Matías R. Bender

Inria Saclay
-
CMAP, École Polytechnique,
Institut Polytechnique de Paris

Concours - Chargé d'Enseignement
Spécialité Informatique

🇫🇷 **Since 2023 - Chargé de recherche (CN)**
Inria Saclay - CMAP, École Polytechnique
Team Tropical, lead by Stéphane Gaubert

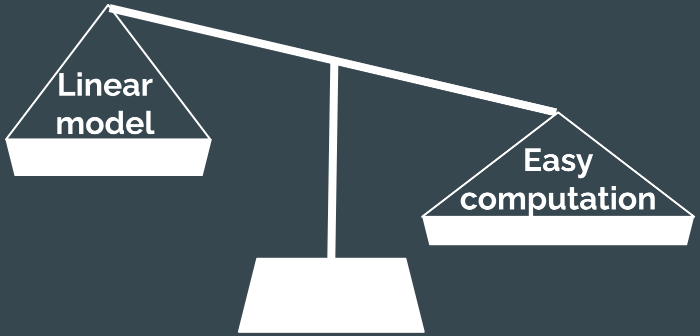
🇩🇪 **2019 - 2022 - Postdoctoral researcher**
Institut für Mathematik - Technische Universität Berlin
Mentored by Peter Bürgisser

🇫🇷 **2016 - 2019 - Ph.D. in Informatics**
Laboratoire d'informatique LIP6, Sorbonne Université
Algorithms for sparse polynomial systems:
Gröbner basis and resultants
Supervised by Jean-Charles Faugère & Elias Tsigaridas

🇦🇷 **2015 - M.Sc. in Computer Science**
CS department - FCEN - Universidad de Buenos Aires

🇪🇸 2014 - Exchange - Univ. Autónoma de Madrid

Slides available at http://mbender.github.io/etc/slidesDIX.pdf

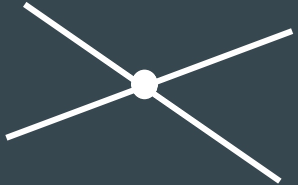


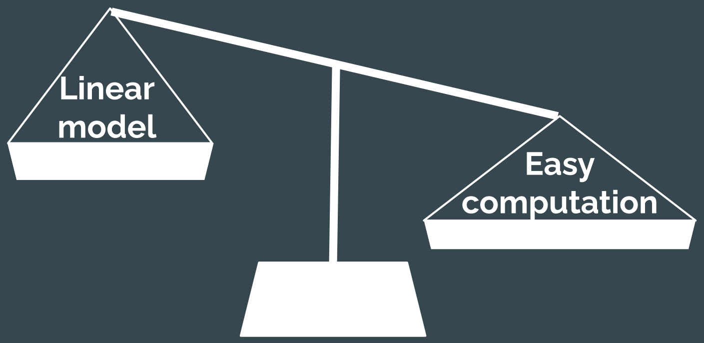
Linear model

Easy computation

Linear algebra

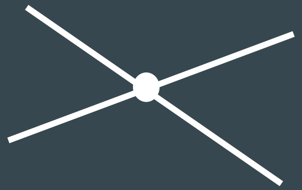
$$\left\{ \begin{array}{c} 3 \cdot x + 2 \cdot y = 5 \\ 4 \cdot x - 3 \cdot y = 2 \end{array} \right\}$$

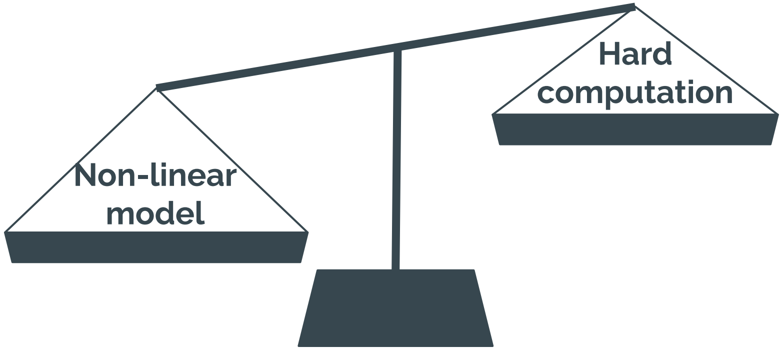




Linear algebra

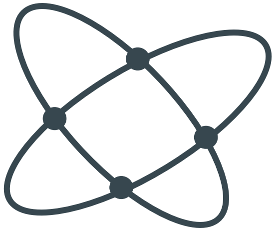
$$\begin{Bmatrix} 3 \cdot x + 2 \cdot y = 5 \\ 4 \cdot x - 3 \cdot y = 2 \end{Bmatrix}$$

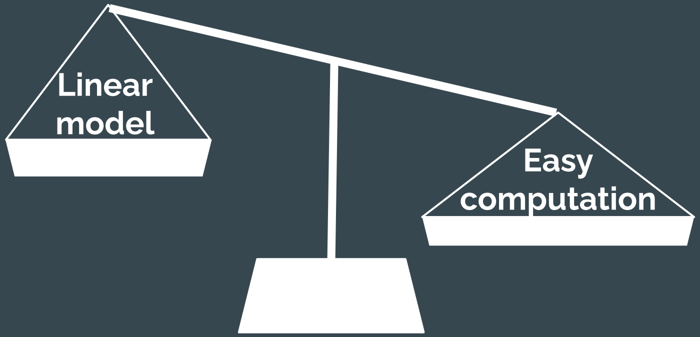




Non-linear algebra

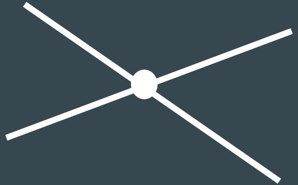
$$\begin{Bmatrix} 3 \cdot x^2 + 4 \cdot x \cdot y + 2 \cdot y^2 + x + 2 \cdot y = 5 \\ 6 \cdot x^2 + 2 \cdot x \cdot y + 3 \cdot y^2 - x + 3 \cdot y = 9 \end{Bmatrix}$$

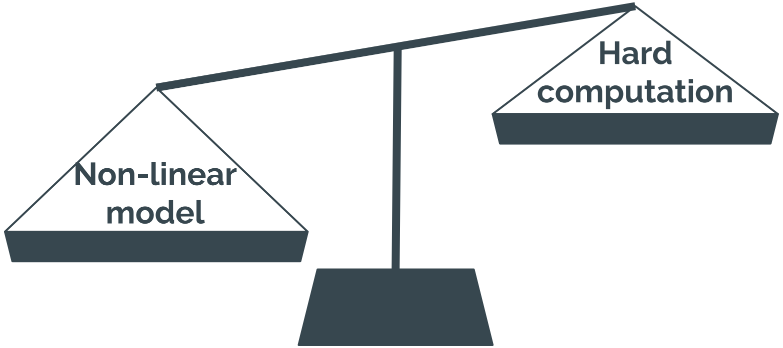




Linear algebra

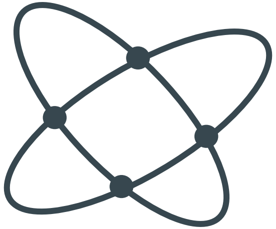
$$\begin{cases} 3 \cdot x + 2 \cdot y = 5 \\ 4 \cdot x - 3 \cdot y = 2 \end{cases}$$





Non-linear algebra

$$\begin{cases} 3 \cdot x^2 + 4 \cdot x \cdot y + 2 \cdot y^2 + x + 2 \cdot y = 5 \\ 6 \cdot x^2 + 2 \cdot x \cdot y + 3 \cdot y^2 - x + 3 \cdot y = 9 \end{cases}$$

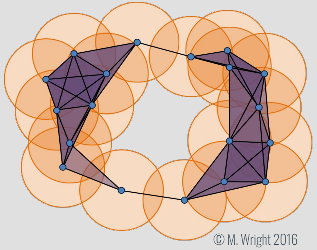


Pragmatic approach: Use non-linear model, if we can compute with it

My objective: Study trade-off for non-linear systems in practice

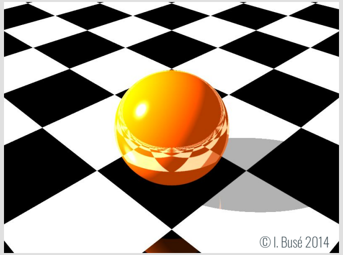
Polynomials systems and applications

Cryptography



© M. Wright 2016

Topological data analysis



© I. Busé 2014

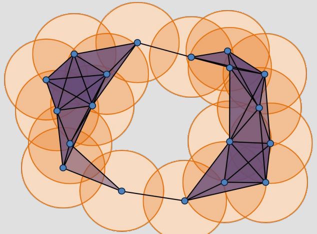
Geometric modeling

© Martinec & Pajdla 2007

Computer vision

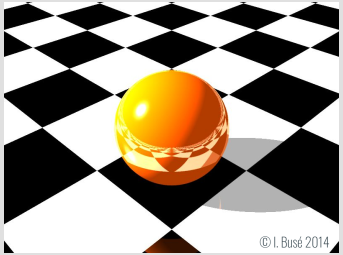
Polynomials systems and applications

Cryptography



© M. Wright 2016

Topological data analysis



© I. Busé 2014

Geometric modeling

© Martinec & Pajdla 2007

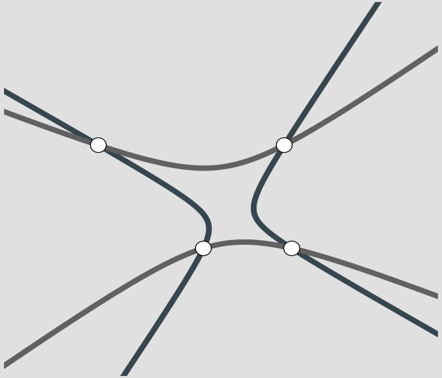
Computer vision

In practice, polynomials have structure

We exploit structure to

- Speed up computations
- Improve quality of approximations
- Study degenerate situations

Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

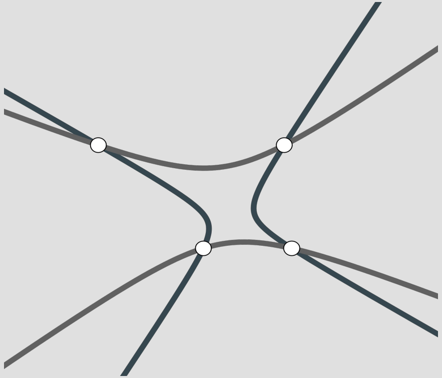
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$

Generalized eigenvalue problem

(2 solutions)

$$\left(\begin{pmatrix} 20 & 6 \\ -12 & 3 \end{pmatrix} - \lambda \begin{pmatrix} -1 & -4 \\ 4 & 2 \end{pmatrix} \right) \begin{pmatrix} 1 \\ x \end{pmatrix} = 0$$

Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

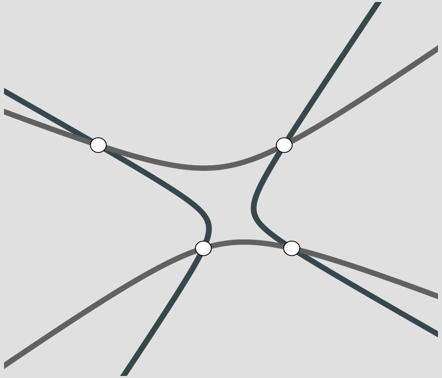
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$

Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$

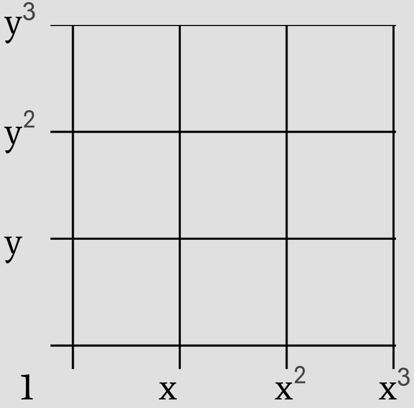
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$

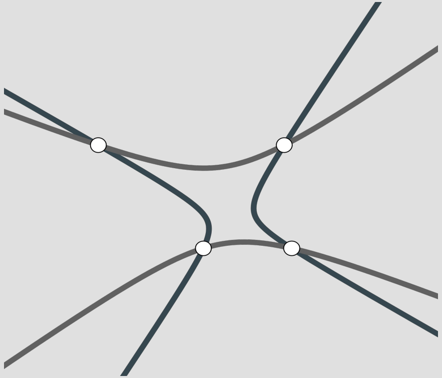


Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$

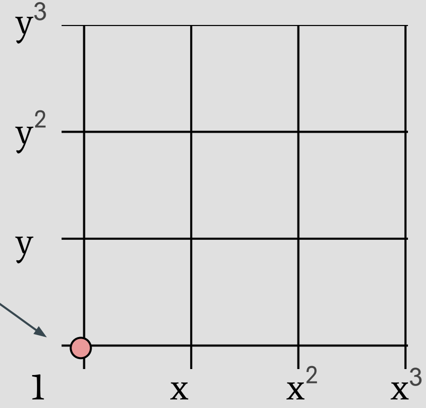
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\left\{\begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array}\right\}$$

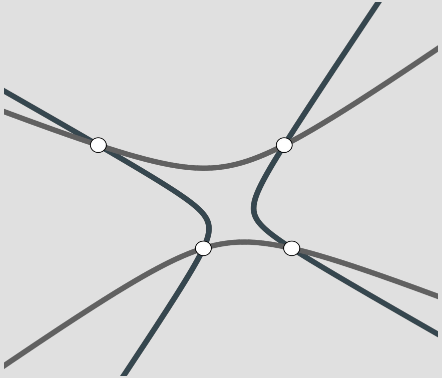


Generalized eigenvalue problem

(2 solutions)

$$\left\{\begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array}\right\}$$

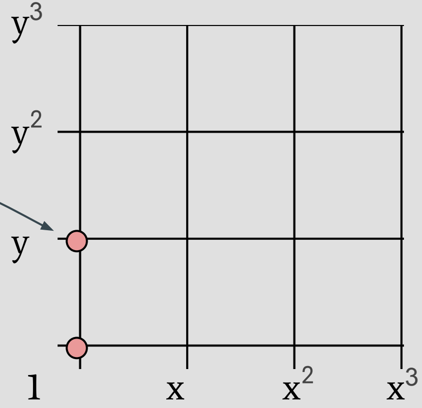
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$

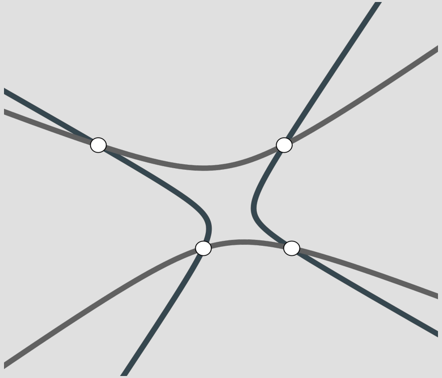


Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$

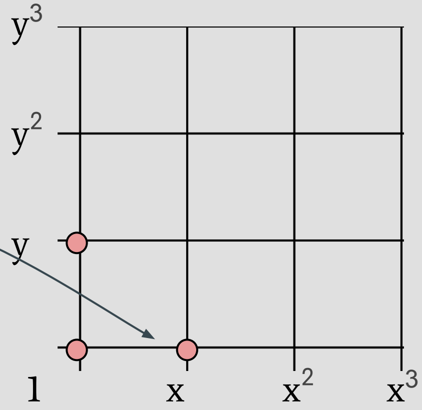
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$

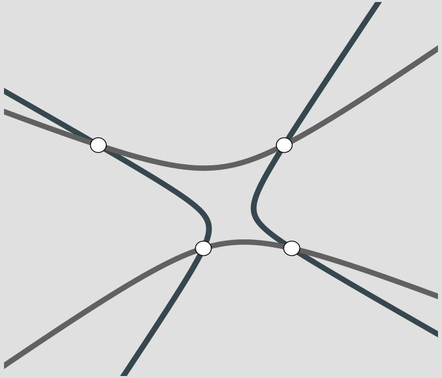


Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$

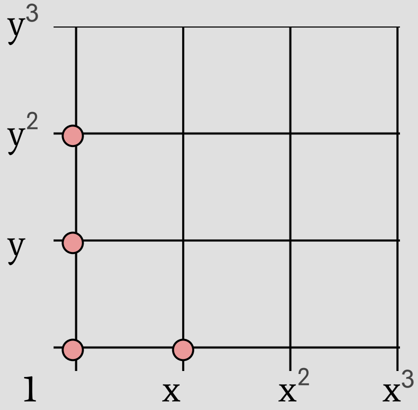
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$

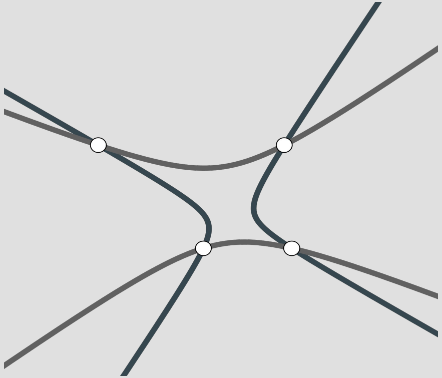


Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$

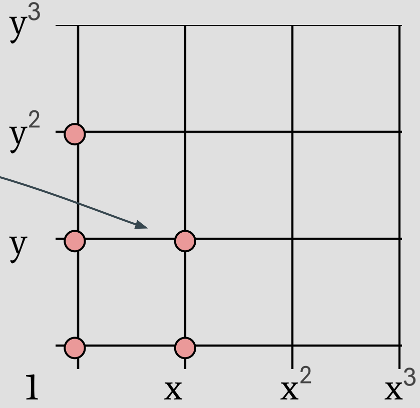
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$

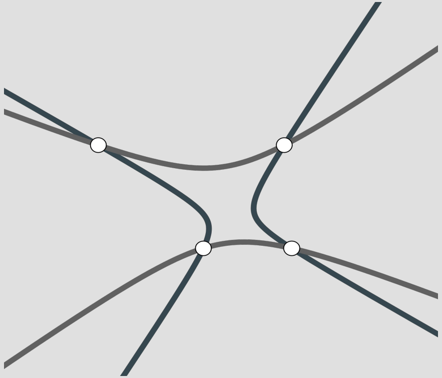


Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$

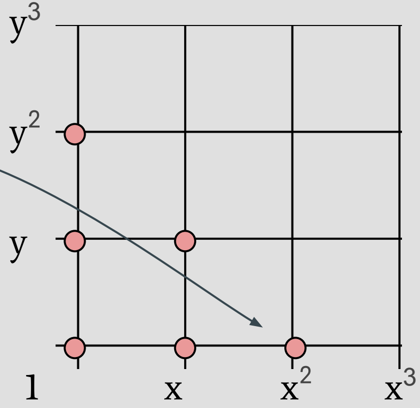
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$

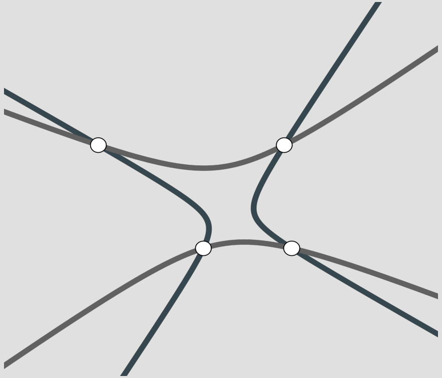


Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$

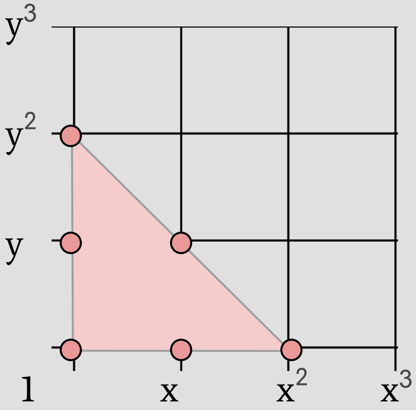
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$

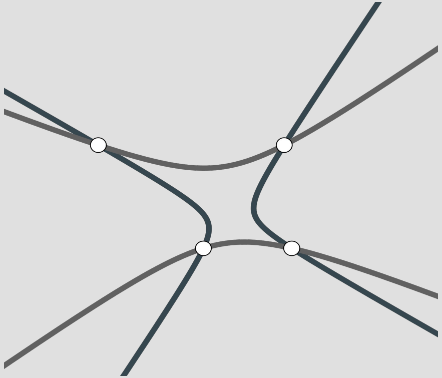


Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$

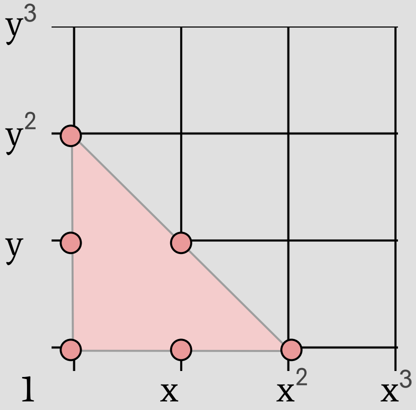
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

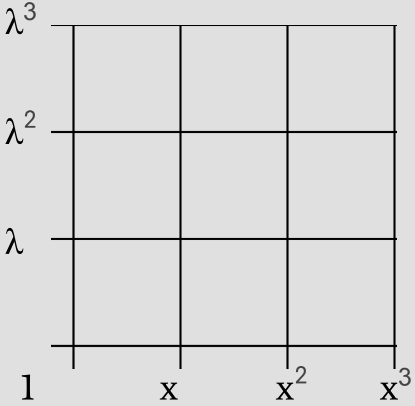
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$



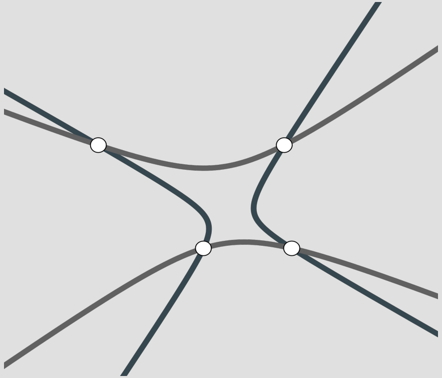
Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$



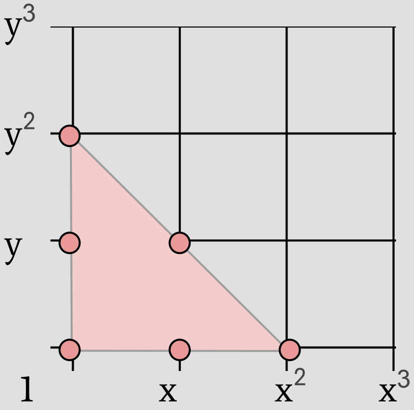
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

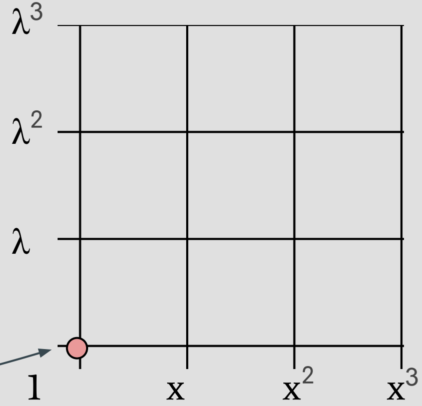
$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$



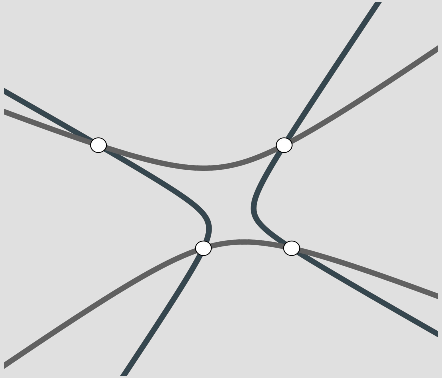
Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$



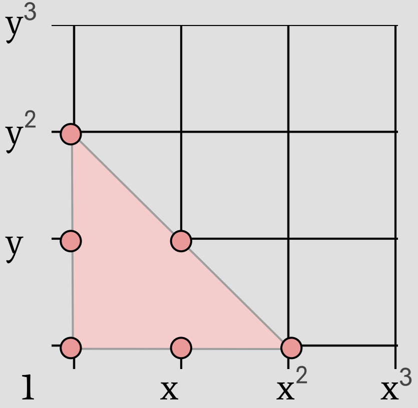
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

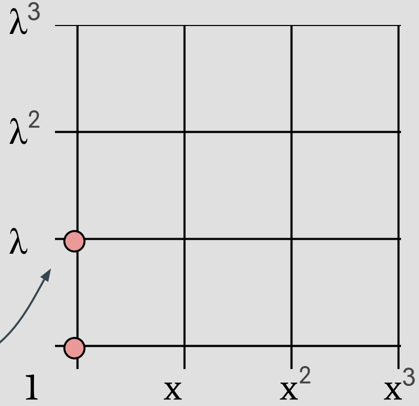
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$



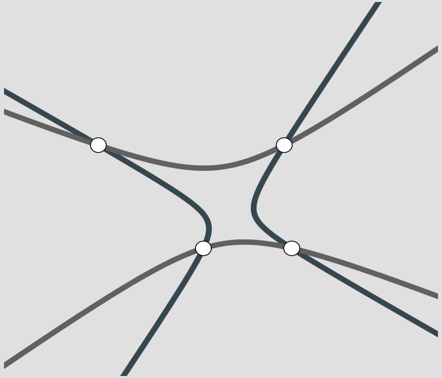
Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$



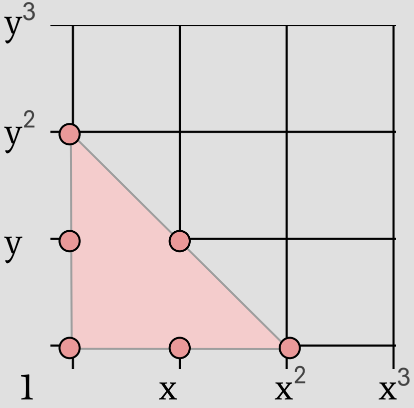
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

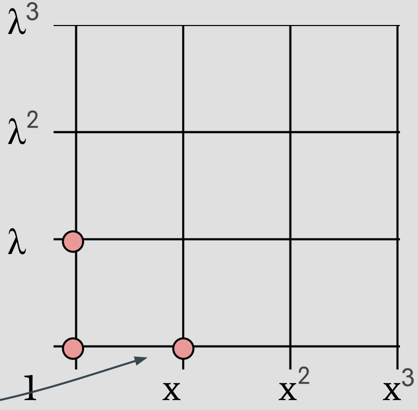
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$



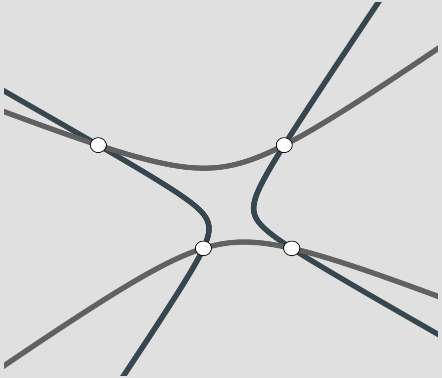
Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$



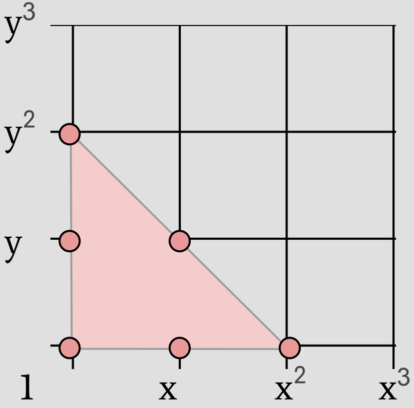
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

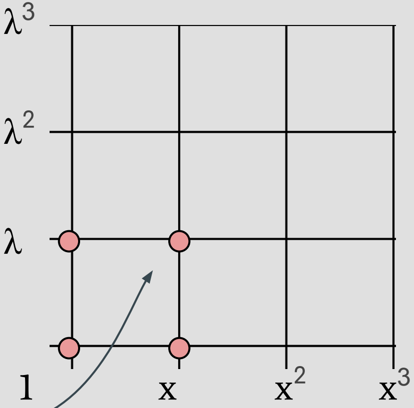
$$\begin{cases} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{cases}$$



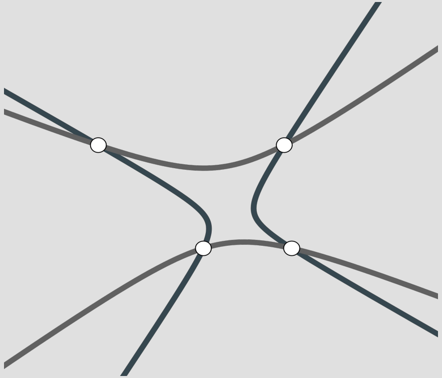
Generalized eigenvalue problem

(2 solutions)

$$\begin{cases} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{cases}$$



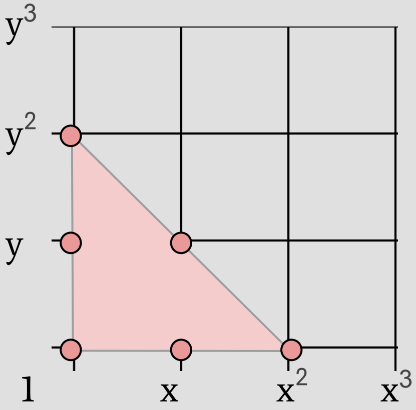
Exploiting sparsity - Polynomial with a few monomials



Generic system

(4 solutions)

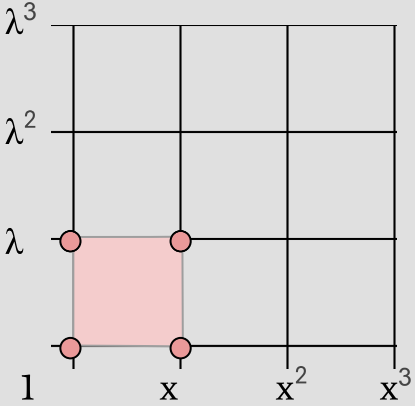
$$\left\{ \begin{array}{l} x^2 + x \cdot y - y^2 - x - 2 \cdot y - 1 = 0 \\ -x^2 - x \cdot y + 3 \cdot y^2 + x + y - 7 = 0 \end{array} \right\}$$



Generalized eigenvalue problem

(2 solutions)

$$\left\{ \begin{array}{l} 0 \cdot x^2 - 4 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 6 \cdot x - \lambda + 20 = 0 \\ 0 \cdot x^2 + 2 \cdot x \cdot \lambda - 0 \cdot \lambda^2 + 3 \cdot x + 4 \cdot \lambda - 12 = 0 \end{array} \right\}$$



Contributions

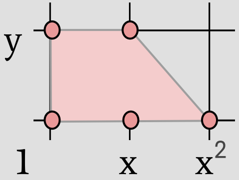
Solving sparse systems

Applications

Contributions

Solving sparse systems

$$\begin{cases} 0 = x^2 + xy + x + y - 1 \\ 0 = 3\,x^2 + 5\,xy + 2\,x + 4\,y - 2 \end{cases}$$



Applications

Contributions

PhD

Symbolic methods (Gröbner bases)

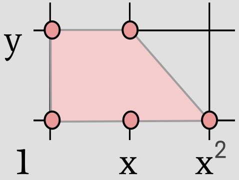
Efficient algorithms (no reds. to zero) + complexity bounds

Invited tutorial,
ISSAC 2022

$$\begin{cases} x = \frac{2}{3}\,y^2 - 3\,y + \frac{1}{3} \\ 0 = 2\,y^3 - 10\,y^2 + 7\,y + 1 \end{cases}$$

Solving sparse systems

$$\begin{cases} 0 = x^2 + xy + x + y - 1 \\ 0 = 3\,x^2 + 5\,xy + 2\,x + 4\,y - 2 \end{cases}$$

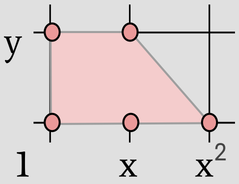


Applications

Contributions

Solving sparse systems

$$\begin{cases} 0 = x^2 + xy + x + y - 1 \\ 0 = 3\,x^2 + 5\,xy + 2\,x + 4\,y - 2 \end{cases}$$



PhD

Postdoc

Symbolic methods (Gröbner bases)

Efficient algorithms (no reds. to zero) + complexity bounds

Invited tutorial,
ISSAC 2022

$$\begin{cases} x = \frac{2}{3}\,y^2 - 3\,y + \frac{1}{3} \\ 0 = 2\,y^3 - 10\,y^2 + 7\,y + 1 \end{cases}$$

Numerical methods

Robust algorithms (accurate even when solutions near infinity)

Software: EigenvalueSolver.jl (Julia)

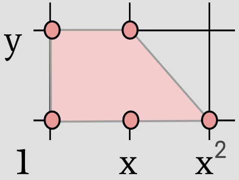
$$(x,y) \in \left\{ \begin{array}{l} (-2.0, 1.0), (0.707106, -0.12132) \\ (-0.707106, 4.12132) \end{array} \right\}$$

Applications

Contributions

Solving sparse systems

$$\begin{cases} 0 = x^2 + xy + x + y - 1 \\ 0 = 3\,x^2 + 5\,xy + 2\,x + 4\,y - 2 \end{cases}$$



PhD

Postdoc

Symbolic methods (Gröbner bases)

Efficient algorithms (no reds. to zero) + complexity bounds

Invited tutorial,
ISSAC 2022

$$\begin{cases} x = \frac{2}{3}\,y^2 - 3\,y + \frac{1}{3} \\ 0 = 2\,y^3 - 10\,y^2 + 7\,y + 1 \end{cases}$$

Numerical methods

Robust algorithms (accurate even when solutions near infinity)

Software: EigenvalueSolver.jl (Julia)

$$(x, y) \in \left\{ \begin{array}{l} (-2.0, 1.0), (0.707106, -0.12132) \\ (-0.707106, 4.12132) \end{array} \right\}$$

Applications

Multi-parameter eigenvalue problem

Software: sylvesterMEP (Matlab)

Symmetric tensor decomposition (binary forms)

Best student paper award
ISSAC 2016

Topological data analysis (TDA)

(multi-param. persistent homology)

Software: Muphasa (C++)

Computational biology

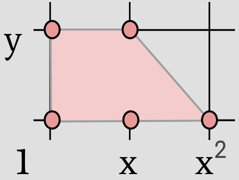
(classifiers on Boolean networks)

Software: Classifier-construction (Python)

Contributions

Solving sparse systems

$$\begin{cases} 0 = x^2 + xy + x + y - 1 \\ 0 = 3\,x^2 + 5\,xy + 2\,x + 4\,y - 2 \end{cases}$$



PhD

Postdoc

Symbolic methods (Gröbner bases)

Efficient algorithms (no reds. to zero) + complexity bounds

Invited tutorial,
ISSAC 2022

$$\begin{cases} x = \frac{2}{3}\,y^2 - 3\,y + \frac{1}{3} \\ 0 = 2\,y^3 - 10\,y^2 + 7\,y + 1 \end{cases}$$

Numerical methods

Robust algorithms (accurate even when solutions near infinity)

Software: EigenvalueSolver.jl (Julia)

$$(x, y) \in \left\{ \begin{array}{l} (-2.0, 1.0), (0.707106, -0.12132) \\ (-0.707106, 4.12132) \end{array} \right\}$$

Applications

Multi-parameter eigenvalue problem
Software: sylvesterMEP (Matlab)

Topological data analysis (TDA)
(multi-param. persistent homology)
Software: Muphasa (C++)

Computational biology
(classifiers on Boolean networks)
Software: Classifier-construction (Python)

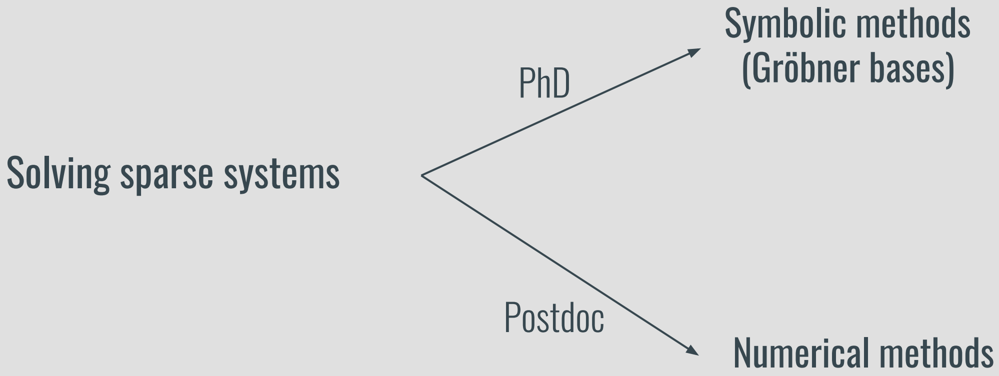
Symmetric tensor decomposition
(binary forms)

Best student paper award
ISSAC 2016

Quasi-optimal algorithm to decompose binary forms (improves Comon, Dür, Mourrain, Helmke, Sylvester)

$$-4704\,x^3 + 3576\,x^2\,y - 912\,x\,y^2 + 78\,y^3 = \frac{1}{6}\,(-26\,x + 7\,y)^3 - \frac{1}{6}\,(22\,x - 5\,y)^3$$

Research project



Solving sparse systems

PhD

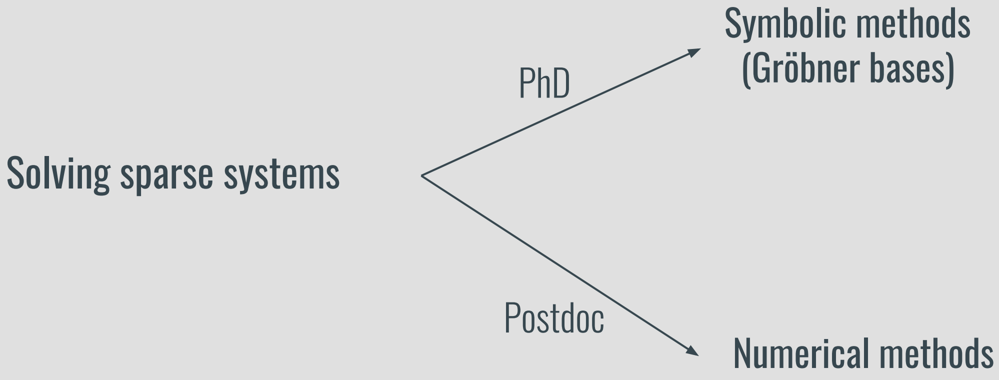
Symbolic methods
(Gröbner bases)

Postdoc

Numerical methods

Research project

Efficient implementations
C/C++ implementation, incorporate other linear algebra speed-ups (F4)



Solving sparse systems

PhD

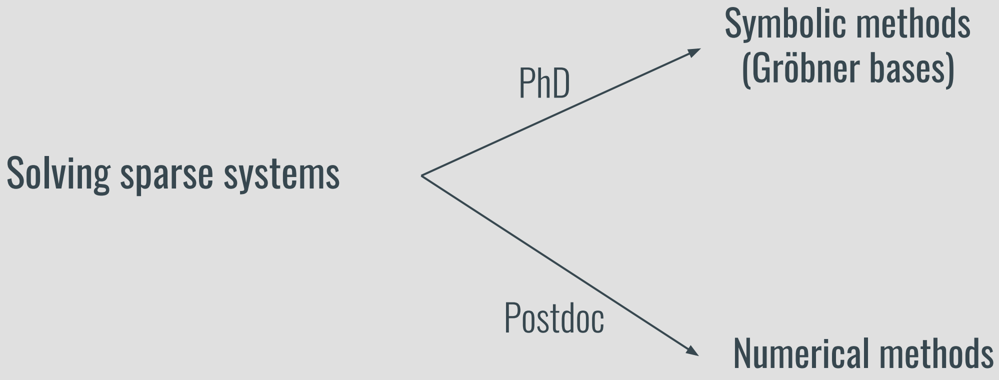
Symbolic methods (Gröbner bases)

Postdoc

Numerical methods

Research project

Efficient implementations
C/C++ implementation, incorporate other linear algebra speed-ups (F4)



Symbolic methods (Gröbner bases)

PhD

Solving sparse systems

Postdoc

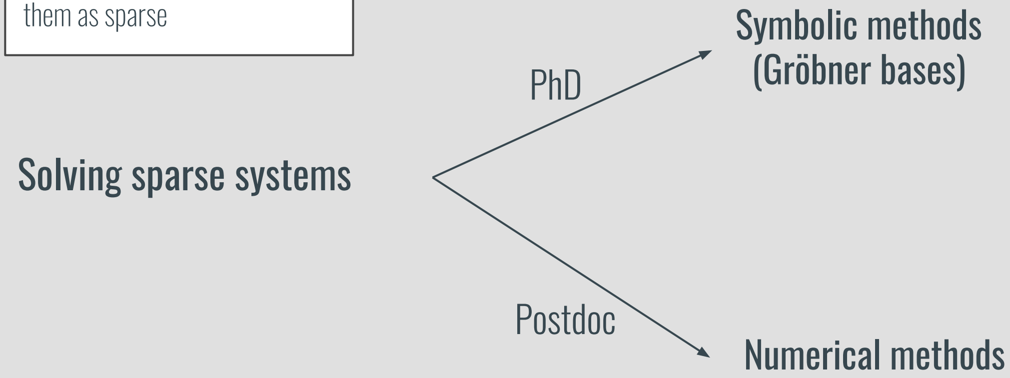
Numerical methods

Conditioning and precision analysis
Relate and improve the condition number of matrices to the conditioning of systems (missing in dense case also!)

Research project

Efficient implementations
C/C++ implementation, incorporate other linear algebra speed-ups (F4)

Structured systems
Use toric degenerations to treat them as sparse



Solving sparse systems

PhD

Symbolic methods
(Gröbner bases)

Postdoc

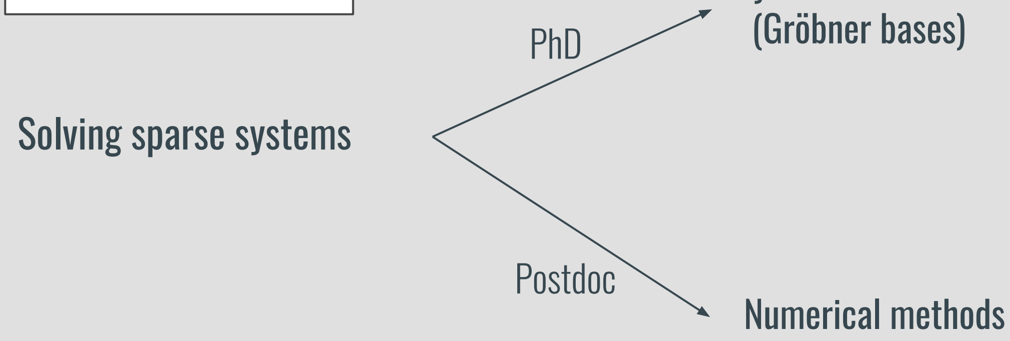
Numerical methods

Conditioning and precision analysis
Relate and improve the condition number of matrices to the conditioning of systems (missing in dense case also!)

Research project

Efficient implementations
C/C++ implementation, incorporate other linear algebra speed-ups (F4)

Structured systems
Use toric degenerations to treat them as sparse



Solving sparse systems

PhD

Symbolic methods (Gröbner bases)

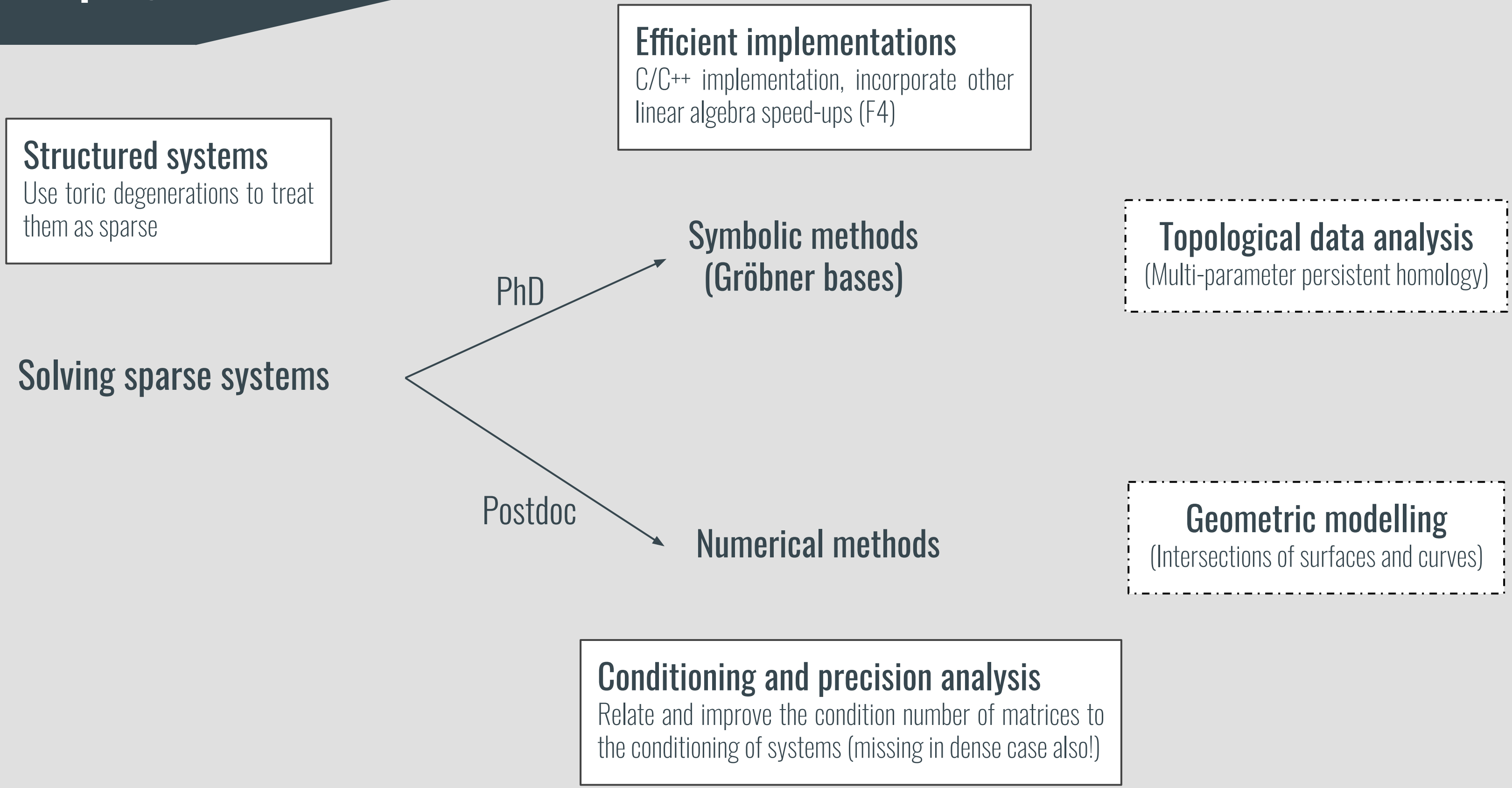
Postdoc

Numerical methods

Geometric modelling
(Intersections of surfaces and curves)

Conditioning and precision analysis
Relate and improve the condition number of matrices to the conditioning of systems (missing in dense case also!)

Research project



Efficient implementations
C/C++ implementation, incorporate other linear algebra speed-ups (F4)

Structured systems
Use toric degenerations to treat them as sparse

Symbolic methods (Gröbner bases)

Topological data analysis
(Multi-parameter persistent homology)

PhD

Solving sparse systems

Postdoc

Numerical methods

Geometric modelling
(Intersections of surfaces and curves)

Conditioning and precision analysis
Relate and improve the condition number of matrices to the conditioning of systems (missing in dense case also!)

Teaching experience

Teaching experience

University of Buenos Aires
Computer Science Department

3 Semester - Algorithms and Data Structures 2 (2nd year)

- Formal specification using abstract data types
- Classical data structures (e.g., AVL tree, heap, trie)
- Classical sorting algorithms (e.g., bubble-, merge-, bucket-sort)
- Divide and conquer techniques
- Complexity analysis

1 Semester - Algorithms and Data Structures 3 (3th year)

- Dynamic programming techniques
- Graph theory
- Complexity theory (P and NP)

Teaching experience

University of Buenos Aires
Computer Science Department

3 Semester - Algorithms and Data Structures 2 (2nd year)

- Formal specification using abstract data types
- Classical data structures (e.g., AVL tree, heap, trie)
- Classical sorting algorithms (e.g., bubble-, merge-, bucket-sort)
- Divide and conquer techniques
- Complexity analysis

1 Semester - Algorithms and Data Structures 3 (3th year)

- Dynamic programming techniques
- Graph theory
- Complexity theory (P and NP)

Technische Universität Berlin
Mathematics Department

Co-advised 1 bachelor 🏅 + 1 master thesis

Teaching experience

University of Buenos Aires
Computer Science Department

3 Semester - Algorithms and Data Structures 2 (2nd year)

- Formal specification using abstract data types
- Classical data structures (e.g., AVL tree, heap, trie)
- Classical sorting algorithms (e.g., bubble-, merge-, bucket-sort)
- Divide and conquer techniques
- Complexity analysis

1 Semester - Algorithms and Data Structures 3 (3th year)

- Dynamic programming techniques
- Graph theory
- Complexity theory (P and NP)

Technische Universität Berlin
Mathematics Department

Co-advised 1 bachelor 🏅 + 1 master thesis

Upcoming! Course “Computational Commutative Algebra” at CIMPA summer school in Oaxaca, Mexico

Teaching experience

University of Buenos Aires
Computer Science Department

3 Semester - Algorithms and Data Structures 2 (2nd year)

- Formal specification using abstract data types
- Classical data structures (e.g., AVL tree, heap, trie)
- Classical sorting algorithms (e.g., bubble-, merge-, bucket-sort)
- Divide and conquer techniques
- Complexity analysis

1 Semester - Algorithms and Data Structures 3 (3th year)

- Dynamic programming techniques
- Graph theory
- Complexity theory (P and NP)

Technische Universität Berlin
Mathematics Department

Co-advised 1 bachelor 🏅 + 1 master thesis

Upcoming! Course “Computational Commutative Algebra” at CIMPA summer school in Oaxaca, Mexico

High school
Intro to programming in Alice

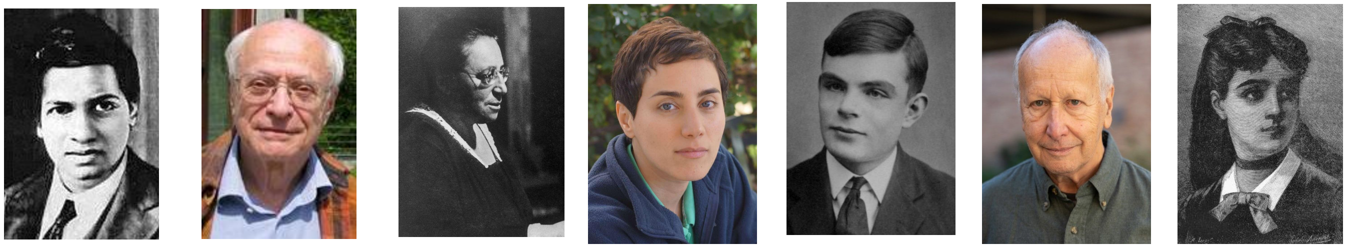
Primary school
Intro to programming in Scratch

Popular science

Computer literacy for Marginalized people

Exercise - Applications of sorting for 1st/2nd year bachelor students

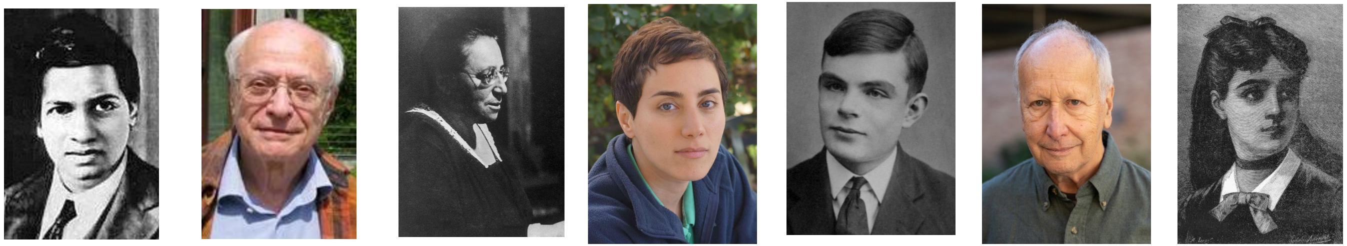
We ask **n** mathematicians their favorite integer number and we want to know the most popular ones.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Exercise - Applications of sorting for 1st/2nd year bachelor students

We ask **n** mathematicians their favorite integer number and we want to know the most popular ones.

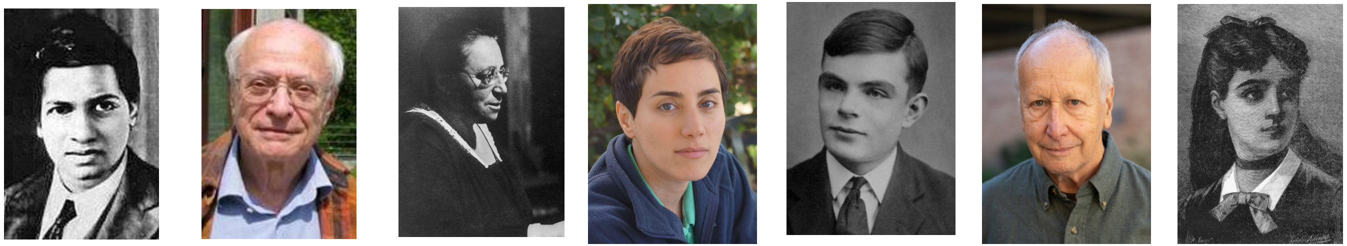


1729	3	2	4	11	3	11
0	1	2	3	4	5	6

A) Given preferences as array of n integers S, find one of the most popular ones!

Exercise - Applications of sorting for 1st/2nd year bachelor students

We ask **n** mathematicians their favorite integer number and we want to know the most popular ones.



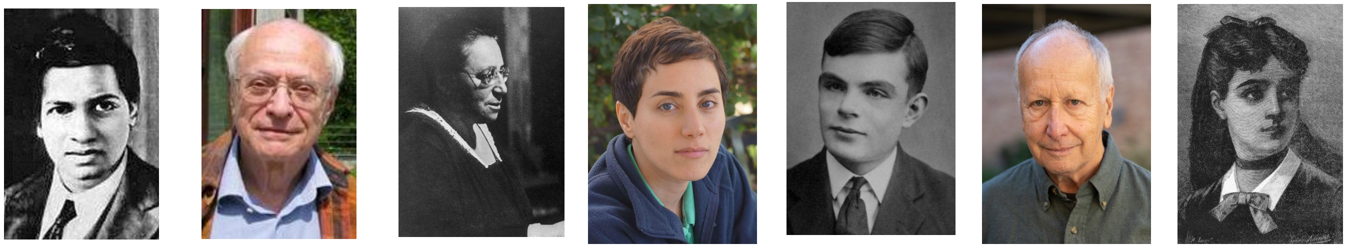
1729	3	2	4	11	3	11
0	1	2	3	4	5	6

A) Given preferences as array of n integers S, find one of the most popular ones!

B) If S is sorted, can you improve the complexity of your algorithm?

Exercise - Applications of sorting for 1st/2nd year bachelor students

We ask **n** mathematicians their favorite integer number and we want to know the most popular ones.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

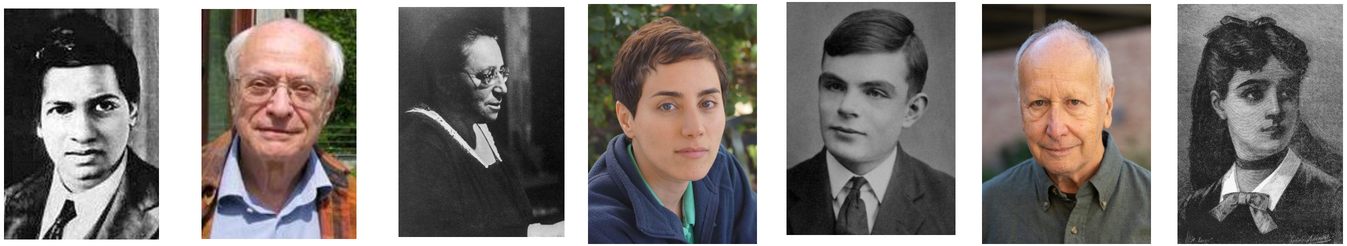
A) Given preferences as array of n integers S, find one of the most popular ones!

B) If S is sorted, can you improve the complexity of your algorithm?

C) Go back to A) and improve your algorithm.

Exercise - Applications of sorting for 1st/2nd year bachelor students

We ask **n** mathematicians their favorite integer number and we want to know the most popular ones.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

A) Given preferences as array of n integers S, find one of the most popular ones!

B) If S is sorted, can you improve the complexity of your algorithm?

C) Go back to A) and improve your algorithm.

D) If all the number in S are between 0 and 9. Can you improve your algorithm?

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.

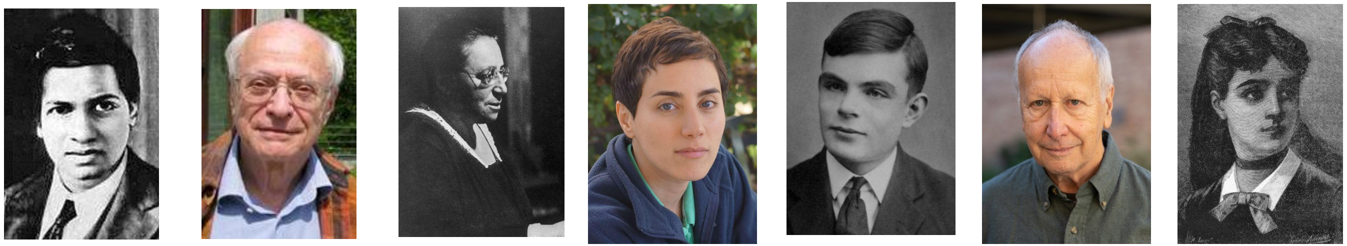
1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such that for all p in $\mathbb{Z}$, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume $n > 0$.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such that for all p in $\mathbb{Z}$, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

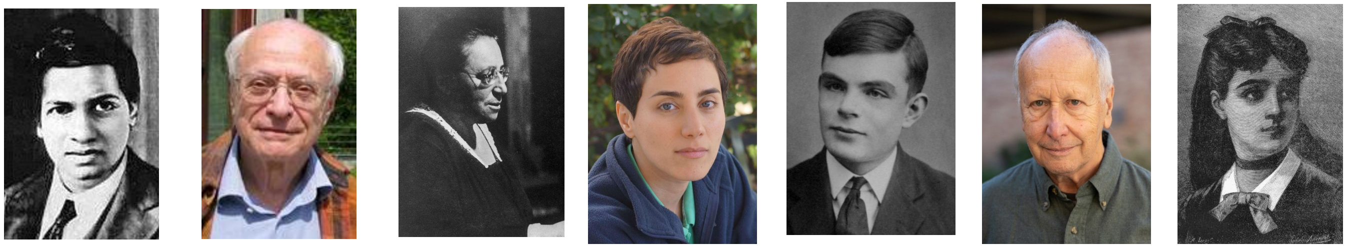
In this example, **3** and **11** are the only valid outputs.

Algorithm:

```
For each favorite number S[i]:
  ↳ Count how many people chose S[i];
Return the most choiced one;
```

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume $n > 0$.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such that for all p in ℤ, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

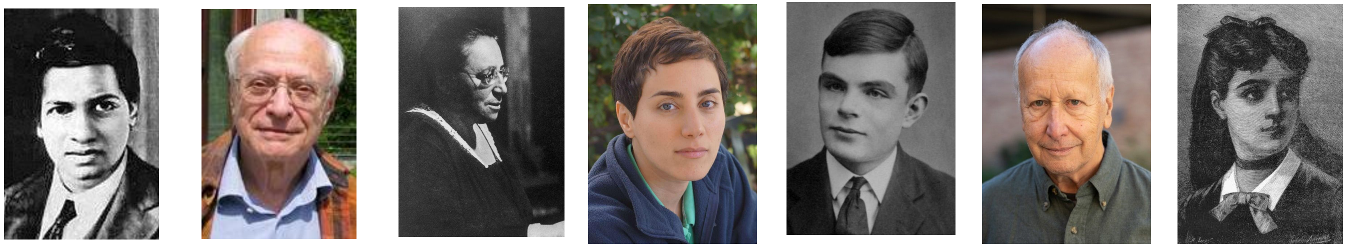
Algorithm:

```
For each favorite number S[i]:                .
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:                .
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such that for all p in $\mathbb{Z}$, $\#\{ i \in \{0 \ldots n-1\} : S[i] == v \} \geq \#\{ i \in \{0 \ldots n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

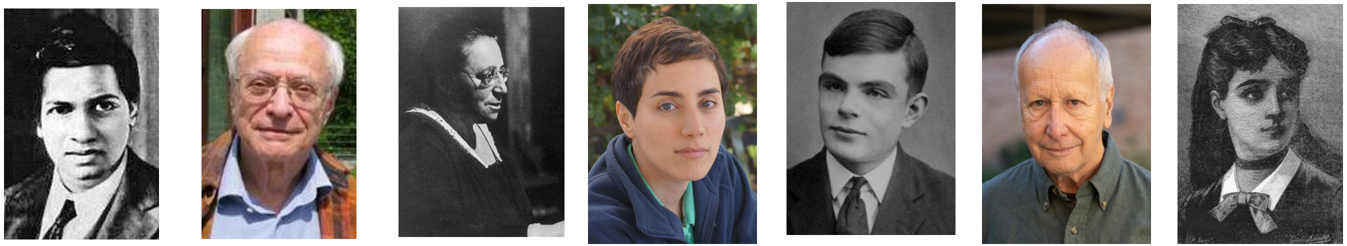
```
For each favorite number S[i]:                .
    ↳ Count how many people chose S[i];  ←
  Return the most choiced one;
```

```
For each element S[j]:                        .
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
  Return number of occurrences of S[i])
```

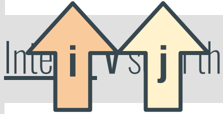
Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6



Output: Inte **v** s that for all p in ℤ, #{ i ∈ {0...n-1} : S[i] == v } ≥ #{ i ∈ {0...n-1} : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

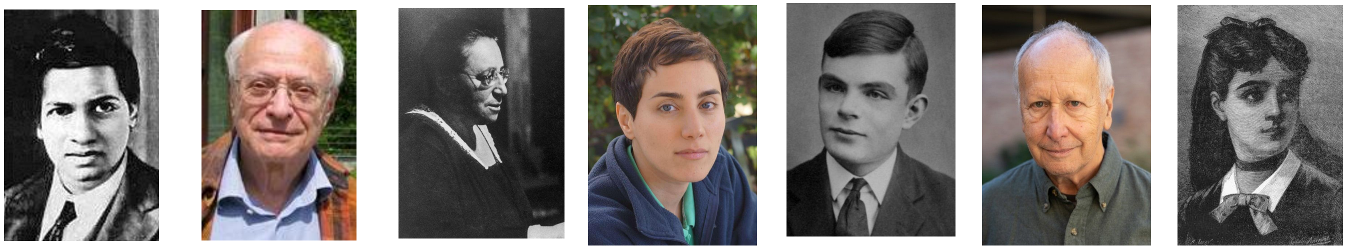
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Inte[illegible] v such that for a[illegible] ℤ, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

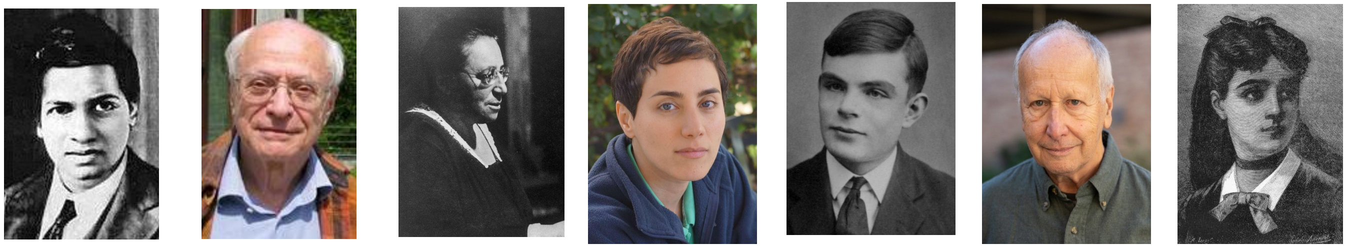
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n^2) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Inte **i** v such that for all p in $\mathbb{Z}$, $\#\{ i \in$ {0 **j** -1} $: S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

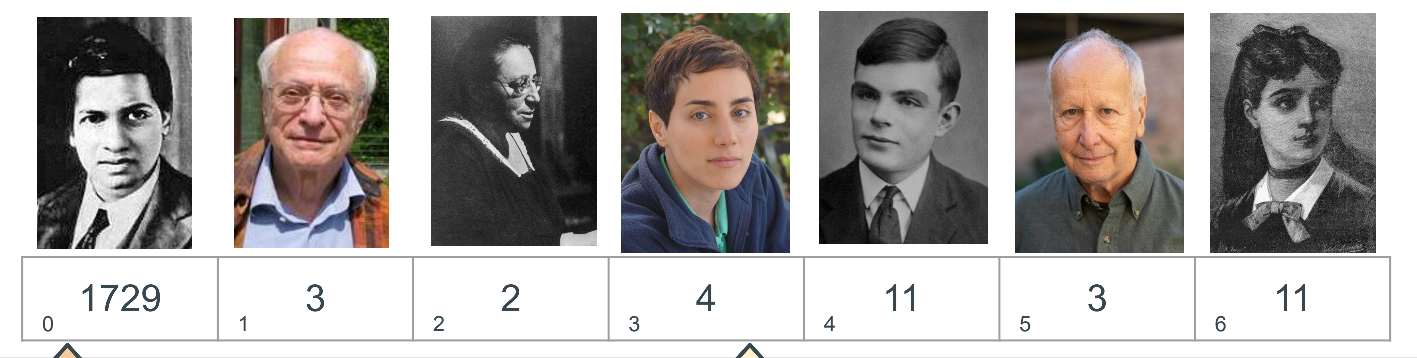
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S.

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Inte **i** v such that for all p in ℤ, #{ i ∈ {0...n-1} : S[i] == **j** ≥ #{ i ∈ {0...n-1} : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

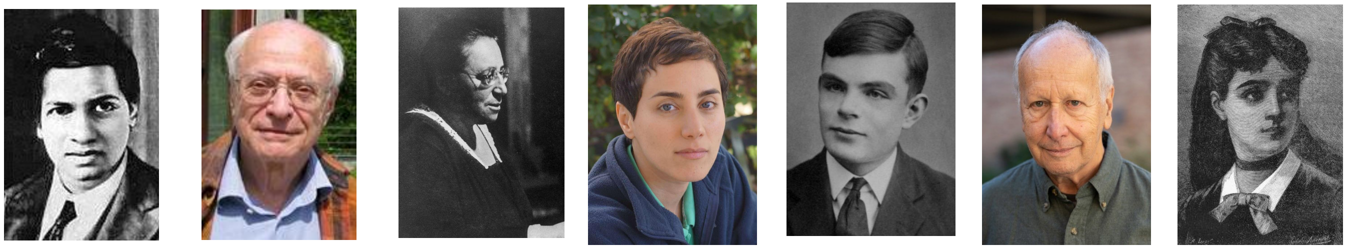
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n^2) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Inte **i** v such that for all p in ℤ, #{ i ∈ {0...n-1} : S[i] == v } ≥ #{ i ∈ {0... **j** } : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

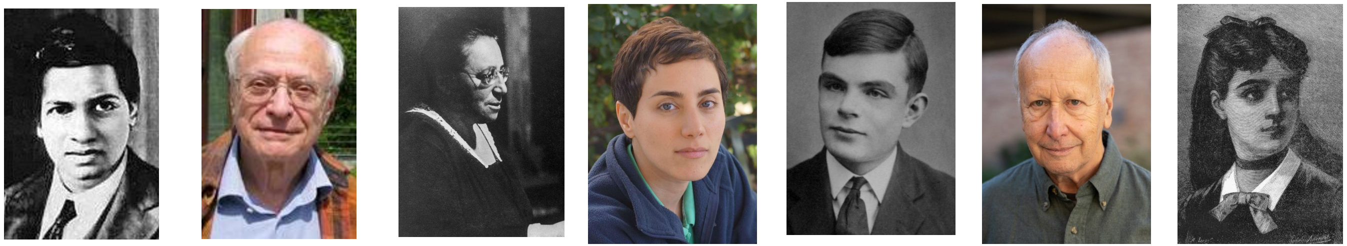
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
    Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
    Return number of occurrences of S[i])
```

Complexity: O(n²) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Inte **i** v such that for all p in $\mathbb{Z}$, $\#\{ i \in \{0 \dots n-1\} : S[i] == v \} \geq \#\{ i \in \{0 \dots n-1\} : S[i] ==$ p **j**

In this example, **3** and **11** are the only valid outputs.

Algorithm:

```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume $n > 0$.

0	1	2	3	4	5	6
1729	3	2	4	11	3	11

Output: Inte v such that for all p in $\mathbb{Z}$, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

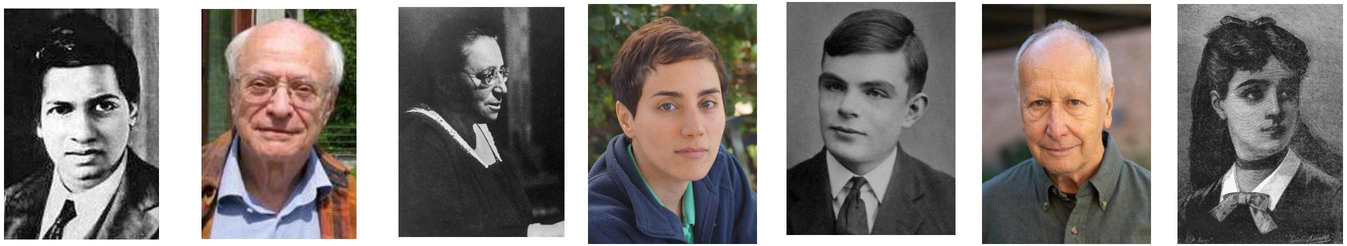
```
For each favorite number S[i]:                .
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:                        .
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** s **j** tha **i** all p in $\mathbb{Z}$, $\#\{ i \in \{0 \ldots n-1\} : S[i] == v \} \geq \#\{ i \in \{0 \ldots n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

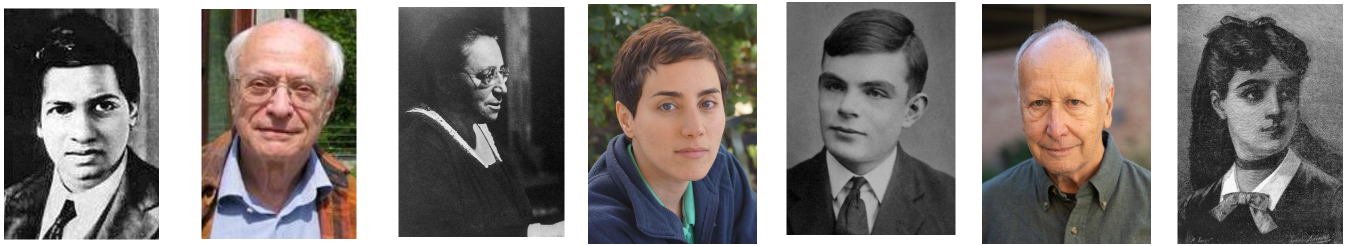
```
For each favorite number S[i]:                .
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:                        .
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

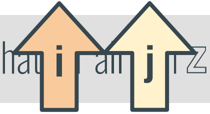
Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



0	1	2	3	4	5	6
1729	3	2	4	11	3	11



Output: Integer **v** such that for all p in ℤ, #{ i ∈ {0...n-1} : S[i] == v } ≥ #{ i ∈ {0...n-1} : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

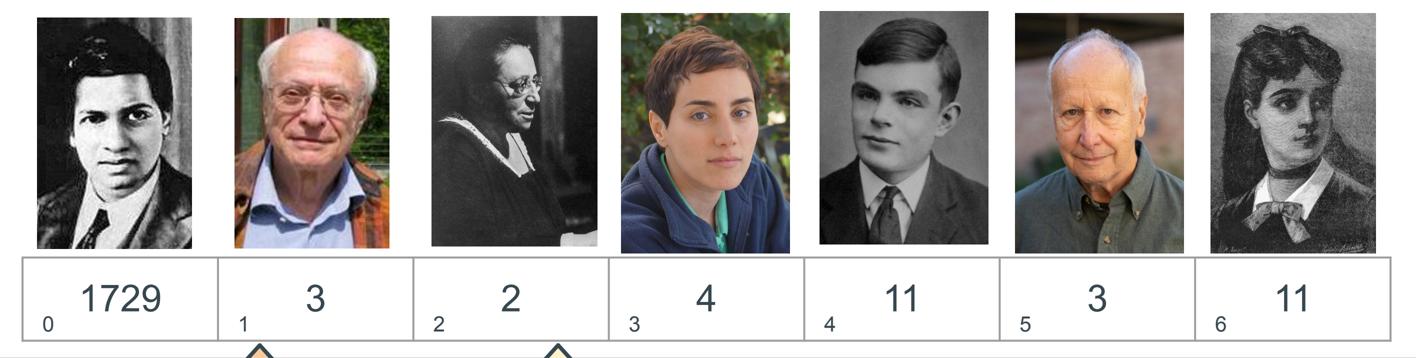
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n²) basic operations ————→ For each element in S, we visit every element in S.

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



0	1	2	3	4	5	6
1729	3	2	4	11	3	11

Output: Integer **v** such tha all p in $\mathbb{Z}$, #{ i ∈ {0 n-1} : S[i] == v } ≥ #{ i ∈ {0...n-1} : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

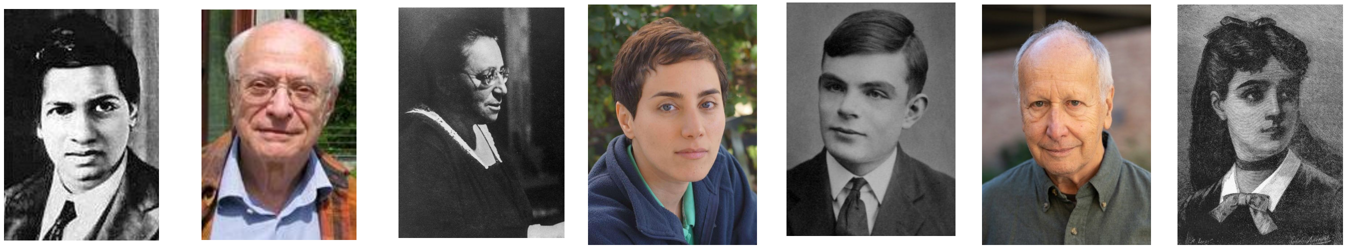
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n^2) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such tha **i** all p in ℤ, #{ i ∈ {0...n-1} : S[i] == **j** ≥ #{ i ∈ {0...n-1} : S[i] == p },

In this example, **3** and **11** are the only valid outputs.

Algorithm:

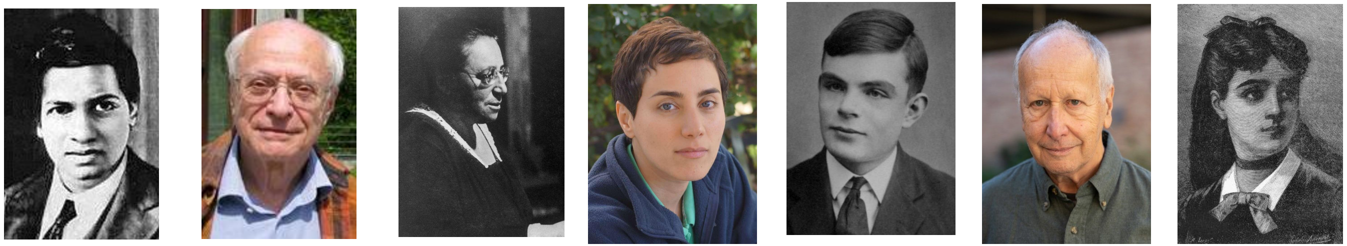
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n²) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



0	1	2	3	4	5	6
1729	3	2	4	11	3	11

Output: Integer **v** such tha **i** all p in ℤ, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...$ **j** $\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

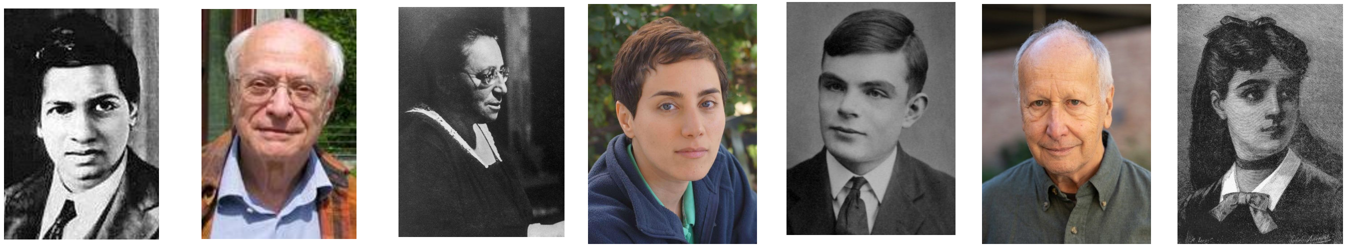
```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: O(n^2) basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.



1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such tha **i** all p in $\mathbb{Z}$, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p$ **j**

In this example, **3** and **11** are the only valid outputs.

Algorithm:

```
For each favorite number S[i]:
    ↳ Count how many people chose S[i];
Return the most choiced one;
```

```
For each element S[j]:
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part A, understanding the statement

Input: An array S of n Integers such that S[i] is the favorite number of the i-th mathematician; Assume n > 0.

1729	3	2	4	11	3	11
0	1	2	3	4	5	6

Output: Integer **v** such tha all p in $\mathbb{Z}$, $\#\{ i \in \{0...n-1\} : S[i] == v \} \geq \#\{ i \in \{0...n-1\} : S[i] == p \}$,

In this example, **3** and **11** are the only valid outputs.

Algorithm:

```
For each favorite number S[i]:                .
    ↳ Count how many people chose S[i];
  Return the most choiced one;
```

```
For each element S[j]:                        .
    ↳ If S[i] == S[j] then: increase repetitions of S[i]
  Return number of occurrences of S[i])
```

Complexity: $O(n^2)$ basic operations ————→ For each element in S, we visit every element in S .

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.

1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.

1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Favorite number	Repetitions

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.

1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

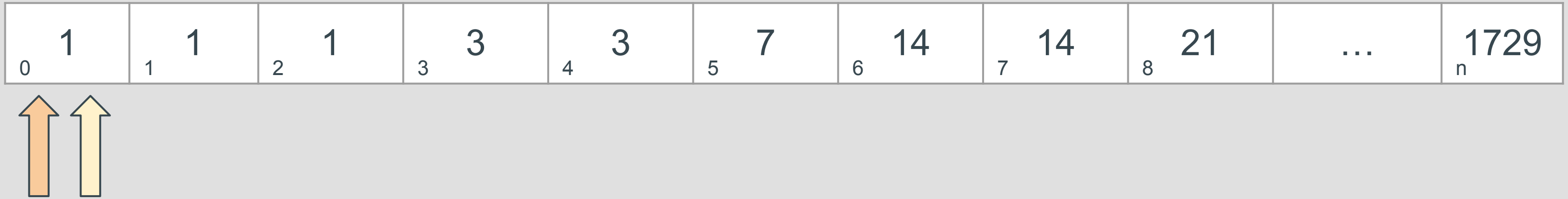
Favorite number	Repetitions
1	

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

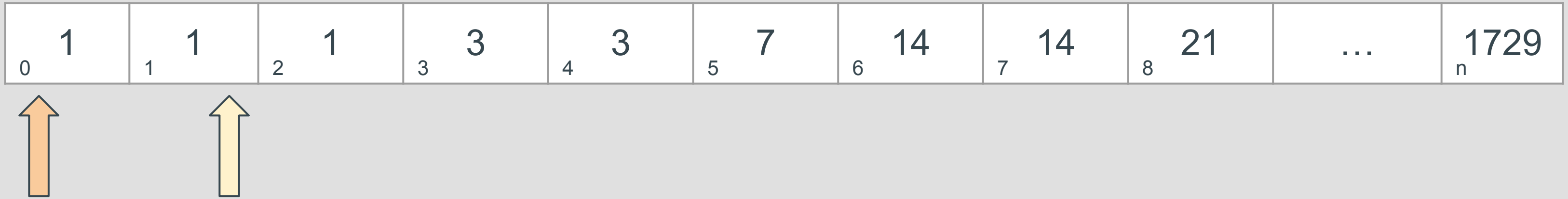
Favorite number	Repetitions
1	1

Exercise - Part B, using ordering

Previous algorithm

- For each favorite number S[i]
  - ↳ Count how many people chose S[i]

Assume that the array S is sorted.



0	1	2	3	4	5	6	7	8		n
1	1	1	3	3	7	14	14	21	…	1729

Favorite number	Repetitions
1	2

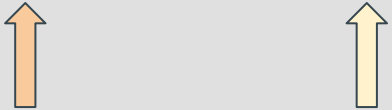
Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.

1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n



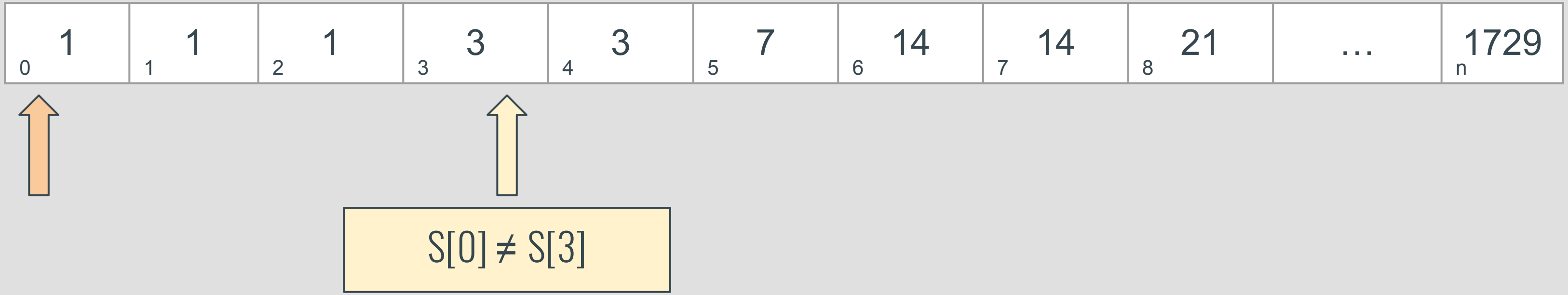
Favorite number	Repetitions
1	3

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

$S[0] \neq S[3]$

Favorite number	Repetitions
1	3

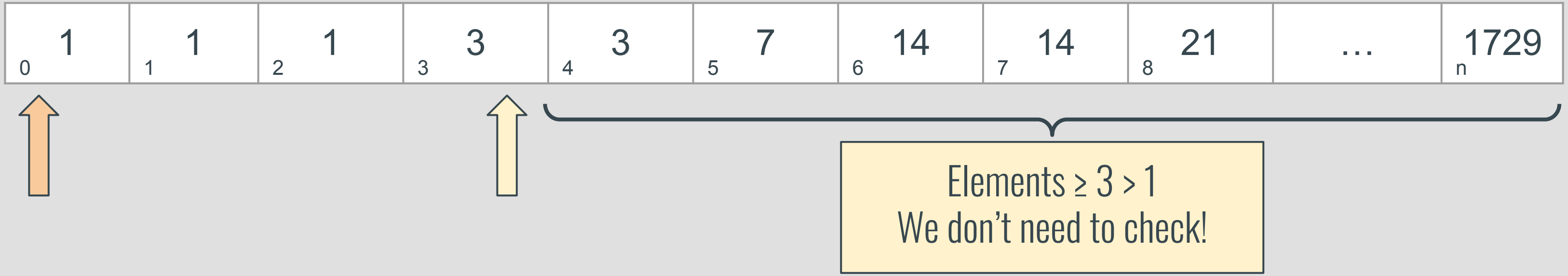
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



0	1	2	3	4	5	6	7	8		n
1	1	1	3	3	7	14	14	21	…	1729

Elements ≥ 3 > 1
We don't need to check!

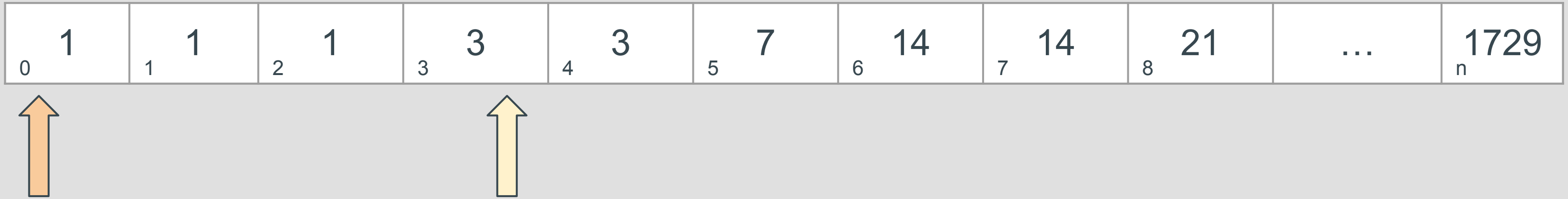
Favorite number	Repetitions
1	3

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

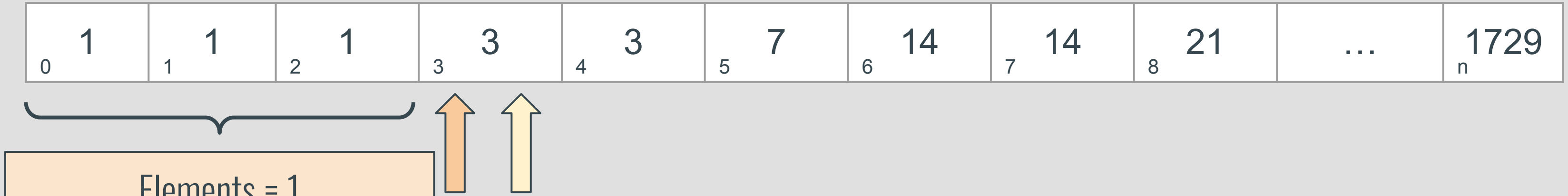
Favorite number	Repetitions
1	3

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Elements = 1
We don't need to count again!

Favorite number	Repetitions
1	3
3	

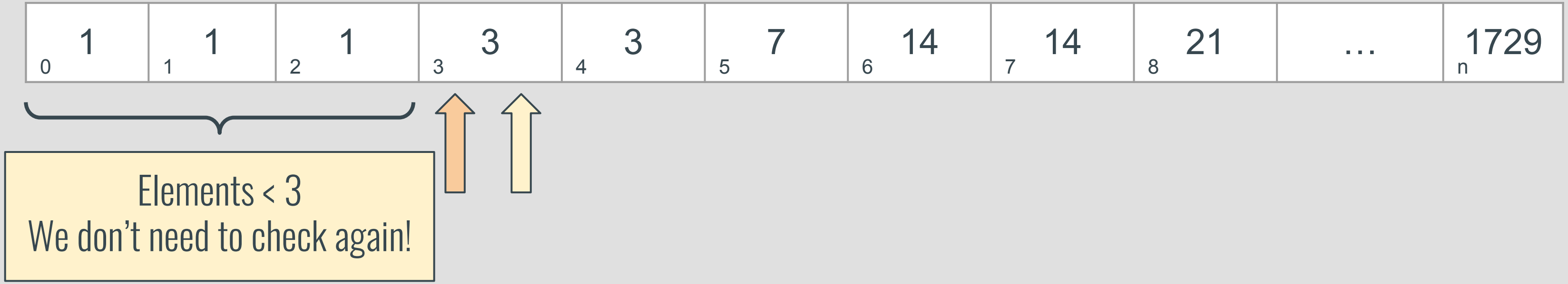
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Elements < 3
We don’t need to check again!

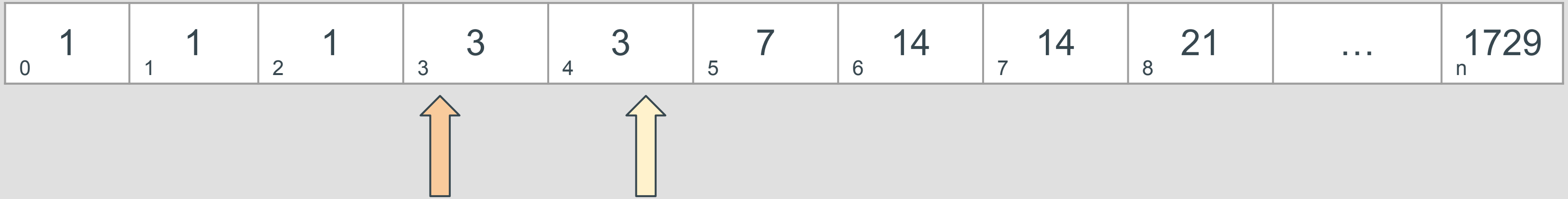
Favorite number	Repetitions
1	3
3	1

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Favorite number	Repetitions
1	3
3	2

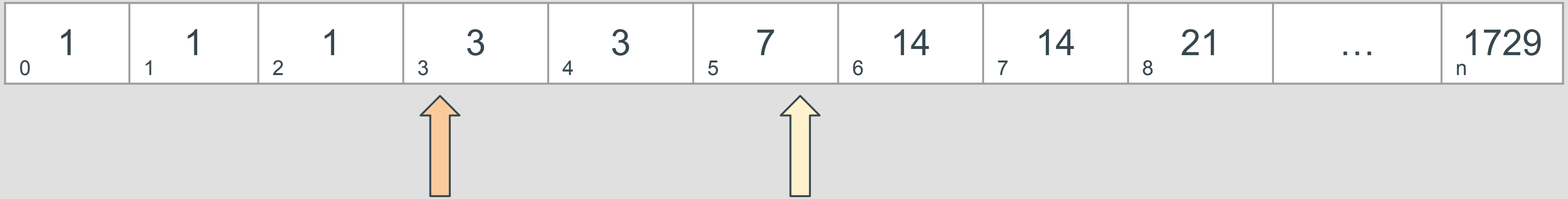
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

S[3] ≠ S[5]

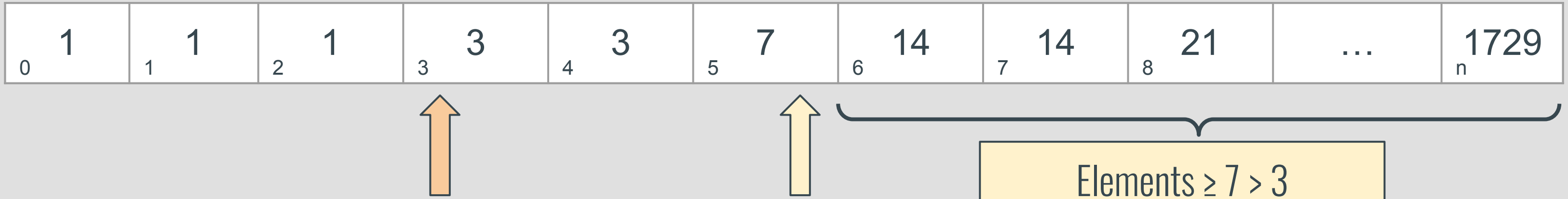
Favorite number	Repetitions
1	3
3	2

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Elements ≥ 7 > 3
We don't need to check!

Favorite number	Repetitions
1	3
3	2

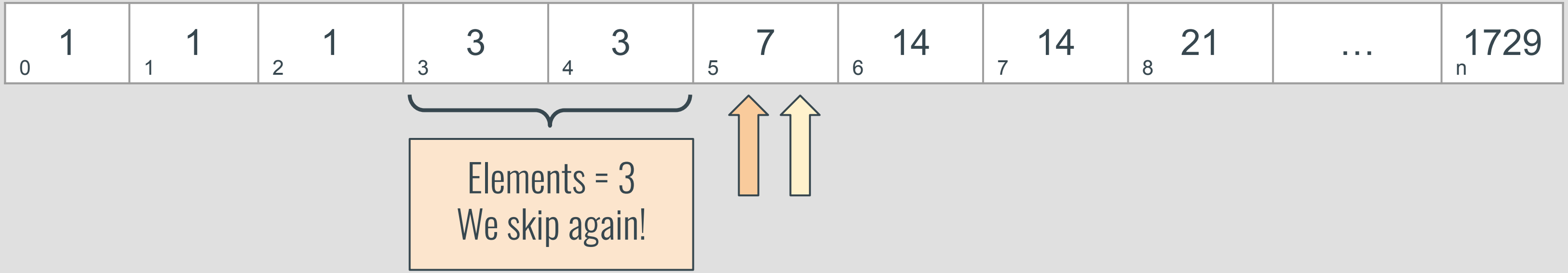
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



0	1	2	3	4	5	6	7	8		n
1	1	1	3	3	7	14	14	21	…	1729

Elements = 3
We skip again!

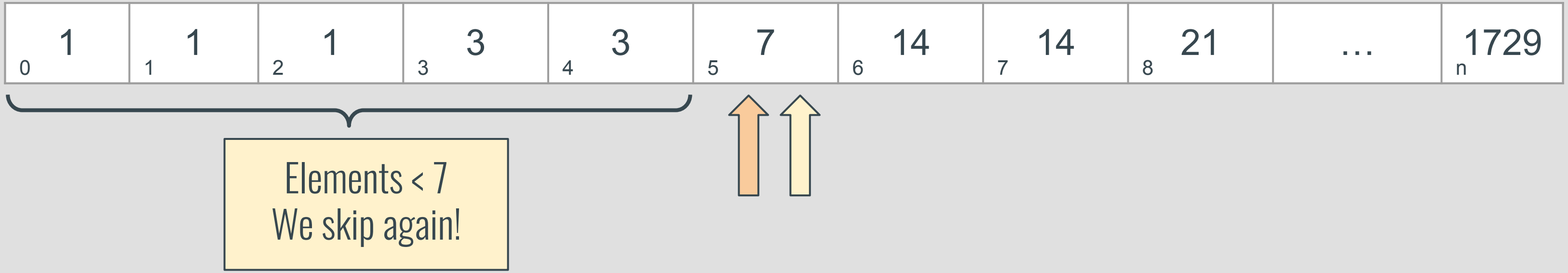
Favorite number	Repetitions
1	3
3	2
7	

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Elements < 7
We skip again!

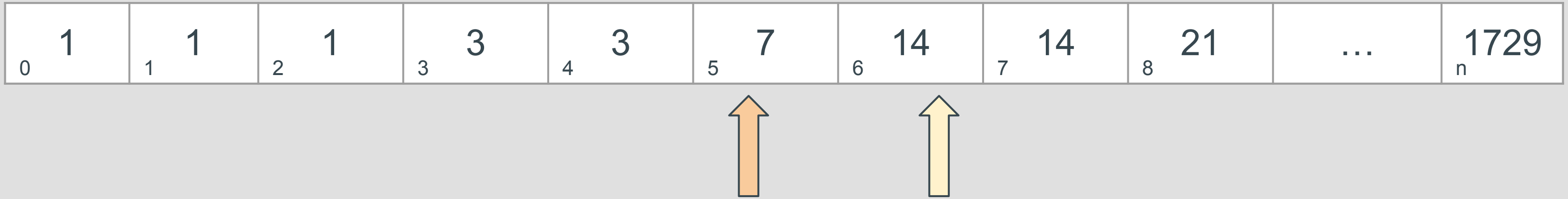
Favorite number	Repetitions
1	3
3	2
7	1

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Favorite number	Repetitions
1	3
3	2
7	1

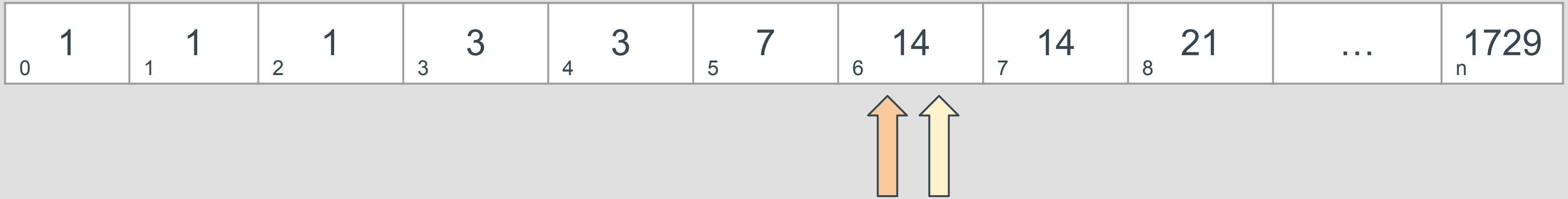
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Favorite number	Repetitions
1	3
3	2
7	1
14	

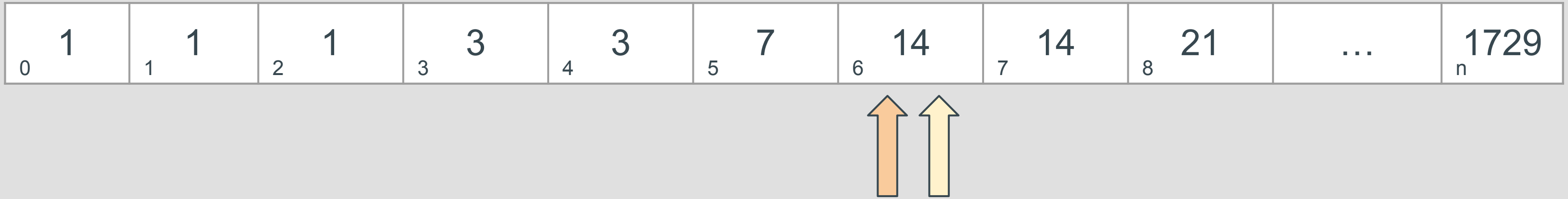
Exercise - Part B, using ordering

Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.



0	1	2	3	4	5	6	7	8		n
1	1	1	3	3	7	14	14	21	…	1729

Favorite number	Repetitions
1	3
3	2
7	1
14	1

Exercise - Part B, using ordering

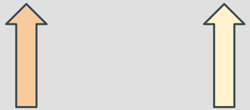
Previous algorithm

For each favorite number S[i]

↳ Count how many people chose S[i]

Assume that the array S is sorted.

1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n



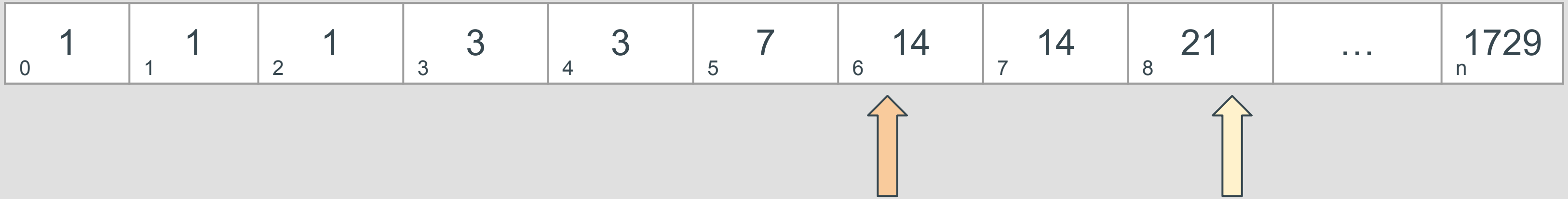
Favorite number	Repetitions
1	3
3	2
7	1
14	2

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

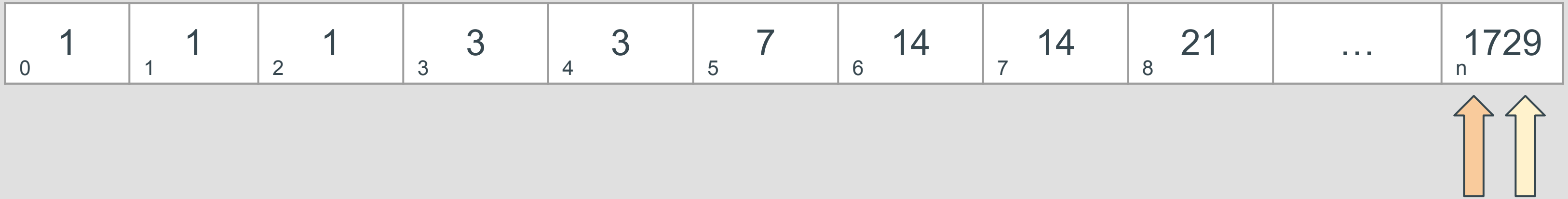
Favorite number	Repetitions
1	3
3	2
7	1
14	2
…	

Exercise - Part B, using ordering

Previous algorithm

```
For each favorite number S[i]
    ↳ Count how many people chose S[i]
```

Assume that the array S is sorted.



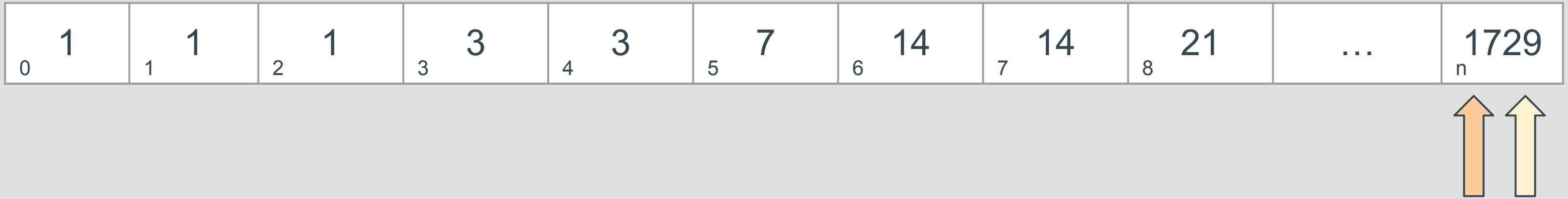
1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Favorite number	Repetitions
1	3
3	2
7	1
14	2
…	

Each **arrow only moves forward**. Hence, **new algorithm** needs **O(n)** basic operations!

Exercise - Part B, using ordering

Assume that the array S is sorted.

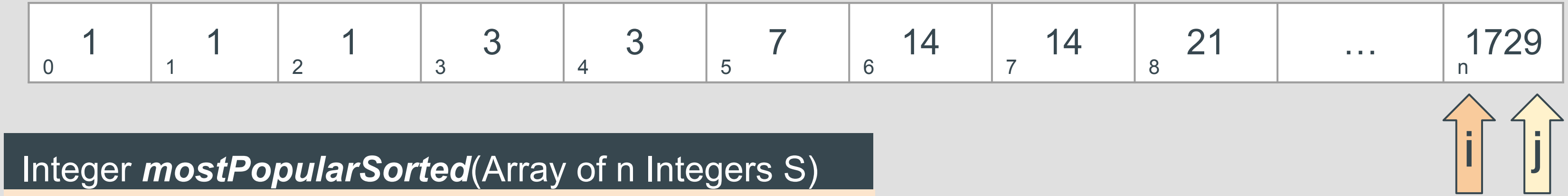


1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Each **arrow only moves forward**. Hence, **new algorithm** needs **O(n)** basic operations!

Exercise - Part B, using ordering

Assume that the array S is sorted.



1	1	1	3	3	7	14	14	21	…	1729
0	1	2	3	4	5	6	7	8		n

Integer ***mostPopularSorted***(Array of n Integers S)

```
Integer popularUntilNow, repetitions;
Integer repsPopular := 0;
Integer i := 0;
while i < n do:
    repetitions := countRepsSorted(S,i);
    if repsPopular < repetitions then:
        popularUntilNow := S[i];
        repsPopular := repetitions;
    ↳ end if;
    i := i + repetitions;
↳ end for;
return popularUntilNow;
```

Integer ***countRepsSorted***(Array of n Integers S, Integer firstOcurr)

```
Integer j := firstOcurr;
while (j+1 < n and S[j+1] == S[j]) do:
    ↳ j := j + 1;
return firstOcurr - j + 1;
```

mostPopularSorted(S) takes O(n) basic operations

Exercise - Part C, a new look into our original problem

```
Integer mostPopularSorted(Array of n Integers S)
Integer popularUntilNow, repetitions;
Integer repsPopular := 0;
Integer i := 0;
while i < n do:
    repetitions := countRepsSorted(S,i);
    if repsPopular < repetitions then:
        popularUntilNow := S[i];
        repsPopular := repetitions;
    end if;
    i := i + repetitions;
end for;
return popularUntilNow;
```

```
Integer countRepsSorted(Array of n Integers S,
                        Integer firstOcurr)
Integer j := firstOcurr;
while (j+1 < n and S[j+1] == S[j]) do:
    j := j + 1;
return firstOcurr - j + 1;
```

mostPopularSorted(S) takes
O(n) basic operations

If S is not sorted?

Exercise - Part C, a new look into our original problem

```
Integer mostPopularSorted(Array of n Integers S)
Integer popularUntilNow, repetitions;
Integer repsPopular := 0;
Integer i := 0;
while i < n do:
|   repetitions := countRepsSorted(S,i);
|   if repsPopular < repetitions then:
|   |   popularUntilNow := S[i];
|   |   repsPopular := repetitions;
|   ↳ end if;
|   i := i + repetitions;
↳ end for;
return popularUntilNow;
```

```
Integer countRepsSorted(Array of n Integers S,
                        Integer firstOcurr)
Integer j := firstOcurr;
while (j+1 < n and S[j+1] == S[j]) do:
    ↳ j := j + 1;
return firstOcurr - j + 1;
```

mostPopularSorted(S) takes
$O(n)$ basic operations

If S is not sorted? By sorting the array S, we can solve our original problem in $O(n \log n)$ basic operations

```
Integer mostPopular(Array of n Integers S)
Array[Integer] sortedS := Sort(S);          O(n log n)
return mostPopularSorted(sortedS);          O(n)
```

Some subjects I could teach

Short term
Mid term
If needed

Algorithmic techniques

- CSE103 Introduction to Algorithms,
- CSE202 Design and Analysis of Algorithms,
- INF411 Les bases de la programmation et de l'algorithmique,
- INF421 Conception et analyse d'algorithmes
- INF576 Algorithms and advanced programming

Programming paradigms and languages

- CSE101 Computer Programming,
- CSE104 Web Programming,
- CSE201 Object-oriented Programming in C++,
- CSE301 Functional Programming

Theoretical CS and logics

- CSE304 Complexity,
- INF412 Fondement de l'informatique : logique, modèles, calculs

Specialized topics

- INF556 Topological Data Analysis
- INF573 Image Analysis and Computer Vision
- INF562 Géométrie algorithmique : de la théorie aux applications.

Some subjects I could teach

Short term
Mid term
If needed

Algorithmic techniques

- CSE103 Introduction to Algorithms,
- CSE202 Design and Analysis of Algorithms,
- INF411 Les bases de la programmation et de l'algorithmique,
- INF421 Conception et analyse d'algorithmes
- INF576 Algorithms and advanced programming

Programming paradigms and languages

- CSE101 Computer Programming,
- CSE104 Web Programming,
- CSE201 Object-oriented Programming in C++,
- CSE301 Functional Programming

Theoretical CS and logics

- CSE304 Complexity,
- INF412 Fondement de l'informatique : logique, modèles, calculs

Specialized topics

- INF556 Topological Data Analysis
- INF573 Image Analysis and Computer Vision
- INF562 Géométrie algorithmique : de la théorie aux applications.

My research

Sparse polynomial systems and applications

My publications

Journals	3
Proc. of Int. Conferences	5
Preprints	3

Since '23 - CRCN Inria
2019-22 - Postdoc
2019 - PhD in CS
2015 - MSc in CS

Awards and Recognitions

Distinguished student author award, ISSAC 2016

Invited tutorial, ISSAC 2022 (on computer algebra)

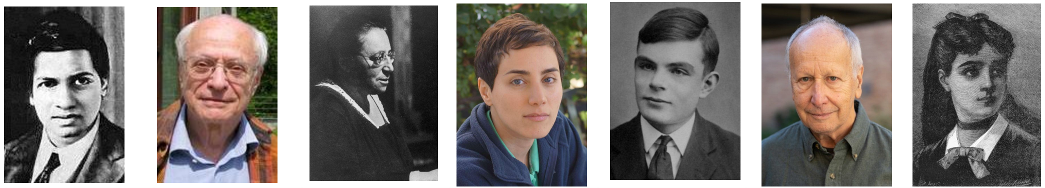
Teaching experience

Teacher assistant at Bachelor in CS (4 semester)
Algorithms and data structures
(Topics sorting, complexity, graph theory, programming techniques)

Primary and high school students (2 + 2 years)
Introduction to programming

Exercise - Part D, intro to counting sort

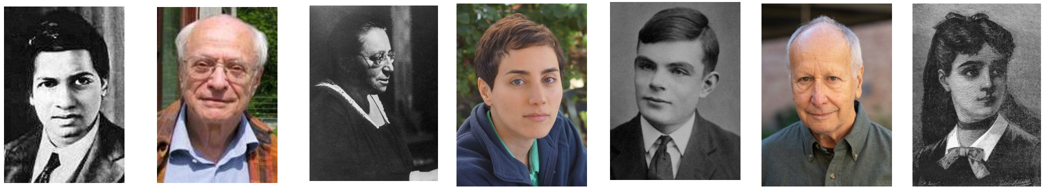
We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



7	3	2	4	4	3	4
0	1	2	3	4	5	6

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                            O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                              O(1)
 |  if reps[mostPopular] < reps[j] then:
 |    ↳ mostPopular := j;
↳ end for;
return mostPopular;            Complexity O(n)
```

Favorite number	0	1	2	3	4	5	6	7	8	9
Repetitions	0	0	0	0	0	0	0	0	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.

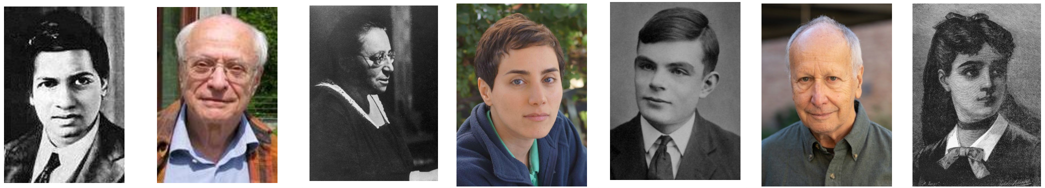
7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                               O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                                 O(1)
 |   if reps[mostPopular] < reps[j] then:
 |     ↳ mostPopular := j;
↳ end for;
return mostPopular;                       Complexity O(n)
```

Favorite number	0	1	2	3	4	5	6	7	8	9
Repetitions	0	0	0	0	0	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



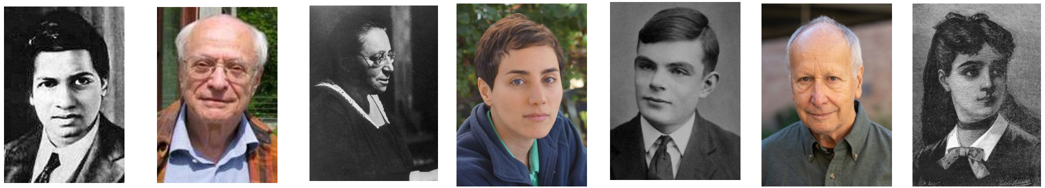
0	1	2	3	4	5	6
7	3	2	4	4	3	4

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                          O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                            O(1)
|   if reps[mostPopular] < reps[j] then:
|     ↳ mostPopular := j;
↳ end for;
return mostPopular;          Complexity O(n)
```

Favorite number	0	1	2	3	4	5	6	7	8	9
Repetitions	0	0	0	1	0	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



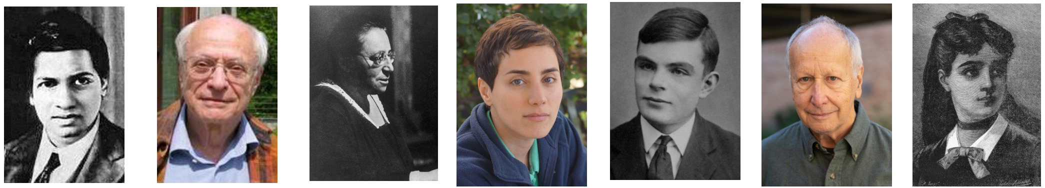
7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                              O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                                O(1)
|   if reps[mostPopular] < reps[j] then:
|     ↳ mostPopular := j;
↳ end for;
return mostPopular;          Complexity O(n)
```

Favorite number	0	1	2	3	4	5	6	7	8	9
Repetitions	0	0	1	1	0	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



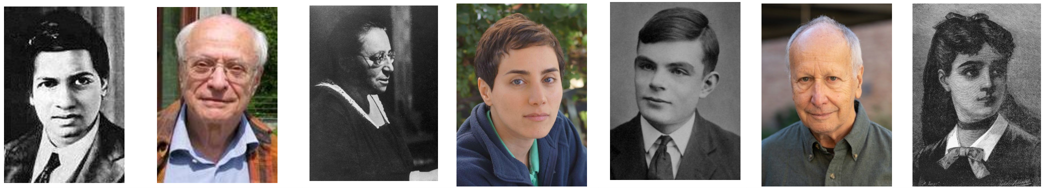
7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                              O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                                O(1)
  |   if reps[mostPopular] < reps[j] then:
  |     ↳ mostPopular := j;
↳ end for;
return mostPopular;                      Complexity O(n)
```

Favorite number	0	1	2	3	**4**	5	6	7	8	9
Repetitions	0	0	1	1	**1**	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



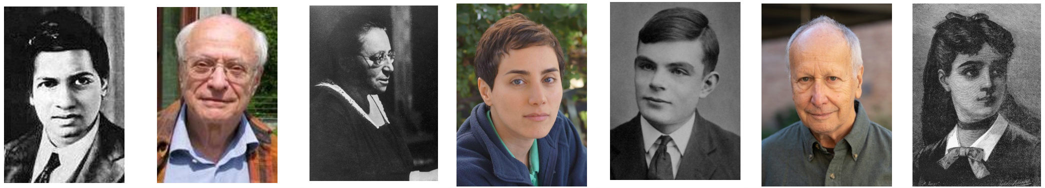
7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                          O(n)
    ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                            O(1)
|   if reps[mostPopular] < reps[j] then:
|       ↳ mostPopular := j;
↳ end for;
return mostPopular;              Complexity O(n)
```

Favorite number	0	1	2	3	**4**	5	6	7	8	9
Repetitions	0	0	1	1	**2**	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



7	3	2	4	4	3	4
0	1	2	3	4	5	6

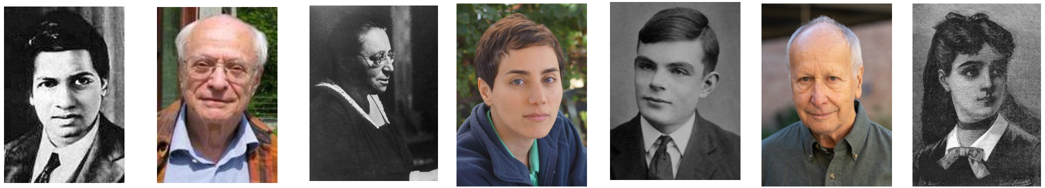
```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                          O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                            O(1)
 |  if reps[mostPopular] < reps[j] then:
 |    ↳ mostPopular := j;
↳ end for;
return mostPopular;
```

Complexity O(n)

Favorite number	0	1	2	**3**	4	5	6	7	8	9
Repetitions	0	0	1	**2**	2	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



7	3	2	4	4	3	4
0	1	2	3	4	5	6

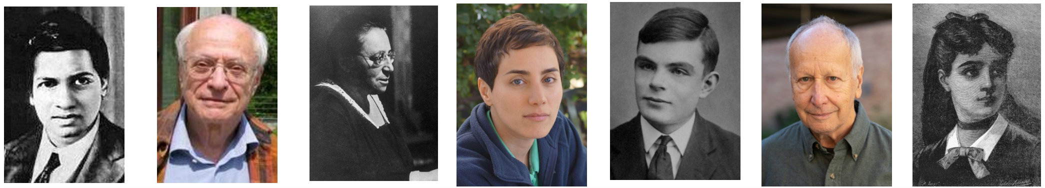
```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                              O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                                O(1)
|   if reps[mostPopular] < reps[j] then:
|     ↳ mostPopular := j;
↳ end for;
return mostPopular;
```

Complexity O(n)

Favorite number	0	1	2	3	**4**	5	6	7	8	9
Repetitions	0	0	1	2	**3**	0	0	1	0	0

Exercise - Part D, intro to counting sort

We ask **n** mathematicians their favorite number between 0 and 9 and we want to know the most popular ones.



7	3	2	4	4	3	4
0	1	2	3	4	5	6

```
Integer mostPop1to9(Array of n Integers S)
Array[Integer] reps[10] := {0, … , 0};
Integer i,j := 0;
Integer mostPopular := 1;
for i from 0 to n-1 do:                                   O(n)
  ↳ reps[S[i]] := reps[S[i]] + 1;
for j from 0 to 9 do:                                     O(1)
|   if reps[mostPopular] < reps[j] then:
|     ↳ mostPopular := j;
↳ end for;
return mostPopular; .
```

Complexity O(n)

mostPopular

Favorite number	0	1	2	3	**4**	5	6	7	8	9
Repetitions	0	0	1	2	3	0	0	1	0	0

Follow up exercise

1) Given function σ : {1,...,n} -> {1,...,n}, check if σ is a permutation, that is, it is a 1-1 map between {1,...,n} to {1,...,n}..

   Input : S - Array of integers
   Restriction: For every i, F[i] ∈ {1...n}.
   The function σ is such that σ(i) := F[i].

   Output: Boolean - Is σ a permutation?

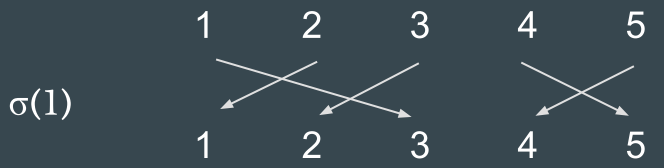
2) Compute the inverse of this map in the same time.

Approach and objectives

- Deduce and prove easy propositions
  (e.g., injective map from {1...n} to {1...n} is bijection).
- Reuse idea of previous exercise: solve by counting.

Input : S

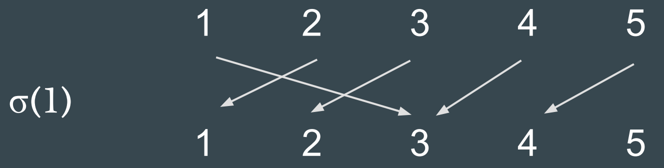
3	1	2	5	4



Output : True - σ is a permutation

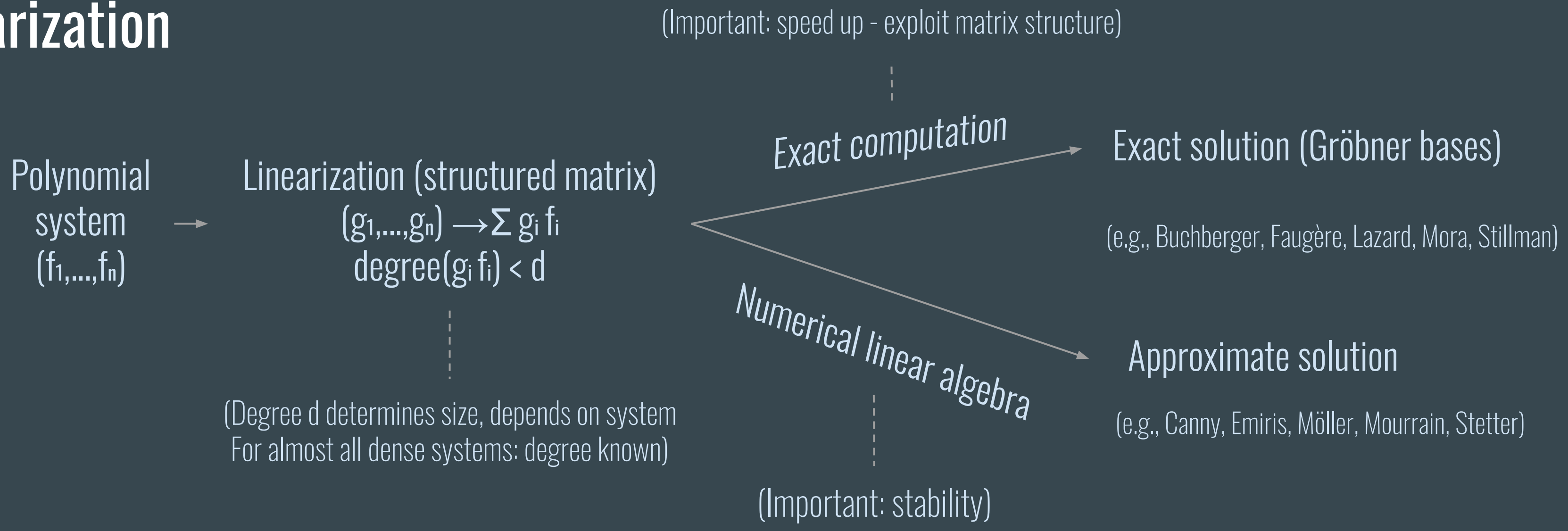
Input: S

3	1	2	3	4



Output : False - σ is NOT a permutation

Solving polynomial systems via linearization



(Important: speed up - exploit matrix structure)

Polynomial system $(f_1,\dots,f_n)$ → Linearization (structured matrix)

$(g_1,\dots,g_n) \longrightarrow \Sigma\, g_i f_i$

degree($g_i f_i$) < d

(Degree d determines size, depends on system
For almost all dense systems: degree known)

Exact computation → Exact solution (Gröbner bases)

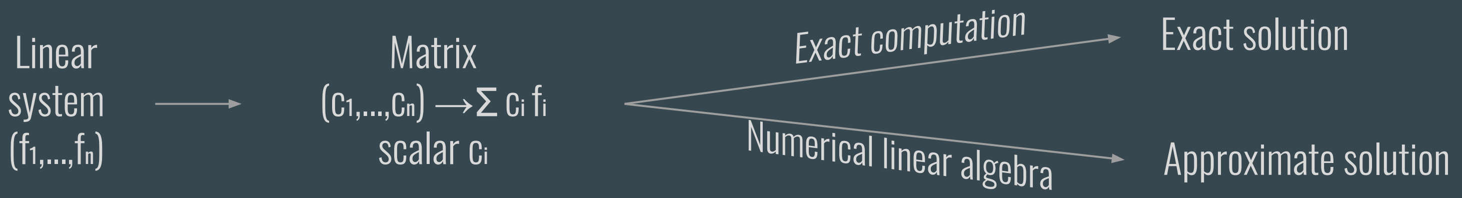
(e.g., Buchberger, Faugère, Lazard, Mora, Stillman)

Numerical linear algebra → Approximate solution

(e.g., Canny, Emiris, Möller, Mourrain, Stetter)

(Important: stability)

Analogy for linear systems



Linear system $(f_1,\dots,f_n)$ → Matrix

$(c_1,\dots,c_n) \longrightarrow \Sigma\, c_i f_i$

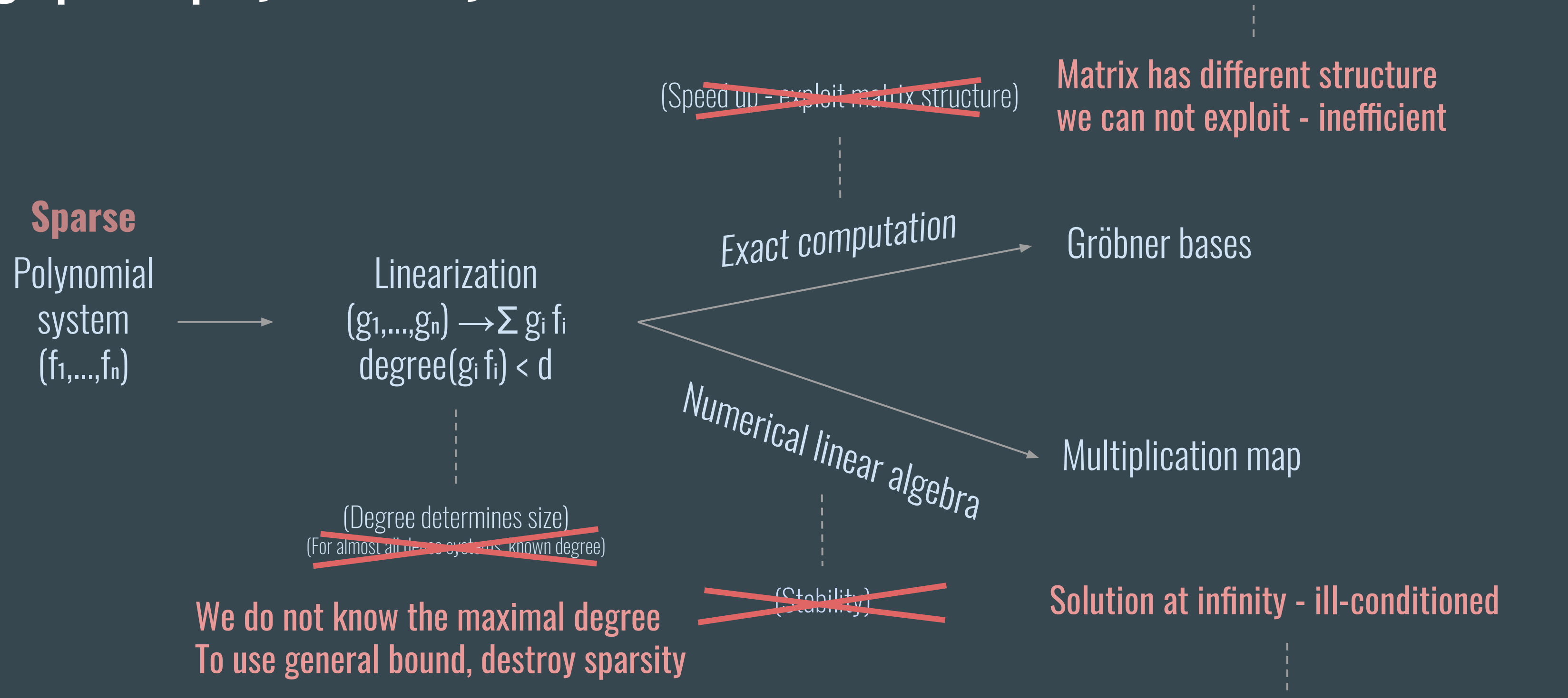
scalar c_i

Exact computation → Exact solution

Numerical linear algebra → Approximate solution

Solving sparse polynomial systems

- Study special systems, e.g., bi- or weighted homogeneous.
- Discover the structure of the matrix “on the fly” (e.g., Eder, Gao, Perry)



Matrix has different structure
we can not exploit - inefficient

~~(Speed up - exploit matrix structure)~~

Sparse

Polynomial
system
$(f_1,\dots,f_n)$

Linearization
$(g_1,\dots,g_n) \longrightarrow \Sigma\, g_i f_i$
$\text{degree}(g_i f_i) < d$

Exact computation

Gröbner bases

Numerical linear algebra

Multiplication map

(Degree determines size)
~~(For almost all dense systems, known degree)~~

We do not know the maximal degree
To use general bound, destroy sparsity

~~(Stability)~~

Solution at infinity - ill-conditioned

- Sparse resultant (e.g., Canny, Gelfand-Kapranov-Zelevinsky, Sturmfels)