BỘ GIÁO DỤC VÀ ĐÀO TẠO

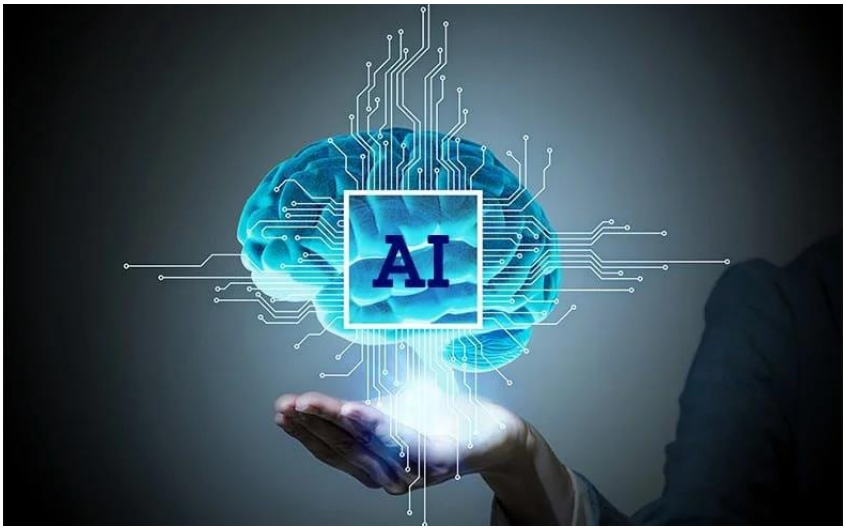
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

KHOA TOÁN - TIN HỌC

-----📖-----

BÀI BÁO CÁO THỰC HÀNH TUẦN 5

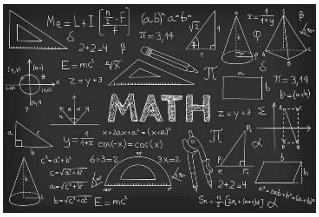


MÔN HỌC: **Phân Tích Thuật Toán**

Sinh Viên: **Trần Công Hiếu - 21110294**

Lớp: **21TTH**

TP.HCM, ngày 11 tháng 05 năm 2024



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA TOÁN – TIN HỌC

BÀI BÁO CÁO THỰC HÀNH TUẦN 5

HK1 - NĂM HỌC: 2024-2025

MÔN: PHÂN TÍCH THUẬT TOÁN

SINH VIÊN: TRẦN CÔNG HIẾU

MSSV: 21110294

LỚP: 21TTH

NHẬN XÉT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.HCM, Ngày……Tháng……Năm 2021

Giảng viên Bộ môn

Mục Lục

Bài 1.

1.1. Chương trình với $O(n^3)$.

```python
import numpy as np

n = int(input("Input n: "))

A = np.random.randint(1, 1001, size=(n,n))
B = np.random.randint(1, 1001, size=(n,n))
C = np.zeros((n,n), dtype=int)

print("- Matrix A is:\n",A)
print("- Matrix B is:\n",B)
print("- Matrix C is:\n",C)

for i in range(0,n):
    for j in range(0,n):
        for k in range(0,n):
            C[i][j] += A[i][k]*B[k][j]
print("- Matrix C=A*B is:\n",C)

```

(File: bai1_1.py).

"import numpy as np": Gọi thư viện numpy để thao tác với mảng trong đoạn mã của chương trình và gọi tắt là "np", điều này cho phép gọi các hàm và đối tượng từ thư viện numpy bằng cách sử dụng tiền tố "np".

"n = int(input("Input n: "))": Yêu cầu nhập giá trị đầu vào n là kích thước của ma trận vuông.

"A = np.random.randint(1, 1001, size=(n,n))": Tạo ma trận A có giá trị ngẫu nhiên từ 1 đến 1000 bằng phương thức randint() và kích thước ma trận là $n \times n$.

"B = np.random.randint(1, 1001, size=(n,n))": Tương tự, ta tạo ma trận B có giá trị ngẫu nhiên từ 1 đến 1000 bằng phương thức randint() và kích thước ma trận là $n \times n$.

"C = np.zeros((n,n))": Tạo ma trận C có kích thước $n \times n$ với các giá trị 0. Và in ra lần lượt 3 ma trận để dễ dàng kiểm tra kết quả.

Tạo lần lượt 3 vòng lặp for cho biến chạy i, j và k trong đại diện index của phần tử thuộc ma trận. Ứng với đó ta gán giá trị cho ma trận C như công thức tính tích 2 ma trận.

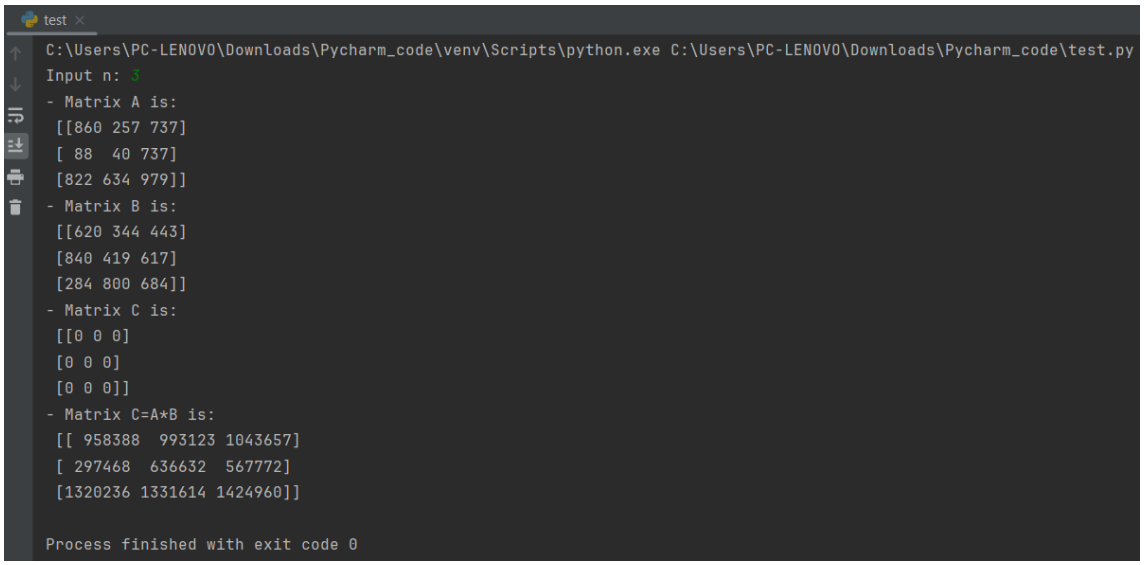
Phép nhân hai ma trận: $A = [a_{ij}]_{n \times p}; B = [b_{ij}]_{p \times m}$
Tích hai ma trận A và B, ký hiệu $A \times B$, là ma trận $C = [c_{ij}]_{n \times m}$ với:

$$c_{ij} = \sum_{k=1}^{p} a_{ik}.b_{kj}$$

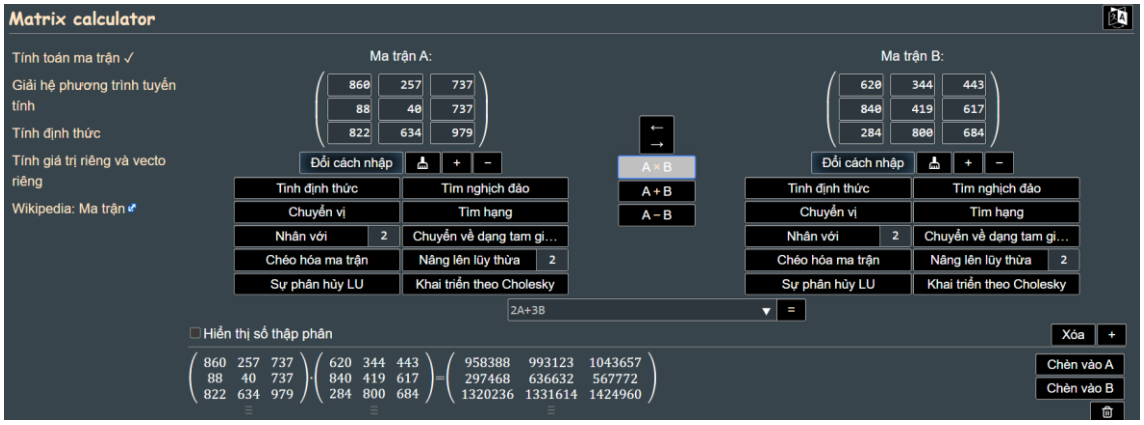
"print("- Matrix C=A*B is:\n",C)": Cuối cùng in kết quả ma trận C ra.

Kết quả.

- Khi chạy đoạn mã.



- Kiểm tra lại bằng tính toán online trên website.

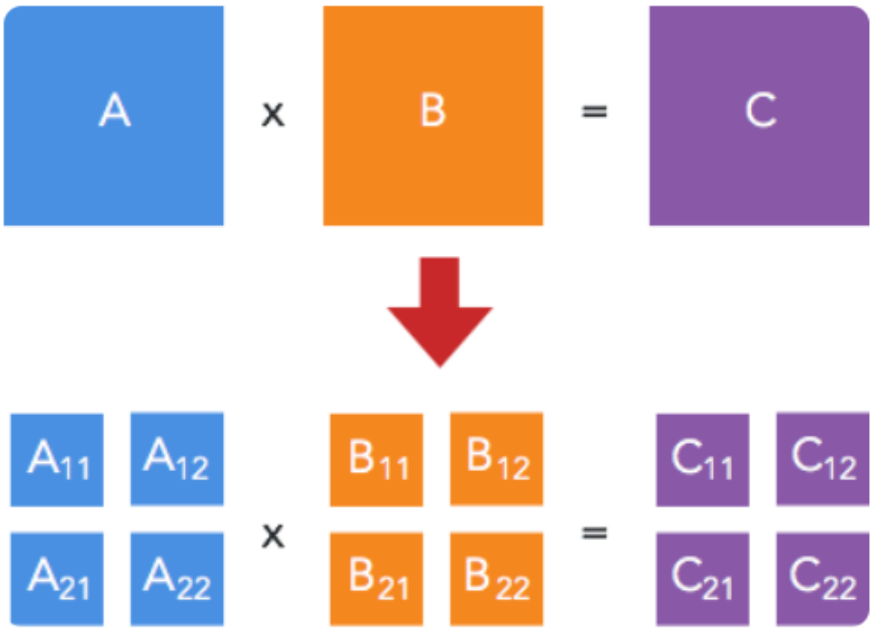


* Chứng minh độ phức tạp thuật toán của đoạn mã trên.

Dễ thấy rằng, đoạn mã chỉ có các thao tác gán $O(1)$ và chủ yếu là 3 vòng lặp lồng nhau với số lần lặp mỗi vòng là n nên độ phức tạp thuật toán là $O(n^3)$.

1.2. Chương trình với $O(n^{log7})$.

*** Ý tưởng:** Thuật toán của Strassen là áp dụng chia để trị để giải quyết bài toán theo hướng của giải thuật cơ bản trên. Cụ thể là: với mỗi ma trận vuông A, B, C có kích thước n×n, chúng ta chia chúng thành 4 ma trận con, và biểu diễn tích A×B=Ctheo các ma trận con đó:



Trong đó:

$$
\begin{aligned}
C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}
$$

Tuy nhiên với cách phân tích này thì chúng ta vẫn cần 8 phép nhân để tính ra ma trận C. Đây là phần quan trọng nhất của vấn đề.

Chúng ta định nghĩa ra các ma trận M mới như sau:

$$
\begin{aligned}
M_1 &= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
M_2 &= (A_{2,1} + A_{2,2})B_{1,1} \\
M_3 &= A_{1,1}(B_{1,2} - B_{2,2}) \\
M_4 &= A_{2,2}(B_{2,1} - B_{1,1}) \\
M_5 &= (A_{1,1} + A_{1,2})B_{2,2} \\
M_6 &= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
M_7 &= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
\end{aligned}
$$

Và biểu diễn lại các phần tử của C theo M như sau:

$$
\begin{aligned}
C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
C_{1,2} &= M_3 + M_5 \\
C_{2,1} &= M_2 + M_4 \\
C_{2,2} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

Giải thích đoạn mã.

```python
import numpy as np

def split(matrix):
    row, col = matrix.shape
    row2, col2 = row//2, col//2
    return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:, col2:]

```

(File: bai1_2.py)

"import numpy as np": Gọi thư viện numpy để thao tác với mảng trong đoạn mã của chương trình và gọi tắt là "np", điều này cho phép gọi các hàm và đối tượng từ thư viện numpy bằng cách sử dụng tiền tố "np".

"def split(matrix):": Định nghĩa hàm split() với đối số truyền vào là ma trận cần chia. Mục đích của hàm là chia ma trận được truyền vào thành ma trận vuông có n bằng một nửa n ban đầu của matrix.

```
def strassen(A, B):
    if (len(A)==1):
        return A*B
    # split matrix A to A1, A2, A3, A4
    A1, A2, A3, A4 = split(A)

    # split matrix B to B1, B2, B3, B4
    B1, B2, B3, B4 = split(B)

    # caculate M
    M1 = strassen((A1+A4), (B1+B4))
    M2 = strassen((A3+A4), B1)
    M3 = strassen(A1, (B2-B4))
    M4 = strassen(A4, (B3-B1))
    M5 = strassen((A1+A2), B4)
    M6 = strassen((A3-A1), (B1+B2))
    M7 = strassen((A2-A4), (B3+B4))

    C1 = M1+M4-M5+M7
    C2 = M3+M5
    C3 = M2+M4
    C4 = M1+M3-M2+M6

    # Combining the 4 quadrants into a single matrix by stacking horizontally and vertically.
    C = np.vstack((np.hstack((C1, C2)), np.hstack((C3, C4))))

    return C

```

"def strassen(A, B):": Định nghĩa hàm strassen() với đối số truyền vào là 2 ma trận. Theo như thuật toán strassen, ta sẽ chia nhỏ lần lượt đến khi n = 2 thì ta sẽ tính như công thức và ghép lại thông qua

"C = np.vstack((np.hstack((C1, C2)), np.hstack((C3, C4)))))": Thực hiện việc kết hợp 4 vùng con của ma trận thành một ma trận duy nhất bằng cách ghép chồng theo chiều dọc và ngang. "np.hstack((C1, C2))": Thực hiện ghép chồng hai vùng con C1 và C2 theo chiều ngang, tức là ghép từng hàng của C1 với

từng hàng của C2. “np.hstack((C3, C4))”: Thực hiện ghép chồng hai vùng con C3 và C4 theo chiều ngang.

Sau đó, hai vùng con đã được ghép chồng theo chiều ngang được ghép chồng tiếp theo nhau theo chiều dọc bằng cách sử dụng np.vstack().

Kết quả là một ma trận mới được tạo ra từ việc ghép chồng các vùng con theo cách mô tả, với các vùng con C1, C2, C3 và C4 đại diện cho các phần tử của ma trận gốc sau khi đã được chia nhỏ.

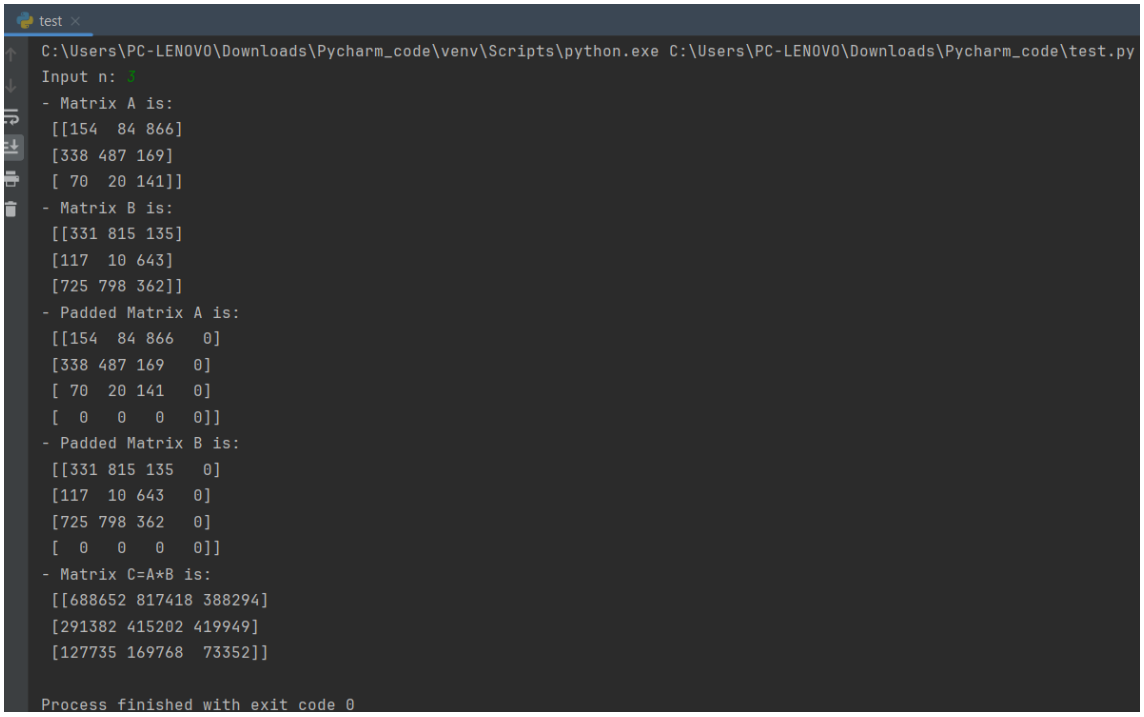
```
36  n = int(input("Input n: "))
37
38  if((n%2)==0):
39      A = np.random.randint(1, 1001, size=(n,n))
40      B = np.random.randint(1, 1001, size=(n,n))
41      print("- Matrix A is:\n",A)
42      print("- Matrix B is:\n",B)
43      C = strassen(A,B)
44  else:
45      A = np.random.randint(1, 1001, size=(n,n))
46      B = np.random.randint(1, 1001, size=(n,n))
47      print("- Matrix A is:\n",A)
48      print("- Matrix B is:\n",B)
49      A_padded = np.pad(A, ((0, 1), (0, 1)), mode='constant')
50      B_padded = np.pad(B, ((0, 1), (0, 1)), mode='constant')
51      print("- Padded Matrix A is:\n",A_padded)
52      print("- Padded Matrix B is:\n",B_padded)
53      C = strassen(A_padded, B_padded)[:2, :2]
54  print("- Matrix C=A*B is:\n",C)
```

“n = int(input(“Input n: ”))”: Nhập n là kích thước của ma trận vuông.

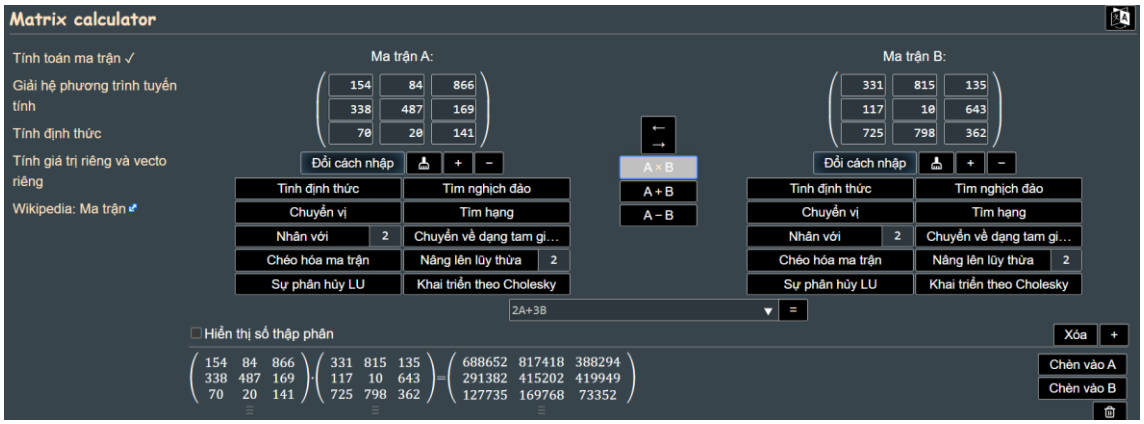
Vì theo như thuật toán Strassen, ta sẽ “chia để trị” cho 2 nên nếu trường hợp n là lẻ thì ta thêm vào bên phải và bên dưới ma trận A và B các dòng 0 để thành ma trận vuông cấp chẵn.

Kết quả:

- Khi chạy đoạn mã.



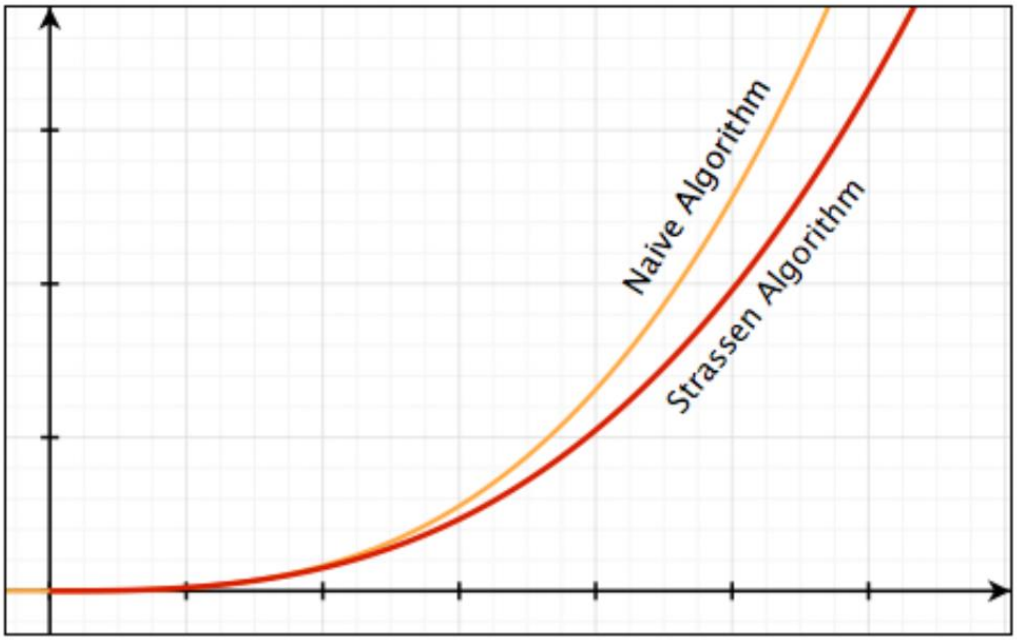
- Kiểm tra lại bằng tính toán online trên website.



* Chứng minh độ phức tạp thuật toán của đoạn mã trên.

Từ đoạn mã, ta có T(n) = 7T(n/2) + O(n^2). Hơn nữa, theo định lý của Master nên ta có độ phức tạp của phương pháp trên là O(n^{log7}).

*** Đồ thị so sánh 2 phương pháp với $n = 2^k, k = 10, 11, \dots, 32.$**



Bài 2.

Không thể nhân hai ma trận vuông cấp n với độ phức tạp O(n^2). Vì để tính toán mỗi phần tử trong ma trận kết quả, ta cần thực hiện phép nhân và cộng của một hàng với một cột tương ứng từ hai ma trận gốc. Điều này yêu cầu n phép nhân và n−1 phép cộng cho mỗi phần tử trong ma trận kết quả. Tổng số phép tính cho một phần tử trong ma trận kết quả là 2n−1, và có n^2 phần tử trong ma trận kết quả. Do đó, tổng số phép tính để tính toán ma trận kết quả là n^2(2n−1), một độ phức tạp là O(n^3).

Tuy nhiên, với tư duy tối ưu thuật toán. Nên ta có được những thuật toán tối ưu hơn so với O(n^3) mà không chỉ dừng lại ở thuật toán Strassen mà còn là thuật toán Coppersmith-Winograd với độ phức tạp $O(n^{2.376})$ và thuật toán CW cải tiến với độ phức tạp $O(n^{2.373})$. Và các nhà khoa học vẫn đang miệt mài nghiên cứu để đưa con số này về O(n^2).

Thuật toán	Input	Độ phức tạp
Naive Algorithm	Ma trận vuông	$O(n^3)$
Naive Algorithm	Ma trận bất kì	$O(nmp)$
Strassen Algorithm	Ma trận vuông	$O(n^{2.807})$
Coppersmith-Winograd Algorithm	Ma trận vuông	$O(n^{2.376})$
Các thuật toán CW cải tiến	Ma trận vuông	$O(n^{2.373})$