Search in a 2D matrix

BRUTE FORCE

Search in entire matrix if element is found return true.

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
    for(int i=0;i<matrix.size();i++)
    {
        for(int j=0;j<matrix[0].size();j++)
        {
            if(matrix[i][j]==target)
                return true;
        }
    }
    return false;
    }
};
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Better Approach :

Firstly consider corner element of row 0 so that there is differentiating factor because element on the left of the right corner element is smaller and down element is greater.

So if curr element is greater than target then we move left else we move right.

There will occur a condition where i or j will get out of bounds of the matrix moving left or down.

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m=matrix.size(),n=matrix[0].size();
        int i=0,j=n-1;
        while(j>=0 && i<m)
        {
            if(matrix[i][j]==target)
            {
                return true;
            }
            else if(matrix[i][j]>target)
            {
                j--;
            }
            else
            {
                i++;
            }

        }
        return false;
    }
};
```

Optimal Approach : Binary Search

Flatten 2d matrix to 1d array then use Binary search to find the element in the flattened array.

index the array from 0 to m*n-1 in 1d array

To find the corresponding index in 2d array

1. finding row - divide the index/ no. of column - we will reach in that row.
2. To find column now index% no. of column gives column index.

now we compare whether elem > mid if yes we move to right half else to left half

```
bool searchMatrix(vector<vector<int>>& mat, int target) {
    int m=mat.size(), n=mat[0].size();
    int low=0, high=(m*n-1);
    while(low<=high)
    {
        int mid=(low+high)/2;
        if(mat[mid/n][mid%n]==target)
        {
            return true;
        }
        else if (mat[mid/n][mid%n]>target)
        {
            high=mid-1;
        }
```

```
          else
          {
              low=mid+1;
          }
      }
       return false;
  }
```

- Time Complexity : O(log(m+n))
- Space complexity : O(1)