

# MA3227 Numerical Analysis II

## Lecture 4: Sparse LU Factorisation

Simon Etter



2019/2020

# Sparse LU Factorisation

## Introduction

We have seen in Lectures 2 and 3 how partial differential equations give rise to very large but sparse linear systems.

So far, we solved these systems using Julia's backslash operator `\`. In this lecture, we will take a closer look at how `\` works.

## Recap from Numerical Analysis I

LU factorisation (aka Gaussian elimination) is the method of choice for solving dense linear systems. Julia's `\` uses LU factorisation.

LU factorisation of a dense  $N \times N$  matrix requires  $\mathcal{O}(N^3)$  floating-point operations. This is way too much for the applications we have in mind.

## Example

Consider the 2D Poisson equation on an  $n \times n$  grid with  $n = 1000$ .

LU factorisation of  $\Delta_n^{(2)}$  requires  $\mathcal{O}(n^6) = \mathcal{O}(10^{18})$  operations.

Assuming  $10^9$  ops/sec (1GHz), this will take approximately 32 years!

**Question:** Can we compute  $A = LU$  more efficiently if  $A$  is sparse?

# Sparse LU Factorisation

## Recap LU factorisation

$$\begin{pmatrix} 4 & 1 & -2 \\ -8 & 2 & 3 \\ 12 & 7 & -5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 4 & -1 \\ 0 & 4 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 4 & -1 \\ 0 & 0 & 2 \end{pmatrix}$$

**Question:** Are  $L$  and  $U$  sparse if  $A$  is sparse?

# Sparse LU Factorisation

## Example

$$A_1 := \begin{pmatrix} 1 & & 1 \\ & 1 & 1 \\ & & 1 \\ 1 & 1 & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ & 1 & 1 \\ & & 1 \\ & & & 1 \end{pmatrix} =: L_1 U_1,$$

$$A_2 := \begin{pmatrix} 1 & & 1 \\ & -1 & 1 \\ & & 1 \\ 1 & 1 & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ & -1 & 1 \\ & & 1 \\ & & & 1 \end{pmatrix} =: L_2 U_2.$$

Observations:

- ▶  $A[i,j] = 0$  does not imply  $L[i,j] = 0$  and  $U[i,j] = 0$ .
- ▶ Some entries of  $L, U$  are zero regardless of the values we assign to the nonzero entries of  $A$ . Other entries of  $L, U$  may be zero or nonzero depending on the values in  $A$ .

# Sparse LU Factorisation

## Terminology

- ▶ *Sparsity pattern/structure of  $A$* : list of all  $(i, j)$  such that  $A[i, j] \neq 0$ .
- ▶ *Fill-in entry of  $A$* :  $(i, j)$  such that  $A[i, j] = 0$  but there exists an  $\tilde{A}$  with the same structure as  $A$  such that  $\tilde{L}[i, j] \neq 0$  or  $\tilde{U}[i, j] \neq 0$ .  
Such an entry  $(i, j)$  is called *structurally nonzero*.

## Outlook

The following slides will present statements regarding the sparsity patterns of polynomials, the inverse and the LU factorisation of  $A$ .

All these statements are meant in the structural sense:  $B[i, j] \neq 0$  means that there is an  $\tilde{A}$  with the same structure as  $A$  such that  $\tilde{B}[i, j] \neq 0$ , but for some  $A$  we may still have  $B[i, j] = 0$ .

Reasons for looking at structure rather than actual values:

- ▶ It is much easier.
- ▶ It provides a worst-case estimate.
- ▶ Cancellation (i.e. a structurally nonzero entry being zero) is rare.

# Sparse LU Factorisation

**Graph of a sparse matrix**  $A \in \mathbb{R}^{n \times n}$

Graph  $G(A) := (V(A), E(A))$  given by

$$V(A) := \{1, \dots, n\}, \quad E(A) := \{j \rightarrow i \mid A[i, j] \neq 0\}.$$

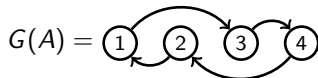
Note transpose in  $E(A)$ : entry  $A[i, j]$  corresponds to edge  $j \rightarrow i$ .

**Path in  $G = (V, E)$**

Ordered sequence  $k_0, \dots, k_p \in V$  such that  $k_{q-1} \rightarrow k_q \in E$  for all  $q$ .  
Number of edges  $p$  is called the length of the path.

**Example**

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & \bullet & \\ \bullet & & 3 & \\ & \bullet & & 4 \end{pmatrix}$$



$2 \rightarrow 1 \rightarrow 3$  is a path of length 2.

# Sparse LU Factorisation

## Path theorem for matrix powers

$$A^p[i, j] \neq 0 \iff \exists \text{ path } j \rightarrow i \text{ of length } p.$$

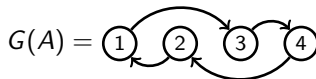
*Proof (not examinable).*

$$A^p[i, j] = \sum_{k_{p-1}} \dots \sum_{k_1} A[i, k_{p-1}] \dots A[k_a, k_{a-1}] \dots A[k_1, j].$$

Each term is nonzero iff  $j \rightarrow k_1 \rightarrow \dots \rightarrow k_{p-1} \rightarrow i$  is a path in  $G(A)$ .

## Example (continued)

$$A^2 = \begin{pmatrix} 1 & \bullet & & \bullet \\ & 2 & \bullet & \bullet \\ \bullet & \bullet & 3 & \\ \bullet & \bullet & & 4 \end{pmatrix}$$



$A^2[4, 1] \neq 0$  because  $1 \rightarrow 3 \rightarrow 4$  is a path of length 2 in  $G(A)$ .

$A^2[2, 1] = 0$  because  $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$  is a path of length 3 in  $G(A)$ .

# Sparse LU Factorisation

## Path theorem for inverse

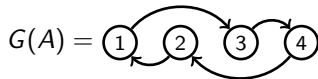
$$A^{-1}[i,j] \neq 0 \iff \exists \text{ path } j \rightarrow i.$$

*Proof (not examinable).*

- ▶  $A^{-1} = p(A)$  for polynomial  $p(x)$  interpolating  $\frac{1}{x}$  on eigenvalues of  $A$ .
- ▶ Hence entry  $(i,j)$  of  $A^{-1} = \sum_{p=0}^{n-1} c_p A^p$  is nonzero if there is a path  $j \rightarrow i$  of arbitrary length.

## Example (continued)

$$A^{-1} = \begin{pmatrix} 1 & \bullet & \bullet & \bullet \\ \bullet & 2 & \bullet & \bullet \\ \bullet & \bullet & 3 & \bullet \\ \bullet & \bullet & \bullet & 4 \end{pmatrix}$$



$A^{-1}[2,1] \neq 0$  because  $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$  is a path in  $G(A)$ .

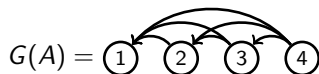


# Sparse LU Factorisation

## Corollaries of path theorem

- ▶ If  $G(A)$  is connected (there exists a path between any pair of vertices), then  $A^{-1}$  is dense.
- ▶ If  $G(A)$  is disconnected, i.e.  $A = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}$ , then  $A^{-1} = \begin{pmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{pmatrix}$ .
- ▶ Inverse of upper/lower triangular matrix is upper/lower triangular.

$$A = \begin{pmatrix} 1 & \bullet & \bullet & \bullet \\ & 2 & \bullet & \bullet \\ & & 3 & \bullet \\ & & & 4 \end{pmatrix}$$



# Sparse LU Factorisation

## Fill path

Path  $i \rightarrow k_1 \rightarrow \dots \rightarrow k_p \rightarrow j$  in  $G(A)$  such that  $k_1, \dots, k_p < \min\{i, j\}$ .

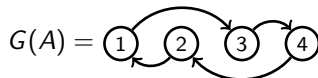
## Fill path theorem

$$(L + U)[i, j] \neq 0 \iff \exists \text{ fill path } j \rightarrow i.$$

*Proof (not examinable).* See next two slides.

## Example (continued)

$$L + U = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & \bullet & 3 & \\ & \bullet & \bullet & 4 \end{pmatrix}$$



$L[3, 2] \neq 0$  because  $2 \rightarrow 1 \rightarrow 3$  is a fill path in  $G(A)$ .

$L[4, 1] = 0$  because  $1 \rightarrow 3 \rightarrow 4$  is not a fill path in  $G(A)$ .

# Sparse LU Factorisation

## Lemma for fill path theorem (not examinable)

Let  $i, j \in \{1, \dots, n\}$  and set  $\ell := \{1, \dots, \min\{i, j\} - 1\}$ . Then,

$$\begin{aligned}U[i, j] &= A[i, j] - A[i, \ell] A[\ell, \ell]^{-1} A[\ell, j] && \text{for } i \leq j, \\L[i, j] U[j, j] &= A[i, j] - A[i, \ell] A[\ell, \ell]^{-1} A[\ell, j] && \text{for } i \geq j.\end{aligned}$$

*Proof.* Block LU factorisation with  $\bar{r} := \{\min\{i, j\}, \dots, n\}$ :

$$\begin{pmatrix} A[\ell, \ell] & A[\ell, \bar{r}] \\ A[\bar{r}, \ell] & A[\bar{r}, \bar{r}] \end{pmatrix} = \begin{pmatrix} I & \\ A[\bar{r}, \ell] A[\ell, \ell]^{-1} & I \end{pmatrix} \begin{pmatrix} A[\ell, \ell] & A[\ell, \bar{r}] \\ A[\bar{r}, \bar{r}] - A[\bar{r}, \ell] A[\ell, \ell]^{-1} A[\ell, \bar{r}] \end{pmatrix}.$$

$$\text{Let } L_1 U_1 = A[\ell, \ell], \quad L_2 U_2 = A[\bar{r}, \bar{r}] - A[\bar{r}, \ell] A[\ell, \ell]^{-1} A[\ell, \bar{r}].$$

Full factorisation is then given by

$$\begin{pmatrix} A[\ell, \ell] & A[\ell, \bar{r}] \\ A[\bar{r}, \ell] & A[\bar{r}, \bar{r}] \end{pmatrix} = \begin{pmatrix} L_1 & \\ A[\bar{r}, \ell] A[\ell, \ell]^{-1} L_1 & L_2 \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1} A[\ell, \bar{r}] \\ & U_2 \end{pmatrix}.$$

Claim follows by noting that  $L[i, j] = L_2[i, j]$  and  $U[i, j] = U_2[i, j]$  have the given form.

# Sparse LU Factorisation

## Fill path theorem (repeated from earlier slide)

$$(L + U)[i, j] \neq 0 \iff \exists \text{ fill path } j \rightarrow i.$$

*Proof (not examinable).* According to lemma on previous slide:

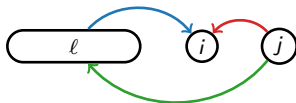
$$U[i, j] = A[i, j] - A[i, \ell] A[\ell, \ell]^{-1} A[\ell, j] \quad \text{for } i \leq j,$$

$$L[i, j] U[j, j] = A[i, j] - A[i, \ell] A[\ell, \ell]^{-1} A[\ell, j] \quad \text{for } i \geq j.$$

1st term makes  $U[i, j]$  nonzero if there is a fill path  $j \rightarrow i$  of length 1.

2nd term makes  $U[i, j]$  nonzero if there is a fill path  $j \rightarrow i$  of length  $> 1$ .

Same arguments apply for  $L[i, j] U[j, j]$ .



# Sparse LU Factorisation

## Exercise

Consider the matrix

$$A = \begin{pmatrix} 1 & & & \bullet & \\ & 2 & \bullet & & \\ \bullet & & 3 & & \\ & & & 4 & \\ & \bullet & & & 5 \end{pmatrix}.$$

- ▶ Determine the sparsity pattern of the LU factorisation of  $A$ .
- ▶ Find a permutation of the rows and columns of  $A$  such that there is no fill-in.

# Sparse LU Factorisation

## Outlook

Our aim for the remainder of this lecture is to use the fill path theorem to analyse the costs of computing the LU factorisation of  $\Delta_n^{(1)}$  and  $\Delta_n^{(2)}$ .

The general outline of this analysis is as follows.

- ▶ Determine the graph  $G(\Delta_n^{(d)})$ .
- ▶ Determine the number of fill-path-neighbours  $n_{\text{fpn}}$  per vertex.

The memory and runtime requirements are then given by

$$\text{memory} = \mathcal{O}(N n_{\text{fpn}}) \quad \text{and} \quad \text{runtime} = \mathcal{O}(N n_{\text{fpn}}^2)$$

where  $N = n^d$  denotes the number of rows/columns.

*Proof.* Memory requirements are obvious.

Runtime: In each column, we have  $\mathcal{O}(n_{\text{fpn}})$  entries to eliminate. Each elimination operates on rows with  $\mathcal{O}(n_{\text{fpn}})$  nonzeros and hence requires  $\mathcal{O}(n_{\text{fpn}})$  operations. Estimate follows by multiplying

(# columns)  $\times$  (# eliminations per column)  $\times$  (# operations per elimination).

# Sparse LU Factorisation

## LU factorisation of $\Delta_n^{(1)}$

$$G(\Delta_5^{(1)}) = \text{graph with 5 nodes (1 to 5) and edges (1,2), (2,1), (2,3), (3,2), (3,4), (4,3), (4,5), (5,4)}$$

Observation:  $G(\Delta_n^{(1)})$  does not contain any fill-paths of length  $> 1$ ;

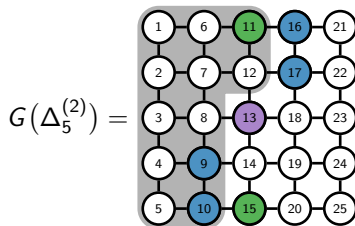
Conclusion:  $n_{\text{fpn}} = \mathcal{O}(1)$ , memory =  $\mathcal{O}(N)$ , runtime =  $\mathcal{O}(N)$ .

## Discussion

- ▶  $\mathcal{O}(N)$  scaling is much better than  $\mathcal{O}(N^3)$  scaling for dense LU.
- ▶  $\mathcal{O}(N)$  scaling is optimal: no algorithm operating on  $\mathcal{O}(N)$  data can do better than  $\mathcal{O}(N)$  memory and runtime.

# Sparse LU Factorisation

LU factorisation of  $\Delta_n^{(2)}$



Marked in gray are all vertices numbered less than 13.

Highlighted in blue and green are the vertices which are fill-path-connected to vertex 13.

Row 13 in  $L + U$  has therefore the sparsity pattern

$$( \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad 13 \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad )$$

and the full sparsity pattern of  $L + U$  is as shown on the next slide.



## Sparse LU Factorisation

### LU factorisation of $\Delta_n^{(2)}$ (continued)

$L + U =$

A 25x25 matrix  $L + U$  is shown, represented as a dot plot. The matrix is lower triangular, with non-zero entries (colored dots) on and below the main diagonal. The dots are colored black, blue, green, and light blue. The main diagonal is black. The first column has 25 black dots. The second column has 24 black dots and 1 blue dot at row 2. The third column has 23 black dots, 2 blue dots at rows 2 and 3, and 1 green dot at row 4. This pattern continues, with the number of colored dots per column increasing until column 12 (which has 12 blue dots, 11 green dots, and 1 light blue dot at row 12) and then decreasing. The last column has 25 black dots. The matrix is enclosed in large parentheses.

Conclusion:

$$\begin{aligned} n_{\text{fpn}} &= \mathcal{O}(n) = \mathcal{O}(N^{1/2}), \\ \text{memory} &= \mathcal{O}(Nn) = \mathcal{O}(N^{3/2}), \\ \text{runtime} &= \mathcal{O}(Nn^2) = \mathcal{O}(N^2). \end{aligned}$$

# Sparse LU Factorisation

## Discussion

Sparse LU applied to  $\Delta_n^{(2)}$  performs better than dense LU ( $\mathcal{O}(N^2)$  vs.  $\mathcal{O}(N^3)$  runtime), but it is still quite expensive.

Corollary of fill path theorem: fill-in depends on order of vertices.

It turns out that we can compute the LU factorisation of  $\Delta_n^{(2)}$  more economically if we reorder the vertices.

The next slide introduces the nested dissection order. This order is asymptotically optimal, i.e. no vertex order leads to less fill-in in the big- $\mathcal{O}$  sense than the nested dissection order.

# Sparse LU Factorisation

---

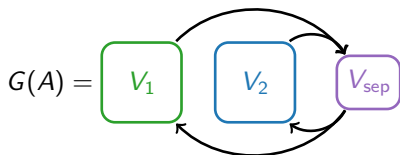
**Algorithm 1** Nested dissection order

---

- 1: Partition the vertices into three sets  $V_1$ ,  $V_2$ ,  $V_{\text{sep}}$  such that there are no edges between  $V_1$  and  $V_2$  (subscript  $\text{sep}$  stands for *separator*).
  - 2: Arrange the vertices in the order  $V_1$ ,  $V_2$ ,  $V_{\text{sep}}$ , where  $V_1$  and  $V_2$  are ordered recursively according to the nested dissection algorithm and  $V_{\text{sep}}$  is ordered arbitrarily.
- 

After reordering:

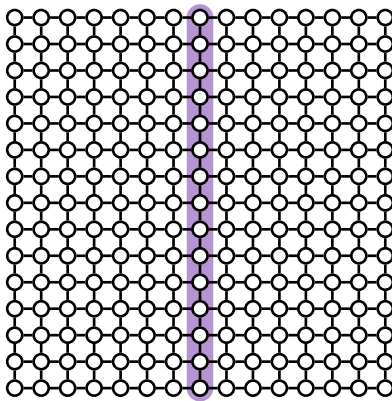
$$A = \begin{pmatrix} \text{green} & & & & & \\ & & & & & \\ & & \text{blue} & & & \\ & & & & & \\ & & & & & \\ & & & & & \text{purple} \end{pmatrix}$$

**Observation**

With this order, we have  $L[V_2, V_1] = U[V_1, V_2] = 0$  because any path from  $V_1$  to  $V_2$  must pass through  $V_{\text{sep}}$  and is therefore not a fill-path.

# Sparse LU Factorisation

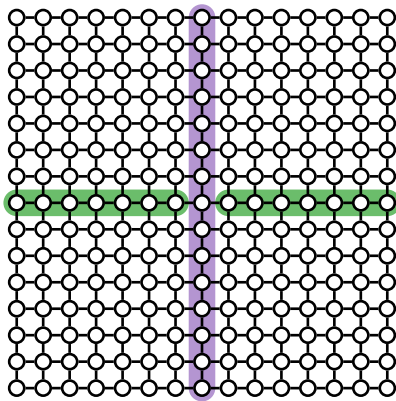
## Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

# Sparse LU Factorisation

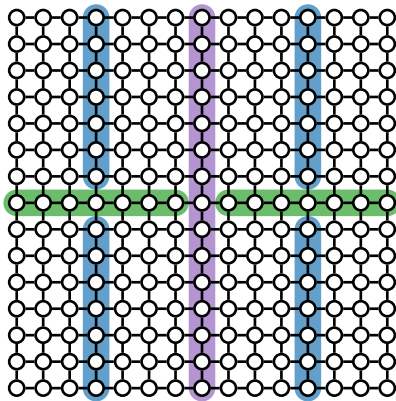
## Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

# Sparse LU Factorisation

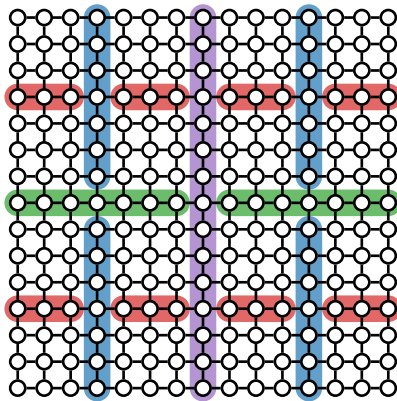
## Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

# Sparse LU Factorisation

## Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

# Sparse LU Factorisation

## LU factorisation of $\Delta_n^{(2)}$ with nested dissection

It can be shown that the requirements of LU factorisation of  $\Delta_n^{(d)}$  with nested dissection ordering are as follows.

	Runtime	Memory
$d = 1$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
$d = 2$	$\mathcal{O}(N^{3/2})$	$\mathcal{O}(N \log(N))$
$d = 3$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^{4/3})$

A rigorous proof of this result is beyond our scope.

Instead, I will present an argument why the runtimes cannot be lower.

Observation: Diagonal blocks associated with the  $V_{\text{sep}}$  are dense.

Factorising these blocks hence requires  $\mathcal{O}(|V_{\text{sep}}|^3)$  operations.

In 2D: largest  $V_{\text{sep}}$  satisfies  $|V_{\text{sep}}| = \mathcal{O}(n) = \mathcal{O}(N^{1/2})$ .

In 3D: largest  $V_{\text{sep}}$  satisfies  $|V_{\text{sep}}| = \mathcal{O}(n^2) = \mathcal{O}(N^{2/3})$ .

Recall:  $n = \#$  mesh points per dimension,  $N = n^d =$  total  $\#$  mesh points.

Taking these  $|V_{\text{sep}}|$  to the third power yields the numbers in the table.



# Sparse LU Factorisation

## Discussion

LU factorisation is popular for solving linear systems because it is very easy to use: all we have to do is provide  $A$  and  $b$  and hit enter.

Unfortunately, this lecture showed that the simplicity of LU comes at the price of superlinear memory and runtime requirements for partial differential equations in dimensions  $d > 1$ .

In the next few weeks, we will look at algorithms which are more complicated to use but which perform better than LU when used correctly.

# Sparse LU Factorisation

## Discussion (continued)

Some further practical complications (not examinable):

- ▶ Computing a nested dissection order can be quite expensive.  
Solution: use Approximate Minimum Degree (AMD) order instead. This order sorts the vertices in increasing order of degree (number of neighbours). Because even computing degrees is expensive, we use approximate degrees instead.
- ▶ LU factorisation requires pivoting (interchanging of rows) to be numerically stable. This may require striking a balance between minimising fill-in and maintaining stability.  
Solution: some matrices do not require pivoting, namely diagonally dominant and symmetric positive definite ones.

Julia's `\` automatically takes care of these complications.

It internally uses the SuiteSparse package which is also used by Matlab.

# Sparse LU Factorisation

## Summary

$$\begin{aligned} A^p[i, j] \neq 0 &\iff \exists \text{ path } j \rightarrow i \text{ of length } p \\ A^{-1}[i, j] \neq 0 &\iff \exists \text{ path } j \rightarrow i \\ (L + U)[i, j] \neq 0 &\iff \exists \text{ fill path } j \rightarrow i \end{aligned}$$

Cost of sparse LU factorisation for partial differential equations:

	Runtime	Memory
$d = 1$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
$d = 2$	$\mathcal{O}(N^{3/2})$	$\mathcal{O}(N \log(N))$
$d = 3$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^{4/3})$