This is a handout I created explaining Strassen's algorithm for matrix multiplication. It includes a breakdown of the problem, inputs and outputs, an example, a comparison with the naive solution, a explanation of Strassen's approach, and a complexity analysis.

Strassen's Algorithm for Matrix Multiplication

The Problem: Matrix Multiplication

Matrix Multiplication involves calculating the product of two matrices A and B to produce a third matrix C.

Inputs:

- Two square matrices A and B, both of size $n \times n$. Each element in these matrices represents a numerical value.

Output:

- The resulting matrix C, also of size $n \times n$, where each element $C_{i,j}$ is the sum of the products of the elements from the i-th row of A and the j-th column of B. Below is a visualization of how this output is calculated:

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \times \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ c_1a_2 + d_1c_2 & c_1b_2 + d_1d_2 \end{bmatrix}$$

Figure 1: source: https://www.cuemath.com/algebra/multiplication-of-matrices/

Small real-world Example:

Consider a situation where we need to edit an image. The image can be represented by a matrix, A. Let B represent the matrix we will be transforming A with through multiplication. With matrix multiplication we can calculate a simple transformation of values in this 2D space:

- **Matrix A**: Transformation coefficients

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- **Matrix B**: Image values

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

- **Output**: Resulting transformed image:

$$C = A \times B = ?$$

Naive Solution (Brute Force)

A naive method for matrix multiplication follows the this formula: For each element $C_{i,j}$, compute:

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \times B_{k,j}$$

Example Calculation:

Using the example matrices A and B from above:

$$C_{1,1} = (1 \times 5) + (2 \times 7) = 5 + 14 = 19$$

$$C_{1,2} = (1 \times 6) + (2 \times 8) = 6 + 16 = 22$$

$$C_{2,1} = (3 \times 5) + (4 \times 7) = 15 + 28 = 43$$

$$C_{2,2} = (3 \times 6) + (4 \times 8) = 18 + 32 = 50$$

So, the resulting matrix from $A * B$ is:

$$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Naive Approach Code

Below is an implementation of brute force matrix multiplication.

```
def matrix_multiply(A, B):
    """
    Multiplies two matrices A and B using brute force. The number of columns in A must be equal to the r

    :param A: (array of arrays of ints or floats) Represents a matrix of size m x n
    :param B: (array of arrays of ints or floats) Represents a second matrix of size n x p

    :return C: (array of arrays of ints or floats) The resulting matrix of size m x p from A multiplied

    >>> A = [[-1,4],[2,3]]
    >>> B = [[9,-3], [6,1]]
    >>> multiply_bf(A, B)
    [[15, 7],[36, -3]]
    """
    num_Acols = len(A[0])
    num_Bcols = len(B[0])

    C = [[0]*num_Bcols for i in range(len(A))]
```

```
for rowA in range(len(A)):
    for colA in range(num_Acols):
        for colB in range(num_Bcols):
            C[rowA][colB] += B[colA][colB] * A[rowA][colA]
return C
```

Complexity:

- **Time Complexity**: $O(n^3)$ for multiplying two $n \times n$ matrices.
- **Space Complexity** $O(n^2)$ for creating a new $n \times n$ matrix to hold the result

Strassen's Algorithm

Strassen's Algorithm is a more efficient method for matrix multiplication using a divide-and-conquer approach. It reduces the number of necessary multiplications by splitting the matrices into smaller submatrices.

Why Strassen is a Divide-and-Conquer Algorithm:

Strassen's algorithm checks off all three requirements of a divide-and-conquer algorithm:

- **Divide**: Split each $n \times n$ matrix into four $\frac{n}{2} \times \frac{n}{2}$ submatrices.
- **Conquer**: Perform matrix multiplications on these submatrices using specific combinations.
- **Combine**: Use the results of these smaller multiplications to compute the final $n \times n$ matrix.

Strassen's Formula:

Strassen's Algorithm divides the input matrices into sub matrices of size N/2 until the matrices are of size 1x1. This reduces the multiplication of two 2×2 matrices into 7 multiplications instead of 8 multiplications as in the naive approach.

For matrices:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Strassen defines the following products:

1. $M_1 = (a + d) \times (e + h)$
2. $M_2 = (c + d) \times e$
3. $M_3 = a \times (f - h)$
4. $M_4 = d \times (g - e)$
5. $M_5 = (a + b) \times h$
6. $M_6 = (c - a) \times (e + f)$
7. $M_7 = (b - d) \times (g + h)$

The resulting matrix C is computed as:

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$$

Pseudo-Code:

Note that the inputs are both $n \times n$ matrices. I use the subtract and add functions as helper functions:

```
Add(A, B) #A and B must be n by n 2D arrays (matrices)
  n = A.length
  result = [[0 for x = 0 to n] for j = 0 to n]
  for i = 0 to n:
      for j = 0 to n:
          result[i][j] = A[i][j] - B[i][j]
  return result

Subtract(A, B) #A and B must be n by n 2D arrays (matrices)
  n = A.length
  result = [[0 for x = 0 to n] for j = 0 to n]
  for i = 0 to n:
      for j = 0 to n:
          result[i][j] = A[i][j] + B[i][j]
  return result
```

```
Strassen(A, B)
  if A.length == 1:
        return A[0][0] * B[0][0]

  C = n x n matrix
  middle = A.length // 2

  # Divide A and B into four submatrices of size n/2 x n/2
  sub_a = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_b = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_c = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_d = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_e = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_f = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_g = [[0 for x = 0 to middle] for j = 0 to middle]
  sub_h = [[0 for x = 0 to middle] for j = 0 to middle]

  # Fill in values of new sub matrices
  for i = 0 to middle
      for j = 0 to middle:
          sub_a[i][j] = A[i][j]
          sub_b[i][j] = A[i][j + middle]
          sub_c[i][j] = A[middle + i][j]
          sub_d[i][j] = A[i + middle][j + middle]
          sub_e[i][j] = B[i][j]
          sub_f[i][j] = B[i][j + middle]
          sub_g[i][j] = B[middle + i][j]
          sub_h[i][j] = B[i + middle][j + middle]

  # Compute M1 through M7 using the formulas above and recursion
  M1 = Strassen(Add(sub_a, sub_d), sub_e + sub_h)
  M2 = Strassen(Add(sub_c, sub_d), sub_e)
  M3 = Strassen(sub_a, Subtract(sub_f, sub_h))
  M4 = Strassen(sub_d, Subtract(sub_g, sub_e))
```

```
M5 = Strassen(Add(sub_a, sub_b), sub_h)
M6 = Strassen(Subtract(sub_c, sub_a), Add(sub_e, sub_f))
M7 = Strassen(Subtract(sub_b, sub_d), Add(sub_g, sub_h))

# Combine results into C
for i = 0 to middle:
      for j = 0 to middle:
          C[i][j] = M1[i][j] + M4[i][j] - M5[i][j] + M7[i][j]
          C[i][j + middle] = M3[i][j] + M5[i][j]
          C[middle + i][j] = M2[i][j] + M4[i][j]
          C[middle + i][j + middle] = M1[i][j] - M2[i][j] + M3[i][j] + M6[i][j]

return C
```

Example Using Strassen's Algorithm

We can use Strassen's algorithm to multiply the matrices A and B:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Step-by-Step Calculation:

1. **Divide**:
    - $a = 1, b = 2, c = 3, d = 4$
    - $e = 5, f = 6, g = 7, h = 8$
2. **Calculate M1 to M7**:
    - $M_1 = (1 + 4)(5 + 8) = 5 \times 13 = 65$
    - $M_2 = (3 + 4) \times 5 = 7 \times 5 = 35$
    - $M_3 = 1 \times (6 - 8) = 1 \times (-2) = -2$
    - $M_4 = 4 \times (7 - 5) = 4 \times 2 = 8$
    - $M_5 = (1 + 2) \times 8 = 3 \times 8 = 24$
    - $M_6 = (3 - 1) \times (5 + 6) = 2 \times 11 = 22$
    - $M_7 = (2 - 4) \times (7 + 8) = (-2) \times 15 = -30$
3. **Combine**:
    - $C_{1,1} = 65 + 8 - 24 - 30 = 19$
    - $C_{1,2} = -2 + 24 = 22$
    - $C_{2,1} = 35 + 8 = 43$
    - $C_{2,2} = 65 - 35 - 2 + 22 = 50$

The result is:

$$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

5. Complexity Analysis

- **Time Complexity**: $O(n^{\log_2 7}) \approx O(n^{2.81})$
  - Strassen's algorithm improves upon the naive $O(n^3)$ by reducing the number of multiplications.
  - It is faster for large matrices but has larger constant factors and stack overhead, making it less practical for small matrices.
- **Space Complexity**: $O(n^{\log_2 7})$ -> Note this is a larger than the $0(n)$ space complexity for the naive approach

---

References:

- https://en.wikipedia.org/wiki/Strassen_algorithm
- https://www.geeksforgeeks.org/strassens-matrix-multiplication/
- https://www.cuemath.com/algebra/multiplication-of-matrices/