```
package Love_Babbar;

public class Matrix {
    class Matrix {
        //
**********************************************************************************************************
***************
        // Spiral traversal on a Matrix
        class Solution {
            // Function to return a list of integers denoting spiral traversal of matrix.
            static ArrayList<Integer> spirallyTraverse(int matrix[][], int r, int c) {
                // code here
                ArrayList<Integer> res = new ArrayList<>();
                int left = 0, right = r - 1, top = 0, bottom = c - 1;
                while (left < right && top < bottom) {
                    for (int i = left; i <= right; i++) {
                        res.add(matrix[top][i]);
                    }
                    top++;

                    for (int i = top; i <= bottom; i++) {
                        res.add(matrix[i][right]);
                    }
                    right--;

                    for (int i = right; i >= left; i--) {
                        res.add(matrix[bottom][i]);
                    }
                    bottom--;
                    for (int i = bottom; i >= top; i--) {
                        res.add(matrix[i][left]);
                    }
                    left++;
                }
                return res;
            }
        }
        // Search an element in a Matrix(Upper Right Corner method)
        class Solution {
            // Function to search a given number in row-column sorted matrix.
            static boolean search(int matrix[][], int n, int m, int x) {
                // code here
                int r = 0, c = m - 1;
                while (r < n && c >= 0) {
                    if (matrix[r][c] == x) {
                        return true;
                    } else if (matrix[r][c] < x) {
                        r++;
                    } else {
```

```
            c--;
        }
    }
    return false;
  }
}

// Find median in a row wise sorted matrix
class Solution {
  int median(int matrix[][], int R, int C) {
    // code here
    int min_ele = Integer.MAX_VALUE;
    int max_ele = Integer.MIN_VALUE;
    for (int i = 0; i < R; i++) {
      if (matrix[i][0] < min_ele) {
        min_ele = matrix[i][0];
      }
      if (matrix[i][C - 1] > max_ele) {
        max_ele = matrix[i][C - 1];
      }
    }
    int half_ele = (R * C + 1) / 2;
    while (min_ele < max_ele) {
      int mid = min_ele + (max_ele - min_ele) / 2;
      int count_less_than_mid = 0;
      for (int i = 0; i < R; i++) {
        int l = 0, r = C - 1;
        while (l < r) {
          int m = (l + r) / 2;
          if (matrix[i][m] <= mid) {
            l = m + 1;
          } else {
            r = m;
          }
        }
        if (matrix[i][l] <= mid) {
          count_less_than_mid++;
        }
        count_less_than_mid+= l;
      }
      if (count_less_than_mid < half_ele) {
        min_ele = mid + 1;
      } else {
        max_ele = mid;
      }
    }
    return min_ele;
  }
}
```

```
        // Find row with maximum no. of 1's
        class Solution {
            int rowWithMax1s(int arr[][], int n, int m) {
                // code here
                int ans = -1;
                int max_ones = 0;
                for (int i = 0; i < n; i++) {
                    int count = f(arr, i, n, m);

                    if (count > max_ones) {
                        ans = i;
                        max_ones = count;
                    }
                }
                return ans;
            }

            int f(int arr[][], int row, int n, int m) {

            }
        }
        // Print elements in sorted order using row-column wise sorted matrix
        // Maximum size rectangle
        // Find a specific pair in matrix NA
        // Rotate matrix by 90 degrees
        // Kth smallest element in a row-column wise sorted matrix
        // Common elements in all rows of a given matrix
    }
}
```