Divide and Conquer | Set 5 (Strassen’s Matrix Multiplication)

Given two square matrices A and B of size n x n each, find their multiplication matrix.

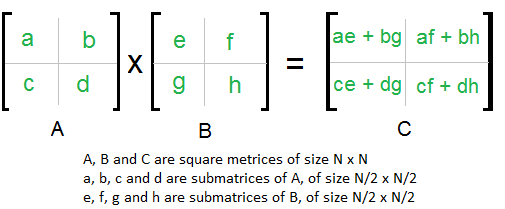
***Naive Method***  
Following is a simple way to multiply two matrices.

|  |
| --- |
| void multiply(int A[][N], int B[][N], int C[][N])  {      for (int i = 0; i < N; i++)      {          for (int j = 0; j < N; j++)          {              C[i][j] = 0;              for (int k = 0; k < N; k++)              {                  C[i][j] += A[i][k]\*B[k][j];              }          }      }  } |

Run on IDE

Time Complexity of above method is O(N3).

***Divide and Conquer***  
Following is simple Divide and Conquer method to multiply two square matrices.  
1) Divide matrices A and B in 4 sub-matrices of size N/2 x N/2 as shown in the below diagram.  
2) Calculate following values recursively. ae + bg, af + bh, ce + dg and cf + dh.

[](http://d21vdchv0ihj7g.cloudfront.net/wp-content/uploads/strassen_new.png)

In the above method, we do 8 multiplications for matrices of size N/2 x N/2 and 4 additions. Addition of two matrices takes O(N2) time. So the time complexity can be written as

T(N) = 8T(N/2) + O(N2)

From [Master's Theorem](http://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/), time complexity of above method is O(N3)

which is unfortunately same as the above naive method.

***Simple Divide and Conquer also leads to O(N3), can there be a better way?***  
In the above divide and conquer method, the main component for high time complexity is 8 recursive calls. The idea of**Strassen’s method** is to reduce the number of recursive calls to 7. Strassen’s method is similar to above simple divide and conquer method in the sense that this method also divide matrices to sub-matrices of size N/2 x N/2 as shown in the above diagram, but in Strassen’s method, the four sub-matrices of result are calculated using following formulae.

[](http://d21vdchv0ihj7g.cloudfront.net/wp-content/uploads/stressen_formula_new_new.png)

**Time Complexity of Strassen’s Method**  
Addition and Subtraction of two matrices takes O(N2) time. So time complexity can be written as

T(N) = 7T(N/2) + O(N2)

From [Master's Theorem](http://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/), time complexity of above method is

O(NLog7) which is approximately O(N2.8074)

Generally Strassen’s Method is not preferred for practical applications for following reasons.  
1) The constants used in Strassen’s method are high and for a typical application Naive method works better.  
2) For Sparse matrices, there are better methods especially designed for them.  
3) The submatrices in recursion take extra space.  
4) Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen’s algorithm than in Naive Method (Source: [CLRS Book](http://www.flipkart.com/introduction-algorithms-3rd/p/itmczynzhyhxv2gs?pid=9788120340077&affid=sandeepgfg))

**References:**  
[Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest](http://www.flipkart.com/introduction-algorithms-3rd/p/itmczynzhyhxv2gs?pid=9788120340077&affid=sandeepgfg)  
<https://www.youtube.com/watch?v=LOLebQ8nKHA>  
<https://www.youtube.com/watch?v=QXY4RskLQcI>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

* - - - - - - - - - - - - - An example

Example step-through of Strassen's method for matrix multiplication on 2x2 matrices

/\* MULTIPLIES A and B =============

using Strassen's O(n^2.81) method

A = [1 3] B = [6 8]

[7 5] [4 2]

C = A \* B = ?

=================================\*/

// Step 1: Split A and B into half-sized matrices of size 1x1 (scalars).

def a11 = 1

def a12 = 3

def a21 = 7

def a22 = 5

def b11 = 6

def b12 = 8

def b21 = 4

def b22 = 2

// Define the "S" matrix.

def s1 = b12 - b22 // 6

def s2 = a11 + a12 // 4

def s3 = a21 + a22 // 12

def s4 = b21 - b11 // -2

def s5 = a11 + a22 // 6

def s6 = b11 + b22 // 8

def s7 = a12 - a22 // -2

def s8 = b21 + b22 // 6

def s9 = a11 - a21 // -6

def s10 = b11 + b12 // 14

// Define the "P" matrix.

def p1 = a11 \* s1 // 6

def p2 = s2 \* b22 // 8

def p3 = s3 \* b11 // 72

def p4 = a22 \* s4 // -10

def p5 = s5 \* s6 // 48

def p6 = s7 \* s8 // -12

def p7 = s9 \* s10 // -84

// Fill in the resultant "C" matrix.

def c11 = p5 + p4 - p2 + p6 // 18

def c12 = p1 + p2 // 14

def c21 = p3 + p4 // 62

def c22 = p5 + p1 - p3 - p7 // 66

/\* RESULT: =================

C = [ 18 14]

[ 62 66]

=========================\*/