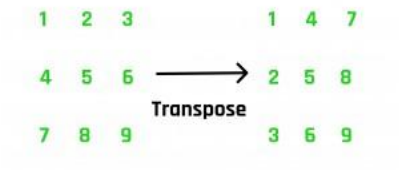
Sample problems on matrix

Problem #1 : Transpose of a Matrix

Description - Transpose of a matrix is obtained by changing rows to columns and columns to rows. In other words, transpose of A[ ][ ] is obtained by changing A[i][j] to A[j][i].We are given a matrix of size m*n, We have to print the transpose of the matrix.



Solution : We have given matrix **A[m][n]**, We will create auxiliary matrix **B[n][m]** for storing the Transpose of the Matrix A. The idea is to place A [j][i] at B [i][j].

Pseudo Code

```
void transpose(A[m][n])
{
    B[n][m] // Transpose Matrix

    for ( i=0 to n-1 )
    {
        for ( j=0 to m-1 )
            B[i][j] = A[j][i]
    }
}
```

Time Complexity : O(m*n)
Auxiliary Space : O(m*n)

Problem #2 : Search Element in Row-wise and Column-wise Sorted Matrix

Description - Given an **n x n** matrix and a number **x**, find the **position of x** in the matrix. In the given matrix, every row and column is sorted in increasing order.

Solution : Idea is to solve problem with row and column elimination reducing the search space. Before jumping at the solution, lets try to understand the concept that is actually allowing us to solve the problem in linear time.

Let's start our search from the top-right corner of the array. There are three possible cases.

1. The number we are searching for is greater than the current number. This will ensure, that all the elements in the current row is smaller than the number we are searching for as we are already at the right-most element and the row is sorted. Thus, the entire row gets eliminated and we continue our search on the next row. Here elimination means we won't search on that row again.
2. The number we are searching for is smaller than the current number. This will ensure, that all the elements in the current column is greater than the number we are searching for. Thus, the entire column gets eliminated and we continue our search on the previous column i.e. the column at the immediate left.
3. The number we are searching for is equal to the current number. This will end our search.

Pseudo Code

```
// matrix size : n*n
void search(mat[][] ,int x)
{
    i = 0, j = n-1
    while(i > n && j >= 0 )
    {
        if (mat[i][j] == x )
        {
            print(i,j)
            break
        }
        else if (mat[i][j] > x )
        {
            j--
        }
        else
        {
            i++
        }
    }

}
```

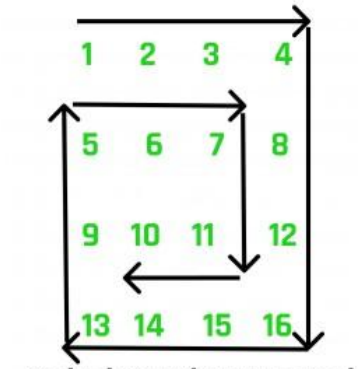
Since, at each step, we are eliminating an entire row or column.

Time Complexity : O(n)

Auxiliary Space : O(1)

Problem #3 : Spiral Traversal of Matrix

Description - We are given a 2D Matrix of size **m*n**. We have to print the Matrix in Spiral form shown in the Example.



Spiral Matrix Traversal

```
Output : 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Solution : We will be traversing the Matrix in Spiral form with the help of 5 variables which includes iterator, starting and ending index of row and columns.

- k - starting row index
- m - ending row index
- l - starting column index
- n - ending column index
- i - iterator

Pseudo Code

```
void print_spiral(A[][], m, n)
{
    k = 0, l = 0
    while (k < m && l < n)
    {
        /* Print the first row from the remaining rows */
        for (i=l to n-1)
            print(A[k][i])
        k++

        /* Print the last column from the remaining columns */
        for (i = k to i = m-1 )
            print(A[i][n-1])
        n--

        /* Print the last row from the remaining rows */
        if ( k < m)
        {
            for (i = n-1; i >= l; --i)
                print(A[m-1][i])
            m--
        }

        /* Print the first column from the remaining columns */
        if (l < n) :
        {
            for (i = m-1; i >= k; --i)
                print(A[i][l])
            l++
        }
    }
}
```