Matrix Multiplication

Advanced Programming and Algorithmic Design

Alberto Casagrande
Email: acasagrande@units.it

a.a. 2018/2019

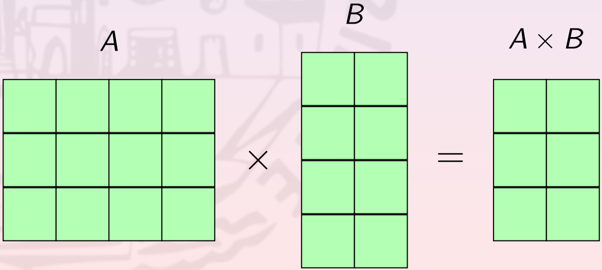
Problem Definition

Matrix Multiplication

Definition (Row-Column Multiplication)

Let A be a $n \times m$ matrix and let B be a $m \times l$ matrix. $A \times B$ is a $n \times l$ matrix s.t.

$$(A \times B)[i,j] = \sum_k A[i,k] * B[k,j]$$

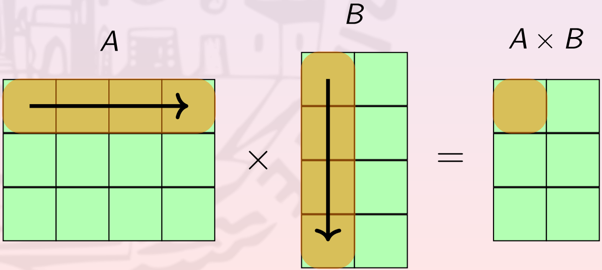


Matrix Multiplication

Definition (Row-Column Multiplication)

Let A be a $n \times m$ matrix and let B be a $m \times l$ matrix. $A \times B$ is a $n \times l$ matrix s.t.

$$(A \times B)[i,j] = \sum_k A[i,k] * B[k,j]$$

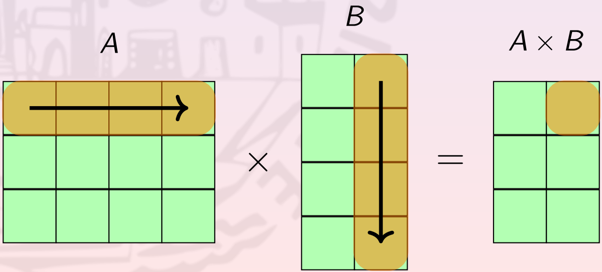


Matrix Multiplication

Definition (Row-Column Multiplication)

Let A be a $n \times m$ matrix and let B be a $m \times l$ matrix. $A \times B$ is a $n \times l$ matrix s.t.

$$(A \times B)[i,j] = \sum_k A[i,k] * B[k,j]$$

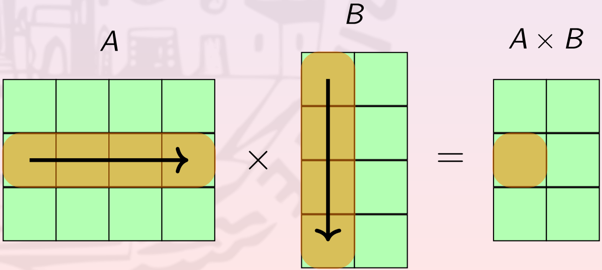


Matrix Multiplication

Definition (Row-Column Multiplication)

Let A be a $n \times m$ matrix and let B be a $m \times l$ matrix. $A \times B$ is a $n \times l$ matrix s.t.

$$(A \times B)[i,j] = \sum_k A[i,k] * B[k,j]$$



Problem Definition

Input: Two $n \times n$ matrices A and B
Output: The $n \times n$ matrix $A \times B$

E.g.,

$$A = \begin{bmatrix} 1 & -1 & 2 \\ 2 & 0 & 3 \\ 0 & -1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & -2 & 2 \\ 2 & 0 & 0 \\ -1 & 3 & 0 \end{bmatrix} \Longrightarrow A \times B = \begin{bmatrix} 0 & -4 & 2 \\ 5 & 5 & 4 \\ -4 & 6 & 0 \end{bmatrix}$$

Square matrices solution can easily be generalized

Naïve Solution: Code

```
def naive_mult(C, A, B):
  for i ← 1..rows(A):
    for j ← 1..cols(B):
      a ← 0
      for k ← 1..cols(A):
        a ← a + A[i,k] * B[k,j]
      endfor

      C[i,j] ← a
    endfor
  endfor

  return C
enddef
```

Naïve Solution: Complexity

The naïve solution mimes row-column definition

3 nested loops with indexes in $[1, n]$

The inner-block takes time $O(1)$

The overall execution takes time $\mathbf{\Theta(n^3)}$

Naïve Solution: Complexity

The naïve solution mimes row-column definition

3 nested loops with indexes in $[1, n]$

The inner-block takes time $O(1)$

The overall execution takes time $\mathbf{\Theta(n^3)}$

Can we find a better algorithm?

Divide-and-Conquer Strategy

Divide-and-Conquer Strategy

What about splitting A and B in blocks?

$$\overset{A}{\begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array}} \times \overset{B}{\begin{array}{|c|c|}\hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline \end{array}} = \overset{A \times B}{\begin{array}{|c|c|}\hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \\ \hline \end{array}}$$

where

$$C_{ij} = (A_{i1} \times B_{1j}) + (A_{i2} \times B_{2j})$$

Divide-and-Conquer Strategy (Cont'd)

$+$ is the elements-wise matrix sum (time complexity $\Theta(n^2)$)

$\times$ is the usual row-column multiplication

A_{ik} and B_{kj} are $\frac{n}{2} \times \frac{n}{2}$ matrices

Divide-and-Conquer Strategy (Cont'd)

$+$ is the elements-wise matrix sum (time complexity $\Theta(n^2)$)

$\times$ is the usual row-column multiplication

A_{ik} and B_{kj} are $\frac{n}{2} \times \frac{n}{2}$ matrices

We can define a recursive algorithm:

- if $rows(A) < 2$, return `naive_mult(C, A, B)`
- for $i, j, k \in [12]$ recursively compute $D_{ijk} = A_{ik} \times B_{kj}$
- for $i, j \in [12]$ compute $C_{ij} = D_{ij1} + D_{ij2}$
- return C

Divide-and-Conquer Strategy: Complexity

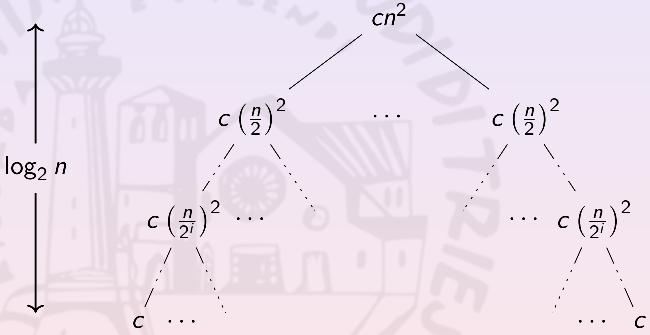
The recursive algorithm requires:

- 8 multiplications between $\frac{n}{2} \times \frac{n}{2}$ matrices
- 4 sums between $\frac{n}{2} \times \frac{n}{2}$ matrices

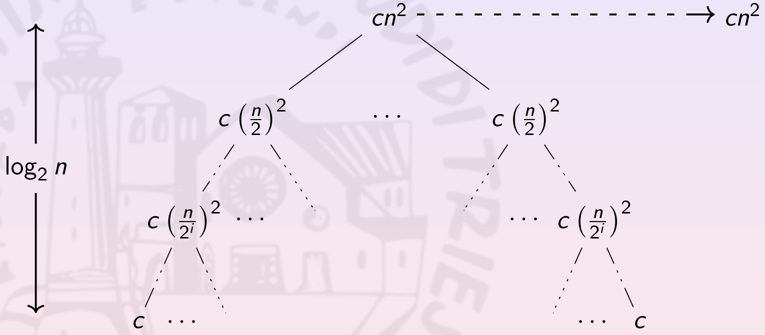
If T_M is the complexity of the algorithm

$$\begin{aligned} T_M(n) &= 8 * T_M(n/2) + 4 * \Theta(n^2) \\ &= 8 * T_M(n/2) + \Theta(n^2) \end{aligned}$$

Divide-and-Conquer Strategy: Complexity (Recursion Tree)

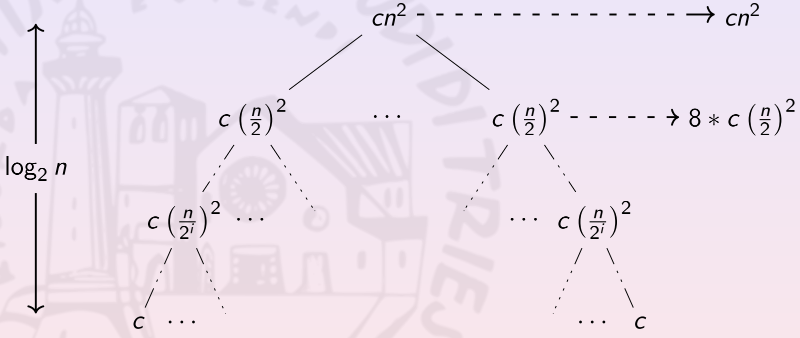


Divide-and-Conquer Strategy: Complexity (Recursion Tree)



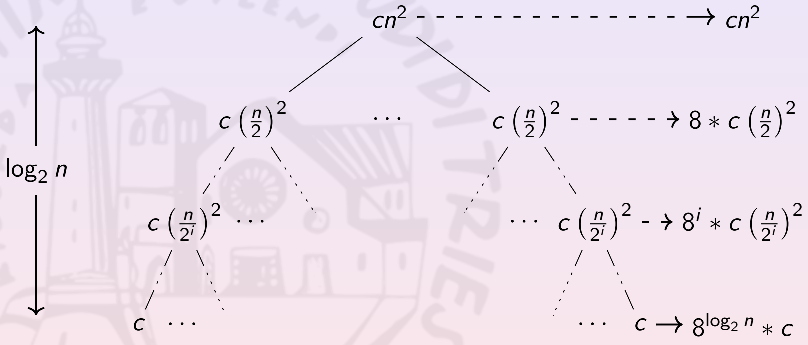
$$T_M(n) = cn^2 \Big(1 + \qquad\qquad\qquad\qquad \Big)$$

Divide-and-Conquer Strategy: Complexity (Recursion Tree)



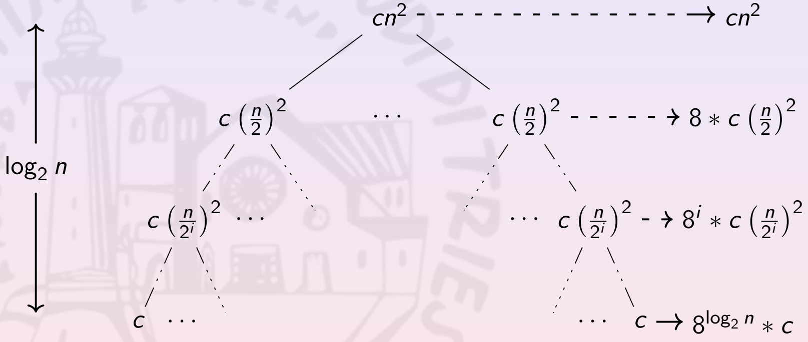
$$T_M(n) = cn^2 \left(1 + 2 + \qquad\qquad\qquad\qquad \right)$$

Divide-and-Conquer Strategy: Complexity (Recursion Tree)



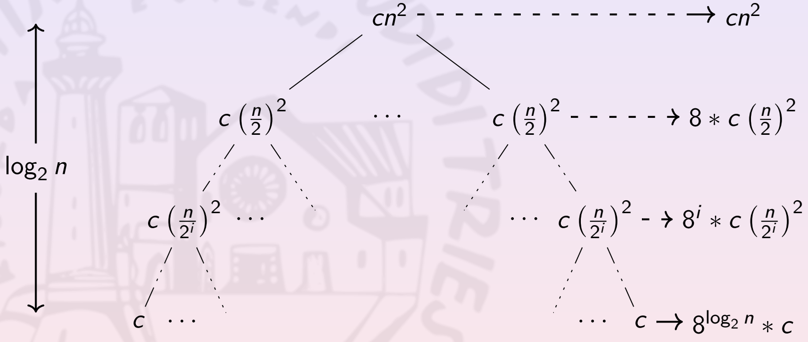
$$T_M(n) = cn^2\left(1 + 2 + \ldots + 2^i + \ldots + 2^{\log_2 n}\right)$$

Divide-and-Conquer Strategy: Complexity (Recursion Tree)



$$
\begin{aligned}
T_M(n) &= cn^2\left(1 + 2 + \ldots + 2^i + \ldots + 2^{\log_2 n}\right)\\
&= cn^2\left(2^{1+\log_2 n} - 1\right) = cn^2\,(2n-1)
\end{aligned}
$$

Divide-and-Conquer Strategy: Complexity (Recursion Tree)



$$
\begin{aligned}
T_M(n) &= cn^2\left(1 + 2 + \ldots + 2^i + \ldots + 2^{\log_2 n}\right)\\
&= cn^2\left(2^{1+\log_2 n} - 1\right) = cn^2\,(2n-1) \in \Theta(n^3)
\end{aligned}
$$

Some Further Thoughts

The divide-and-conquer approach had too many recursive calls

Can it be "rephrased" to decrease them?

Some Further Thoughts

The divide-and-conquer approach had too many recursive calls

Can it be “rephrased” to decrease them?

Yes, it can!!!

Strassen's Algorithm

Strassen's Algorithm

Sums ($\Theta(n^2)$)

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}\\
S_2 &= A_{11} + A_{12}\\
S_3 &= A_{21} + A_{22}\\
S_4 &= B_{21} - B_{11}\\
S_5 &= A_{11} + A_{22}\\
S_6 &= B_{11} + B_{22}\\
S_7 &= A_{12} - A_{22}\\
S_8 &= B_{21} + B_{22}\\
S_9 &= A_{11} - A_{21}\\
S_{10} &= B_{11} + B_{12}
\end{aligned}
$$

$$\Longrightarrow$$

Recursion Calls

$$
\begin{aligned}
P_1 &= A_{11} \times S_1\\
P_2 &= S_2 \times B_{22}\\
P_3 &= S_3 \times B_{11}\\
P_4 &= A_{22} \times S_4\\
P_5 &= S_5 \times S_6\\
P_6 &= S_7 \times S_8\\
P_7 &= S_9 \times S_{10}
\end{aligned}
$$

Strassen's Algorithm

Sums ($\Theta(n^2)$)

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

Strassen's Algorithm

Sums ($\Theta(n^2)$)

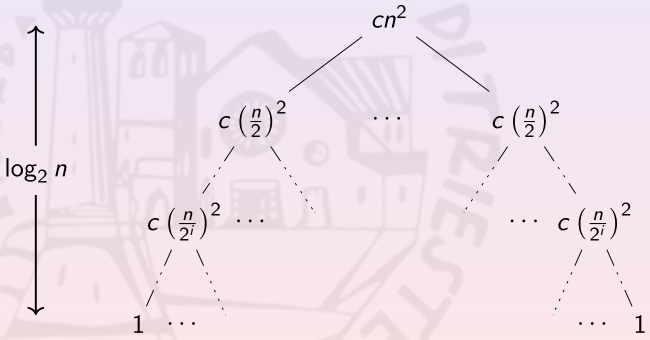
$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_5 + P_1 - P_3 - P_7$$

The complexity equation is:

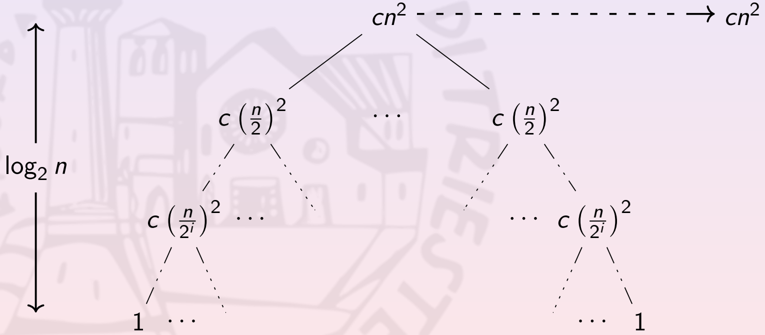
$$T_M(n) = 7 * T_M(n/2) + \Theta(n^2)$$

because the S's and the C's are computed by sums

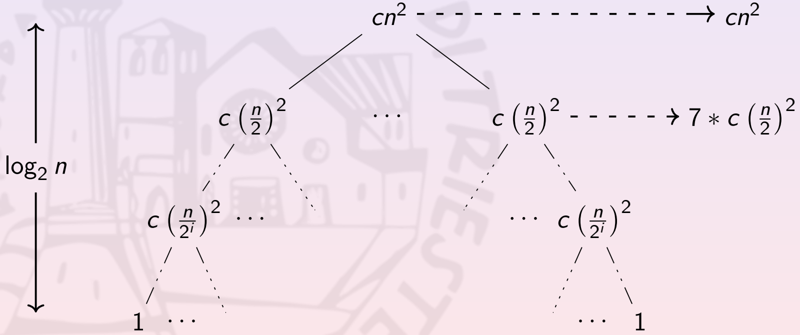
Strassen's Algorithm: Complexity (Recursion Tree)



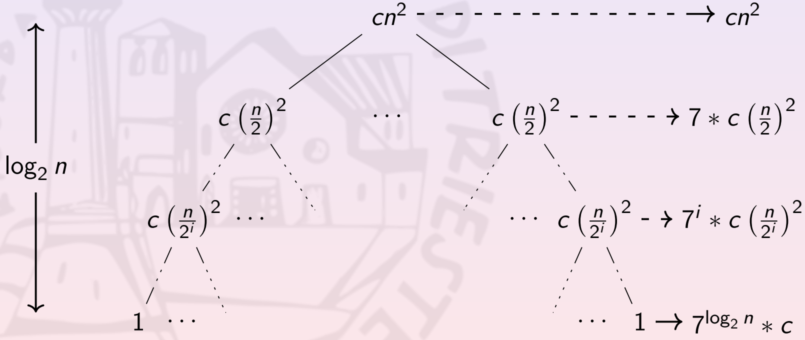
Strassen's Algorithm: Complexity (Recursion Tree)



Strassen's Algorithm: Complexity (Recursion Tree)



Strassen's Algorithm: Complexity (Recursion Tree)



Strassen's Algorithm: Complexity (Cont'd)

$$
\begin{aligned}
T_M(n) &= cn^2 \left(1 + \frac{7}{4} + \ldots + \left(\frac{7}{4}\right)^i + \ldots + \left(\frac{7}{4}\right)^{\log_2 n}\right) \\
&= c'n^2 \left(\left(\frac{7}{4}\right)^{1+\log_2 n} - 1\right) && c' = \frac{4}{3}c \\
&= c'n^2 \left(\frac{7}{4}\right)^{1+\log_2 n} - c'n^2 \\
&= c''4^{\log_2 n} \left(\frac{7}{4}\right)^{\log_2 n} - c'n^2 && c'' = \frac{7}{4}c' \\
&= c''7^{\log_2 n} - c'n^2 \\
&= c''7^{\frac{\log_7 n}{\log_7 2}} - c'n^2 = c''n^{\log_2 7} - c'n^2
\end{aligned}
$$

Strassen's Algorithm: Complexity (Cont'd)

$$T_M(n) = cn^2 \left(1 + \frac{7}{4} + \ldots + \left(\frac{7}{4}\right)^i + \ldots + \left(\frac{7}{4}\right)^{\log_2 n}\right)$$

$$= c'n^2 \left(\left(\frac{7}{4}\right)^{1+\log_2 n} - 1\right) \qquad c' = \frac{4}{3}c$$

$$= c'n^2 \left(\frac{7}{4}\right)^{1+\log_2 n} - c'n^2$$

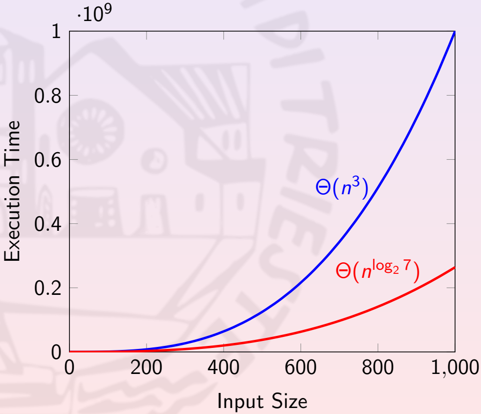
$$= c''4^{\log_2 n} \left(\frac{7}{4}\right)^{\log_2 n} - c'n^2 \qquad c'' = \frac{7}{4}c'$$

$$= c''7^{\log_2 n} - c'n^2$$

$$= c''7^{\frac{\log_7 n}{\log_7 2}} - c'n^2 = c''n^{\log_2 7} - c'n^2 \in \Theta\left(n^{\log_2 7}\right)$$

Final Considerations

Strassen's algorithm ($\Theta(n^{\log_2 7})$) improves asymptotic complexity of naïve algorithm ($\Theta(n^3) = \Theta(n^{\log_2 8})$)



Final Considerations

However, it is not **in-place** i.e., it requires a non-constant amount of additional memory

A careful handling of the auxiliary memory may make the difference in implementation.