Search In a Matrix II LeetCode

You are given a sorted 2D matrix, where each row and each column is sorted in ascending order. Your task is to determine whether a given target value exists in the matrix. If the target is present, return true; otherwise, return false.

matrix = [

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

], Target: 9

Output: True

Approach 1: Function to search for a target using the linear search approach

- Method: This approach uses a linear search method. It iterates through each element in the matrix and checks if the current element is equal to the target.
- **Time Complexity: O(rows * cols) where rows is the number of rows and cols is the number of columns in the matrix.**
- **Space Complexity: O(1) since no additional space is used.**
- This approach is not efficient for large sorted matrices.

Approach 2: Function to search for a target using the binary search approach

- Method: This approach flattens the 2D matrix into a 1D array and performs binary search on it. It maps the mid index back to the corresponding row and column indices using modulo and integer division.
- **Time Complexity: O(log(rows * cols)) where rows is the number of rows and cols is the number of columns in the matrix.**
- **Space Complexity: O(1) since no additional space is used.**
- This approach is efficient for large sorted matrices but involves additional calculations and complexity compared to Approach 3.

Approach 3: Function to search for a target using the Divide and Conquer approach

- Method: This approach efficiently searches for the target in the sorted 2D matrix. It starts the search from the top-right corner of the matrix and uses binary search-like steps. Based on comparisons with the target, it narrows down the search space either by moving down to the next row or to the left to the previous column.
- **Time Complexity: O(rows + cols) where rows is the number of rows and cols is the number of columns in the matrix.**
- **Space Complexity: O(1) since no additional space is used.**
- This approach is the best among the three for searching in a sorted 2D matrix. It efficiently leverages the sorted property of the matrix.
- **Explanation of Time Complexity:**
    - In this approach, we start the search from the top-right corner of the matrix, and at each step, we either move down to the next row or to the left to the previous column. We can never move up or to the right, ensuring that we never revisit any element. Hence, the search will cover at most rows + cols elements.
    - In the worst case, we might need to traverse all the rows and all the columns of the matrix. Therefore, the time complexity of this approach is O(rows + cols).

- **Approach 3 Walk through on the given Matrix:**

```
matrix = [
  [1, 4, 7, 11, 15],
  [2, 5, 8, 12, 19],
  [3, 6, 9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
], target = 9
```

1. Start from the top-right corner: We start the search from **matrix[0][4]**, which is the top-right element (**15**).
2. Compare with the target:
    - **15** is greater than the target **9**, so we need to move to the left to explore smaller values.
    - We move to **matrix[0][3]**, which is **11**.
3. Compare with the target:
    - **11** is greater than the target **9**, so we move to the left again.

- We move to **matrix[0][2]**, which is **7**.

4. Compare with the target:
    - **7** is less than the target **9**, so we need to explore greater values.
    - Since we cannot move right anymore, we move down to **matrix[1][2]**, which is **8**.
5. Compare with the target:
    - **8** is less than the target **9**, so we need to explore greater values.
    - Since we cannot move right anymore, we move down to **matrix[2][2]**, which is **9**.
6. Target found:
    - The element at **matrix[2][2]** is **9**, which is the target we are looking for.
    - We return **true** as the target is found.