2D Array Problems -1

Assignment Solutions

Q1 - Given two integer matrices, multiply the matrices, if possible, else return "Invalid Input". (Medium)

Input1:

n1 = 2

m1 = 3

arr1 = {{2,4,1}, {3,5,6}}

n2 = 3

m2 = 2

arr2 = {{1,2}, {3,4}, {5,7}}

Expected Output:

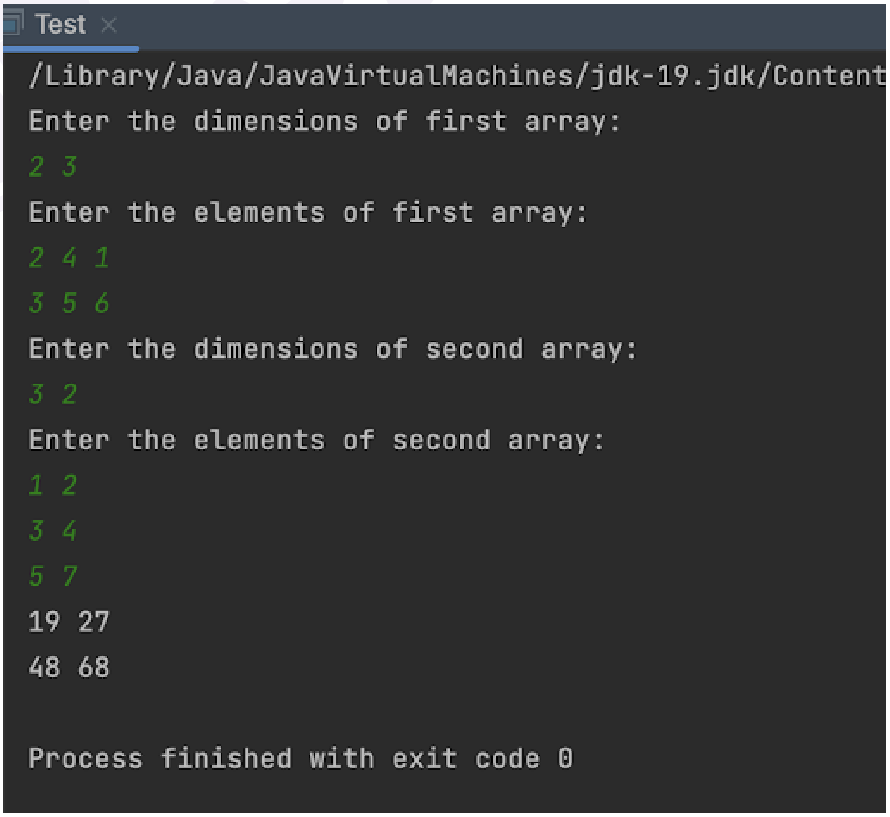
```
19 27
48 68
```

Explanation:

- To multiply 2 matrices which are not square matrices, the number of columns in the 1st matrix is equal to the rows in the 2nd matrix.
- If the condition is not satisfied, we print Invalid Input and return.
- Else use a blank array with rows = number of rows in 1st matrix and columns = number of columns in 2nd matrix.
- Multiply and store the answer like we do for a square matrix.

Code:

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter the dimensions of first array: ");
    int n1 = scn.nextInt();
    int m1 = scn.nextInt();
    int[][] arr1 = new int[n1][m1];
    System.out.println("Enter the elements of first array: ");
    for (int i = 0; i < n1; i++) {
      for (int j = 0; j < m1; j++) {
        arr1[i][j] = scn.nextInt();
      }
    }
    System.out.println("Enter the dimensions of second array: ");
    int n2 = scn.nextInt();
    int m2 = scn.nextInt();
    int[][] arr2 = new int[n2][m2];
    System.out.println("Enter the elements of second array: ");
    for (int i = 0; i < n2; i++) {
      for (int j = 0; j < m2; j++) {
        arr2[i][j] = scn.nextInt();
      }
    }
    if (m1 != n2) { //basic condition for multiplication to be possible
      System.out.print("Invalid input");
      return;
    }
    int[][] ans = new int[n1][m2];
    for (int i = 0; i < n1; i++) {
      for (int j = 0; j < m2; j++) {
       for (int k = 0; k < m1; k++) {
        ans[i][j] += (arr1[i][k] * arr2[k][j]);
       }
      }
    }
    for (int i = 0; i < n1; i++) {
      for (int j = 0; j < m2; j++) {
        System.out.print(ans[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Content
Enter the dimensions of first array:
2 3
Enter the elements of first array:
2 4 1
3 5 6
Enter the dimensions of second array:
3 2
Enter the elements of second array:
1 2
3 4
5 7
19 27
48 68

Process finished with exit code 0
```

Q2 - Given a square matrix, rotate it by 90 degrees in anti clockwise direction. (Medium)

Input1:

n = 3

m = 3

matrix = [[1,2,3],[4,5,6],[7,8,9]]

Expected Output:

3 6 9

2 5 8

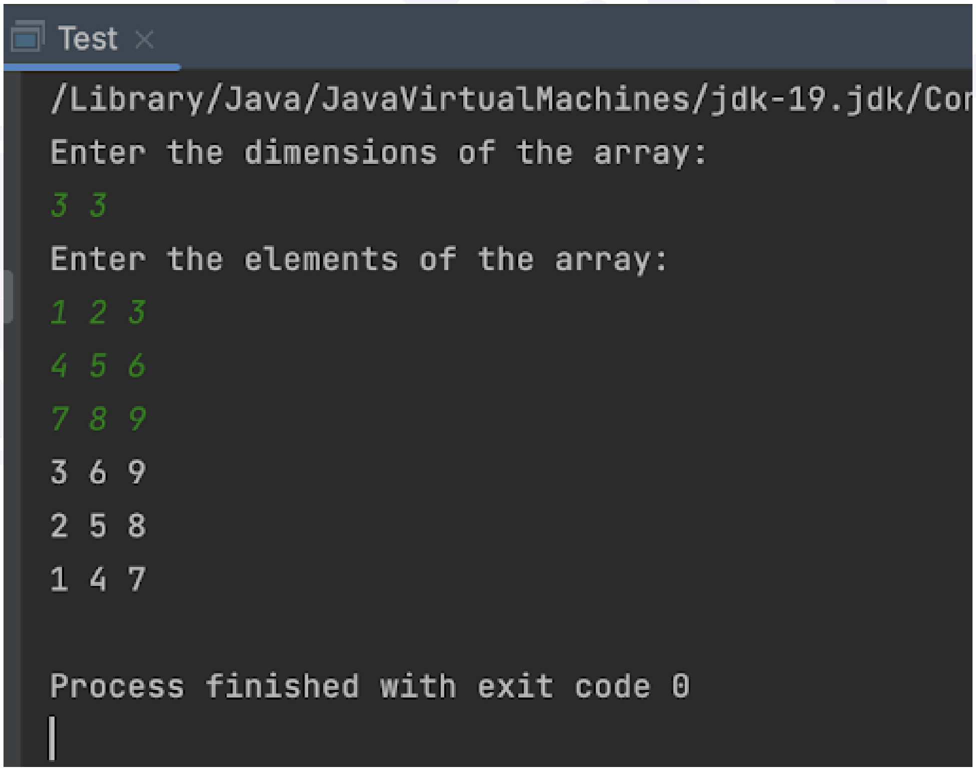
1 4 7

Explanation:

- There are n/2 squares or cycles in a matrix of side n. Process a square one at a time. Run a loop to traverse the matrix a cycle at a time, i.e loop from 0 to n/2 – 1, loop counter is i
- Consider elements in group of 4 in current square, rotate the 4 elements at a time. So the number of such groups in a cycle is n – 2*i.
- So run a loop in each cycle from x to n – x – 1, loop counter is y
- The elements in the current group is (x, y), (y, n-1-x), (n-1-x, N-1-y), (n-1-y, x), now rotate the these 4 elements, i.e (x, y) <- (y, n-1-x), (y, n-1-x)<- (n-1-x, n-1-y), (n-1-x, n-1-y)<- (n-1-y, x), (n-1-y, x)<- (x, y)
- Print the matrix.

Code:

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter the dimensions of the array: ");
    int n = scn.nextInt();
    int m = scn.nextInt();
    int[][] mat = new int[n][m];
    System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        mat[i][j] = scn.nextInt();
      }
    }
    for (int x = 0; x < n / 2; x++) {
      for (int y = x; y < n - x - 1; y++) {
        // Store current cell in temp variable
        int temp = mat[x][y];
        // Move values from right to top
        mat[x][y] = mat[y][n - 1 - x];
        // Move values from bottom to right
        mat[y][n - 1 - x] = mat[n - 1 - x][n - 1 - y];
        // Move values from left to bottom
        mat[n - 1 - x][n - 1 - y] = mat[n - 1 - y][x];
        // Assign temp to left
        mat[n - 1 - y][x] = temp;
      }
    }
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        System.out.print(mat[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Co
Enter the dimensions of the array:
3 3
Enter the elements of the array:
1 2 3
4 5 6
7 8 9
3 6 9
2 5 8
1 4 7

Process finished with exit code 0
```

Q3 - Given a n*m matrix, return true if the matrix is a Toeplitz matrix. A matrix is called Toeplitz if every diagonal from top-left to bottom-right has the same elements. (Medium)

Input1:

n = 3

m = 4

arr[]=[[1, 2, 3, 4],[5, 1, 2, 3],[9, 5, 1, 2]]

Expected Output:

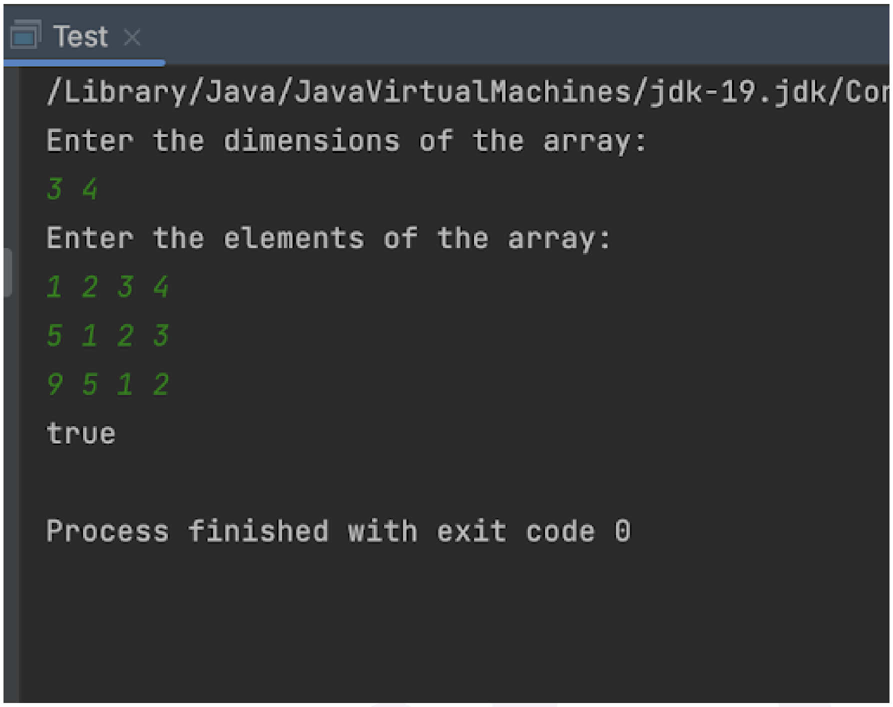
true

Explanation:

- For every cell, check if diagonally previous cell has the same element or not. If not return, false then and there, else continue.
- Start with 1st row and 1st column as we are matching with the previous row and column.

Code:

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter the dimensions of the array: ");
    int n = scn.nextInt();
    int m = scn.nextInt();
    int[][] mat = new int[n][m];
    System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        mat[i][j] = scn.nextInt();
      }
    }
    for (int i = 1; i < n; i++) {
      for (int j = 1; j < m; j++) {
        if(mat[i][j] != mat[i-1][j-1]){
          System.out.println(false);
          return;
        }
      }
    }
    System.out.println(true);
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Cor
Enter the dimensions of the array:
3 4
Enter the elements of the array:
1 2 3 4
5 1 2 3
9 5 1 2
true

Process finished with exit code 0
```

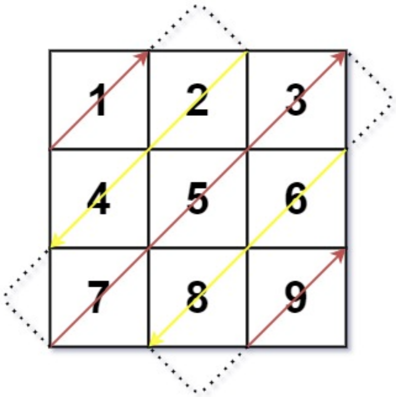
Q4 - Given a n*m matrix, return an array of elements containing diagonal traversal of the matrix. (Medium)

Input1:

n = 3

m = 3

arr[]=[[1, 2, 3],[4, 5, 6],[7, 8, 9]]



Expected Output:

[1, 4, 2, 7, 5, 3, 8, 6, 9]

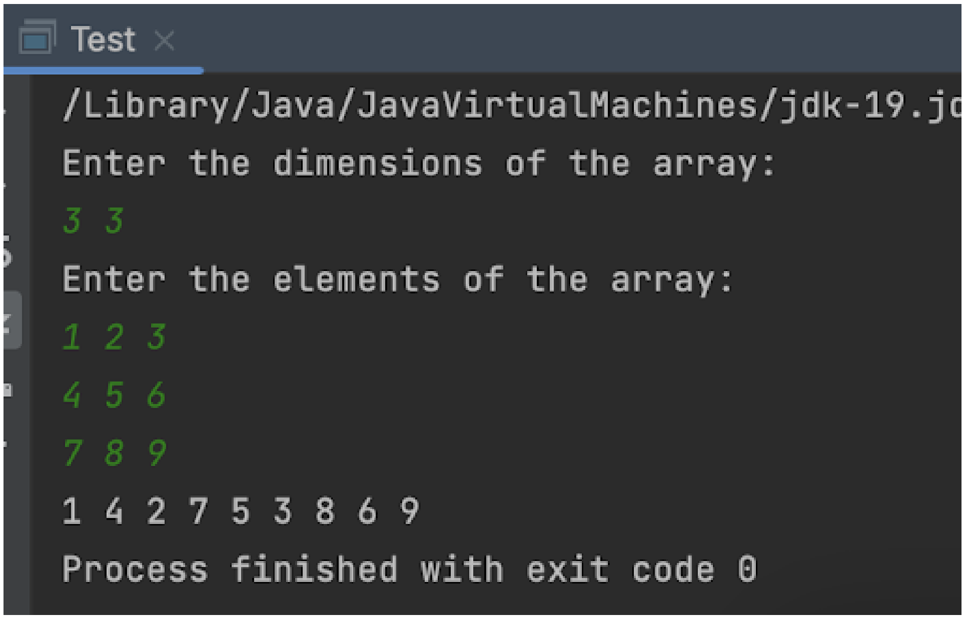
Explanation:

- Create a blank array of length n*m to store the diagonally traversed elements.
- Use 3 pointers, idx to keep track of current index of blank arr, row and col to point to current row and column of the matrix.
- In a while loop, use 2 new pointers i and j which will be initialized with the current row and col respectively.

- While, i and j are in bounds, diagonally traverse, by decrementing i and incrementing j and store at idx in arr, increment idx everytime.
- Increment current row and check if row exceeds the limit and if so, we move to next column and start from the last row.

Code:

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter the dimensions of the array: ");
    int n = scn.nextInt();
    int m = scn.nextInt();
    int[][] mat = new int[n][m];
    System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        mat[i][j] = scn.nextInt();
      }
    }
    int[] arr = new int[n*m];
    int idx = 0;
    int row = 0;
    int col = 0;
    while(col < m){
      int i = row;
      int j = col;
      while(i >= 0 && j < m){
        arr[idx] = mat[i][j];
        idx++;
        i--;
        j++;
      }
      row++;
      if(row >= n){
        row = n-1;
        col++;
      }
    }
    for(int i = 0; i < arr.length; i++){
      System.out.print(arr[i] + " ");
    }
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jd
Enter the dimensions of the array:
3 3
Enter the elements of the array:
1 2 3
4 5 6
7 8 9
1 4 2 7 5 3 8 6 9
Process finished with exit code 0
```

Q5 - Given an array of intervals where intervals[i] = [start, end], merge all overlapping intervals, and return the count of the non-overlapping intervals that cover all the intervals in the input. (Medium)

Input1:

n = 4

m = 2

arr[]=[[1,4],[2,3],[5,8],[6,9]]

Expected Output:

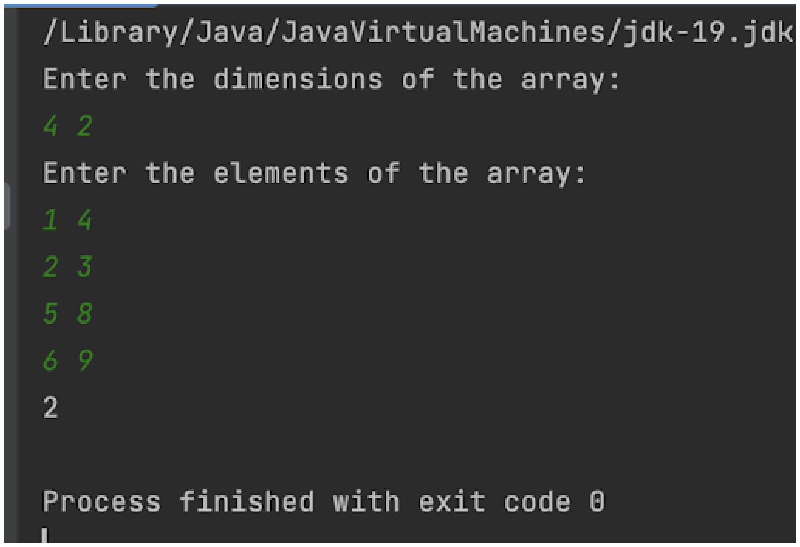
2

Explanation:

- Sort the matrix using java inbuilt method, Arrays.sort(), enter mat as one argument and method to sort as second argument.
- Since here, we want to sort the intervals by start time in ascending order, we pass our second argument as (a,b) -> a[0]-b[0], meaning when sorting two arrays, sort them in increasing order on the basis of the element at 0th index.
- Traverse the matrix, if start time of the next interval is less than or equal to finish time of current interval, then they can be merged.
- While merging, start time would be that of current interval and end time will be the max of finish time of both the intervals.
- Keep a count of merged intervals, and print it in the end.

Code:

```
import java.util.Scanner;
import java.util.Arrays;
public class Test {
  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    System.out.println("Enter the dimensions of the array: ");
    int n = scn.nextInt();
    int m = scn.nextInt();
    int[][] mat = new int[n][m];
    System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        mat[i][j] = scn.nextInt();
      }
    }
    Arrays.sort(mat, (a,b) -> a[0]-b[0]);
    int count = 0;
    int i = 0;
    while(i < n){
      while(i < n-1 && mat[i+1][0] <= mat[i][1]){
        mat[i+1][0] = mat[i][0];
        mat[i+1][1] = Math.max(mat[i][1], mat[i+1][1]);
        i++;
      }
      i++;
      count++;
    }
    System.out.println(count);
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk
Enter the dimensions of the array:
4 2
Enter the elements of the array:
1 4
2 3
5 8
6 9
2

Process finished with exit code 0
```