# COMP 3804

Alina Shaikhet

# Lecture 3 - Divide & Conquer Algorithms

# Recurrences - D&C Pattern

**D&C-Algorithm**$(n)\{$

If $n$ is small, then solve and return ← **base case**

otherwise: **D&C-Algorithm**$\left(\frac{n}{b_1}\right)$

**D&C-Algorithm**$\left(\frac{n}{b_2}\right)$

$\vdots$

**D&C-Algorithm**$\left(\frac{n}{b_a}\right)$

**Combine** solution and return

$\}$

Without base case your algorithm will never stop

Dividing the problem into smaller sub-problems to solve them recursively.

# Recurrences - D&C Pattern

Lets split a problem into $\boldsymbol{a} \geq \mathbf{1}$ subproblems, each on the input of size ${}^{\boldsymbol{n}}/_{\boldsymbol{b}}$ where $\boldsymbol{b} > \mathbf{1}$

**D&C-Algorithm**$(\boldsymbol{n})\{$

If $\boldsymbol{n}$ is small, then solve and return ← $\boldsymbol{T}(\mathbf{1}) = \boldsymbol{O}(\mathbf{1})$

otherwise: **D&C-Algorithm**$\left(\frac{\boldsymbol{n}}{\boldsymbol{b}}\right)$

**D&C-Algorithm**$\left(\frac{\boldsymbol{n}}{\boldsymbol{b}}\right)$

$\vdots$

**D&C-Algorithm**$\left(\frac{\boldsymbol{n}}{\boldsymbol{b}}\right)$

$\boldsymbol{a}$ recursive calls

**Combine** solution and return

$\}$

$$\boldsymbol{T}(\boldsymbol{n}) = \boldsymbol{a}\boldsymbol{T}\left({}^{\boldsymbol{n}}/_{\boldsymbol{b}}\right) + \boldsymbol{f}(\boldsymbol{n})$$

# How to Solve Recurrences

- Iterative method or Unfolding
- Recursion tree method
- Substitution – guess the solution, then prove it by induction
- Master theorem

# Master Theorem

$d$ is the exponent in the "extra work"

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ n^d + a \cdot T\left(\frac{n}{b}\right) & \text{if } n \geq 2. \end{cases}$$

$$a \geq 1, \;\; b > 1, \;\; d \geq 0.$$

If $d > \log_b a$: $T(n) = O\left(n^d\right)$

Extra work is already big, recursive calls increase the total amount of work by a constant factor

If $d = \log_b a$: $T(n) = O\left(n^d \log n\right)$

If $d < \log_b a$: $T(n) = O\left(n^{\log_b a}\right)$

# Master Theorem

$$T(n) \le \begin{cases} 1 & \text{if } n = 1, \\ n^d + a \cdot T\left(\frac{n}{b}\right) & \text{if } n \ge 2. \end{cases}$$

$$a \ge 1,\ b > 1,\ d \ge 0.$$

If $d > \log_b a$: $T(n) = O(n^d)$

If $d = \log_b a$: $T(n) = O(n^d \log n)$

If $d < \log_b a$: $T(n) = O(n^{\log_b a})$

**Merge Sort:** $T(n) = n + 2T\left(\frac{n}{2}\right)$

$a = 2,\ b = 2,\ d = 1,\ \log_2 2 = 1$

$d = \log_b a$: $T(n) = O(n \log n)$

**Multiplying Integers:** $T(n) = n + 4T\left(\frac{n}{2}\right)$; $\quad a = 4,\ b = 2,\ d = 1$

school method

$\log_2 4 = 2,\ d < \log_b a$: $\quad T(n) = O(n^2)$

**Karatsuba:** $T(n) = n + 3T\left(\frac{n}{2}\right)$; $\quad a = 3,\ b = 2,\ d = 1$

$\log_2 3 \approx 1.58,\ d < \log_b a$: $\quad T(n) = O(n^{\log_2 3})$

# Master Theorem

$$\boldsymbol{T(n) \le \begin{cases} 1 & \text{if } n = 1, \\ n^d + a \cdot T\left(\frac{n}{b}\right) & \text{if } n \ge 2. \end{cases}}$$

$$\boldsymbol{a \ge 1,\ \ b > 1,\ \ d \ge 0.}$$

If $\boldsymbol{d > \log_b a}$: $\boldsymbol{T(n) = O\left(n^d\right)}$

If $\boldsymbol{d = \log_b a}$: $\boldsymbol{T(n) = O\left(n^d \log n\right)}$

If $\boldsymbol{d < \log_b a}$: $\boldsymbol{T(n) = O\left(n^{\log_b a}\right)}$

**Binary Tree Traversal:**

$$\boldsymbol{T(n) = 1 + 2T\left(\frac{n}{2}\right)}$$

$$a = 2,\ b = 2,\ d = 0,\ \log_2 2 = 1$$

$$\boldsymbol{d < \log_b a:\ T(n) = O(n)}$$

**Binary Search:** $\boldsymbol{T(n) = 1 + T\left(\frac{n}{2}\right)}$; $\quad a = 1,\ b = 2,\ d = 0$

$$\log_2 1 = 0,\ \boldsymbol{d = \log_b a:\ \ T(n) = O(\log n)}$$

# Master Theorem – Limitations

These recurrences cannot be solved using the Master Theorem:

$T(n) = 2^n T\left(\frac{n}{2}\right) + n^3$ — $a$ is not a constant. The number of subproblems should be fixed.

$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$ — $\frac{n}{\log n}$ is not of the form $n^d$

$T(n) = 0.5T\left(\frac{n}{2}\right) + n^2$ — $a < 1$ we cannot have less than one subproblem

$T(n) = 5T\left(\frac{n}{7}\right) - n^2$ — $f(n)$ is negative (time we spend to combine solution)

$T(n) = 2T(n) + 3$ — $b = 1$ $b$ should be strictly bigger than $\mathbf{1}$

# Searching

Searching is a fundamental problem in computer science.

Linear Search **sequentially** checks each element of the given list until a match is found or the whole list has been searched.

Binary Search searches for an element in a **sorted** array

The search space of the problem is the set of possible solutions at a given time. Every time we check an element in the input list, we reduce the size of the search space by **1** (for linear search) and by half (for binary search).

# Binary Search

**Input:** $A = [x_0, x_1, \ldots, x_{(n-1)}]$ sorted list of $n$ elements, element $y$.

**Output:** the index in the array of where the element is or $-1$ if the element is not in the array

# Searching

Let’s search for the element **52**

When the data is **sorted** we can significantly reduce the size of the search space with each element we check.

$$\boldsymbol{A} = [\overset{0}{\mathbf{1}}, \overset{1}{\mathbf{2}}, \overset{2}{\mathbf{5}}, \overset{3}{\mathbf{7}}, \overset{4}{\mathbf{11}}, \overset{5}{\mathbf{12}}, \overset{6}{\mathbf{16}}, \overset{7}{\mathbf{19}}, \overset{8}{\mathbf{21}}, \overset{9}{\mathbf{25}}, \overset{10}{\mathbf{33}}, \overset{11}{\mathbf{52}}, \overset{12}{\mathbf{57}}, \overset{13}{\mathbf{61}}, \overset{14}{\mathbf{74}}, \overset{15}{\mathbf{79}}, \overset{16}{\mathbf{99}}]$$

$$\boldsymbol{n} = \mathbf{17}$$

1. Check the middle element.
2. If it is not what we are looking for then throw away that element and every other element that we now know cannot hold the target.
3. Repeat on the resulting search space.

# Searching

Let’s search for the element **52**

When the data is **sorted** we can significantly reduce the size of the search space with each element we check.

$$\begin{array}{c} \\ A = \end{array} \begin{array}{c} \\ {[} \end{array}\begin{array}{ccccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1, & 2, & 5, & 7, & 11, & 12, & 16, & 19, & 21, & \mathbf{25}, & \mathbf{33}, & \mathbf{52}, & \mathbf{57}, & \mathbf{61}, & \mathbf{74}, & \mathbf{79}, & \mathbf{99} \end{array}\begin{array}{c} \\ {]} \end{array}$$

$$n = 17$$

1. Check the middle element.
2. If it is not what we are looking for then throw away that element and every other element that we now know cannot hold the target.
3. Repeat on the resulting search space.

# Searching

Let's search for the element **52**

When the data is **sorted** we can significantly reduce the size of the search space with each element we check.

$$\begin{array}{cccccccccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \boldsymbol{A} = [ & 1, & 2, & 5, & 7, & 11, & 12, & 16, & 19, & 21, & \mathbf{25}, & \mathbf{33}, & \mathbf{52}, & 57, & 61, & 74, & 79, & 99] \end{array}$$

$\boldsymbol{n = 17}$

1. Check the middle element.
2. If it is not what we are looking for then throw away that element and every other element that we now know cannot hold the target.
3. Repeat on the resulting search space.

# Searching

Let's search for the element **52**

When the data is **sorted** we can significantly reduce the size of the search space with each element we check.

$$A = [\overset{0}{1}, \overset{1}{2}, \overset{2}{5}, \overset{3}{7}, \overset{4}{11}, \overset{5}{12}, \overset{6}{16}, \overset{7}{19}, \overset{8}{21}, \overset{9}{\mathbf{25}}, \overset{10}{\mathbf{33}}, \overset{11}{\mathbf{52}}, \overset{12}{57}, \overset{13}{61}, \overset{14}{74}, \overset{15}{79}, \overset{16}{99}]$$

$$\boldsymbol{n} = \mathbf{17}$$

1. Check the middle element.
2. If it is not what we are looking for then throw away that element and every other element that we now know cannot hold the target.
3. Repeat on the resulting search space.

# Searching

Let's search for the element **52**

When the data is **sorted** we can significantly reduce the size of the search space with each element we check.

$$\begin{array}{c} \\ A = \end{array} \begin{array}{cccccccccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ [ & 1, & 2, & 5, & 7, & 11, & 12, & 16, & 19, & 21, & 25, & 33, & \mathbf{52}, & 57, & 61, & 74, & 79, & 99 \end{array} ]$$

$$n = 17$$

How many comparisons did we do? **4**

1. Check the middle element.
2. If it is not what we are looking for then throw away that element and every other element that we now know cannot hold the target.
3. Repeat on the resulting search space.

# Searching – Worst-case Analysis

**Linear Search** checks each element in the list in the worst case.

That is, it checks all $\boldsymbol{n}$ elements.

**Binary Search** checks at most $\mathbf{\log_2 \boldsymbol{n} + 1}$ elements in the worst case.

$$\left(\left((n/2)/2\right)/2\right)/2 \quad \dots \quad = 1 \qquad \frac{n}{\underbrace{2 \times 2 \times 2 \times \cdots \times 2}_{y \text{ times}}} = 1 \qquad \frac{n}{2^y} = 1$$

$$n = 2^y \qquad \log_2 n = \log_2 2^y = y$$

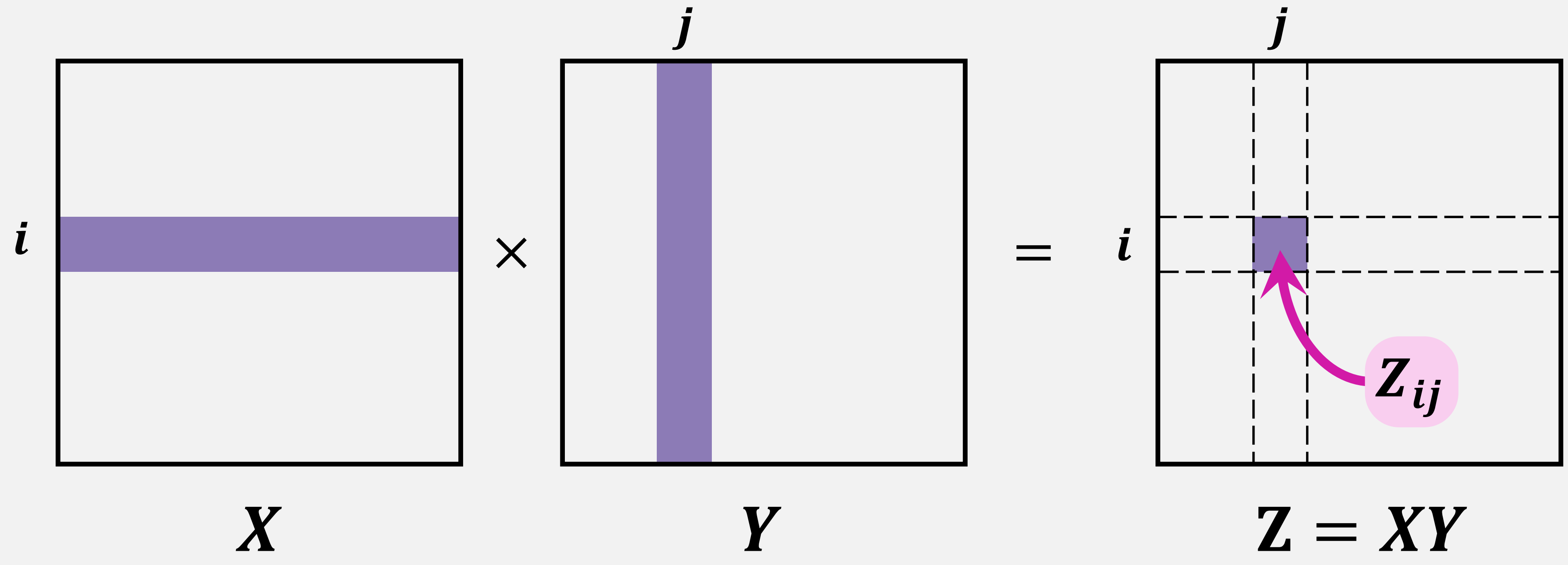
**OR:** $T(n) = 1 + T\left(\frac{n}{2}\right) = O(\log n)$

# Matrix Multiplication

**Input:** $n \times n$ matrices $X$ and $Y$.

**Output:** product $Z = XY$, which is an $n \times n$ matrix.

$Z_{ij}$ = dot product of row $i$ of $X$ and column $j$ of $Y$.




$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$

# Matrix Multiplication

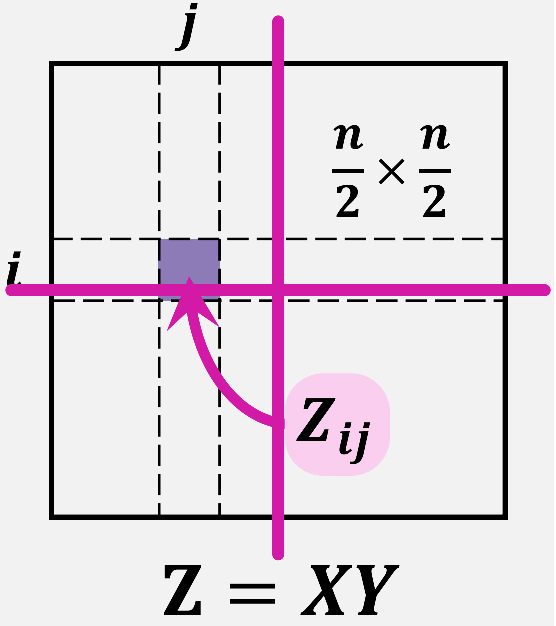
**Input:** $n \times n$ matrices $X$ and $Y$.

**Output:** product $Z = XY$, which is an $n \times n$ matrix.

$Z_{ij}$ can be computed in $O(n)$ time.

$Z$ has $n^2$ entries

$Z$ can be computed in $O(n^3)$ time.




Can we do better? **YES**

Using **Divide & Conquer**

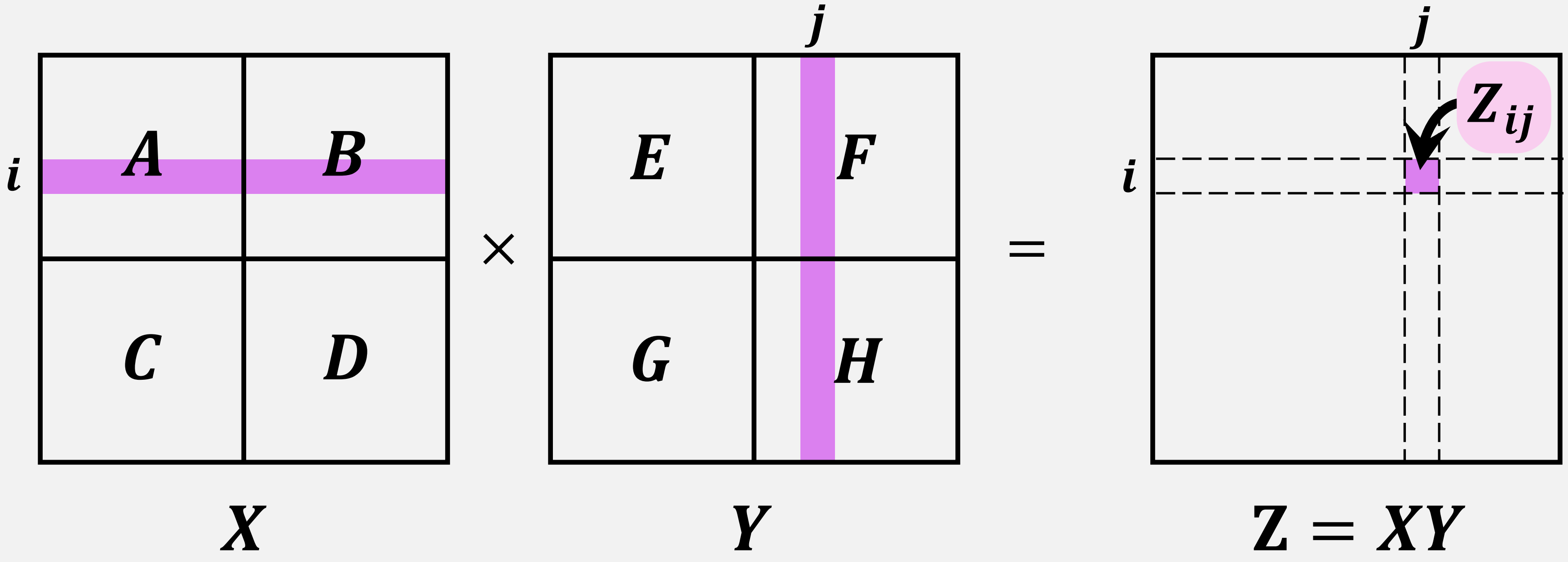
# Matrix Multiplication – Divide & Conquer

Assume $\boldsymbol{n}$ is a power of $\mathbf{2}$.

Divide both $\boldsymbol{X}$ and $\boldsymbol{Y}$ into $\mathbf{4}$ matrices each of size $\frac{\boldsymbol{n}}{\mathbf{2}} \times \frac{\boldsymbol{n}}{\mathbf{2}}$:

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}$$

$$\mathbf{Y} = \begin{pmatrix} \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} \end{pmatrix}$$




Then, $$\boldsymbol{XY} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix}\begin{pmatrix} \boldsymbol{E} & \boldsymbol{F} \\ \boldsymbol{G} & \boldsymbol{H} \end{pmatrix} = \begin{pmatrix} \boldsymbol{AE} + \boldsymbol{BG} & \boldsymbol{AF} + \boldsymbol{BH} \\ \boldsymbol{CE} + \boldsymbol{DG} & \boldsymbol{CF} + \boldsymbol{DH} \end{pmatrix}$$

# Matrix Multiplication – Divide & Conquer

$$XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

To multiply two $n \times n$ matrices:

**8** times: recursively multiply two $\frac{n}{2} \times \frac{n}{2}$ matrices,

**4** times: add two $\frac{n}{2} \times \frac{n}{2}$ matrices. $\boxed{O(n^2)}$

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 8 \cdot T\left(\frac{n}{2}\right) + O(n^2) & \text{if } n \geq 2. \end{cases}$$

$$\boxed{T(n) = O(n^3)}$$

# Matrix Multiplication – Divide & Conquer

$$XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Strassen (**1969**):

To multiply two $n \times n$ matrices:

**7** times: recursively multiply two $\frac{n}{2} \times \frac{n}{2}$ matrices,

**18** times: add two $\frac{n}{2} \times \frac{n}{2}$ matrices. $\boxed{O(n^2)}$

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 7 \cdot T\left(\frac{n}{2}\right) + O(n^2) & \text{if } n \geq 2. \end{cases}$$

$$\boxed{T(n) = O\left(n^{\log_2 7}\right)}$$

$\log_2 7 \approx 2.807$

# Matrix Multiplication – Divide & Conquer

$$XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Strassen (1969):

$$P_1 = A(F - H)$$
$$P_2 = (A + B)H$$
$$P_3 = (C + D)E$$
$$P_4 = D(G - E)$$
$$P_5 = (A + D)(E + H)$$
$$P_6 = (B - D)(G + H)$$
$$P_7 = (A - C)(E + F)$$

$$XY = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

$$\begin{aligned} P_3 + P_4 &= (C + D)E + D(G - E) \\ &= CE + DE + DG - DE \\ &= CE + DG \end{aligned}$$

# Conclusion: multiply two $\boldsymbol{n} \times \boldsymbol{n}$ matrices

- Iterative method (using definition): $O(n^3)$
- Strassen $(\mathbf{1969})$: $O(n^{2.807}) = \mathrm{O}(n^{\log_2 7})$

  it is faster than the iterative method when $\boldsymbol{n} > \mathbf{100}$
- Coppersmith & Winograd $(\mathbf{1990})$: $O(n^{2.376})$

  $(\mathbf{2014})$: $O(n^{2.373})$
- Alman & Williams (January $\mathbf{2021}$): $O(n^{2.3728596})$
- Lower bound: $\Omega(n^2)$