Stack-sorting:
A polynomial decision algorithm

Adeline Pierrot

LRI, Université Paris Sud

Séminaire de combinatoire Philippe Flajolet, october 2014

Joint work with Dominique Rossin, during my PHD at LIAFA
1. Introduction to stack sorting

2. Pushall sorting (tri par sas)

3. General sorting
Permutations and patterns

Permutation of size $n$ : Order on $[1..n]$
Example : $\sigma = 3 \ 1 \ 2 \ 8 \ 5 \ 4 \ 7 \ 9 \ 6$
**Permutations and patterns**

**Permutation** of size $n$: Order on $[1..n]$

**Example**: $\sigma = 3\ 1\ 2\ 8\ 5\ 4\ 7\ 9\ 6$

**Pattern**: extracted sub-structure (cf subword)

**Example**: $1\ 3\ 2\ 4 \preceq 3\ 1\ 2\ 8\ 5\ 4\ 7\ 9\ 6$ since $2\ 5\ 4\ 9 \equiv 1\ 3\ 2\ 4$. 
Permutations and patterns

**Permutation** of size $n$: Order on $[1..n]$

Example: $\sigma = 3 \ 1 \ 2 \ 8 \ 5 \ 4 \ 7 \ 9 \ 6$

**Pattern**: extracted sub-structure (cf subword)

Example: $1 \ 3 \ 2 \ 4 \ 
succeq \ 3 \ 1 \ 2 \ 8 \ 5 \ 4 \ 7 \ 9 \ 6$ since $2 \ 5 \ 4 \ 9 \ \equiv \ 1 \ 3 \ 2 \ 4$.

**Remark**: $\sigma$, $\pi$ as input, deciding whether $\pi \preceq \sigma$ is NP-complete.
Class of permutations: set downward closed for $\preceq$

Equivalently: $\sigma \in C$ and $\pi \preceq \sigma \Rightarrow \pi \in C$
Class of permutations: set downward closed for ≼

Equivalently: $\sigma \in \mathcal{C}$ and $\pi \prec \sigma \Rightarrow \pi \in \mathcal{C}$

$Av(B)$: the class of perm. avoiding all the patterns in the set $B$. 
Class of permutations: set downward closed for $\preceq$

Equivalently: $\sigma \in C$ and $\pi \preceq \sigma \Rightarrow \pi \in C$

$Av(B)$: the class of perm. avoiding all the patterns in the set $B$.

Prop.: Every class $C$ is characterized by its basis:

$$C = Av(B) \text{ for } B = \{\sigma \notin C | \forall \pi \preceq \sigma \text{ with } \pi \neq \sigma, \pi \in C\}$$

Basis may be finite or infinite.
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**: 

```
4132
```

```
output       \( \mu \)
\( \rho \)  \sigma_1 \ldots \sigma_n
```
Stack sorting

**Stack:** last-in first-out device introduced by Knuth (1968).

**Definition:** \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{\rho, \mu\}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example:**

\[
\begin{array}{c}
\sigma_1 \ldots \sigma_n \\
\mu \\
\rho \\
\end{array}
\]

\[
\begin{array}{c}
\text{output} \\
\end{array}
\]

\[
\begin{array}{c}
4132 \\
\end{array}
\]
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**:

```
  4
  132
```

```
output \( \mu \) \( \rho \) \( \sigma_1 \ldots \sigma_n \)
```
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**: 

```
4
132
```

```
\sigma_1 \ldots \sigma_n
\downarrow
\rho
\mu
output
```

Definition: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

Example:

\[
\begin{array}{c|c}
1 & 32 \\
4 &
\end{array}
\]
**Stack sorting**

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example:**

```

1 4 3 2
```

\( \sigma_1 \ldots \sigma_n \)
Stack sorting

**Stack:** last-in first-out device introduced by Knuth (1968).

**Definition:** $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

```
1  4
32
```

Diagram:
```
output
     \rightarrow\downarrow
     \rightarrow
\sigma_1 \ldots \sigma_n
```
Stack sorting


Definition: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

Example:

\[
\begin{array}{c}
 & & 1 & & 32 \\
 & 4 & & & \\
\end{array}
\]
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

```
  1
  3 4
  2
```
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example**:

```
1  3  4
2
```

```
µ  ρ  σ₁...σₙ
```
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
 \end{array}
\]
**Stack sorting**

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example**:

```
1
2
3
4
```

```
    \[\mu\]
   /  \   \rho
sigma_1 \ldots sigma_n
```
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

```
12

3 4
```

$\sigma_1 \ldots \sigma_n$
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example**:

```
12
3
4
```
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

```
123
  |   |
  |   |
  |   |
  4   |
```

output $\sigma_1 \ldots \sigma_n$
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**:

\[
\begin{array}{c}
123 \\
\downarrow \\
4
\end{array}
\]
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**:

\[
\begin{array}{c}
1234 \\
\end{array}
\]

\[
\begin{array}{c}
\sigma_1 \ldots \sigma_n \\
\end{array}
\]

\[
\begin{array}{c}
\text{output} \\
\end{array}
\]

\[
\begin{array}{c}
\rho \\
\mu \\
\end{array}
\]
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example**:

```
1234 4132
```

4132 is sortable

```
\mu \rho \sigma_1 \ldots \sigma_n
```

output
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{\rho, \mu\}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**: 

\[
\begin{align*}
1234 &\rightarrow 4132 &\rightarrow 2413 \\
4132 &\text{ is sortable}
\end{align*}
\]
**Stack sorting**

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{\rho, \mu\}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**: 

\[
\begin{array}{c}
1234 \\
4132 \\
2413
\end{array}
\]

4132 is sortable
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \(\sigma\) is sortable if \(\exists\) a sequence of moves \(m \in \{\rho, \mu\}^*\) s.t. the output \(m(\sigma)\) is the identity.

**Example**:  

\[
\begin{array}{c}
1234 \\
\hline
\hline
\hline
4132 \\
\hline
\hline
\hline
4132 \text{ is sortable}
\end{array}
\]
Stack sorting


Definition: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

Example:

$1234$ $4132$

$4132$ is sortable
Stack sorting

**Stack:** last-in first-out device introduced by Knuth (1968).

**Definition:** $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

$$
\begin{array}{c}
1234 \\
\hline
4132
\end{array}
\quad
\begin{array}{c}
\hline
4132 \text{ is sortable}
\end{array}
\quad
\begin{array}{c}
4 \quad 2 \\
\hline
13
\end{array}
$$

**Definition:** $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example:**

$$\begin{align*}
1234 & \quad 4132 \\
\quad 4132 & \quad 13 \\
\text{4132 is sortable}
\end{align*}$$
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**: 

\[
\begin{array}{c}
1234 \\
\hline
4132 \\
\hline
4132 \text{ is sortable}
\end{array}
\]

\[
\begin{array}{c}
1 \\
\hline
4 \\
\hline
2 \\
\hline
3
\end{array}
\]
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

**Example**: $4132$ is sortable

$4132$ is sortable
Stack sorting


Definition: $\sigma$ is sortable if $\exists$ a sequence of moves $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity.

Example:

$1234 \quad \quad \quad 4132 \quad \quad \quad 1 \quad \quad \quad 4 \quad \quad \quad 3$

$4132$ is sortable
Stack sorting

**Stack**: last-in first-out device introduced by Knuth (1968).

**Definition**: \( \sigma \) is sortable if \( \exists \) a sequence of moves \( m \in \{ \rho, \mu \}^* \) s.t. the output \( m(\sigma) \) is the identity.

**Example**:

\[
\begin{array}{ccc}
1234 & 4132 & 1 \quad 4 \quad 2 \\
\hline
4132 \text{ is sortable} & 2413 \text{ is not sortable}
\end{array}
\]
Sorting with one stack: a linear algorithm

**Question**: How to decide if $\sigma$ is sortable?

Find $m \in \{\rho, \mu\}$ s.t. the output $m(\sigma)$ is the identity $|\sigma| = n \Rightarrow |m(\sigma)|_{\rho} = |m(\sigma)|_{\mu} = n$.

**Naive algorithm**: Check if $m(\sigma)$ is the identity $\forall m \in \{\rho, \mu\} \rightarrow$ exponential algorithm.

Key: At most one way to sort a permutation: do move $\mu$ if and only if the top of the stack is the next element to be output.$\rightarrow$ A linear algorithm to test whether a permutation is sortable.
Question: How to decide if $\sigma$ is sortable?

Find $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity

$|\sigma| = n \Rightarrow |m(\sigma)|_\rho = |m(\sigma)|_\mu = n.$
Question: How to decide if $\sigma$ is sortable?

Find $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity

$\left|\sigma\right| = n \Rightarrow \left|m(\sigma)\right|_{\rho} = \left|m(\sigma)\right|_{\mu} = n.$

Naive algorithm: Check if $m(\sigma)$ is the identity $\forall m \in \{\rho, \mu\}^{2n}$

$\rightarrow$ exponential algorithm.
Question: How to decide if $\sigma$ is sortable?

Find $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity

$|\sigma| = n \Rightarrow |m(\sigma)|_\rho = |m(\sigma)|_\mu = n$.

Naive algorithm: Check if $m(\sigma)$ is the identity $\forall m \in \{\rho, \mu\}^{2n}$

$\rightarrow$ exponential algorithm.

Key: At most one way to sort a permutation:
Do move $\mu$ if and only if the top of the stack is the next element to be output.
Question: How to decide if $\sigma$ is sortable?

Find $m \in \{\rho, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity

$|\sigma| = n \Rightarrow |m(\sigma)|_{\rho} = |m(\sigma)|_{\mu} = n$.

Naive algorithm: Check if $m(\sigma)$ is the identity $\forall \ m \in \{\rho, \mu\}^{2n}$

$\rightarrow$ exponential algorithm.

Key: At most one way to sort a permutation:
Do move $\mu$ if and only if the top of the stack is the next element to be output.

$\rightarrow$ A linear algorithm to test whether a permutation is sortable.
Question: How many sortable permutations of size $n$?
Question: How many sortable permutations of size $n$?

$\sigma$ sortable $\Leftrightarrow$ $\sigma$ avoids 231
Question: How many sortable permutations of size $n$?

$\sigma$ sortable $\iff \sigma$ avoids 231

$\sigma_1 \ldots \sigma_n$
Question: How many sortable permutations of size \( n \)?

\( \sigma \) sortable \( \iff \sigma \) avoids 231

\[ \ldots 2 \ldots 3 \ldots 1 \ldots \]
Question: How many sortable permutations of size $n$?

$\sigma$ sortable $\iff \sigma$ avoids 231
Sorting with one stack: a mathematical characterization

**Question:** How many sortable permutations of size $n$?

$\sigma$ sortable $\Leftrightarrow$ $\sigma$ avoids 231

\[
\begin{array}{cccc}
\vdots & 2 & \cdots & 3 \\
\vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \vdots & 1 \\
\end{array}
\]

The set of permutations sortable with one stack: $\text{Av}(231)$

enumerated by **Catalan numbers**: $c_n = \frac{1}{n+1} \binom{2n}{n} \approx 4^n \ll n! \approx n^n$
**Question:** How many sortable permutations of size $n$?

$\sigma$ sortable $\iff$ $\sigma$ avoids 231

\[
\begin{array}{ccc}
\vdots & \vdots & \vdots \\
2 & 3 & 1 \\
\end{array}
\begin{array}{c}
3 \\
\vdots \\
2 \\
\vdots \\
\end{array}

The set of permutations sortable with one stack: $Av(231)$ enumerated by **Catalan numbers**: $c_n = \frac{1}{n+1} \binom{n}{2n} \approx 4^n << n! \approx n^n$

**Generalized** by Tarjan, Pratt...
Natural questions for sorting devices

- **Decision**: what is the complexity of the problem consisting of deciding whether a given permutation is sortable or not?

- **Characterization**: can one characterize permutations that are sortable?

- **Counting**: how many sortable permutations of size $n$?
Sorting with two stacks in serie

**Definition**: $\sigma$ is sortable if $\exists \ m \in \{\rho, \lambda, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity (Knuth 1973).

**Question**: $\sigma$ a given permutation, is $\sigma$ sortable with two stacks?
Definition: $\sigma$ is sortable if $\exists m \in \{\rho, \lambda, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity (Knuth 1973).

Question: $\sigma$ a given permutation, is $\sigma$ sortable with two stacks?

Naive algorithm: Check if $m(\sigma)$ is the identity $\forall m \in \{\rho, \lambda, \mu\}^{3n}$ s.t. $|m(\sigma)|_\rho = |m(\sigma)|_\lambda = |m(\sigma)|_\mu = n$

$\rightarrow$ exponential algorithm ($3^{3n}$ tests).
Sorting with two stacks in serie

Definition: $\sigma$ is sortable if $\exists \ m \in \{\rho, \lambda, \mu\}^*$ s.t. the output $m(\sigma)$ is the identity (Knuth 1973).

Question: $\sigma$ a given permutation, is $\sigma$ sortable with two stacks?

Naive algorithm: Check if $m(\sigma)$ is the identity $\forall \ m \in \{\rho, \lambda, \mu\}^{3n}$ s.t. $|m(\sigma)|_\rho = |m(\sigma)|_\lambda = |m(\sigma)|_\mu = n$

$\rightarrow$ exponential algorithm ($3^{3n}$ tests).

Is there a polynomial algorithm?
Sorting with two stacks in serie

**Definition:** \( \sigma \) is sortable if \( \exists m \in \{\rho, \lambda, \mu\}^* \) s.t. the output \( m(\sigma) \) is the identity (Knuth 1973).

**Question:** \( \sigma \) a given permutation, is \( \sigma \) sortable with two stacks?

**Naive algorithm:** Check if \( m(\sigma) \) is the identity \( \forall m \in \{\rho, \lambda, \mu\}^{3n} \) s.t. \( |m(\sigma)|_\rho = |m(\sigma)|_\lambda = |m(\sigma)|_\mu = n \)

\[ \rightarrow \text{exponential algorithm (}3^{3n}\text{ tests)}.\]

Is there a polynomial algorithm?

**Conjectured NP-complete in the literature**

A canonical way to sort?

- Non unique way to sort.

\[ \sigma_1 \ldots \sigma_n \]

\[ \mu \quad \lambda \quad \rho \]

\[ \text{V} \quad \text{H} \]
A canonical way to sort?

- **Non unique** way to sort.

  Example: moves $\mu$ and $\rho$ commute.

\[ \mu \leftrightarrow \top \] of $V$ is the next element to be output.

Some permutations still have an exponential number of sortings: $n(n-1)...1$ can be sorted in $2^{n-1}$ different ways.

No way to choose between move $\lambda$ and move $\rho$.

Several weaker variants have been studied: Greedy algorithm (West 93), Increasing stacks (Murphy 02) ...
A canonical way to sort?

- **Non unique** way to sort.
  Example: moves $\mu$ and $\rho$ commute.

- **Canonical** sorting?
A canonical way to sort?

- Non unique way to sort.
  Example: moves $\mu$ and $\rho$ commute.

- Canonical sorting?

$\mu \Leftrightarrow$ top of $V$ is the next element to be output.
A canonical way to sort?

- **Non unique** way to sort.
  Example: moves $\mu$ and $\rho$ commute.

- **Canonical** sorting?

  $\mu \Leftrightarrow$ top of $V$ is the next element to be output.

  Some permutations still have an exponential number of sortings: $n \ (n-1) \ldots \ 1$ can be sorted in $2^{(n-1)}$ different ways.
A canonical way to sort?

- Non unique way to sort.
  Example: moves \( \mu \) and \( \rho \) commute.

- Canonical sorting?

  \( \mu \Leftrightarrow \) top of \( V \) is the next element to be output.

  Some permutations still have an exponential number of sortings:
  \( n \, (n - 1) \ldots 1 \) can be sorted in \( 2^{(n-1)} \) different ways.

  No way to choose between move \( \lambda \) and move \( \rho \)
A canonical way to sort?

- **Non unique** way to sort.
  
  Example: moves $\mu$ and $\rho$ commute.

- **Canonical** sorting?
  
  $\mu \Leftrightarrow$ top of $V$ is the next element to be output.

  Some permutations still have an **exponential number of sortings**: $n \ (n - 1) \ldots \ 1$ can be sorted in $2^{(n-1)}$ differents ways.

  **No way to choose** between move $\lambda$ and move $\rho$

  Several **weaker variants** have been studied:
  Greedy algorithm (West 93), Increasing stacks (Murphy 02)...

A permutation class

Let $\pi \prec \sigma$ (pattern)

sorting procedure for $\sigma \rightarrow$ sorting procedure for $\pi$
A permutation class

Let $\pi \prec \sigma$ (pattern)

sorting procedure for $\sigma \rightarrow$ sorting procedure for $\pi$

$\rightarrow \sigma$ sortable and $\pi \prec \sigma \Rightarrow \pi$ sortable

$\rightarrow$ sortable permutations form a class $Av(B)$
A permutation class

Let $\pi \prec \sigma$ (pattern)

sorting procedure for $\sigma \rightarrow$ sorting procedure for $\pi$

$\Rightarrow \sigma$ sortable and $\pi \prec \sigma \Rightarrow \pi$ sortable

$\Rightarrow$ sortable permutations form a class $Av(B)$

But $B$ infinite and not characterised

<table>
<thead>
<tr>
<th>length</th>
<th>sortable</th>
<th>unsortable</th>
<th>basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \leq 6$</td>
<td>$n!$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>5018</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>39374</td>
<td>946</td>
<td>51</td>
</tr>
<tr>
<td>9</td>
<td>336870</td>
<td>26010</td>
<td>146</td>
</tr>
<tr>
<td>10</td>
<td>3066695</td>
<td>562105</td>
<td>604</td>
</tr>
</tbody>
</table>
Decomposition

- $\sigma = \oplus[\pi_1, \ldots, \pi_n]$
Decomposition

- $\sigma = \oplus[\pi_1, \ldots, \pi_n]$ is sortable $\iff \forall i, \pi_i$ is sortable.

\[ \begin{array}{c}
\pi_1 \\
\pi_2 \\
\pi_n \\
\end{array} \quad \begin{array}{c}
s(\pi_1) \cdots s(\pi_n) \\
\pi_1 \cdots \pi_n
\end{array} \]
Decomposition

• $\sigma = \oplus[\pi_1, \ldots, \pi_n]$ is sortable $\iff \forall i, \pi_i$ is sortable.

• $\sigma = \ominus[\pi_1, \ldots, \pi_n]$ is sortable $\Rightarrow \forall i, \pi_i$ is sortable.
Decomposition

- $\sigma = \oplus [\pi_1, \ldots, \pi_n]$ is sortable $\iff \forall i, \pi_i$ is sortable.

- $\sigma = \ominus [\pi_1, \ldots, \pi_n]$ is sortable $\Rightarrow \forall i, \pi_i$ is sortable.

Converse not true: $\pi_i$ has to admit a special sorting in 2 steps:
Decomposition

• $\sigma = \oplus[\pi_1, \ldots, \pi_n]$ is sortable $\iff \forall i, \pi_i$ is sortable.

Converse not true: $\pi_i$ has to admit a special sorting in 2 steps:

$\sigma = \ominus[\pi_1, \ldots, \pi_n]$ is sortable $\iff \forall i < n, \pi_i$ is pushall sortable and $\pi_n$ is sortable.
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$
A sorting in 2 parts: first one \( \in \{\rho, \lambda\}^* \), second one \( \in \{\lambda, \mu\}^* \)
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:

\[
\begin{array}{c}
\text{sortie} \\
\mu \\
\lambda \\
\rho \\
\sigma_1 \ldots \sigma_n \\
V \\
H
\end{array}
\]

2 4 1 3
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:

```
  4
 3 1 2
```
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
Pushall sorting

A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

**Example**: 2 4 1 3 is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: $2413$ is pushall sortable:
A sorting in 2 parts: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

Example: 2 4 1 3 is pushall sortable:
**Pushall sorting**

A sorting in **2 parts**: first one $\in \{\rho, \lambda\}^*$, second one $\in \{\lambda, \mu\}^*$

**Example**: $2413$ is pushall sortable:

\[\begin{array}{c}
1 & 2 & 3 & 4 \\
\end{array}\]
Encoding a pushall sorting

\[
\begin{align*}
2 & 4 & 3 & 1 \\
\rightarrow & & \rightarrow & & \rightarrow & & \rightarrow & & \rightarrow \\
2 & 4 & 3 & 1 & 4 & 2 & 3 & 1 & 4 & 2 & 1 \\
\rightarrow & & \rightarrow & & \rightarrow & & \rightarrow & & \rightarrow & & \rightarrow \\
1 & 3 & 4 & 2 & 1 & 3 & 4 & 2 & 1 & 3 & 4 \end{align*}
\]
Encoding a pushall sorting

2 4 3 1 → 2 4 3 1 → 2 4 3 1 → 2 4 3 1 → 2 4 3 1

1 3 4 2 → 1 3 4 2 → 1 3 4 2 → 1 3 4 2 → 1 3 4 2

Test in linear time whether a coloring is valid.

2^n colorings to test → reduce this number.
Encoding a pushall sorting

\[ \begin{array}{c}
2431 & \rightarrow & 12431 & \rightarrow & 12431 & \rightarrow & 12431 & \rightarrow & 12431 \\
1342 & \rightarrow & 1342 & \rightarrow & 1342 & \rightarrow & 1342 & \rightarrow & 1342 \\
\end{array} \]

total configuration
Encoding a pushall sorting

Pushall sorting process $\Leftrightarrow$ valid configuration
Encoding a pushall sorting

$\begin{array}{c}
2 4 3 1 \\
1 3 4 2 \\
\end{array} \rightarrow 
\begin{array}{c}
\begin{array}{c}
2 \\
4 \\
1 \\
3 \\
\end{array} \\
\begin{array}{c}
3 \\
4 \\
2 \\
1 \\
\end{array} \\
\end{array}$

$\begin{array}{c}
3 1 \\
2 1 \\
\end{array} \rightarrow 
\begin{array}{c}
\begin{array}{c}
3 \\
4 \\
1 \\
2 \\
\end{array} \\
\begin{array}{c}
4 \\
3 \\
2 \\
1 \\
\end{array} \\
\end{array}$

$\begin{array}{c}
1 2 3 4 \\
1 2 3 4 \\
\end{array} \rightarrow 
\begin{array}{c}
\begin{array}{c}
1 2 3 4 \\
1 2 3 4 \\
\end{array} \\
\begin{array}{c}
1 2 3 4 \\
1 2 3 4 \\
\end{array} \\
\end{array}$

$\begin{array}{c}
2 4 3 1 \\
1 2 3 4 \\
\end{array}$

Pushall sorting process $\Leftrightarrow$ valid configuration
Encoding a pushall sorting

\[
\begin{array}{c}
\begin{array}{c}
\text{2 4 3 1} \\
\text{1 3 4 2} \\
\text{2 4 3 1}
\end{array}
\rightarrow
\begin{array}{c}
\text{2 4 3 1} \\
\text{1 3 4 2} \\
\text{1 3 4 2}
\end{array}
\rightarrow
\begin{array}{c}
\text{2 4 3 1} \\
\text{1 3 4 2} \\
\text{1 2 3 4}
\end{array}
\rightarrow
\begin{array}{c}
\text{2 4 3 1} \\
\text{1 2 3 4} \\
\text{1 2 3 4}
\end{array}
\rightarrow
\begin{array}{c}
\text{2 4 3 1}
\end{array}
\end{array}
\]

(1 stack-sorting) \quad \Rightarrow \quad 2 4 3 1

Pushall sorting process \iff valid configuration \iff valid coloring
Encoding a pushall sorting

```
\[ \begin{array}{c}
2 & 4 & 3 & 1 \\
\end{array} \]
→ \[ \begin{array}{c}
2 \\
4 & 3 & 1 \\
\end{array} \]
→ \[ \begin{array}{c}
2 \\
4 & 3 \\
1 \\
\end{array} \]
→ \[ \begin{array}{c}
2 \\
4 \\
3 & 1 \\
\end{array} \]
→ \[ \begin{array}{c}
2 \\
4 \\
3 \\
1 \\
\end{array} \]
→ \[ \begin{array}{c}
3 & 1 \\
2 \\
\end{array} \]
→ \[ \begin{array}{c}
3 \\
4 \\
1 \\
2 \\
\end{array} \]
→ \[ \begin{array}{c}
3 \\
4 \\
1 \\
2 \\
\end{array} \]
→ \[ \begin{array}{c}
1 & 2 & 3 & 4 \\
\end{array} \]
```

Pushall sorting process \( \iff \) valid configuration \( \iff \) valid coloring

→ Test in **linear time** whether a coloring is valid.
Encoding a pushall sorting

2 4 3 1 → 2 4 3 1 → 2 4 3 1 → 2 4 3 1 → 2 4 3 1

1 3 4 2 → 1 3 4 2 → 1 3 4 2 → 12 3 4 12 3 4 12 3 4

2 4 3 1 (1 stack-sorting) → 3 4 1 2 → (1 stack-sorting) 1 2 3 4 1 2 3 4

Pushall sorting process ⇔ valid configuration ⇔ valid coloring

→ Test in linear time whether a coloring is valid.
2^n colorings to test → reduce this number.
Valid coloring: coloring of $\sigma$ with two colors $G$ and $R$ s.t.

- no pattern $132$ in $R$
- no pattern $213$ in $G$
- no point of $R$ lying vertically between a pattern $12$ of $G$
- no point of $G$ lying horizontally between a pattern $12$ of $R$

$\Rightarrow$ coloring with forbidden patterns $132$, $213$, $1X2$ and $2/13$
Valid coloring: coloring of $\sigma$ with two colors $G$ and $R$ s.t.

- no pattern $132$ in $R$
- no pattern $213$ in $G$
- no point of $R$ lying vertically between a pattern $12$ of $G$
- no point of $G$ lying horizontally between a pattern $12$ of $R$

$\Rightarrow$ coloring with forbidden patterns $132$, $213$, $1X2$ and $2/13$

Proof: $R =$ right stack and $G =$ left stack $\Rightarrow$ bijection between these colorings and valid stack-configurations.
Proof

Sortable stack-configuration $\iff$ avoids $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$
Proof

Sortable stack-configuration \iff\ avoids \begin{align*} &1\ 2\ \ \ 3 \\ &2\ 3\ \ 1 \end{align*}

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

  \begin{align*}
  &1\ 3\ 2 \\
  &1\ 3\ 2
  \end{align*}

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of \(R\) in the right stack and thoses of \(G\) in the left stack in the right order).
Proof

Sortable stack-configuration $\iff$ avoids $\begin{array}{c} 2 \\
1 \end{array}$, $\begin{array}{c} 2 \\
3 \\
1 \end{array}$ and $\begin{array}{c} 2 \\
3 \\
1 \end{array}$

Recall: Forbidden colored patterns $= 132, \ 213, \ 1X2$ and $2/13$.

Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

  $\begin{array}{c} 2 \\
1 \end{array}$, $\begin{array}{c} 2 \\
3 \\
1 \end{array}$ and $\begin{array}{c} 2 \\
3 \\
1 \end{array}$

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids \[ \begin{array}{c|c|c} 2 & 1 & \emptyset \\ \hline & & \emptyset \end{array}, \begin{array}{c|c} 2 & 3 \\ \hline & 1 \end{array} \text{ and } \begin{array}{c|c|c} 2 & \emptyset & 3 \\ \hline & 1 \end{array} \]

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:
  \[ \begin{array}{c|c|c} 2 & 1 & \emptyset \\ \hline & & \emptyset \end{array}, \begin{array}{c|c} 2 & 3 \\ \hline & 1 \end{array} \]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of \( R \) in the right stack and thoses of \( G \) in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids $\begin{array}{c}
2 \\
1
\end{array}$, $\begin{array}{c}
2 \\
3 \\
1
\end{array}$ and $\begin{array}{c}
2 \\
3
\end{array}$

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

\[
\begin{array}{c}
2 \\
1
\end{array}, \quad \begin{array}{c}
2 \\
3 \\
1
\end{array}, \quad \begin{array}{c}
2 \\
3
\end{array}
\]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration \iff avoids \[
\begin{array}{c}
2 \\
1 \\
\end{array},
\begin{array}{c}
3 \\
1 \\
\end{array}
\text{ and }
\begin{array}{c}
2 \\
3 \\
1 \\
\end{array}
\]

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids \( \begin{array}{c} 2 \\ 1 \end{array}, \begin{array}{c} 2 \\ 3 \end{array} \) and \( \begin{array}{c} 2 \\ 3 \end{array} \)

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

\[
\begin{array}{c}
| & | \\
| & | \\
| & |
\end{array}
\quad 1X2
\]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of \( R \) in the right stack and thoses of \( G \) in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids \[\begin{array}{c}
\begin{array}{c}
2 \\
1 \\
\end{array}
\end{array}, \begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1 \\
\end{array}
\end{array}\] and \[\begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1 \\
\end{array}
\end{array}\]

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.

Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:
  \[\begin{array}{c}
\begin{array}{c}
2 \\
1 \\
\end{array}
\end{array}, \begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1 \\
\end{array}
\end{array}\]
  \[\begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1 \\
\end{array}
\end{array}\]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration $\iff$ avoids $\begin{array}{c}
\begin{array}{c}
2 \\
1
\end{array}
\end{array}$, $\begin{array}{c}
\begin{array}{c}
2 \\
3
\end{array}
\end{array}$ and $\begin{array}{c}
\begin{array}{c}
2 \\
3
\end{array}
\end{array}$

Recall: Forbidden colored patterns $= \begin{array}{c}
\begin{array}{c}
132 \\
213 \\
1X2 \\
2/13
\end{array}
\end{array}$. Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:
  $\begin{array}{c}
\begin{array}{c}
2 \\
1
\end{array}
\end{array}$ and $\begin{array}{c}
\begin{array}{c}
2 \\
3
\end{array}
\end{array}$

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and those of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids \[
\begin{array}{c}
\begin{array}{c}
2 \\
1
\end{array}
\end{array},
\begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1
\end{array}
\end{array}
\text{ and } \begin{array}{c}
\begin{array}{c}
2 \\
3 \\
1
\end{array}
\end{array}
\]

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

\[
\begin{array}{c}
\begin{array}{c}
1 \\
X
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
2 \\
\end{array}
\end{array}
\]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of \( R \) in the right stack and those of \( G \) in the left stack in the right order).
Proof

Sortable stack-configuration $\iff$ avoids $\begin{array}{|c|c|} \hline 1 & \vphantom{1} \\ \hline \end{array}$, $\begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array}$, $\begin{array}{|c|c|} \hline 3 & 1 \\ \hline \end{array}$ and $\begin{array}{|c|c|} \hline 2 & 3 \\ \hline \end{array}$

Recall: Forbidden colored patterns $= 132$, $213$, $1X2$ and $2/13$.

Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

  $\begin{array}{|c|c|} \hline 1 & \vphantom{1} \\ \hline \end{array}$, $\begin{array}{|c|c|} \hline 2 & \vphantom{1} \\ \hline \end{array}$

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration $\Leftrightarrow$ avoids $\begin{array}{c} 2 \\ 1 \end{array}$, $\begin{array}{c} 3 \\ 1 \end{array}$ and $\begin{array}{c} 2 \\ 3 \end{array}$

Recall: Forbidden colored patterns $= 132$, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

• If the coloring comes from a sorting process, then it avoids the colored patterns:

\[
\begin{array}{c} 2 \\ 1 \end{array} \quad \begin{array}{c} X \end{array}
\]

• If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration $\iff$ avoids $\begin{bmatrix} 2 & 1 \end{bmatrix}$, $\begin{bmatrix} 2 & 3 & 1 \end{bmatrix}$ and $\begin{bmatrix} 2 & 3 \end{bmatrix}$

Recall: Forbidden colored patterns $= 132, 213, 1X2$ and $2/13$.

Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

\[
\begin{bmatrix} 2 & 1 \\ 3 & & \end{bmatrix}
\]

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration $\Leftrightarrow$ avoids $\begin{array}{c|c}
2 & 1 \\
\hline
\end{array}$, $\begin{array}{c|c|c}
2 & 3 & 1 \\
\hline
\end{array}$ and $\begin{array}{c|c|c}
2 & 3 \\
\hline
1 & 1
\end{array}$

Recall: Forbidden colored patterns $= 132, 213, 1X2$ and $2/13$.
Correspondence between stack-patterns and colored patterns.

• If the coloring comes from a sorting process, then it avoids the colored patterns:

$\begin{array}{c|c}
1 & 2 \\
\hline
3 & 3
\end{array}$

• If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration $\iff$ avoids $\begin{array}{|c|c|} \hline 2 & 1 \\ \hline \end{array}$, $\begin{array}{|c|c|} \hline 2 & 3 \\ \hline \end{array}$ and $\begin{array}{|c|c|} \hline 2 & 3 \\ \hline \end{array}$

Recall: Forbidden colored patterns $= 132, 213, 1X2$ and $2/13$.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

  $\begin{array}{|c|c|} \hline \hline & 1 \\ \hline \hline \end{array}$ $\begin{array}{|c|c|} \hline 3 & 2 \\ \hline \hline \end{array}$

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Proof

Sortable stack-configuration ⇔ avoids $\begin{array}{c} 2 \\ 1 \end{array}$, $\begin{array}{c} 2 \\ 3 \end{array}$ and $\begin{array}{c} 2 \\ 3 \end{array}$

Recall: Forbidden colored patterns = 132, 213, 1X2 and 2/13.
Correspondence between stack-patterns and colored patterns.

- If the coloring comes from a sorting process, then it avoids the colored patterns:

  $\begin{array}{c} 2 \\ 1 \end{array}$, $\begin{array}{c} 2 \\ 3 \end{array}$ and $\begin{array}{c} 2 \\ 3 \end{array}$

- If the coloring avoids the colored patterns, then we obtain a sorting process (no choice to put the elements of $R$ in the right stack and thoses of $G$ in the left stack in the right order).
Forbidden colored patterns:

\[ \text{Col}(\sigma) = \text{the set of valid colorings of } \sigma \]

\[ \#\text{Col}(n(n-1) \ldots 1) = 2^n \]
Decomposition

Forbidden colored patterns:

\[ \text{Col}(\sigma) = \{ \text{valid colorings of } \sigma \} \]

\[ \#\text{Col}(n(n-1)\ldots1) = 2^n \]

\[ \ominus[\pi_1, \ldots, \pi_k] = \begin{array}{c}
\pi_1 \\
\pi_2 \\
\vdots \\
\pi_k
\end{array} \]

Example: \( \ominus[1, \ldots, 1] = n(n-1)\ldots1 \)

Theorem:
\[ \sigma = \ominus[\pi_1, \ldots, \pi_k] \Rightarrow \text{Col}(\sigma) \approx \text{Col}(\pi_1) \times \cdots \times \text{Col}(\pi_k) \]
Restrict the number of bicolorings to test

Add hypothesis
to ensure a polynomial number of bicolorings to test.
Restrict the number of bicolorings to test

Add hypothesis
to ensure a polynomial number of bicolorings to test.

- Assume $\sigma$ is $\ominus$-indecomposable.
  Otherwise $\sigma = \ominus[\pi_1 \ldots \pi_n]$ with $\pi_i$ $\ominus$-indecomposable
  $\sigma = \ominus[\pi_1, \ldots, \pi_k] \Rightarrow \text{Col}(\sigma) \approx \text{Col}(\pi_1) \times \cdots \times \text{Col}(\pi_k)$
  So replace $\sigma$ by the $\pi_i$. 
Restrict the number of bicolorings to test

Add hypothesis
to ensure a polynomial number of bicolorings to test.

• Assume $\sigma$ is $\Theta$-indecomposable.
  Otherwise $\sigma = \Theta[\pi_1 \ldots \pi_n]$ with $\pi_i$ $\Theta$-indecomposable
  $\sigma = \Theta[\pi_1, \ldots, \pi_k] \Rightarrow \text{Col}(\sigma) \approx \text{Col}(\pi_1) \times \cdots \times \text{Col}(\pi_k)$
  So replace $\sigma$ by the $\pi_i$.

• Separate distinct cases:
  Each pattern 12 is unicolor
  There are patterns 12 but no pattern 12
  There are patterns 12 but no pattern 12
  There are patterns 12 and patterns 12.
Forbidden colored patterns $\Rightarrow$ implication rules

(i) $\emptyset$

(ii) $\emptyset$

(iii) $\emptyset$

(iv) $\emptyset$

(v) $\emptyset$

(vi) $\emptyset$

(vii) $\emptyset$

(viii) $\emptyset$
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$. [Diagram showing $\sigma_i$ and $\sigma_j$]
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\implies C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$. 

\[ \sigma_i \quad \sigma_j \]
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

![Diagram](image)

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$. Zone $A$ is non-empty as $\sigma$ is $\ominus$-indecomposable.
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma$ $\Theta$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\implies C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$. Zone $A$ is non-empty as $\sigma$ is $\Theta$-indecomposable.
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$. Zone $A$ is non-empty as $\sigma$ is $\ominus$-indecomposable. Let $\sigma_k$ in this zone and $c$ its color,
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$.

Zone $A$ is non-empty as $\sigma$ is $\ominus$-indecomposable.

Let $\sigma_k$ in this zone and $c$ its color,
First case: Each pattern 12 is unicolor

**Proposition:** \( \sigma \Theta \)-indecomposable and \( C \) a right coloring of \( \sigma \) where each pattern 12 is unicolor \( \Rightarrow \) \( C \) is unicolor.

**Proof:** Left-to-right minima of \( \sigma \) are unicolor.

Let \( \sigma_i \) and \( \sigma_j \) consecutive left-to-right minima of \( \sigma \).

Zone \( A \) is non-empty as \( \sigma \) is \( \Theta \)-indecomposable.

Let \( \sigma_k \) in this zone and \( c \) its color, then \( \sigma_i \) and \( \sigma_j \) also have color \( c \).
First case: Each pattern 12 is unicolor

**Proposition:** $\sigma \ominus$-indecomposable and $C$ a right coloring of $\sigma$ where each pattern 12 is unicolor $\Rightarrow C$ is unicolor.

**Proof:** Left-to-right minima of $\sigma$ are unicolor.

Let $\sigma_i$ and $\sigma_j$ consecutive left-to-right minima of $\sigma$.

Zone $A$ is non-empty as $\sigma$ is $\ominus$-indecomposable.

Let $\sigma_k$ in this zone and $c$ its color, then $\sigma_i$ and $\sigma_j$ also have color $c$.

**Consequence:** We just have to check the 2 unicolor colorings (all points in $\mathbb{R}$ or all points in $\mathbb{G}$).
Other cases

• There are patterns 12 but no pattern 12: Position of the down-rightmost pattern 12 determines all colors:

\[ \emptyset \quad \emptyset \quad \emptyset \]
\[ \emptyset \quad \sigma_j \quad \emptyset \]
\[ \emptyset \quad \sigma_i \quad \emptyset \]

Moreover, knowing the position of \( \sigma_i \) is sufficient to recover \( \sigma_j \) and determine all colors.
Other cases

- There are patterns 12 but no pattern 12: Position of the down-rightmost pattern 12 determines all colors:

Moreover, knowing the position of $\sigma_i$ is sufficient to recover $\sigma_j$ and determine all colors.
Other cases

- There are patterns 12 but no pattern 12: Position of the down-rightmost pattern 12 determines all colors:

Moreover, knowing the position of $\sigma_i$ is sufficient to recover $\sigma_j$ and determine all colors.

- Similar results for the other cases.
8 kinds of colorings for $\sigma \ominus$-indecomposable

**Theorem**: $c$ valid coloring of $\sigma \Rightarrow \exists m, p \text{ s.t. } c = C_m(p)$. 

\begin{align*}
C_1 & \quad C_2 & \quad C_3 & \quad C_4 \\
C_5 & \quad C_6 & \quad C_7 & \quad C_8
\end{align*}
Quadratic algorithm

Algorithm:

Input: $\sigma \ominus$-indecomposable.
Output: All valid colorings of $\sigma$:

For $i$ from 1 to 8
    For $p$ from 1 to $n = |\sigma|$
        Test if $C_i(p)$ is a valid coloring of $\sigma$
Quadratic algorithm

Algorithm:

*Input:* $\sigma \ominus$-indecomposable.
*Output:* All valid colorings of $\sigma$:

For $i$ from 1 to 8
  
  For $p$ from 1 to $n = |\sigma|$  
  
  Test if $C_i(p)$ is a valid coloring of $\sigma$

*Complexity:*

Test if a coloring is valid $=$ linear
Quadratic algorithm

Algorithm:

Input: $\sigma$ $\ominus$-indecomposable.

Output: All valid colorings of $\sigma$:

For $i$ from 1 to 8
    For $p$ from 1 to $n = |\sigma|
        Test if $C_i(p)$ is a valid coloring of $\sigma$

Complexity:

Test if a coloring is valid = linear

$\sigma$ $\ominus$-indecomposable $\Rightarrow |Col(\sigma)| \leq 8|\sigma|$ computed in $O(|\sigma|^2)$
Quadratic algorithm

Algorithm:

*Input*: $\sigma$ $\Theta$-indecomposable.

*Output*: All valid colorings of $\sigma$:

For $i$ from 1 to 8
  
  For $p$ from 1 to $n = |\sigma|
  
  Test if $C_i(p)$ is a valid coloring of $\sigma$

Complexity:

Test if a coloring is valid $=$ linear

$\sigma$ $\Theta$-indecomposable $\implies |Col(\sigma)| \leq 8|\sigma|$ computed in $O(|\sigma|^2)$

$\sigma = \Theta[\pi_1, \ldots, \pi_k] \implies Col(\sigma) \approx Col(\pi_1) \times \cdots \times Col(\pi_k)$

$\implies Col(\sigma)$ described by $(Col(\pi_1), \ldots, Col(\pi_k))$
Quadratic algorithm

Algorithm:

*Input:* $\sigma \ominus$-indecomposable.

*Output:* All valid colorings of $\sigma$:

For $i$ from 1 to 8
   For $p$ from 1 to $n = |\sigma|$
      Test if $C_i(p)$ is a valid coloring of $\sigma$

Complexity:

Test if a coloring is valid $=$ linear

$\sigma \ominus$-indecomposable $\Rightarrow |\text{Col}(\sigma)| \leq 8|\sigma|$ computed in $O(|\sigma|^2)$

$\sigma = \ominus[\pi_1, \ldots, \pi_k] \Rightarrow \text{Col}(\sigma) \approx \text{Col}(\pi_1) \times \cdots \times \text{Col}(\pi_k)$

$\rightarrow \text{Col}(\sigma)$ described by $(\text{Col}(\pi_1), \ldots, \text{Col}(\pi_k))$

$\rightarrow$ computed in *quadratic* time: $8|\pi_1|^2 + \cdots + 8|\pi_k|^2 \leq 8|\sigma|^2$. 
Outline

1. Introduction to stack sorting
2. Pushall sorting (tri par sas)
3. General sorting
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

**Configuration** when \( \sigma_{k_i} \) enters the stacks

\[ \sigma_{k_i} = \{ \sigma_j | j < k_i \text{ and } \sigma_j > \sigma_{k_i} \} \]

\( \sigma_{k_i} \)-sortable \( \Rightarrow \) push-all sortable \( \forall i \)

The push-all sortings of the \( \sigma_{(i)} \) must be compatibles

Recursive algorithm

Compatibility test = linear.

Exponential number of tests?
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

Configuration when \( \sigma_{k_i} \) enters the stacks

\[ \sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \} \]
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

Configuration when \( \sigma_{k_i} \) enters the stacks = total for \( \sigma^{(i)} \)

\[ \sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \} \]
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

Configuration when \( \sigma_{k_i} \) enters the stacks = total for \( \sigma^{(i)} \)

\[ \sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \} \]

\( \sigma \) sortable \( \Rightarrow \) \( \sigma^{(i)} \) push-all sortable \( \forall i \)
From pushall sorting to general sorting

\( \sigma_{k_i} = \text{right-left minima of } \sigma \)

**Configuration** when \( \sigma_{k_i} \) enters the stacks = total for \( \sigma^{(i)} \)

\[ \sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \} \]

\( \sigma \) sortable \( \Rightarrow \sigma^{(i)} \) push-all sortable \( \forall i \)

The push-all sortings of the \( \sigma^{(i)} \)
must be *compatibles*
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

**Configuration** when \( \sigma_{k_i} \) enters the stacks = total for \( \sigma^{(i)} \)

\[ \sigma^{(i)} = \{ \sigma_j \mid j < k_i \land \sigma_j > \sigma_{k_i} \} \]

\( \sigma \) sortable \( \Rightarrow \) \( \sigma^{(i)} \) push-all sortable \( \forall i \)

The push-all sortings of the \( \sigma^{(i)} \)

must be **compatibles**

Recursive algorithm
From pushall sorting to general sorting

\[ \sigma_{k_i} = \text{right-left minima of } \sigma \]

**Configuration** when \( \sigma_{k_i} \) enters the stacks = total for \( \sigma^{(i)} \)

\( \sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \} \)

\( \sigma \) sortable \( \Rightarrow \sigma^{(i)} \) push-all sortable \( \forall i \)

The push-all sortings of the \( \sigma^{(i)} \) must be **compatibles**

Recursive algorithm

Compatibility test = **linear**.
From pushall sorting to general sorting

$\sigma_{k_i} =$ right-left minima of $\sigma$

**Configuration** when $\sigma_{k_i}$ enters the stacks = total for $\sigma^{(i)}$

$\sigma^{(i)} = \{ \sigma_j \mid j < k_i \text{ et } \sigma_j > \sigma_{k_i} \}$

$\sigma$ sortable $\Rightarrow$ $\sigma^{(i)}$ push-all sortable $\forall i$

The push-all sortings of the $\sigma^{(i)}$ must be **compatibles**

Recursive algorithm

Compatibility test = **linear**. Exponentiel number of tests?
Reduce the number of tests

\[ \sigma_k \]

\[ \sigma_{k_i+1} \]

\[ \sigma_{i+2} \]

\[ \sigma^{(i)} \]

\[ \sigma_{k_i} \]
Reduce the number of tests

\[ \sigma_k \]

\[ \sigma_{i+1} \]

\[ \sigma_{k_{i+1}} \]

\[ \sigma_i \]

\[ B_i \]

\[ \sigma^{(i)} \]

\[ \sigma^{(i+1)} \]

\[ Col(\sigma^{(i)}) \approx \]

\[ Col(B_1^{(i)}) \times \cdots \times Col(B_k^{(i)}) \]
Reduce the number of tests

\[
\sigma_{k_i} \rightarrow \text{linear number of tests}
\]

Configurations of \( C_{i+1} \) linked to those of \( D_{i+1} \) → sorting graph

\[
\begin{align*}
\text{Col}(\sigma^{(i)}) & \approx \\
\text{Col}(B^{(i)}_1) \times \cdots \times \text{Col}(B^{(i)}_k)
\end{align*}
\]
Reduce the number of tests

\[ \text{Col}(\sigma^{(i)}) \approx \text{Col}(B_1^{(i)}) \times \cdots \times \text{Col}(B_k^{(i)}) \]
Reduce the number of tests

$\sigma_k \iota \sigma_{k+1}

D(i+1)

B(i)

$\sigma(i)\iota \sigma_{k+1}

Col(\sigma(i)) \approx \underbrace{\text{Col}(B_1(i)) \times \cdots \times \text{Col}(B_k(i))}_{\text{linear number of tests}}$

It is enough to test compatibility on $B(i)$ and $D(i+1)$
Reduce the number of tests

\[ \sigma_{ki} \approx \sigma_{ki+1} + 1 \]

\[ D(i+1) \]

\[ B(i) \]

\[ \sigma(i) \]

\[ \sigma(k_i) \]

\[ \sigma(k_{i+1}) \]

\[ \text{Col}(\sigma(i)) \approx \text{Col}(B_1(i)) \times \cdots \times \text{Col}(B_k(i)) \]

It is enough to test compatibility on \( B(i) \) and \( D(i+1) \)

\[ \rightarrow \text{linear number of tests} \]
Reduce the number of tests

It is enough to test compatibility on $B(i)$ and $D(i+1)$

$\rightarrow$ linear number of tests

Configurations of $C(i+1)$ linked to those of $D(i+1)$
Reduce the number of tests

\[
\text{Col}(\sigma^{(i)}) \approx \text{Col}(B^{(i)}_1) \times \cdots \times \text{Col}(B^{(i)}_k)
\]

It is enough to test compatibility on \(B^{(i)}\) and \(D^{(i+1)}\)

→ linear number of tests

Configurations of \(C^{(i+1)}\) linked to those of \(D^{(i+1)}\)

→ sorting graph
Sorting graph for $\sigma^{(i)}$

\[
\sigma^{(i)} = \ominus [B_1, B_2, \ldots B_s]
\]

Stack configurations of $B_1$

Stack configurations of $B_2$

Stack configurations of $B_3$

Links between compatibles stack configurations

→ a path gives a valid stack configuration of $\sigma^{(i)}$ which is a part of a sorting procedure of $\sigma_1 \ldots \sigma_{k_i}$. 
Algorithm

\[ \sigma = \ldots \sigma_{k_1} \ldots \sigma_{k_2} \ldots \sigma_{k_\ell} \quad (\sigma_{k_i} = \text{right-to-left minima of } \sigma) \]

At step \(i\), the algorithm returns false if \(\sigma_1 \ldots \sigma_{k_i}\) is not 2-stack sortable.

Otherwise it computes the sorting graph of \(\sigma^{(i)}\) describing all the possible stack configurations when \(\sigma_{k_i}\) enters the stacks in a sorting procedure of \(\sigma\) verifying some conditions.

Sorting graph of \(\sigma^{(i)}\) computed from the one of \(\sigma^{(i-1)}\) by checking compatibility between configurations.
Conclusion

Polynomial decision algorithm for 2 stacks in series

- **New notion**: push-all sorting
- Characterization through bicolorings with excluded patterns
- **Optimal** quadratic algorithm to compute all push-all sortings
- **Decomposition** along right-left minima
- One gets all sortings satisfying a property $P$. 
Perspectives

- **Simplify** the algorithm?
Perspectives

• Simplify the algorithm?
• Characterize the permutations sortable with 2 stacks in series?
  Enumeration?

Thank you for your attention
Perspectives

- Simplify the algorithm?
- Characterize the permutations sortable with 2 stacks in series? Enumeration?
- Enumerate the push-all sortable permutations?
Perspectives

- Simplify the algorithm?
- Characterize the permutations sortable with 2 stacks in series? Enumeration?
- Enumerate the push-all sortable permutations?

- Complexity of the decision algorithm for $k$ stacks in series:
Perspectives

• **Simplify** the algorithm?

• **Characterize** the permutations sortable with 2 stacks in series? Enumeration?

• **Enumerate** the push-all sortable permutations?

• **Complexity** of the decision algorithm for $k$ stacks in series:
  - **Generalize** to more than 2 stacks?
Perspectives

• **Simplify** the algorithm?

• **Characterize** the permutations sortable with 2 stacks in series? Enumeration?

• **Enumerate** the push-all sortable permutations?

• **Complexity** of the decision algorithm for $k$ stacks in series:
  • **Generalize** to more than 2 stacks?
  • For fixed $k$, is the problem still polynomial? Is there a threshold?
Perspectives

• Simplify the algorithm?
• Characterize the permutations sortable with 2 stacks in series? Enumeration?
• Enumerate the push-all sortable permutations?

• Complexity of the decision algorithm for $k$ stacks in series:
  • Generalize to more than 2 stacks?
  • For fixed $k$, is the problem still polynomial? Is there a threshold?

Thank you for your attention