**1. Problem Discussion:**

We have to write an efficient algorithm that searches for a value in a m x n matrix. This matrix has the following properties: a) Integers in each row are sorted from left to right. b) The first integer of each row is greater than the last integer of the previous row.

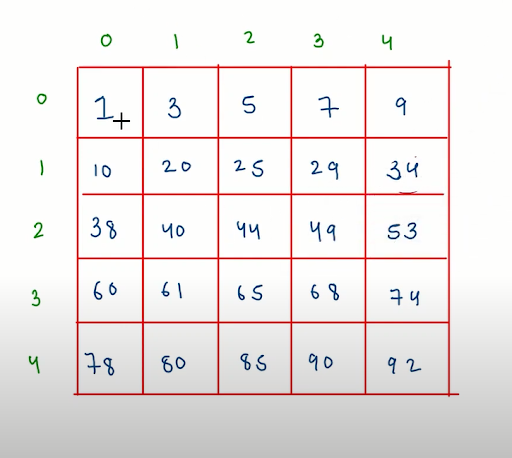
Approach:

Simple Approach:

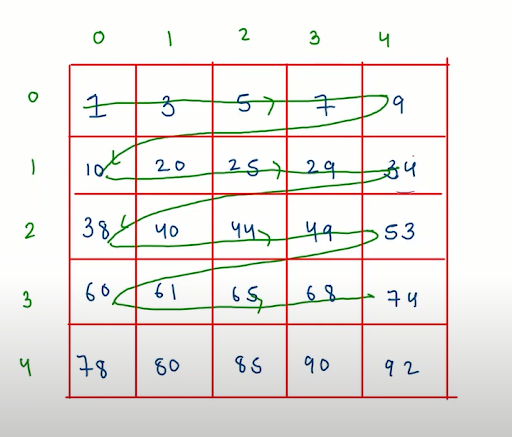
1• Traverse in the m x n matrix and search the element one by one. 2• Run two nested for loop one for row and other for column and check each element one by one with the target. If the element is equal to the target return true and if the target is not in the matrix return false.

Efficient Approach:

We have given the elements in the matrix where each row is sorted from left to right and the first element of each row is greater than the last element of the previous row. We can use this property of the matrix to find the element in the matrix.

****

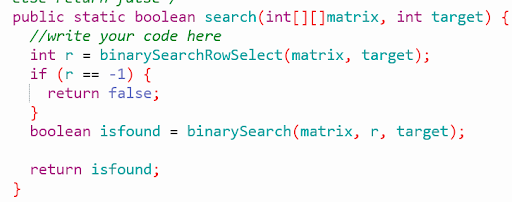
The sorted array elements in the above image is like:

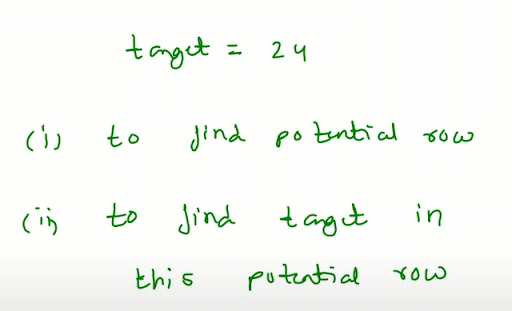
****

Here in the above image we can see that the matrix given is sorted. With this property we have the matrix in the sorted form and we have to just find the target in a sorted 2D array. We have a target to find the matrix the approach is to use two level binary search in a 2D matrix the steps for this are: 1. To find the potential row where the target element can be found. This first step will find the target that belongs to which row by using binary search in rows. This is first level binary search. 2. To find the target element in this potential row. This step will find the target in that row we have found in the 2D array. For finding the target we will use binary search because each row is sorted already. This is second level binary search.

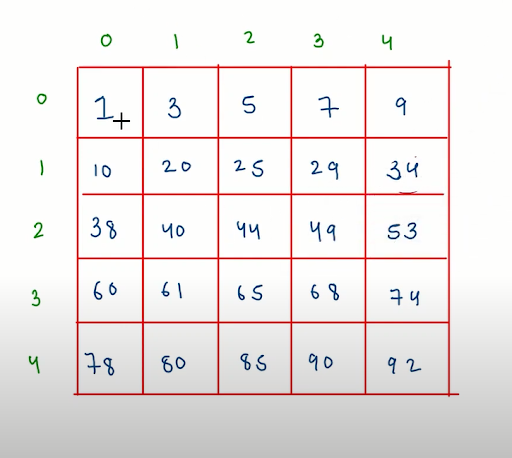
Code Discussion:

1• Find the row of the target i.e. the targets belong to which row we have to find that first. For that make a function which will find the row of the target and if the target is not in the row that return -1 for that case. 2• Next if row = -1 this means target element is not in the matrix so return false by checking this condition. 3• If the row is not equal to -1 this means we have to find the target in that row and if the target is in that row it will return true else it will return false.

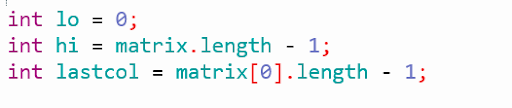
****

****

4• Let’s say we have this given matrix and we have to find 24 as the target.

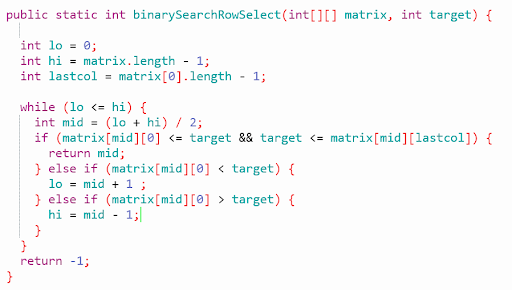
****

5• The two steps are to find the potential row i.e. the row where the target belongs to. For that we will use binary search.

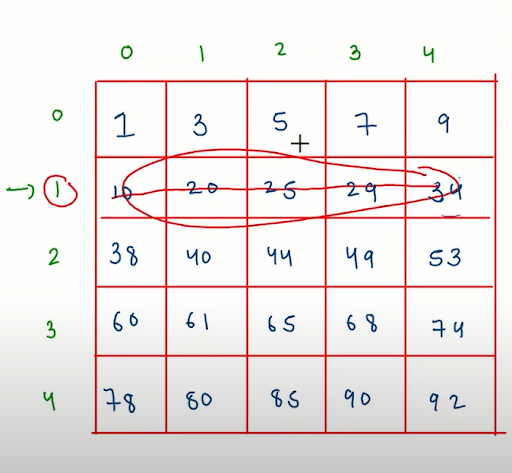
****

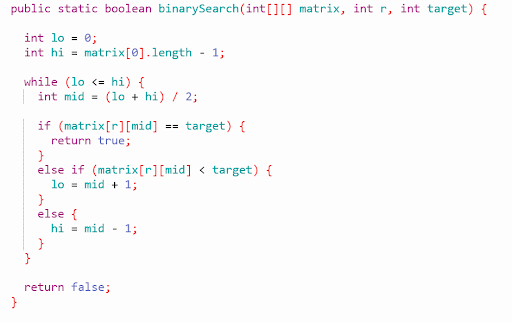
We have low and high of the row in the matrix and we need the lastcol also to check the range for the target. This is for we need check the target at mid row it should be in range of first column and the last column of the that row if it is true than we will return the mid from there.

6• If the target is not in the range of matrix[mid][0] and matrix[mid][lastcol] than we need to check for the first column of the matrix for the binary search. If the target is greater than lo = mid +1 and if the target is smaller than hi = mid-1.

****

7• When the row is found then we will check for that row by using second level binary search.

****

****

Code:

ConsoleJava

import java.util.\*;

import java.io.\*;

public class Main {

/\*your task is to complete this function which returns true if target exists in the matrix

else return false\*/

public static boolean search(int[][]matrix,int target) {

//write your code here

//select a row using binary search

int r = binarySearchRow(matrix,target);

if(r == -1) {

return false;

}

//search in rth row using binary search

boolean found = binarySearch(matrix,target,r);

return found;

}

public static int binarySearchRow(int[][] matrix,int target) {

int lo = 0;

int hi = matrix.length-1;

int lc = matrix[0].length-1;

while(lo <= hi) {

int mid = (lo+hi)/2;

if(matrix[mid][0] <= target && matrix[mid][lc] >= target) {

return mid;

}

else if(matrix[mid][0] < target) {

lo = mid+1;

}

else if(matrix[mid][0] > target) {

hi = mid-1;

}

}

return -1;

}

public static boolean binarySearch(int[][] matrix,int target,int r) {

int lo = 0;

int hi = matrix[0].length-1;

while(lo <= hi) {

int mid = (lo+hi)/2;

if(matrix[r][mid] == target) {

return true;

}

else if(matrix[r][mid] < target) {

lo = mid+1;

}

else {

hi = mid-1;

}

}

return false;

}

public static void main(String[]args) {

//input work

Scanner scn = new Scanner(System.in);

int m = scn.nextInt();

int n = scn.nextInt();

int[][]matrix = new int[m][n];

for(int i=0; i < m;i++) {

for(int j=0; j < n;j++) {

matrix[i][j] = scn.nextInt();

}

}

int target = scn.nextInt();

boolean isFound = search(matrix,target);

System.out.println(isFound);

}

}

Analysis:

Time Complexity: O(log m + log n) The log m is for the binary search complexity to find the row of the target in the matrix The log n is for the binary search complexity to find the target in that row we have found in first level binary search. Space Complexity: O(1) We have not used any extra space that's why the space complexity is O(1).