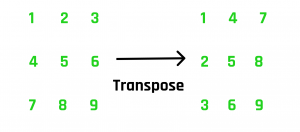
***Sample Problems on Matrix***

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

The steps of the **algorithm** are as follows:

1. Start the search at the topmost and rightmost corner in the matrix, i.e., i=0, j=n-1.
2. If the target value is equal to the value at the position we are at, then return the position of the current row and column.
3. If the target value is less than the value at the position we are at, then we can ignore all the values in the current column and continue the search to the next column. As the matrix is sorted row-wise and column-wise in ascending order, all the other values in the current column are greater than the target value.
4. If the target value is greater than the value at the position we are at, then we can ignore all the values in the current row and continue the search to the next row. As the matrix is sorted row-wise and column-wise in ascending order, all the other values in the row are lesser than the target value.