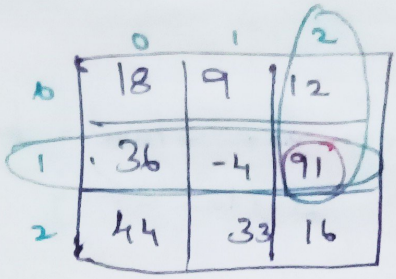
# Day-15

## Binary Search in 2D Array

### Search in Matrix



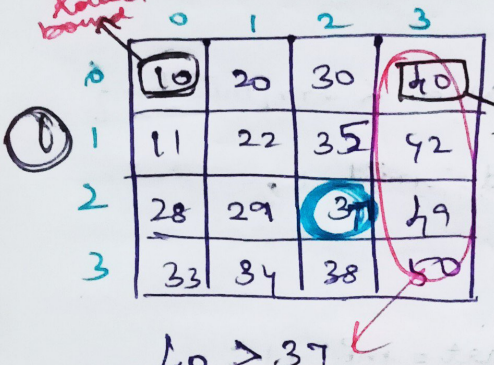

Target = 91

Ans: [1, 2]

N * N = $N^2$ => $O(N^2)$

```
for(row=0; r<n; r++){
  for(c=0; c<n; c++){
    if arr[r][c] = target {
      found ans
    }
  }
}
return -1;
```

**Prob:** Matrix is sorted in row & col wise manner. 37 - Target




Target = 37

40 > 37

Ignore entire col

We we get large no. of search, we need to run the search.

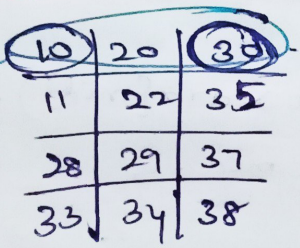
Case 1 : If element == target
ans found

Case 2 : If element < target
Ignore row Row++

Case 3 : If element > target
Ignore col Col--

(1)

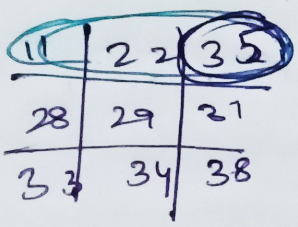
(2) Rem Remove




30 < 37

Remove row

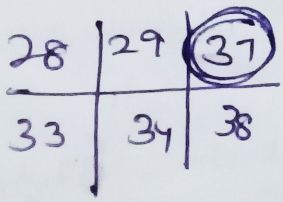
(3)




25 < 37

Remove row

(4)




37 = 37

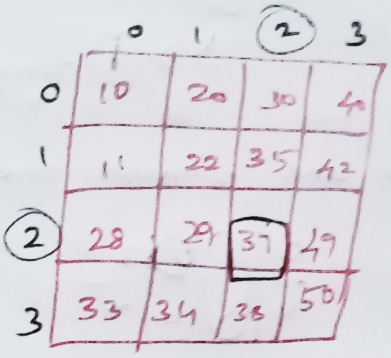
Found Target

Time Complexity:
O(N)

Space Complexity:
O(1) → Constant
No additional space taken

## Program

```
import java.util.Arrays;
class binary2d {
    public static void main (String [] args) {
        int [][] arr = {
                {10, 20, 30, 40},
                {11, 22, 35, 42},
                {28, 29, 37, 49},
                {33, 34, 38, 50}
        };
        int target = 49;
        System.out.println (Arrays.toString (search (arr, target)));
    }
    static int[] search (int [][] matrix, int target) {
        int r = 0;
        int c = matrix.length - 1;
        while (r < matrix.length && c >= 0) {
            if (matrix [r] [c] == target) {
                return new int [] {r, c};
            }
            if (matrix [r] [c] < target) {
                r++;
            }
            else {
                c--;
            }
        }
        return new int [] {-1, -1};
    }
}
```




**olp:**

{2, 2}

If target = 38

{3, 2}

If target = 9

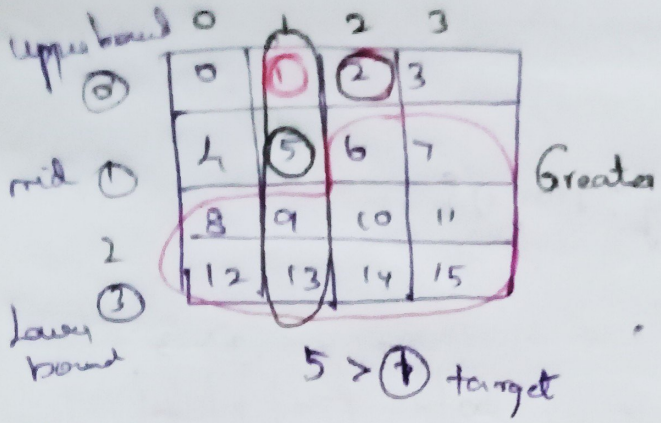
{-1, -1}

**Boolean**

```
public boolean search (int [][]
        matrix, int target){
    for (int i = 0; i < matrix.length;
            i++) {
        for (int j = 0; j < matrix[0].
                length; j++) {
            if (matrix [i] [j] == target){
                return true;
            }
        }
    }
    return false;
}
```

olp will be true or false

# Search in sorted matrix

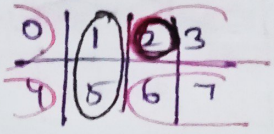



① If element == target
ans

② If element > target
Remove rows after it

③ If element < target
Remove above rows

In the end 2 rows are remaining:




① Check whether the mid col your are at contains the ans [1, 5]

② Consider the four parts

⇒ Go with simple binary search

To get boolean o/p

Program:

```
import java.util.Arrays;
public class SortedMatrixSearch {
    public static void main(String[] args){
        int[][] arr = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int target = 5;   o/p: [1, 1]   [-1, -1]
        System.out.println(Arrays.toString(search(arr, target)));
    }
```

not required

boolean

```
    static int[] binarySearch(int[][] matrix, int row, int cstart,
                              int cend, int target) {
```

Binary Search on col

```
        while (cstart <= cend) {
            int mid = cstart + (cend - cstart) / 2;
            if (matrix[row][mid] == target) {
                return new int[] {row, mid};   return true;
            } else if (matrix[row][mid] < target) {
                cstart = mid + 1;
            } else {
                cend = mid - 1;
            }
```

```
        }
        return new int[]{-1,-1};        return false;
    }
    Boolean
    static int[] search(int[][] matrix, int target){
Empty Matrix
        int rows = matrix.length;
        if(rows==0){
            return new int[]{-1,-1};    return false;
        }
        int cols = matrix[0].length;
        int rstart = 0;
        int rend = rows-1;
Binary Search on rows
        while(rstart < rend){
            int mid = rstart + (rend - rstart)/2;
            if(matrix[mid][0] <= target && target <= matrix
                                        [mid][cols-1]){
                rstart = mid;
                break;
            } else if (matrix[mid][0] < target) {
                rstart = mid+1;
            } else {
                rend = mid-1;
            }
        }
Out-of-Bounds
        if(rstart >= rows){
            return new int[]{-1,-1};    return false;
        }
Search in identified row
        return binarySearch(matrix, rstart, cstart:0, cols-1,
                                                target);
    }
}
```