TASK 9

AIM: To compute the block sum of each element in a given 2D matrix using an efficient prefix sum technique.

ALGORITHM:

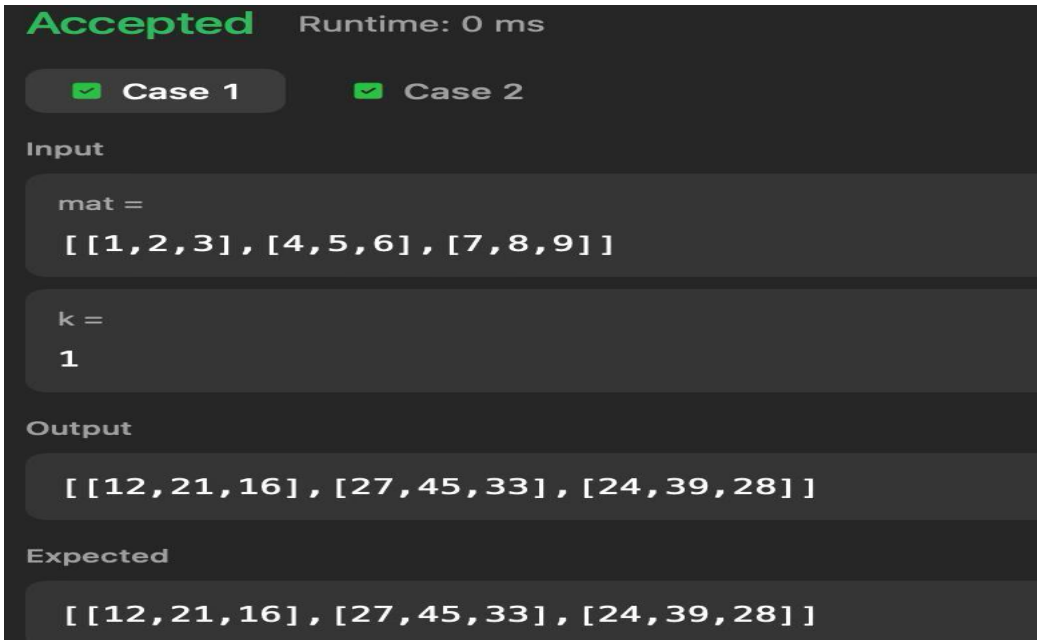
1. Read the matrix mat and integer k.
2. Get dimensions m (rows) and n (columns).
3. Create a prefix sum matrix of size (m+1) x (n+1).
4. Fill the prefix matrix using:
   - prefix[i][j] =
   - mat[i-1][j-1]
   - prefix[i-1][j]
   - prefix[i][j-1]
   - prefix[i-1][j-1]
5. Create result matrix answer[m][n].
6. For each element (i, j):
7. Calculate block boundaries:
   - r1 = max(0, i-k)
   - c1 = max(0, j-k)
   - r2 = min(m-1, i+k)
   - c2 = min(n-1, j+k)
8. Convert to prefix indexing.
9. Compute block sum using prefix formula.
10. Store the result in answer[i][j].
11. Return the result matrix.

PROGRAM:

```
class Solution {
    public int[][] matrixBlockSum(int[][] mat, int k) {
        int m = mat.length;
        int n = mat[0].length;

        int[][] prefix = new int[m + 1][n + 1];

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                prefix[i][j] = mat[i - 1][j - 1]
                        + prefix[i - 1][j]
                        + prefix[i][j - 1]
                        - prefix[i - 1][j - 1];
            }
        }

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

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

                int r1 = Math.max(0, i - k);
                int c1 = Math.max(0, j - k);
```

```
            int r2 = Math.min(m - 1, i + k);
            int c2 = Math.min(n - 1, j + k);

            r1++; c1++; r2++; c2++;

            answer[i][j] = prefix[r2][c2]
                        - prefix[r1 - 1][c2]
                        - prefix[r2][c1 - 1]
                        + prefix[r1 - 1][c1 - 1];
        }
      }

      return answer;
   }
}
```

OUTPUT



RESULT : The program successfully computes the sum of all elements within a k distance block for each cell in the matrix using an optimized O(m × n) approach.

TASK 10

AIM:To rotate a given 2D matrix r times in the anti-clockwise direction by rotating each layer (ring) independently.

ALGORITHM:

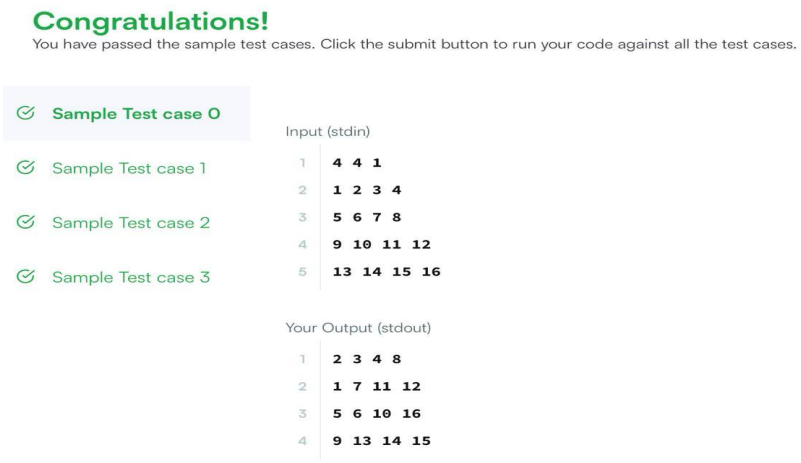
1. Read the values of:
   - m → number of rows
   - n → number of columns
   - r → number of rotations
2. Input the matrix elements into a 2D array.
3. Calculate the number of layers:
   - layers = min(m, n) / 2
4. For each layer:
   - Define boundaries:
   - top = layer
   - bottom = m - 1 - layer
   - left = layer
   - right = n - 1 - layer
5. Extract elements of the current layer in order:
   - Top row (left to right)
   - Right column (top+1 to bottom-1)
   - Bottom row (right to left)
   - Left column (bottom-1 to top+1)
   - Store extracted elements in a list.
6. Compute effective rotations:
   - rotations = r % size_of_layer
   - Rotate the list anti-clockwise using:
7. Collections.rotate(list, -rotations)
8. Place the rotated elements back into the matrix in the same order.
9. Repeat for all layers.
10. Print the final rotated matrix.

PROGRAM

```
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int m = sc.nextInt();
        int n = sc.nextInt();
        int r = sc.nextInt();
```

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

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        int layers = Math.min(m, n) / 2;

        for (int layer = 0; layer < layers; layer++) {

            List<Integer> elements = new ArrayList<>();

            int top = layer;
            int left = layer;
            int bottom = m - 1 - layer;
            int right = n - 1 - layer;
            for (int i = left; i <= right; i++)
                elements.add(matrix[top][i]);
            for (int i = top + 1; i <= bottom - 1; i++)
                elements.add(matrix[i][right]);
            for (int i = right; i >= left; i--)
                elements.add(matrix[bottom][i]);
            for (int i = bottom - 1; i >= top + 1; i--)
                elements.add(matrix[i][left]);

            int size = elements.size();
            int rotations = r % size;
            Collections.rotate(elements, -rotations);

            int index = 0;
            for (int i = left; i <= right; i++)
                matrix[top][i] = elements.get(index++);
            for (int i = top + 1; i <= bottom - 1; i++)
                matrix[i][right] = elements.get(index++);
            for (int i = right; i >= left; i--)
                matrix[bottom][i] = elements.get(index++);
            for (int i = bottom - 1; i >= top + 1; i--)
                matrix[i][left] = elements.get(index++);
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

OUTPUT:



RESULT :The program successfully rotates the matrix in the anti-clockwise direction by r times using an efficient layer-by-layer approac