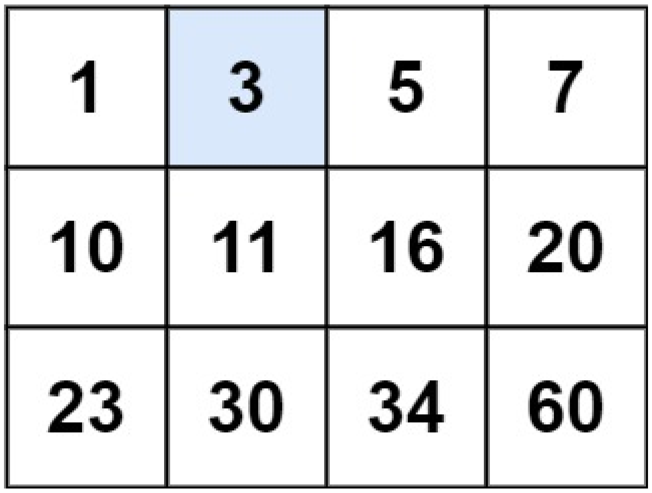
Search a 2D Matrix

Write an efficient algorithm that searches for a value `target` in an `m x n` integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.
- **Example 1:**



1	3	5	7
10	11	16	20
23	30	34	60

- **Input:** `matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3`
- **Output:** `true`

Naive approach: The simple idea is to traverse the array and to search elements one by one.

Algorithm:

Run a nested loop, outer loop for row and inner loop for the column

Check every element with x and if the element is found then print “element found”

If the element is not found, then print “element not found”.

```java
static boolean search(int[][] mat, int x)
 {
   for (int i = 0; i < mat.length; i++) {
     for (int j = 0; j < mat[i].length; j++)
       // if the element is found
       if (mat[i][j] == x) {
         return true;
       }
   }
   return false;
 }
```

Time Complexity: O(n2)
Auxiliary Space: O(1)

Efficient approach: The simple idea is to remove a row or column in each comparison until an element is found. Start searching from the top-right corner of the matrix. There are three possible cases.

1. **The given number is greater than the current number:** This will ensure that all the elements in the current row are smaller than the given number as the pointer is already at the right-most elements and the row is sorted. Thus, the entire row gets eliminated and continues the search for the next row. Here, elimination means that a row needs not be searched.
2. **The given number is smaller than the current number:** This will ensure that all the elements in the current column are greater than the given number. Thus, the entire column gets eliminated and continues the search for the previous column, i.e. the column on the immediate left.
3. **The given number is equal to the current number:** This will end the search.

Algorithm:

1. Let the given element be x, create two variable *i = 0, j = n-1* as index of row and column
2. Run a loop until i = n
3. Check if the current element is greater than x then decrease the count of j. Exclude the current column.
4. Check if the current element is less than x then increase the co

Time Complexity: O(n), Only one traversal is needed, i.e, i from 0 to n and j from n-1 to 0 with at most 2*n steps. The above approach will also work for m x n matrix (not only for n x n). Complexity would be O(m + n).
Auxiliary Space: O(1), No extra space is required.

```
public boolean searchMatrix(int[][] matrix, int target) {

    int n=matrix.length;

    int m=matrix[0].length;

    int i=n-1;

    int j=0;

    while(i>=0&&j<m)

    {

      if(matrix[i][j]==target)

      {

        return true;

      }

      else if(matrix[i][j]>target)

      {
            i--;
        }
        else
        {
           j++;
        }
    }
    return false;
}
```