice (once for transposing and once for reversing).
3. The space complexity is **O(1)** since no additional data structures are used apart from a few variables.

Final Answer: The solution works by first transposing the matrix and then reversing each row, achieving a 90-degree clockwise rotation in-place.