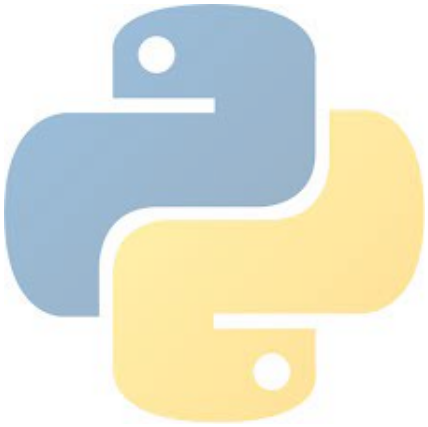
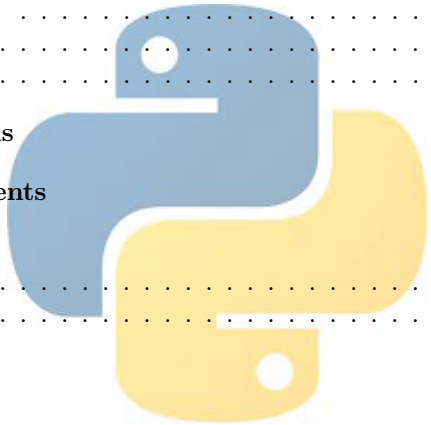
Computer Science (083)
Project Work



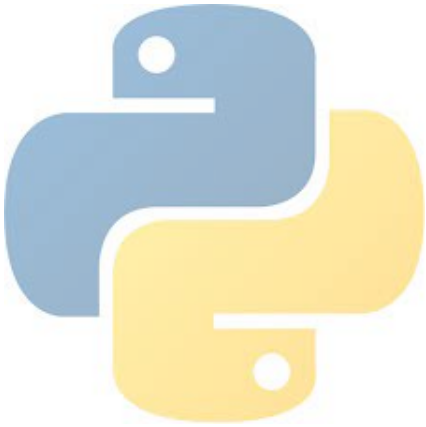
Matrices and their Operations in Python

Contents



Certificate of Completion

This is to certify that **Srijan Mahajan** of Class 12-A has prepared the project on "**Matrices**". The project is found worthy of acceptance as final project report for the subject Computer Science. He has prepared the report under my guidance.

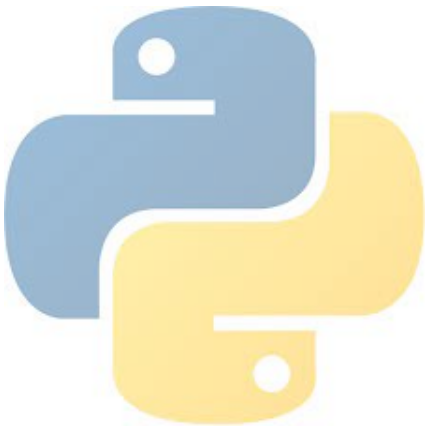


(Ms. Namita Chuttani)
PGT (Computer Science)
The Indian Heights Schoola
Sector-22, Dwarka, Delhi-77

Acknowledgement

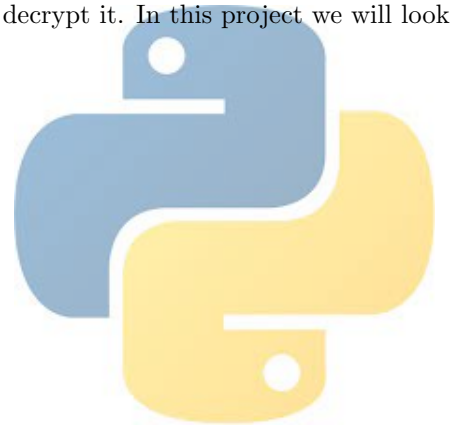
We would like to express our sincerest gratitude to our beloved teacher, Ms. Namita Chuttani (PGT Computer Science) for her guidance and motivation throughout. We would also thank school coordinators Mr. Rajesh Kumar Jha and Ms. Hema Raghav and the school principal, Ms. Archana Narayan for always being a beacon of support and giving us the opportunity to complete the project.

The project would not have been possible without the inputs of our friends, parents, and the participants of the survey without which the project could not have been completed on time.



Abstract

Matrices are a part of Linear Algebra which is used everywhere in Computer Science. It is used in computer graphics to create 2D/3D models, animations, etc. It is used in cryptography (making data secure) by using matrices to store data and a key matrix to encrypt it and its inverse to decrypt it. In this project we will look at operations on matrices in Python.



1 What is a Matrix?

A matrix is a rectangular array of data[1] arranged in rows and columns.
A matrix looks like the following:

$$A = \begin{bmatrix} 1 & 10 & 12 \\ 2 & 20 & 22 \end{bmatrix}_{2\times 3}$$

A matrix is represented by capital letters and the subscript represents Number of rows $\times$ Number of columns and the matrix A is called *a 2 by 3 matrix*.
In Python, to enter a matrix, we use nested lists like:

```
Matrix=[
  [1,10,12],
  [2,10,22]
]
```

1.1 Accessing a Matrix

A matrix a is generally written as $A = [a_{ij}]_{m\times n}$ where $1 \leq i \leq m \wedge 1 \leq j \leq n$. Thus, if we know the location of an element say, *element in row 2 and column 1*, we can write it as a_{21}. In Python as well, if we need to find the *element in row i and column j*, we can return it as:

```
def find_element(matrix: list[list[int|float]], row: int, column: int):
  row_index=row-1 #we need to use row-1 as indexes begin from 0
  column_index=column-1
  return matrix[row_index][column_index]
```

Problem. We can also access the elements of a matrix, either row or column wise.

Code. To print the matrix row-wise:

```
def row_wise(matrix: list[list[int|float]]):
  for i in range(len(matrix)):
    for j in range(len(matrix[0])):
      print(matrix[i][j],end="\t")
  print("\n")
```

If matrix is $A = \begin{bmatrix} 1 & 10 & 12 \\ 2 & 20 & 22 \end{bmatrix}$, then, the output is,

```
1       10      12

2       20      22
```

To print the matrix column-wise:

```
def column_wise(matrix: list[list[int|float]]):
  for i in range(len(matrix[0])):
    for j in range(len(matrix)):
      print(matrix[j][i],end="\t")
  print("\n")
```

The output in this case is:

```
1       2

10      20

12      22
```

1.2 Null Matrix

A null matrix is one such that,

$$a_{ij} = 0 \quad \forall\, i, j$$

Such a matrix is represented by O.

[1] Data may be of any form, like numbers, expressions or alphabets.

Problem. Creating a null matrix of a given order.

Code.

```
def null(rows: int,columns: int) -> list[int|float]:
  null_matrix=[[0.0 for i in range(columns)] for i in range(rows)] #loop over columns then over rows
  return null_matrix
```

1.3 Upper and Lower Triangular Matrices

The upper triangular matrix is a matrix in which all the entries below the diagonal are zero, i.e.,

$$a_{ij} = 0 \quad \forall\, i \geq j$$

Or,

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

The lower triangular matrix is a matrix in which all the entries above the diagonal are zero[2], i.e.,

$$a_{ij} = 0 \quad \forall\, i \leq j$$

Or,

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Problem. Creating an upper and lower triangular matrix.

Code. First for an upper triangular matrix,

```
def upper_triangular(matrix: list[list[int|float]]) -> list[list[int|float]]:
    rows=len(matrix)
    columns=len(matrix[0])
    upper_matrix=null(rows,columns) #null matrix
    for i in range(rows):
        for j in range(columns):
            if i>=j: #Condition for a null matrix
                upper_matrix[i][j]+=matrix[i][j]
            else:
                continue
    return upper_matrix
```

If the matrix is $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, then, the output is,

```
[[1, 0, 0], [4, 5, 0]]
```

Now, for a lower triangular matrix,

```
def lower_triangular(matrix: list[list[int|float]]) -> list[list[int|float]]:
  rows=len(matrix)
  columns=len(matrix[0])
  lower_matrix=null(rows,columns) #null matrix
  for i in range(rows):
    for j in range(columns):
      if j>=i: #Condition for a null
        lower_matrix[i][j]+=matrix[i][j]
      else:
        continue
  return lower_matrix
```

[2]These definitions are open for discussion. Some authors claim that the matrix must be square, while some do not restrict the matrix. Even though a 'triangle' would not be formed in the case of rectangular matrices, it is acceptable. We have chosen not to restrict the matrices to only square matrices.

The output in this case is,

```
[[1, 2, 3], [0, 5, 6]]
```

1.4 Transpose of a Matrix

The transpose of a matrix $A = [a_{ij}]_{m\times n}$ is given by,

$$A^T = [a_{ji}]_{n\times m}$$

Problem. Create another matrix which is the transpose of a given matrix.

Code.

```
def transpose(matrix: list[list[int|float]]) -> list[list[int|float]]:
  rows=len(matrix)
  columns=len(matrix[0])
  transpose_matrix=null(columns, rows) #null matrix
  for i in range(rows):
    for j in range(columns):
      transpose_matrix[j][i]+=matrix[i][j] #definition of transpose

  return transpose_matrix
```

If the matrix is $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$

, the output is,

```
[[1, 4, 1], [2, 5, 1], [3, 6, 1]]
```

1.5 Addition of Two Matrices

Given two matrices $A = [a_{ij}]_{n_1\times m_1}$ and $B = [b_{ij}]_{n_2\times m_2}$, the sum $A + B$ is defined only if $(n_1, m_1) = (n_2, m_2) = (n, m)$ (Say), i.e. the matrices have the same order.

$$C = A + B = [a_{ij} + b_{ij}]_{n\times m}$$

Problem. Find the sum of two matrices.

Code. It returns the sum of the matrices if it exists, else, returns -1.

```
def Matrix_Sum(matrix1: list[list[int|float]],matrix2: list[list[int|float]]) -> list[list[int|float]]
     | int:
    if len(matrix1)==len(matrix2) and len(matrix1[0])==len(matrix2[0]): #Checking for compatibility
    for addition
        rows, columns=len(matrix1), len(matrix1[0])
        sum_matrix=null(rows,columns) #null matrix
        for i in range(rows):
            for j in range(columns):
                sum_matrix[i][j]+=matrix1[i][j]+matrix2[i][j] #definition of sum of matrices
        return sum_matrix
    else:
        return -1
```

If the input matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 2 & 2 & 2 \end{bmatrix}$, the output is,

```
[[2, 4, 6], [8, 10, 12], [3, 3, 3]]
```

If the matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, the output is,

-1

1.6 Subtraction of Two Matrices

Matrices also have am additive inverse, i.e., there exists a matrix B such that,

$$A + B = 0 \implies B = -A$$

The matrix B is defined as $B = [-a_{ij}]_{n\times m}$ if the matrix A js defined as $A = [a_{ij}]_{n\times m}$.
Thus, we can define the operation $A - B$ as $A + (-B)$ which is simple matrix addition.

Problem. Find the difference of two matrices.

Code. It is similar to the sum of two matrices program.

```
def Matrix_Difference(matrix1: list[list[int|float]],matrix2: list[list[int|float]]) -> list[list[int|
    float]]:
#The program is matrix1-matrix2, since subtraction is not commutative
matrix2_new=null(len(matrix2),len(matrix2[0]))
for i in range(len(matrix2)):
  for j in range(len(matrix2[0])):
    matrix2_new[i][j]=-matrix2[i][j] #From the definition of the negative of a matrix
return(Matrix_Sum(matrix1,matrix2_new))#From the definition of difference of two matrices
```

If the input matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 2 & 2 & 2 \end{bmatrix}$, the output is,

```
[[0, 0, 0], [0, 0, 0], [-1, -1, -1]]
```

2 Division of a Matrix by a Scalar

When a matrix is divided by a scalar, each element of the matrix is reduced by a factor of the scalar, i.e.,

$$\frac{A}{k} = \left[\frac{a_{ij}}{k}\right]$$

Problem. Find the result of a matrix divided by a scalar.

Code. Using the above definition,

```
def Division(mx: list[list[int]], scalar: int) -> list[list[int]]:
    new_matrix = null(len(mx), len(mx[0]))
    for i in range(len(mx)):
        for j in range(len(mx[0])):
            new_matrix[i][j] += mx[i][j] / scalar
    return new_matrix
```

2.1 Minor Matrix of an Element

The minor matrix of an element is the matrix obtained from removing the i^{th} row and j^{th} column of a matrix.

Problem. Find the minor matrix of a matrix given the location of the matrix,

Code. For this, first we exclude the i^{th} row. Then, we exclude the j^{th} column .

```
def Minor_Matrix(mx: list[list[int|float]],r; int,c: int) -> list[list[int|float]]:
    return [row[:c]+row[c+1:] for row in (mx[:r]+mx[r+1:])]
```

The following is an incredible one-liner.

2.2 Determinant of a Matrix

The determinant is a scalar value that is a function of the entries of a square matrix. It is denoted by $det(A)$, $|A|$ or Δ.

If $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ then, $\Delta = a_{11}a_{22} - a_{21}a_{12}$.

If $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ then, $\Delta = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + a_{22}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{33}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$

Before finding the determinant of a matrix, let us look at Cofactors.

2.2.1 Cofactors

The cofactor of an element a_{ij} is denoted by A_{ij} and is given by,

$$A_{ij} = (-1)^{i+j} M_{ij}$$

Where, M_{ij} is the determinant of the minor matrix of the element a_{ij}. The determinant of a matrix is thus given by,

$$\Delta = \sum_{j=0}^{n} a_{1j} A_{1j}$$

Thus, the determinant of a matrix is the sum of the elements multiplied with their cofactors.

Problem. Find the determinant of a matrix.

Code. Since we have a closed form for the determinant of a 2×2 matrix, we can reduce all determinants[a] to finding the determinant of a smaller matrix by using cofactors.

```
def Determinant(mx: list[list[int|float]]) -> int|float:
  # for the base case when the matrix is 2*2
  if len(mx) == 2:
    return mx[0][0] * mx[1][1] - mx[0][1] * mx[1][0]
  determinant = 0
  # We will only expand along the first row
  for i in range(len(mx)):
    determinant += (
      ((-1) ** i) * mx[0][i] * Determinant(Minor_Matrix(mx, 0, i))
    )  # By definition
  return determinant
```

[a] So, we need to use recursion.

3 Inverse of a Matrix

If A is a square matrix of order m, and there exists another square matrix B of the same order m, such that $AB = BA = I$, then B is called the inverse matrix of A and it is denoted by A^{-1}. In that case A is said to be invertible. The inverse of a matrix if it exists is unique.

3.1 Adjoint of a Matrix

The adjoint of a square matrix $A = [a_{ij}]_{n\times n}$ is defined as the transpose of the matrix $[A_{ij}]_{n\times n}$, i.e.,

$$\text{adj } A = [A_{ij}]_{n\times n}$$

Problem. Find the adjoint of a matrix.

Code.

```
def Adjoint(mx: list[list[int|float]]) -> list[list[int|float]]:
    temp_matrix = []  # Empty matrix before the transpose
    for i in range(len(mx)):
        row = []  # Initialize each row
        for j in range(len(mx)):
            row.append(
                ((-1) ** (i + j)) * Determinant(Minor_Matrix(mx, i, j))
            )  # from the definition of adjoint
        temp_matrix.append(row)
    return transpose(temp_matrix)
```

The inverse of a matrix can be calculated by using:

$$A^{-1} = \frac{1}{|A|}\text{adj } A$$

Problem. Find the inverse of a matrix.

Code. Using the above definition,

```
def Inverse(mx):
  if Determinant(mx) == 0:
    return None  # The inverse does not exist in this case
  else:
    return Division(Adjoint(mx), Determinant(mx))  # By definition
```

4 Multiplication of Two Matrices

The product of two matrices is defined if the number of columns of A is equal to the number of rows of B. If $A = [a_{ij}]_{m\times n}$ and $B = [b_{jk}]_{n\times p}$, then the i^{th} row of A is $\begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix}$ and the k^{th} column of B is $\begin{bmatrix} b_{1k} \\ b_{2k} \\ \vdots \\ b_{nk} \end{bmatrix}$, then $c_{ik} = \sum_{j=1}^{n} a_{ij}b_{jk}$.

The matrix $C = [c_{ik}]_{m\times p}$ is the product AB. Matrix Multiplication is not commutative but is associative and distributive. Moreover, $AB = 0 \nRightarrow A = 0 \vee B = 0$.
Here, we will take two approaches to find the product of two matrices.

Problem. Find the product of two matrices.

Code. Here, we will use the exact definition used above. We will loop 3 times.

```
def Multiplication(mx1: list[list[int|float]], mx2: list[list[int|float]]) -> list[list[int|float]]:
    if len(mx1[0]) != len(mx2):
        return "The matrices are incompatible for multiplication."
    else:
        result = null(len(mx1), len(mx2[0]))
        for i in range(len(mx1)):  # Loop through rows of first matrix
            for j in range(len(mx2[0])):  # Loop through each column of second matrix
                for k in range(len(mx2)):  # Loop through each row of second matrix
                    result[i][j] += mx1[i][k] * mx2[k][j]  # By definition
        return result
```

The following can also be done one line as follows:

```
def Matrix_Multiplication(mx1,mx2):
    return [[sum([mx1[i][k]*mx2[k][j] for k in range(len(mx1[0]))]) for j in range(len(mx2[0]))] for i
     in range(len(mx1))]
```

This method is however very slow, as it loops three times. If the matrices are square matrices of the same order (n), then the program takes $O(n * n * n) = O(n^3)$ time.

4.1 Strassen Algorithm

Strassen Algorithm[3] is an algorithm for multiplication of two matrices. It is much faster than the traditional multiplication algorithms (like the one mentioned above). However, it only works for square matrices of the order 2^n, where $n \in \mathbb{N}$.

4.1.1 Joining Matrices

Problem. Join any two matrices horizontally.

Code.

```
def joining_horizontally(a: list[list[int|float]], b: list[list[int|float]]) -> list[list[int|float]]:
  n = len(a)
  new_matrix = null(
    len(a), len(a[0]) + len(b[0])
  )  # Null matrix of the same order as the expected output
  for i in range(n):
    for j in range(n):
      new_matrix[i][j] = a[i][j]  # First matrix
      new_matrix[i][j + n] = b[i][j]  # Second matrix
  return new_matrix
```

If the input matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ and $B = \begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$, then the output matrix is, $\begin{bmatrix} 1 & 2 & 3 & 10 & 11 & 12 \\ 4 & 5 & 6 & 13 & 14 & 15 \\ 7 & 8 & 9 & 16 & 17 & 18 \end{bmatrix}$.

Problem. Join any two matrices vertically.

Code.

```
def joining_vertically(a: list[list[int|float]], b: list[list[int|float]]) -> list[list[int|float]]:
    n = len(mx1)
    new_matrix = []  # Empty list
    for i in mx1:
        new_matrix.append(i)  # First the first matrix
    for j in mx2:
        new_matrix.append(j)  # Now the second matrix
    return new_matrix
```

If the input matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ and $B = \begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$, then the output matrix is, $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$.

4.2 Splitting a Matrix into 4 Smaller Matrices

Problem. Split a given matrix into 4 smaller matrices, given the order of the matrix is of the form $2^n \times 2^n$.

Code. We use simple list slicing to achieve the desired output.

```
def split(mx: list[list[int|float]]) -> list[list[int|float]] -> Any:
  if len(mx) == len(mx[0]) and pow_2(len(mx)):
    n = len(mx) // 2
    a = mx[:n]
    b = mx[n:]
    a_11 = [a[i][:n] for i in range(n)]
    a_12 = [a[i][n:] for i in range(n)]
    a_13 = [b[i][:n] for i in range(n)]
    a_14 = [b[i][n:] for i in range(n)]
  return a_11, a_12, a_13, a_14
```

. Product of two 2×2 matrices.

[3] named after the German mathematician Volker Strassen.

. Let the matrices be:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \wedge \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Then the product matrix is given, by,

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Where, the elements are,

$$\begin{aligned} C_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ C_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ C_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ C_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

However, this uses 8 multiplications. Strassen found a way to use only 7 multiplications. This is a bit faster as additions are faster than multiplications. For this, we calculate the following 7 products:

$$\begin{aligned} P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ P_2 &= a_{22}(b_{21} - b_{11}) \\ P_3 &= (a_{11} + a_{12})b_{22} \\ P_4 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\ P_5 &= a_{11}(b_{12} - b_{22}) \\ P_6 &= (a_{21} + a_{22})b_{11} \\ P_7 &= (a_{11} - a_{21})(b_{11} + b_{12}) \end{aligned}$$

Now, calculating the elements,

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 - P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_6 \\ C_{22} &= P_1 + P_5 - P_6 - P_7 \end{aligned}$$

Now, for a bigger matrix, i.e. of the order n, the elements a_{ij} will be matrices and the product will be calculated recursively.
For the time complexity of the previous method,

We reduce the order of the matrix to $\frac{n}{2}$ in each step and calculate 8 products and we add n^2 times, thus,

$$T(n) = 8T(\frac{n}{2}) + n^2$$

On solving the recurrence relation, we get,

$$T(n) = O(n^3)$$

Which is the same as using 3 loops.
Now, for the time complexity of Strassen Algorithm. Similarly,

$$T(n) = 7T(\frac{n}{2}) + n^2$$

Here, we get,

$$T(n) = O(n^{\log_2 7}) = O(2^{2.81})$$

Which is clearly faster than the above mentioned methods! □

Problem. Using the Strassen Algorithm.

Code. It works on the principle of divide and conquer.

```
def strassen(mx1: list[list[int|float]], mx2: list[list[int|float]]) -> list[list[int|float]]:
```

```
  if len(mx1) == 1:
    return [[mx1[0][0] * mx2[0][0]]]
  a_11, a_12, a_21, a_22 = split(mx1)
  b_11, b_12, b_21, b_22 = split(mx2)
  p_1, p_2, p_3, p_4, p_5, p_6, p_7 = (
    strassen(Matrix_Sum(a_11, a_22), Matrix_Sum(b_11, b_22)),
    strassen(a_22, Matrix_Difference(b_21, b_11)),
    strassen(Matrix_Sum(a_11, a_12), b_22),
    strassen(Matrix_Difference(a_12, a_22), Matrix_Sum(b_21, b_22)),
    strassen(a_11, Matrix_Difference(b_12, b_22)),
    strassen(Matrix_Sum(a_21, a_22), b_11),
    strassen(Matrix_Difference(a_11, a_21), Matrix_Sum(b_11, b_12)),
  )  # Exactly from the theory
  c_11, c_12, c_21, c_22 = (
    Matrix_Difference(Matrix_Sum(Matrix_Sum(p_1, p_2), p_4), p_3),
    Matrix_Sum(p_3, p_5),
    Matrix_Sum(p_2, p_6),
    Matrix_Difference(Matrix_Sum(p_1, p_5), Matrix_Sum(p_6, p_7)),
  )  # Exactly from the theory
  return joining_vertically(
    joining_horizontally(c_11, c_12), joining_horizontally(c_21, c_22)
  )
```

5 Binary Files

A binary file is one which stores the data in the form of stream of bytes. It contains information in the same format in which the information is held in memory.

5.1 Opening and Closing Binary Files

Problem. Open a binary file in read and write mode and read and append mode.

Code.

```
<filehandle>=open("<filename.dat", "wb+") # Opens in write and read mode
<filehandle>=open("<filename.dat", "ab+") # Opens in append and read mode
```

Writing on a file opened with "wb+" will remove the contents previously stored and overwrite the data. But. files opened with "ab+" will append data without overwriting the contents of the file.
To close a file,

```
<filehandle>.close()
```

To work with binary files we will import a module "pickle". This module is used for serialization and and de-serialization of byte stream.

5.2 Writing in Binary files

Problem. Store a matrix in a binary file.

Code. Consider a matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ to be stored in a binary file named, "matrix.dat",

```
import pickle
f=open("matrix.dat","wb+")
matrix=matrix=[[1,2,3],[3,4,5],[6,7,8]]
for i in matrix:
    pickle.dump(i,f)
f.close()
```

We can do the same without needing to close the file manually by,

```
import pickle
f=open("new.dat", "wb")
matrix=[[1,2,3],[3,10,5],[6,7,8]]
with open("new.dat", "wb+"):
    for i in matrix:
```

```
        pickle.dump(i,f)
```

To add the data (append) to the binary file, we can use "ab+" in place of "wb+".

5.3 Reading from a Binary File

Problem. Read data from a pre-existing binary file.

Code. Consider the binary file created above,

```
import pickle
from basicmatrices import *
f = open("new.dat", "rb")
while True:
    try:
        row = pickle.load(f)
        print(row)
    except EOFError:  # As Python raises an exception if we try to read after the file is over
        break
```

6 Solving System of Linear Equations

Consider the following system of equations (non-homogenous[4]),

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

This can be represented as,

$$Ax = B$$

Where,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}_{m\times n} \quad \wedge \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n\times 1} \quad \wedge \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}_{m\times 1}$$

If $m = n$, i.e. A is a square matrix, then we can find teh solution of the system of linear equations by,

$$\begin{aligned} Ax &= B \\ \therefore x &= A^{-1}B \end{aligned}$$

We can also check the consistency of the system of linear equation by checking the determinant of the matrix A.

- If the determinant of the matrix is 0, then the system is inconsistent.
- Else the system is consistent and has a unique solution.

Problem. Find the solution of the system of equations given the coefficients of the variables and the constant terms separately.

Code. Using the above theory,

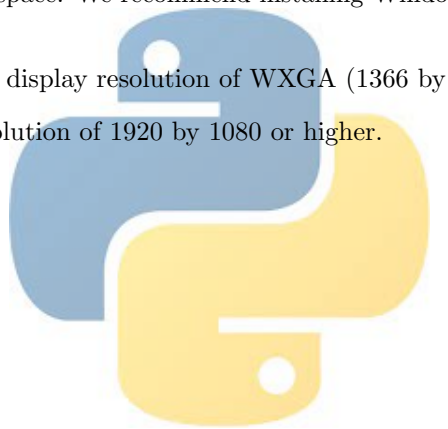
```
def solution(A:list[list[int|float]], B:list[list[int|float]])-> list[list[int|float]]|int:
    if Determinant(A)==0:
        return 0
    else:
        return Matrix_Multiplication(Inverse(A),B)
```

[4] with a unique solution

7 Hardware and Software Requirements

The platform used is Python. Hence we decided to use Microsoft Visual Studio Code 2022. Visual Studio 2022 System Requirements:- Supported Operating System:

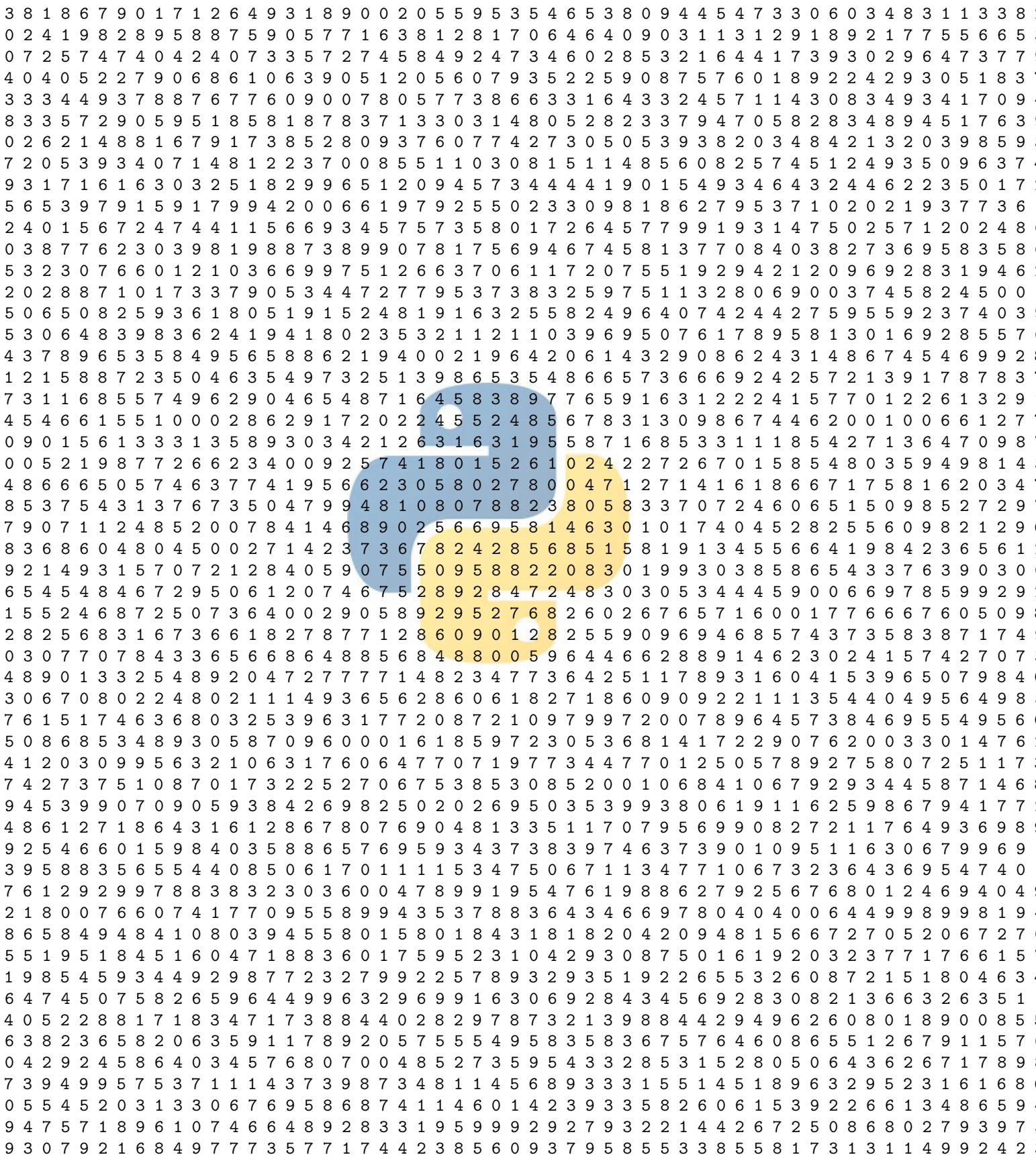
- Windows 11 version 21H2 or higher: Home, Pro, Pro Education, Pro for Workstations, Enterprise, and Education
- Windows 10 version 1909 or higher: Home, Professional, Education, and Enterprise.
- Windows Server 2022: Standard and Datacenter.
- Windows Server 2019: Standard and Datacenter.
- Windows Server 2016: Standard and Datacenter.
- Windows 11 version 21H2 or higher Hardware:
- ARM64 processor or 1.8 GHz or faster x64 processor (quad-core or better recommended). ARM32 processors are not supported.
- Minimum of 4 GB of RAM. Many factors impact resources used; we recommend 16 GB RAM for typical professional solutions.
- Windows 365 %: Minimum 2 vCPU and 8 GB RAM. 4 vCPU and 16 GB of RAM recommended.
- Hard disk space: Minimum of 850 MB up to 210 GB of available space, depending on features installed; typical installations require 20-50 GB of free space. We recommend installing Windows and Visual Studio on a solid-state drive (SSD) to increase performance.
- Video card that supports a minimum display resolution of WXGA (1366 by 768);
- Visual Studio will work best at a resolution of 1920 by 1080 or higher.



8 Input Files

The input files are: (In text form):

8.1 Input.dat



9 5 6 2 4 5 3 2 7 5 8 2 1 2 4 2 3 9 3 9 0 0 6 6 4 4 1 3 1 2 7 5 8 2 7 0 2 1 4 7 2 9 4 4 9 2 5 0 3 0 5 6 0 4 0
7 6 6 4 9 1 6 2 8 8 3 8 7 4 0 5 4 4 1 4 8 5 4 6 7 5 3 4 5 5 8 8 7 8 3 8 0 0 0 8 8 6 0 8 5 7 3 1 5 8 9 1 6 3 5
1 4 0 0 4 5 3 1 1 4 9 5 4 8 5 2 5 5 5 9 3 6 3 2 1 0 1 6 5 9 6 8 7 7 1 1 9 0 4 8 4 0 7 7 5 0 3 5 2 6 7 6 0 6 6
4 7 4 1 4 1 7 9 7 8 6 3 5 5 3 5 3 9 2 9 7 2 0 6 2 3 8 1 7 0 6 4 2 9 2 3 9 6 5 6 4 7 0 4 0 7 5 9 6 4 4 2 6 2 4
4 0 4 6 3 9 6 4 1 4 1 3 3 6 8 6 8 5 4 1 3 0 6 2 1 5 5 2 8 8 5 0 5 2 2 4 8 9 3 0 1 6 5 9 5 1 0 9 6 5 8 7 6 0 9
7 8 5 1 6 9 7 7 2 2 2 8 6 5 4 4 5 6 0 4 3 7 2 7 5 3 3 1 4 7 3 4 8 3 1 3 1 6 9 7 3 7 2 2 8 6 6 5 6 5 2 1 1 1 9
7 3 9 1 0 0 5 2 4 2 7 8 7 5 2 5 8 0 5 8 7 1 3 5 9 0 3 3 7 8 1 1 0 0 0 2 9 7 0 0 5 1 3 5 7 3 1 1 2 4 2 0 7 6 6
6 7 0 6 0 2 9 9 5 8 6 8 2 3 4 1 1 2 8 6 1 6 3 3 1 7 1 0 1 9 6 8 3 3 5 9 2 3 8 1 4 2 1 2 7 5 0 1 8 3 3 1 1 2 6
5 0 9 5 3 5 4 4 2 9 2 6 3 8 4 4 7 6 3 6 7 5 9 6 2 6 0 5 2 8 8 6 1 5 9 2 3 8 1 6 0 0 1 2 2 1 7 0 3 5 0 2 2 8 1
3 5 3 1 4 5 5 4 6 2 1 8 6 6 3 6 2 1 1 8 6 1 6 0 2 6 5 6 0 6 7 5 3 9 7 7 8 3 6 3 0 0 7 1 9 4 0 0 1 9 5 5 4 9 7

8.2 Inputn.dat

0 1 8 4 7 9 2 4 0 9 1 5 8 3 7 0 8 0 1 7 7 4 7 0 6 6 6 6 1 2 7 2 5 1 8 5 7 5 1 0 7 4 5 8 5 2 0 8 5 0 2 1 3 1 1
8 2 2 1 7 9 6 3 3 0 6 2 7 4 9 4 7 1 5 5 2 8 6 7 2 7 8 7 8 0 1 4 1 5 3 6 4 2 6 4 1 4 7 6 0 5 0 5 5 9 5 4 5 4 7
0 5 3 5 7 0 9 7 0 1 6 6 8 2 9 0 3 0 2 8 0 7 3 4 3 5 0 7 8 4 6 5 8 9 3 0 7 8 1 7 7 5 4 9 1 6 3 2 5 5 1 6 6 6 3
5 3 3 7 5 4 5 3 8 8 6 2 6 9 4 5 3 0 6 4 2 7 5 5 6 5 9 5 3 0 7 5 6 3 9 0 0 5 6 8 4 3 9 5 4 6 9 9 0 0 1 4 7 9 2
5 4 7 9 2 8 1 9 6 5 3 3 6 9 9 4 8 5 1 4 3 3 4 1 0 1 0 5 4 5 4 5 5 8 6 7 6 7 4 0 6 0 5 0 5 1 9 3 1 6 2 8 3 8 8
4 4 2 5 4 4 8 1 8 7 6 2 2 0 3 6 0 7 5 6 4 4 5 2 8 4 0 7 0 7 7 4 6 8 1 6 0 4 4 7 1 3 8 1 3 3 4 5 5 1 9 4 5 9 7
2 1 5 0 2 6 0 3 8 5 2 3 9 2 0 1 1 7 1 3 2 5 5 6 3 6 9 7 5 6 3 1 3 1 1 0 6 9 6 2 0 2 7 9 7 7 8 1 1 7 4 8 3 3 3
6 6 1 2 3 5 8 3 8 9 5 8 2 4 2 3 5 0 7 1 6 0 1 0 6 5 6 6 4 9 7 7 6 9 4 5 7 4 2 9 0 5 1 4 7 4 6 2 6 8 6 4 6 0 7
4 2 6 2 6 7 8 2 4 6 5 4 1 0 0 8 3 3 0 6 4 8 0 4 6 6 3 2 4 1 1 7 0 6 4 7 6 3 9 8 0 6 1 8 2 2 7 9 5 1 7 2 2 6 5
7 0 4 7 8 8 7 6 8 7 6 1 5 2 2 7 9 8 7 3 2 3 5 1 6 9 1 5 0 4 6 2 5 4 9 8 3 8 9 6 0 2 8 2 2 2 1 1 9 7 0 0 2 4 9
1 8 2 5 7 8 9 9 7 8 1 2 8 2 0 1 0 4 4 5 1 8 4 6 4 0 6 6 0 8 7 3 8 3 1 7 8 9 9 3 3 7 8 7 5 4 9 5 0 8 1 2 0 1 1
6 4 1 7 1 0 6 6 2 3 7 4 4 5 1 9 6 9 4 6 3 9 2 7 5 4 0 6 4 7 5 7 8 4 5 4 5 2 6 0 7 5 4 4 5 3 8 4 9 5 0 0 9 4 9
8 0 1 0 7 0 6 5 7 9 2 0 5 0 0 0 9 0 3 9 5 8 9 9 3 2 8 1 1 0 5 7 5 1 0 5 1 3 0 0 1 2 8 0 8 8 3 8 4 2 7 6 9 6 4
7 9 4 9 8 0 0 7 9 0 1 6 7 7 9 6 8 4 3 4 0 1 7 4 5 1 1 2 2 1 3 1 9 9 0 9 5 0 9 3 3 4 8 4 7 6 4 7 1 7 3 6 2 5 3
5 8 4 8 8 0 2 7 7 0 9 7 4 4 9 9 4 4 1 0 0 5 5 0 5 6 5 2 6 3 5 6 9 3 1 5 7 0 0 7 2 7 8 3 0 0 2 0 8 9 3 0 3 0 1
4 2 2 3 6 9 9 5 4 8 5 2 4 1 7 6 6 5 3 4 6 6 2 8 0 5 1 4 8 6 9 8 9 1 2 8 9 9 7 9 6 0 3 1 2 0 6 1 9 6 8 3 3 0 2
6 3 4 9 4 8 3 4 4 2 8 7 0 1 3 6 0 1 5 1 3 6 4 1 2 9 7 7 1 4 7 5 6 8 5 9 6 0 2 8 7 2 5 6 8 7 9 1 6 3 7 4 1 0 1
9 1 5 8 2 4 7 6 7 7 2 1 4 2 8 3 5 2 9 7 7 9 6 7 2 5 1 3 9 3 8 4 0 5 7 5 3 2 8 4 1 4 5 6 0 4 4 9 2 4 6 3 3 5 6
8 1 8 1 9 1 0 6 2 7 5 1 6 7 1 0 8 6 7 9 0 7 6 7 4 2 3 5 3 1 3 8 0 5 9 4 7 1 2 3 9 5 0 1 5 0 7 5 8 8 4 9 0 1 4
1 4 3 9 1 4 8 9 7 6 1 6 7 6 3 2 5 8 8 7 3 4 2 3 3 5 9 2 5 7 1 2 5 2 6 6 8 0 1 2 6 5 0 7 0 8 4 5 1 9 6 9 3 6 3
4 6 9 5 4 3 4 6 2 1 0 2 5 1 8 9 4 5 2 5 5 7 5 6 4 4 7 3 2 5 4 3 0 1 5 7 4 0 4 6 1 4 7 5 4 1 8 5 8 4 3 0 7 8 1
1 9 9 3 7 9 9 4 8 8 9 6 5 9 3 8 7 4 6 6 7 8 0 8 4 3 9 9 5 5 7 3 7 2 9 1 7 2 1 1 5 7 0 2 4 7 5 9 1 7 8 6 6 7 8
8 5 5 3 0 4 9 9 3 6 4 2 5 7 9 3 5 6 5 7 0 5 1 5 1 1 9 1 1 2 5 1 2 4 7 7 1 0 9 6 8 4 7 0 8 5 2 5 0 3 4 3 6 9 8
9 1 9 7 8 5 9 9 0 9 3 2 9 4 0 2 7 1 3 6 3 4 4 8 4 9 4 4 6 8 0 1 5 7 4 0 0 8 7 3 2 2 4 1 3 5 3 0 5 7 2 7 0 5 3
0 0 9 5 6 4 5 8 9 0 0 0 4 5 1 9 8 0 9 0 7 8 6 4 0 4 1 9 0 8 5 8 0 0 4 8 0 4 4 7 9 9 6 3 3 7 4 7 7 6 6 0 5 5 5
6 9 1 9 4 0 9 4 7 9 4 5 0 9 0 4 4 5 7 0 2 3 5 2 7 3 1 2 0 0 6 0 3 1 2 8 7 8 3 5 4 0 7 9 1 4 6 8 3 3 1 6 0 2 0
2 8 3 0 1 3 4 8 5 5 5 2 1 7 1 9 6 5 3 6 5 2 5 4 5 5 4 1 6 9 3 5 2 1 9 0 2 3 4 3 9 7 8 2 3 0 4 5 0 7 4 7 4 4 3
7 9 2 3 2 8 6 3 7 4 9 9 3 4 7 3 6 0 7 6 1 2 6 7 2 1 1 0 2 0 8 7 4 9 0 4 4 1 2 4 9 6 9 5 3 1 9 5 7 5 5 4 3 3 9
9 6 5 4 2 4 7 7 7 5 6 3 4 6 6 7 6 3 7 0 6 4 9 7 2 2 5 6 7 8 8 0 8 3 6 3 2 7 9 0 0 5 5 3 0 1 1 3 7 4 2 6 4 3 6
1 0 3 5 9 7 9 6 9 5 2 5 8 2 5 8 2 3 6 5 4 8 0 8 0 7 3 0 3 6 6 8 6 0 7 7 9 4 8 9 8 5 6 0 8 6 4 7 5 6 1 5 2 0 2
5 2 5 8 3 2 5 9 9 5 8 7 0 7 9 4 1 4 0 6 9 1 4 0 8 4 3 1 5 9 5 9 2 6 1 6 1 3 4 7 2 8 0 1 4 9 6 9 3 8 8 4 0 9 6
8 2 8 4 3 8 0 3 4 1 1 3 0 8 7 5 2 8 3 8 7 1 1 4 0 7 9 8 1 9 1 1 2 3 1 6 3 5 3 0 7 7 7 5 4 8 6 9 7 8 7 1 8 1 4
0 5 8 9 3 6 1 5 4 3 7 8 2 0 0 2 7 3 1 9 5 2 6 0 8 4 6 5 2 9 4 2 1 0 1 9 6 8 9 6 9 4 9 5 6 0 5 1 0 6 9 9 7 7 8
6 1 4 0 5 5 7 4 4 9 1 9 1 4 4 1 8 4 8 2 3 2 5 9 7 7 4 7 8 4 5 2 0 1 3 0 9 0 8 5 9 1 1 8 7 7 0 4 2 7 8 9 5 2 7
7 6 7 6 4 1 3 7 5 5 4 4 3 1 8 6 6 8 1 1 2 3 3 0 1 0 3 1 6 8 1 9 0 9 2 4 2 4 4 2 9 7 8 4 1 5 7 8 8 8 5 6 5 9 1
6 2 9 0 1 9 1 0 4 9 2 6 9 9 7 3 9 5 4 6 7 1 7 4 4 6 7 7 1 3 5 1 0 9 9 9 7 8 4 9 7 6 4 6 9 9 7 8 3 8 8 2 0 0 5
3 6 1 4 1 7 1 0 3 5 0 2 3 6 9 6 3 3 0 3 8 3 3 8 0 1 2 2 7 0 4 6 1 2 0 4 9 6 7 8 5 8 2 4 9 3 0 5 9 4 6 5 4 7 3
3 4 3 3 0 5 0 2 8 2 2 9 6 8 3 3 4 5 3 2 9 0 2 3 5 4 5 3 4 9 0 1 8 0 1 0 4 3 9 0 0 2 4 6 0 6 1 3 2 0 0 2 9 2 6
0 5 1 2 7 4 9 8 5 1 4 7 5 2 2 5 7 6 9 9 8 4 3 8 3 2 4 5 3 1 8 1 5 5 0 3 9 5 7 6 8 9 3 3 1 3 9 8 8 3 5 3 9 6 0
1 2 5 0 4 0 0 7 6 1 9 3 4 1 1 9 4 2 3 0 4 8 1 0 3 4 8 3 5 4 5 4 3 8 7 4 3 2 7 6 5 7 8 2 4 2 3 9 5 1 8 9 4 8 9
5 9 3 3 1 0 7 0 7 0 2 8 5 5 0 4 9 3 9 4 7 7 6 1 3 3 2 8 6 5 0 8 1 6 5 6 5 6 9 7 3 3 2 6 9 8 8 7 9 0 9 2 3 0 9
3 5 6 9 0 6 0 8 8 9 4 5 9 4 2 2 4 0 3 7 7 0 6 9 9 1 0 1 6 0 2 2 0 8 1 7 6 9 1 4 5 4 2 9 3 6 1 3 6 7 3 7 8 5 2
0 1 4 9 8 4 1 9 8 5 6 8 8 0 8 6 4 3 4 8 7 8 1 1 6 8 3 5 5 6 9 1 2 1 2 4 1 4 0 3 1 0 3 8 3 2 8 8 7 5 8 5 4 6 1
4 9 0 1 3 8 2 7 8 8 9 2 4 1 9 7 3 3 7 4 9 0 1 9 8 4 2 8 6 4 3 3 5 4 2 7 4 8 2 1 8 5 4 3 4 7 8 2 4 7 4 7 6 2 9
5 9 7 0 5 2 1 9 6 9 2 5 3 6 0 4 1 1 6 5 8 0 6 1 1 5 5 1 4 0 9 6 4 7 8 9 8 0 4 0 1 2 3 4 9 8 3 2 3 9 5 5 3 9 4
4 5 4 5 2 1 6 6 5 3 5 6 8 9 9 9 5 3 6 3 9 9 9 8 5 8 9 9 9 7 2 0 9 5 8 3 6 6 9 5 8 8 3 8 0 7 4 1 6 3 3 7 3 7 8
2 6 0 3 1 3 1 9 0 3 2 0 9 5 2 7 0 5 6 0 2 9 2 1 9 7 0 7 8 2 6 4 4 0 5 8 4 3 6 0 5 3 2 8 4 2 6 3 6 6 4 0 1 5 5

```
8 0 2 4 2 2 9 8 0 7 9 2 4 1 0 2 3 1 5 3 1 0 6 2 5 4 3 1 6 3 7 6 9 5 5 1 3 5 9 0 4 2 0 7 3 8 6 1 2 0 5 5 6 4 3
1 3 3 3 5 8 3 4 1 9 4 3 6 6 6 3 7 5 3 3 4 9 5 7 5 4 4 5 8 7 5 6 2 5 9 5 6 9 1 2 9 8 8 3 4 7 1 7 0 1 6 4 8 6 2
9 5 8 2 7 2 8 5 5 6 0 3 7 3 0 4 4 4 9 1 2 5 4 0 4 7 7 8 5 8 9 5 9 5 8 7 5 0 7 7 6 7 1 1 4 0 1 5 9 4 4 8 1 7 8
1 9 6 2 0 4 4 7 0 9 8 8 7 1 4 8 7 4 2 6 0 6 5 6 5 4 8 8 6 7 0 4 9 6 6 5 2 1 0 1 9 4 2 9 9 5 2 2 4 5 5 9 9 3 5
0 6 0 0 0 7 3 0 7 5 6 1 3 8 2 6 5 8 6 8 1 4 3 2 3 5 4 7 3 3 3 3 0 2 9 5 7 7 8 6 7 8 9 7 8 6 3 8 6 0 8 9 5 2 1
9 4 9 8 2 2 2 2 5 8 8 4 4 0 7 8 2 5 1 9 4 3 5 0 0 7 1 2 7 7 2 5 5 7 9 1 1 4 1 4 3 3 3 0 8 8 6 6 2 4 1 0 5 0 1
6 1 2 3 7 4 8 2 6 9 3 1 3 8 2 5 7 7 9 8 5 1 8 4 1 1 2 9 4 2 0 5 7 2 3 2 2 3 2 0 6 0 5 3 5 1 8 2 9 3 4 5 9 5 6
3 3 2 3 6 7 8 6 7 6 9 6 8 7 8 5 3 7 9 0 4 6 7 6 3 8 9 4 9 0 7 6 8 2 4 4 9 0 5 3 6 4 7 3 2 5 1 4 1 2 7 6 5 9 7
4 0 3 4 5 7 0 3 8 5 6 7 3 4 9 7 0 8 5 6 6 9 6 0 2 8 7 8 9 3 6 6 7 4 2 9 4 1 2 1 7 9 0 5 5 2 4 0 5 7 5 2 7 9 1
5 4 0 1 5 3 4 6 8 1 0 5 1 7 1 3 2 0 2 4 5 4 3 1 1 0 8 1 3 8 7 3 6 8 1 0 4 8 9 7 8 5 7 3 0 2 8 4 8 0 6 9 6 9 4
8 7 0 2 4 1 7 8 0 8 7 1 5 8 0 2 0 3 7 5 6 6 4 4 4 0 7 0 5 3 6 8 6 6 4 3 1 7 4 2 8 1 9 4 4 1 0 1 1 6 3 7 1 8 1
8 8 7 5 1 9 8 3 3 8 7 2 9 1 6 7 6 8 4 1 0 5 2 1 1 2 8 8 0 2 7 3 1 2 8 9 0 1 3 2 6 4 8 0 5 4 5 0 0 7 1 7 3 5 6
3 1 1 9 6 4 3 5 8 0 6 7 6 4 5 1 6 7 3 6 4 3 9 7 3 0 8 9 7 3 4 6 0 1 0 1 1 5 5 8 8 9 7 8 5 1 4 3 8 6 7 0 3 4 1
9 4 3 3 2 3 7 0 8 6 4 8 3 9 3 0 1 5 1 9 1 3 2 5 3 2 6 0 1 2 7 8 1 7 1 1 4 2 7 7 2 8 7 1 8 1 8 3 1 8 3 1 4 6 6
2 3 7 9 2 9 7 8 9 5 4 7 1 7 7 0 7 1 1 9 7 3 5 6 3 6 7 1 3 5 3 0 8 7 6 1 6 4 9 0 3 1 1 2 3 0 3 0 5 0 6 1 6 6 1
9 7 8 5 5 1 7 6 2 4 9 2 9 3 7 9 0 0 9 4 6 5 5 9 5 2 9 2 4 4 1 1 1 1 9 5 1 8 5 7 5 0 4 0 0 9 9 7 3 4 4 8 0 0 9
2 7 2 7 1 8 2 5 4 1 1 4 2 6 3 6 5 3 6 9 1 6 1 6 9 8 6 4 2 9 4 5 9 5 8 0 6 5 0 5 6 6 6 7 6 6 0 3 2 8 3 1 8 7 4
```

9 Output

The following program produces the output, i.e. the product of the above two matrices:

```
from basicmatrices import *
import pickle as p

f1=open("Input.dat", "rb")
f2=open("Inputn.dat", 'rb')

matrix1=[]
matrix2=[]

while True:
  try:
    row=p.load(f1)
    matrix1.append(row)
  except EOFError:
    break
while True:
  try:
    row=p.load(f2)
    matrix2.append(row)
  except EOFError:
    break

temp_result=strassen(matrix1,matrix2)
result=[]
for i in temp_result:
  result.append(list(map(int,i)))
```

```
1233    1069    1161    1106    1033    1338    1283    1063    1359    1308    1153    1244    1243    1132

1319    1184    1216    1238    1302    1396    1418    1178    1666    1543    1312    1299    1401    1408

1365    987     1050    1083    1113    1201    1419    1073    1449    1552    1239    1289    1286    1246

1298    1115    1266    1292    1238    1397    1531    1034    1475    1471    1239    1183    1347    1248

1252    1147    1325    1390    1229    1384    1419    1239    1503    1416    1372    1346    1410    1405

1302    1262    1211    1296    1243    1464    1319    1143    1566    1476    1438    1227    1361    1244

1281    1098    1102    1034    1084    1227    1222    997     1458    1260    1232    1146    1305    1213

1117    897     1111    1072    996     1105    1081    945     1264    1287    1158    1066    1180    1166
```

1124	868	1112	1070	1013	1158	1264	1078	1252	1310	1065	1193	1125	1147
1266	1104	1295	1282	1207	1497	1579	1246	1543	1534	1497	1408	1438	1488
1356	1051	1129	1192	1208	1272	1369	993	1417	1375	1201	1262	1256	1189
1522	1324	1412	1181	1364	1577	1452	1352	1792	1514	1415	1512	1505	1462
1208	1035	1231	1102	1049	1129	1241	1009	1368	1322	1137	1127	1111	1223
1007	990	1163	1090	1060	1125	1264	1032	1327	1250	1237	1057	1170	1220
1359	1215	1233	1218	1234	1466	1460	1216	1526	1573	1260	1331	1387	1283
1347	1074	1282	1372	1279	1406	1395	1116	1514	1477	1279	1305	1347	1338
1333	1113	1381	1279	1257	1441	1394	1206	1501	1454	1460	1284	1513	1436
1533	1272	1278	1333	1262	1378	1432	1208	1641	1652	1420	1306	1431	1413
1300	1159	1287	1284	1262	1329	1492	1351	1431	1354	1254	1333	1463	1351
1069	1019	990	1067	1031	1219	1246	1002	1314	1414	1124	1197	1268	1138
1197	1132	1110	1064	1104	1279	1296	1129	1419	1474	1161	1191	1304	1354
1103	923	1146	1170	1068	1175	1284	929	1437	1161	1096	1152	1112	1204
1225	1159	1147	1233	1201	1282	1487	1070	1423	1510	1270	1294	1365	1256
1329	1178	1219	1235	1252	1295	1404	1269	1516	1431	1313	1440	1477	1341
1024	918	1048	1142	1139	1160	1315	1218	1374	1283	1113	1401	1218	1173
1348	1133	1175	1166	1261	1257	1431	1136	1442	1591	1311	1294	1310	1305
1125	1049	1137	1070	1222	1334	1421	1111	1408	1455	1195	1261	1248	1125
1232	1175	1363	1351	1251	1393	1509	1218	1558	1397	1426	1362	1404	1384
1259	1061	1161	1131	1109	1389	1431	1107	1431	1348	1241	1270	1296	1248
1518	1343	1290	1409	1485	1642	1757	1389	1776	1761	1364	1420	1500	1457
1335	1087	1360	1292	1258	1375	1509	1273	1588	1536	1331	1335	1320	1450
1388	1169	1293	1206	1211	1391	1450	1225	1542	1315	1342	1218	1317	1347
1286	1128	1292	1277	1168	1260	1385	1034	1399	1455	1137	1334	1343	1237
1374	1210	1396	1411	1290	1440	1452	1271	1532	1674	1349	1371	1457	1434
1242	968	1051	1089	1178	1180	1280	1018	1255	1377	1184	1104	1314	1243
1250	1054	1093	1146	1233	1291	1424	1111	1508	1450	1196	1255	1308	1314
1141	1079	962	1195	1212	1221	1388	1074	1329	1392	1129	1114	1170	1168
1318	1119	1305	1269	1334	1418	1429	1245	1571	1560	1389	1420	1398	1434

1377	1254	1250	1273	1209	1284	1421	1155	1587	1529	1275	1210	1351	1354
1440	1241	1350	1279	1405	1416	1529	1234	1668	1602	1386	1480	1515	1550
1114	1036	1083	1085	1087	1204	1283	1050	1311	1363	1207	1204	1206	1236
1455	1293	1265	1331	1313	1563	1743	1328	1717	1794	1548	1302	1591	1419
1454	1360	1422	1436	1362	1414	1559	1349	1574	1398	1486	1433	1434	1354
1287	1114	1215	1123	1208	1263	1392	1098	1466	1441	1206	1275	1355	1371
1286	1085	1120	1107	994	1351	1354	1103	1343	1245	1265	1207	1180	1163
1418	1266	1248	1186	1282	1296	1417	1241	1525	1582	1373	1269	1329	1319
1439	1192	1384	1316	1275	1578	1657	1276	1650	1788	1412	1484	1451	1453
1521	1350	1346	1234	1368	1358	1514	1230	1608	1421	1395	1322	1460	1351
1489	1372	1362	1363	1375	1539	1624	1217	1740	1705	1345	1425	1522	1515
1330	1107	1209	1090	1137	1379	1303	1132	1576	1478	1274	1338	1348	1319
1420	1227	1228	1216	1269	1425	1576	1156	1509	1571	1353	1359	1415	1351
1096	1077	1195	1128	1083	1171	1272	1061	1394	1330	1101	1242	1319	1223
1464	1246	1370	1401	1433	1614	1777	1415	1751	1750	1456	1490	1538	1597
1287	1101	1357	1208	1204	1369	1456	1395	1521	1559	1277	1336	1435	1436
1174	954	873	912	1018	1079	1245	1023	1235	1268	1050	966	1096	1102
1352	1176	1337	1314	1322	1480	1623	1389	1583	1620	1416	1459	1584	1418
1231	1143	1165	1226	1201	1318	1364	1156	1357	1399	1177	1270	1369	1323
1233	1022	1289	1361	1207	1376	1521	1301	1393	1340	1412	1220	1316	1322
1244	1050	1140	1231	1082	1201	1239	1079	1398	1281	1133	1298	1200	1186
1304	989	1202	1251	1080	1354	1487	1152	1366	1502	1318	1257	1280	1364
1134	926	1069	1103	1060	1207	1284	1028	1274	1200	1189	1166	1239	1146
1180	1023	1143	1164	1156	1303	1355	1160	1358	1330	1093	1246	1219	1236
1237	1114	1124	989	1159	1115	1111	937	1342	1297	1159	1018	1219	1163
1288	1057	1177	1081	1138	1230	1293	1179	1434	1405	1176	1257	1283	1255

10 Bibliography

- google.com
- wikipedia.com
- onlinemathtools.com
- Computer Science with Python (Sumita Arora)