2D Array Challenges

Challenge 1 - Matrix Transpose

Problem

Given a square matrix A & its number of rows (or columns) N, return the transpose of A.
The transpose of a matrix is the matrix flipped over it's main diagonal, switching the row and column indices of the matrix.

Constraints

1 <= N <= 1000

Sample Input1

```
A = [
        [1,2,3],
        [4,5,6],
        [7,8,9]
    ]
N = 3
```

Sample Output1

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

Approach

Transpose of a matrix means swapping its rows with columns & columns with rows. But this swap is to be done only for the upper triangle of a matrix i.e. swap half of the elements of the diagonally upper half of the matrix with the diagonally lower half once. In this, each (row, col) & (col, row) pair will be swapped exactly once and the transpose of the square matrix could be obtained.

Code

```
#include<iostream>
using namespace std;

int main() {
   int N = 3;
   int A[N][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

   for(int i=0; i<N; i++) {
       for(int j=i; j<N; j++) {
           //swap
           int temp = A[i][j];
           A[i][j] = A[j][i];
           A[j][i] = temp;
       }
   }

   //print transpose
   for(int i=0; i<N; i++) {
       for(int j=0 ;j<N; j++) {
           cout << A[i][j] << " ";
       }
       cout << "\n";
   }
   return 0;
}
```

Time complexity : O(N^2)

Space complexity : O(1), as no extra space for a new matrix was used

Challenge 2 - Matrix Multiplication

Problem

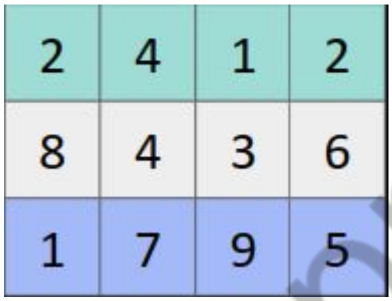
Given two 2-Dimensional arrays of sizes n1Xn2 and n2Xn3. Your task is to multiply these matrices and output the multiplied matrix.

Constraints

1 <= n1,n2,n3 <= 1,000

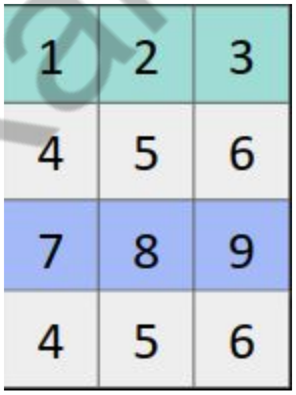
Sample test case

Input



2	4	1	2
8	4	3	6
1	7	9	5

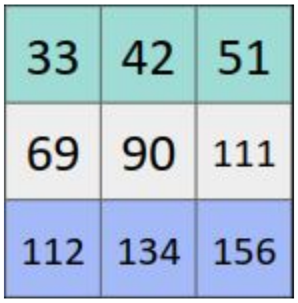
3X4



1	2	3
4	5	6
7	8	9
4	5	6

4X3

Output



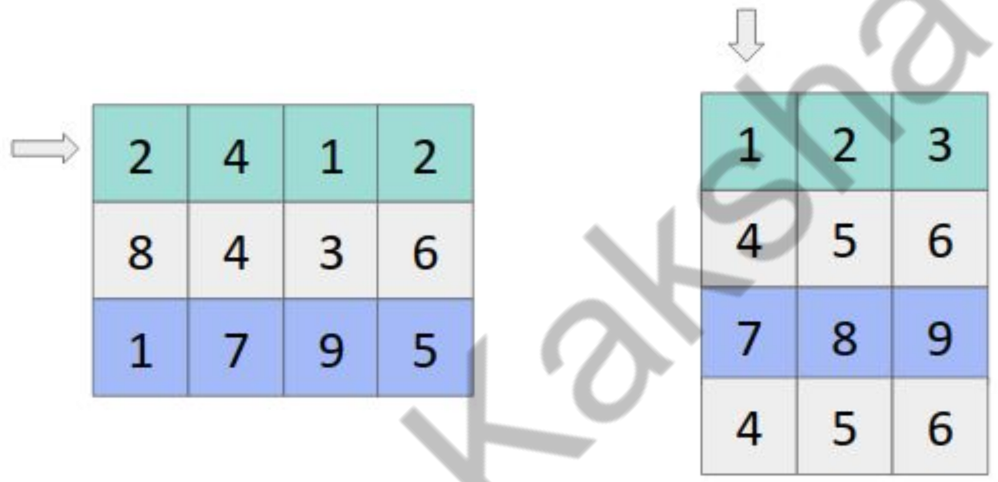
33	42	51
69	90	111
112	134	156

Approach

1. Make a nested loop of order 3. In the outer loop iterate over rows of the first matrix and in the inner loop iterate over columns of the second matrix.
2. Multiply rows of the first matrix with columns of the second matrix in the innermost loop and update in the answer matrix.

Dry Run

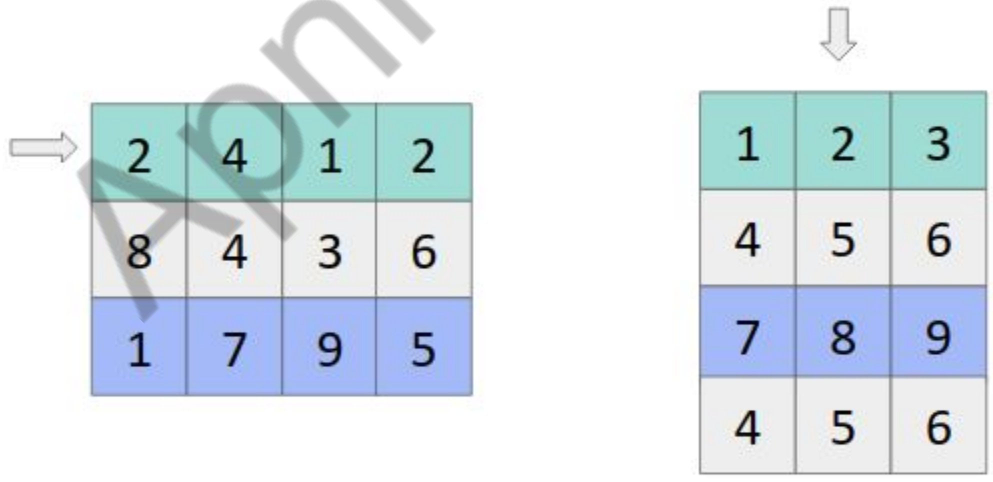
First Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

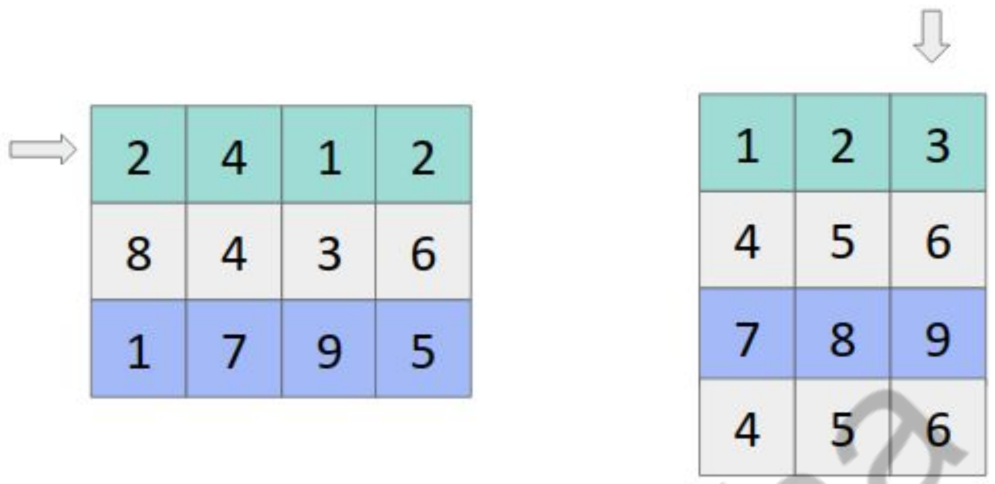
Second Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

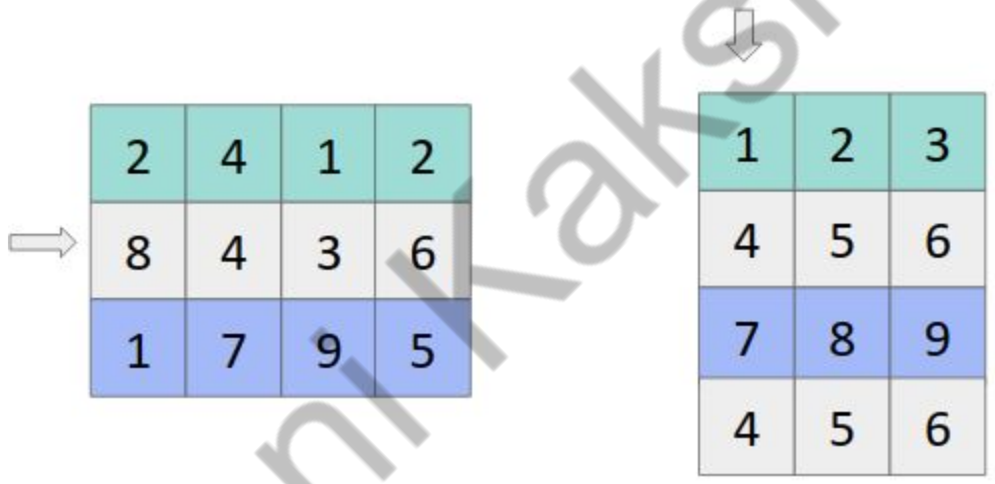
Third Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

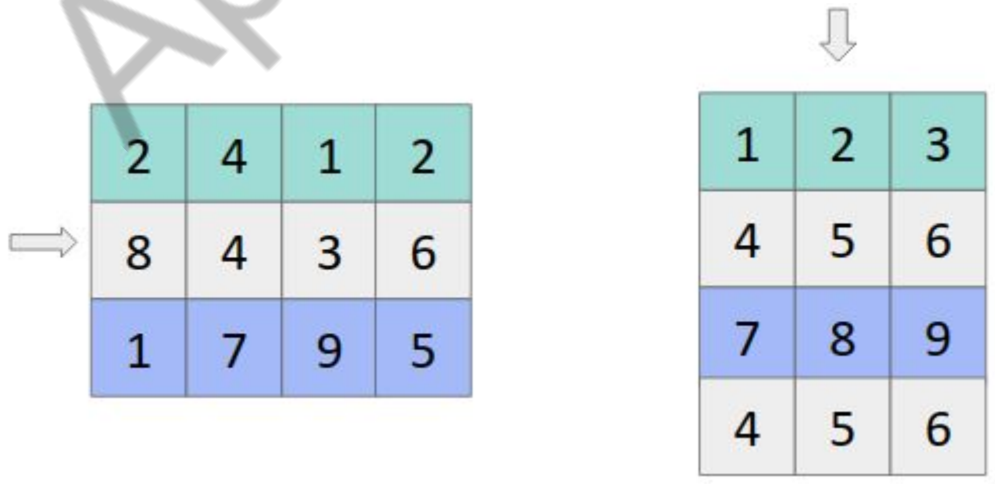
Fourth Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

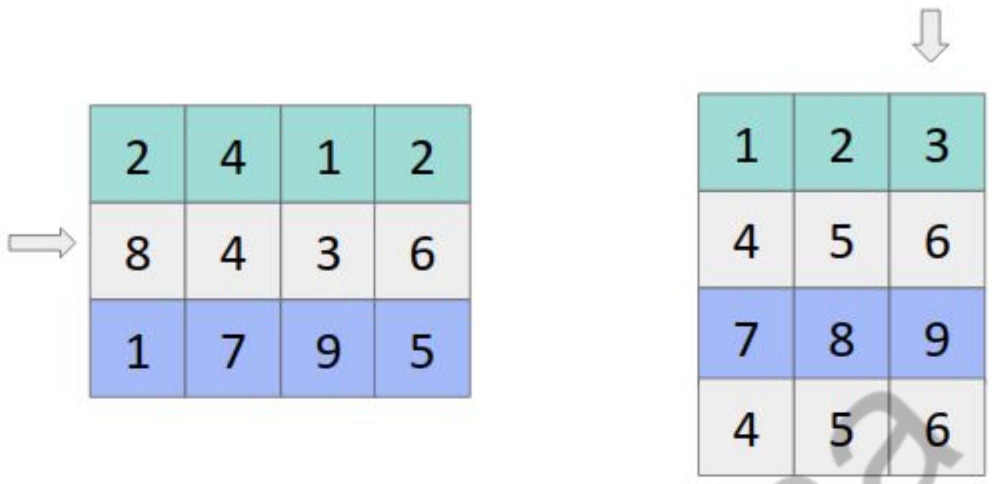
Fifth Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

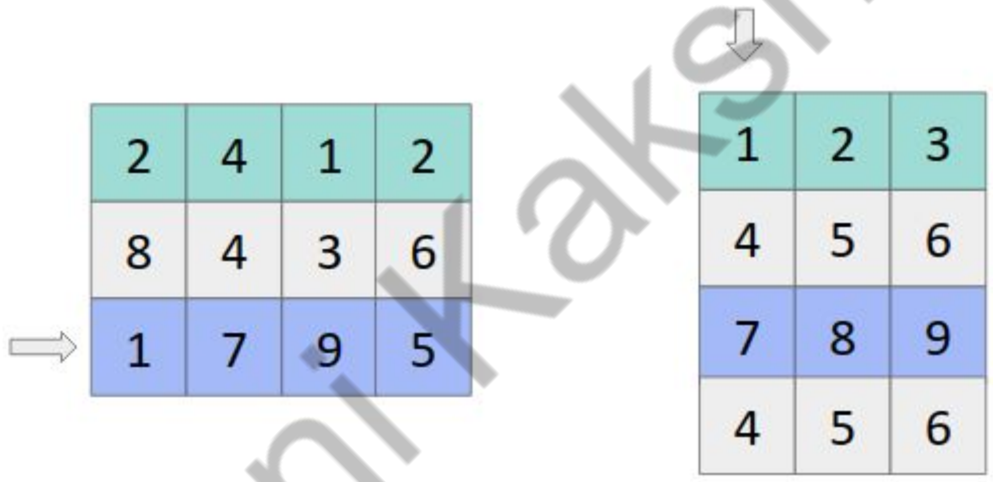
Sixth Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

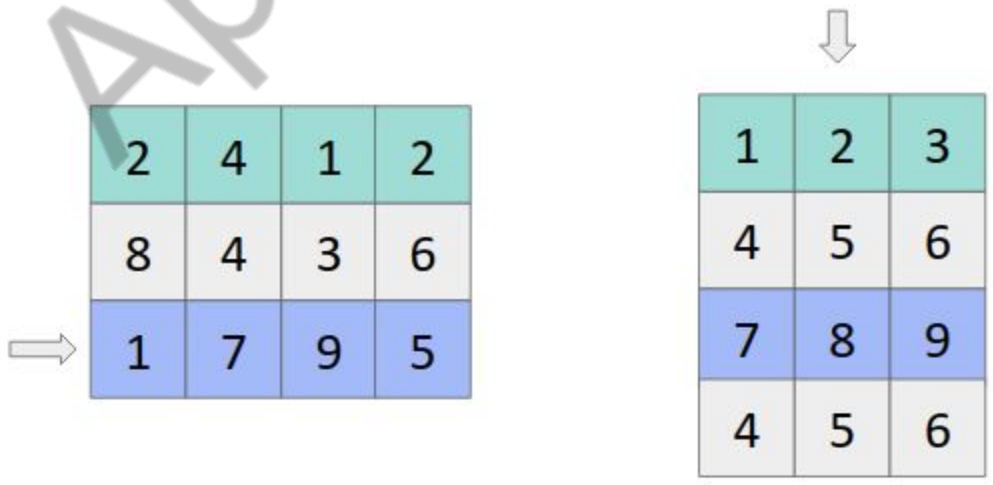
Seventh Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

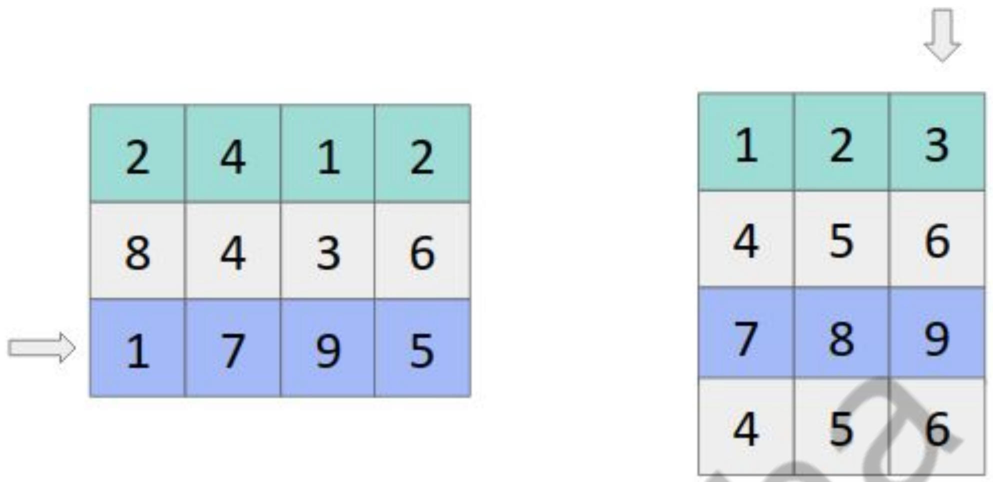
Eighth Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

Ninth Iteration



2	4	1	2
8	4	3	6
1	7	9	5

1	2	3
4	5	6
7	8	9
4	5	6

Code

```
void multiplyMatrices()
{
   int n1,n2,n3;
   cin >> n1 >> n2 >> n3;
   int m1[n1][n2]; int m2[n2][n3]; int ans[n1][n3];
   for(int i=0; i<n1; i++) {
       for(int j=0; j<n2; j++)
           cin >> m1[i][j];
   }
   for(int i=0; i<n2; i++) {
       for(int j=0; j<n3; j++)
           cin >> m2[i][j];
   }
   for(int i=0; i<n1; i++) {
       for(int j=0; j<n3; j++)
           ans[i][j] = 0;
   }

   for(int i=0; i<n1; i++) {
       for(int j=0; j<n3; j++)
       {
           for(int k=0; k<n2; k++) {
               ans[i][j] += m1[i][k]*m2[k][j];
           }
       }
   }

   for(int i=0; i<n1; i++) {
       for(int j=0; j<n3; j++)
           cout << ans[i][j] <<" ";
       cout << endl;
   }
}
```

Time Complexity: O(n1*n2*n3)

Challenge 3 - 2D matrix Search

Problem

Given a nxm matrix.

Write an algorithm to find that the given value exists in the matrix or not.

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

Constraints

1 <= N,M <= 1,000

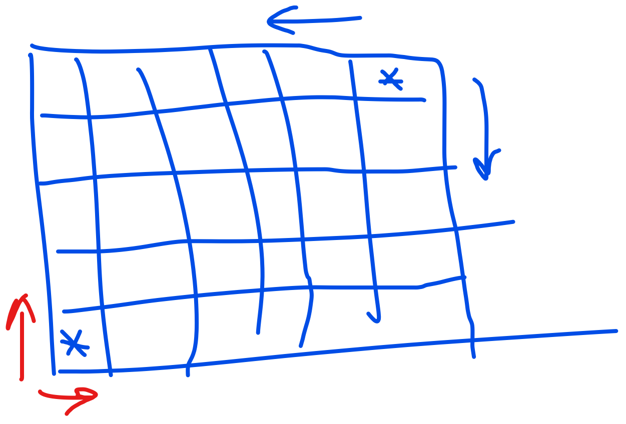
Sample Test Case:

Consider the following matrix:

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

Given target = 5, return true.

Given target = 20, return false.



Brute Force Approach

Linear search in a 2D Array.

Code:

```
#include "bits/stdc++.h"
using namespace std;

int32_t main() {
    int n, m; cin >> n >> m;
    int target; cin >> target;
    int mat[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> mat[i][j];
        }
    }
    bool found = false;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (mat[i][j] == target)
                found = true;
        }
    }
    if (found)
        cout << "Found";
    else
        cout << "Not Found";
}
```

Time complexity : O(N*M)

Optimised Approach [IMP]

```
1. Start from the top right element.
2. You are at (r,c)
   if(matrix[r][c] == target)
           return true
   If (matrix[r][c] > target)
           c--
   else
           r++;
```

At (r,c), you can go to (r-1,c) or (r,c-1), depending on the value of matrix[i][j] and target.

```cpp
#include "bits/stdc++.h"
using namespace std;
int32_t main() {
    int n, m; cin >> n >> m;
    int target; cin >> target;
    int mat[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> mat[i][j];
        }
    }
    bool found = false;
    int r = 0, c = n - 1;
    while (r < m && c >= 0) {
        if (mat[r][c] == target) {
            found = true;
        }
        mat[r][c] > target ? c-- : r++;
    }
    if (found)
        cout << "Found";
    else
        cout << "Not Found";
}
```

Time complexity : O(N + M)