

# Matrix Multiplication

## Problem Introduction:

The goal in this problem is to find the product of two  $n \times n$  matrices  $X$  and  $Y$  is a third  $n \times n$  matrix  $Z = XY$ , with  $(i, j)$  entry.

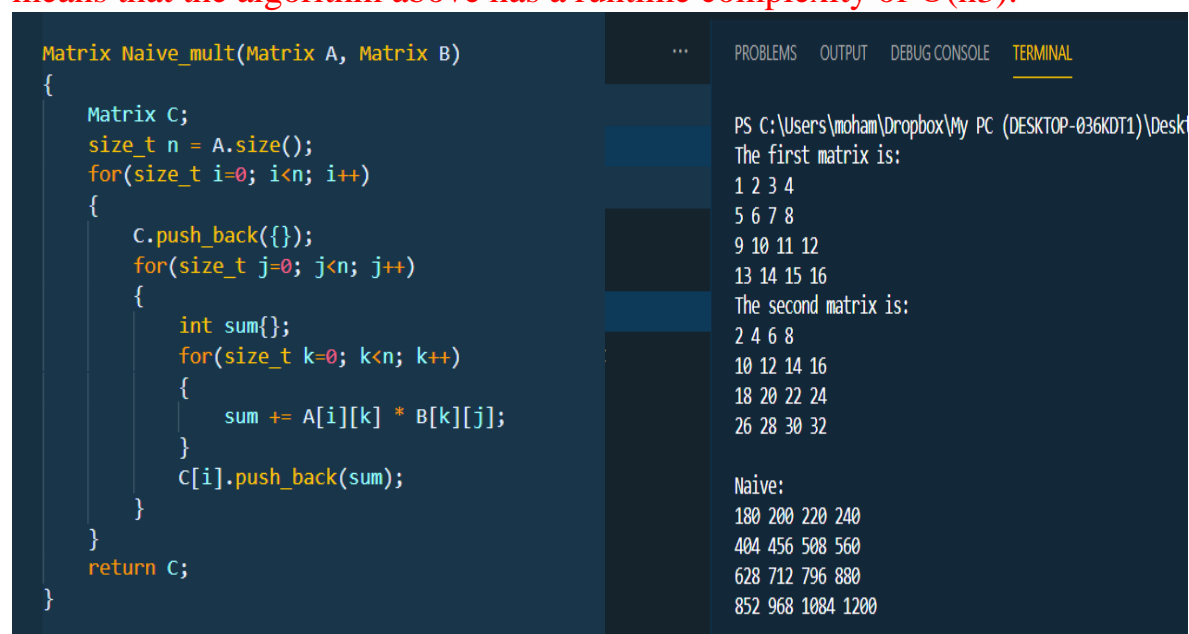
The multiplication can be done using 3 different methods.

## 1st Method: (Naive Approach)

Let us start with two square matrices  $A$  and  $B$  which are both of size  $n$  by  $n$ . In the product  $C = A \times B$  we define the entry  $c_{ij}$ , the entry in the  $i$ th row and the  $j$ th column of  $A$ , as being the dot product of the  $i$ th row of  $A$  with the  $j$ th column of  $B$ .

Taking the dot product of vectors just means that you take the products of the individual components and then add up the results.

Each of the triply nested for loops above runs for exactly  $n$  iterations, which means that the algorithm above has a runtime complexity of  $\Theta(n^3)$ .



```
Matrix Naive_mult(Matrix A, Matrix B)
{
    Matrix C;
    size_t n = A.size();
    for(size_t i=0; i<n; i++)
    {
        C.push_back({});
        for(size_t j=0; j<n; j++)
        {
            int sum{};
            for(size_t k=0; k<n; k++)
            {
                sum += A[i][k] * B[k][j];
            }
            C[i].push_back(sum);
        }
    }
    return C;
}
```

PS C:\Users\moham\Dropbox\My PC (DESKTOP-036KDT1)\Desk  
The first matrix is:  
1 2 3 4  
5 6 7 8  
9 10 11 12  
13 14 15 16  
The second matrix is:  
2 4 6 8  
10 12 14 16  
18 20 22 24  
26 28 30 32  
Naive:  
180 200 220 240  
404 456 508 560  
628 712 796 880  
852 968 1084 1200

## 2nd Method: (Divide & Conquer Approach)

Let us first assume that  $n$  is an exact power of 2 in each of the  $n \times n$  matrices for  $A$  and  $B$ . This simplifying assumption allows us to break a big  $n \times n$  matrix into smaller blocks or quadrants of size  $n/2 \times n/2$ , while also ensuring that the dimension  $n/2$  is an integer. This process is termed as block partitioning and the good part about it is that once matrices are split into blocks and multiplied, the blocks behave as if they were atomic elements. The product of  $A$  and  $B$  can then be expressed in terms of its quadrants. This gives us an idea of a recursive approach where we can keep partitioning matrices into quadrants until they are small enough to be multiplied in the naive way.

For multiplying two matrices of size  $n \times n$ , we make 8 recursive calls above, each on a matrix/subproblem with size  $n/2 \times n/2$ . Each of these recursive calls multiplies two  $n/2 \times n/2$  matrices, which are then added together. For the addition, we add two matrices of size  $n^2/4$ , so each addition takes  $\Theta(n^2/4)$  time.

```
Matrix Divide_Conquer(Matrix A, Matrix B)
{
    if(A.size()==1)
    {
        Matrix c_min = {{A[0][0] * B[0][0]}};
        return c_min;
    }

    vector<Matrix> split_A = Split(A);
    vector<Matrix> split_B = Split(B);
    Matrix C1 = Add_matrix(Divide_Conquer(split_A[0], split_B[0]), Divide_Conquer(split_A[1], split_B[2]));
    Matrix C2 = Add_matrix(Divide_Conquer(split_A[0], split_B[1]), Divide_Conquer(split_A[1], split_B[3]));
    Matrix C3 = Add_matrix(Divide_Conquer(split_A[2], split_B[0]), Divide_Conquer(split_A[3], split_B[2]));
    Matrix C4 = Add_matrix(Divide_Conquer(split_A[2], split_B[1]), Divide_Conquer(split_A[3], split_B[3]));
    vector<Matrix> c_array = {C1, C2, C3, C4};
    Matrix C = Concatenate(c_array);
    return C;
}
```

```
The first matrix is:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
The second matrix is:
2 4 6 8
10 12 14 16
18 20 22 24
26 28 30 32

Divide&Conquer:
180 200 220 240
404 456 508 560
628 712 796 880
852 968 1084 1200
```

### 3rd Method: (Strassen's Algorithm Approach)

Strassen's algorithm makes use of the same divide and conquer approach as above, but instead uses only 7 recursive calls rather than 8. This is enough to reduce the runtime complexity to sub-cubic time!

However, unlike the first case, the runtime complexity here is big  $O(n \cdot \log(a))$ , which is big  $O(n \log 27)$  or big  $O(n^{2.81})$ .

```
Matrix Strassen(Matrix A, Matrix B)
{
    if(A.size()==1)
    {
        Matrix c_min = {{A[0][0] * B[0][0]}};
        return c_min;
    }
    vector<Matrix> split_A = Split(A);
    vector<Matrix> split_B = Split(B);
    Matrix P1 = Strassen(split_A[0], Subtract_matrix(split_B[1], split_B[3]));
    Matrix P2 = Strassen(Add_matrix(split_A[0], split_A[1]), split_B[3]);
    Matrix P3 = Strassen(Add_matrix(split_A[2], split_A[3]), split_B[0]);
    Matrix P4 = Strassen(split_A[3], Subtract_matrix(split_B[2], split_B[0]));
    Matrix P5 = Strassen(Add_matrix(split_A[0], split_A[3]), Add_matrix(split_B[0], split_B[3]));
    Matrix P6 = Strassen(Subtract_matrix(split_A[1], split_A[3]), Add_matrix(split_B[2], split_B[3]));
    Matrix P7 = Strassen(Subtract_matrix(split_A[0], split_A[2]), Add_matrix(split_B[0], split_B[1]));

    Matrix C1 = Add_matrix(Add_matrix(P5, P6), Subtract_matrix(P4, P2));
    Matrix C2 = Add_matrix(P1, P2);
    Matrix C3 = Add_matrix(P3, P4);
    Matrix C4 = Subtract_matrix(Add_matrix(P1, P5), Add_matrix(P3, P7));
    vector<Matrix> c_array = {C1, C2, C3, C4};
    Matrix C = Concatenate(c_array);
    return C;
}
```

The first matrix is:

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

The second matrix is:

```
2 4 6 8
10 12 14 16
18 20 22 24
26 28 30 32
```

Strassen:

```
180 200 220 240
404 456 508 560
628 712 796 880
852 968 1084 1200
```