

Preimages of Sorting Operators

Hjalti Magnússon Henning Úlfarsson



HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY

School of Computer Science

Permutation Patterns 2013
1 July 2013

Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

Outline

1 Introduction

- Permutations and Patterns
- Stack Sort

2 Preimage of stack sort

- Stack of limited depth

3 Preimage of queue sort

4 Preimage of other sorting operators


- Pop-stack sort
- Insertion sort
- Pancake sort

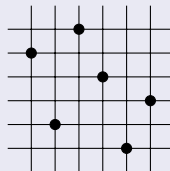
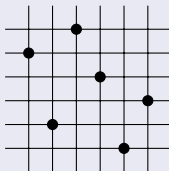
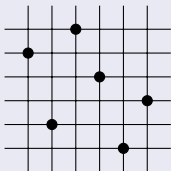
Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

Patterns as graphs


Example

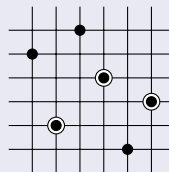
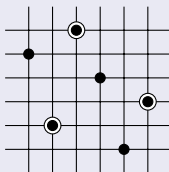
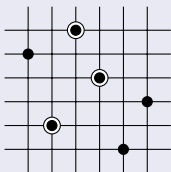
The pattern  occurs in the permutation 526413



Patterns as graphs


Example

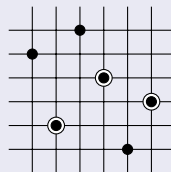
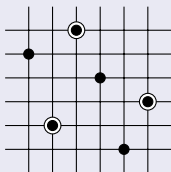
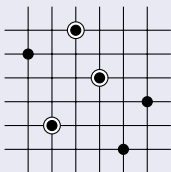
The pattern  occurs in the permutation 526413




Patterns as graphs

Example


The pattern  occurs in the permutation 526413



The same permutation avoids the pattern 


Mesh patterns

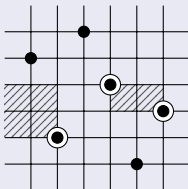
Mesh patterns (Brändén and Claesson, 2010) may forbid certain boxes from containing elements.

The pattern  occurs in the permutation 526413

Mesh patterns

Mesh patterns (Brändén and Claesson, 2010) may forbid certain boxes from containing elements.


The pattern  occurs in the permutation 526413

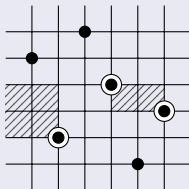


This is an occurrence

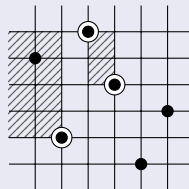
Mesh patterns

Mesh patterns (Brändén and Claesson, 2010) may forbid certain boxes from containing elements.

The pattern  occurs in the permutation 526413



This is an occurrence



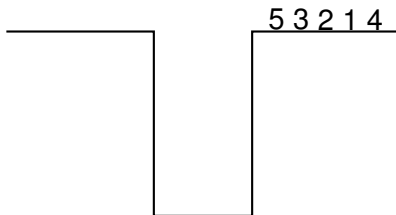
This is not an occurrence

Outline

- 1 Introduction
 - Permutations and Patterns
 - **Stack Sort**
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

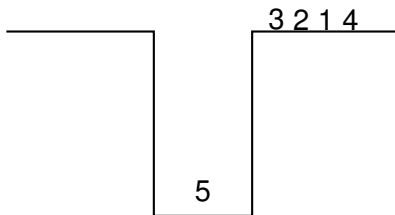
An example

Let's stack-sort 53214



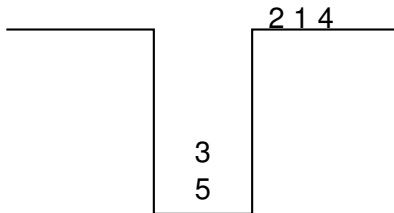
An example

Let's stack-sort 53214



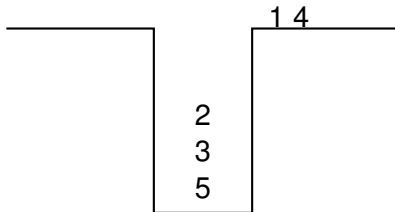
An example

Let's stack-sort 53214



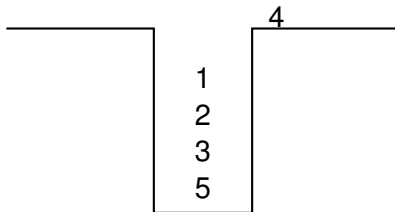
An example

Let's stack-sort 53214



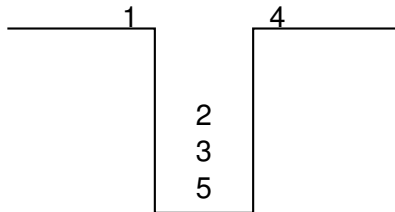
An example

Let's stack-sort 53214



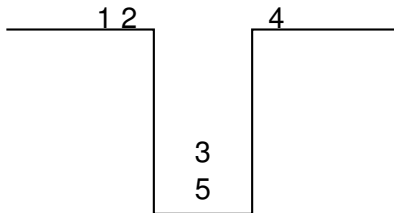
An example

Let's stack-sort 53214



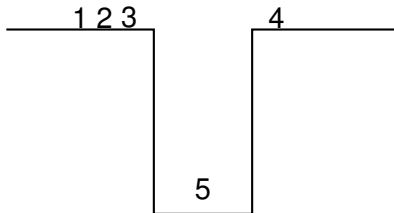
An example

Let's stack-sort 53214



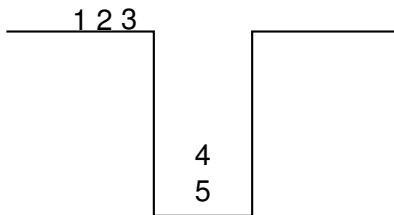
An example

Let's stack-sort 53214



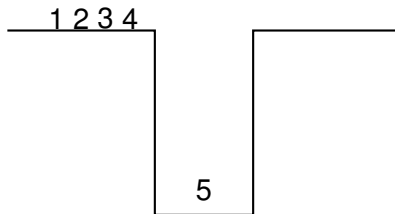
An example

Let's stack-sort 53214



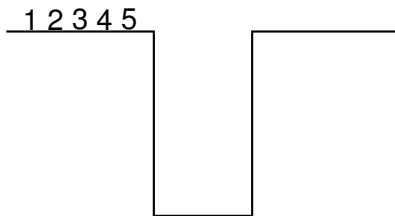
An example

Let's stack-sort 53214



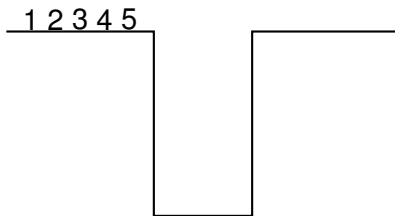
An example

Let's stack-sort 53214



An example

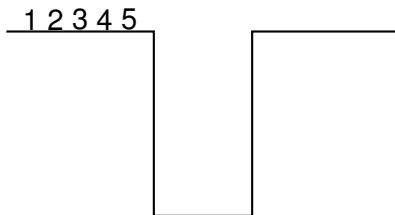
Let's stack-sort 53214



So $S(53214) = 12345$

An example

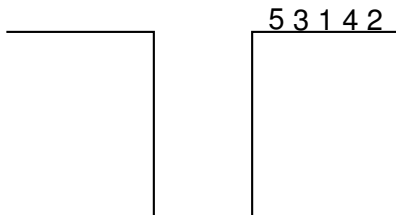
Let's stack-sort 53214



So $S(53214) = 12345$, which is great!

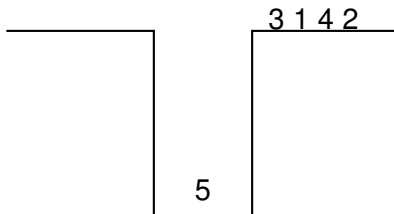
Another example

What about 53142?



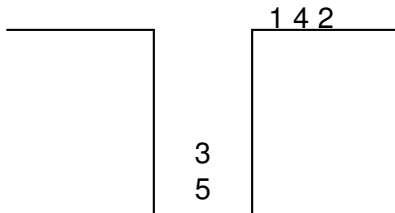
Another example

What about 53142?



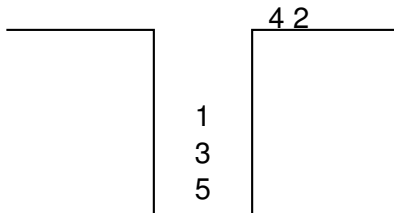
Another example

What about 53142?



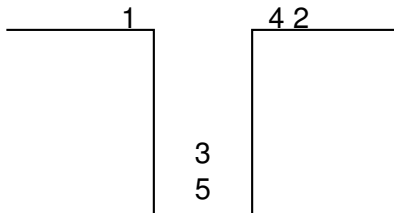
Another example

What about 53142?



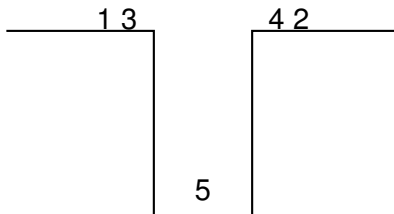
Another example

What about 53142?



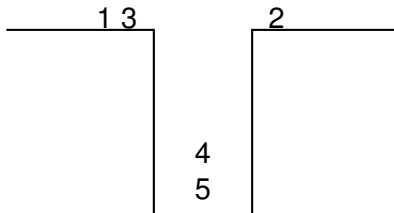
Another example

What about 53142?



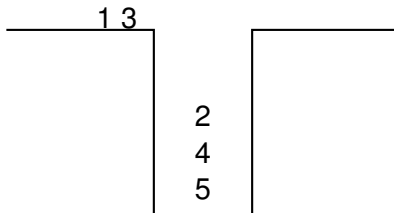
Another example

What about 53142?



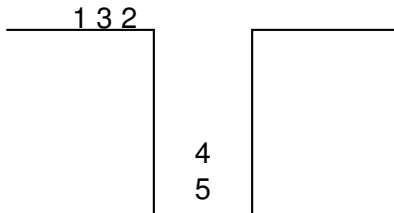
Another example

What about 53142?



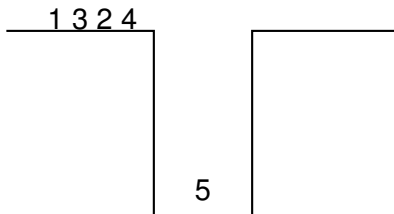
Another example

What about 53142?



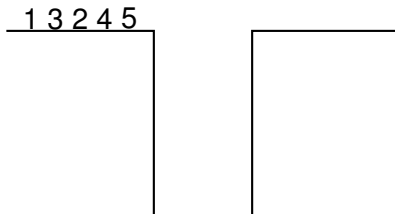
Another example

What about 53142?



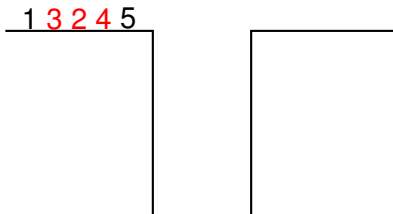
Another example

What about 53142?



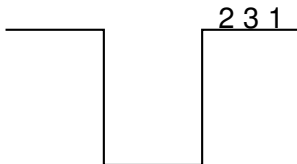
Another example

What about 53142?



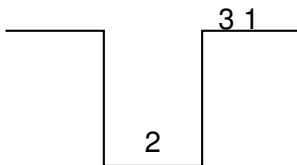
The culprit

The culprit is the pattern 231



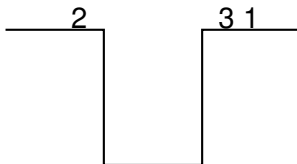
The culprit

The culprit is the pattern 231



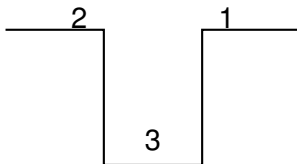
The culprit

The culprit is the pattern 231

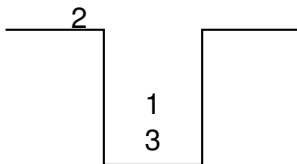


The culprit

The culprit is the pattern 231

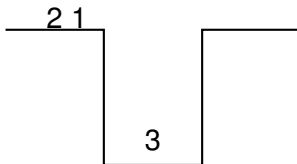


The culprit is the pattern 231



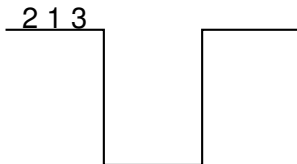
The culprit

The culprit is the pattern 231



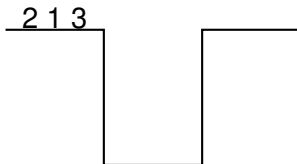
The culprit

The culprit is the pattern 231



The culprit

The culprit is the pattern 231



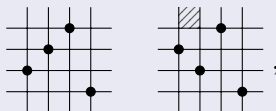
Theorem (Knuth, 1973)

A permutation is stack-sortable if and only if it avoids the classical pattern 231

West-2-stack-sortable

Theorem (West, 1993)

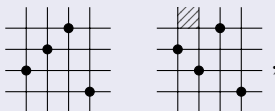
A permutation is sortable with two passes through a stack if and only if it avoids the following two mesh patterns



West-2-stack-sortable

Theorem (West, 1993)

A permutation is sortable with two passes through a stack if and only if it avoids the following two mesh patterns



or using Knuth's result

$$S^{-1}(\text{Av}(231)) = \text{Av} \left(\begin{array}{c} \text{5x5 grid with points at (2,3), (3,2), (4,1), (5,4)} \\ \text{5x5 grid with shaded (1,1) and points at (2,2), (3,1), (4,3), (5,4)} \end{array} \right)$$

Outline

1 Introduction

- Permutations and Patterns
- Stack Sort

2 Preimage of stack sort

- Stack of limited depth

3 Preimage of queue sort

4 Preimage of other sorting operators

- Pop-stack sort
- Insertion sort
- Pancake sort

An algorithm to find preimages

- In 2012 Claesson and Úlfarsson introduced an algorithm, preim_S , to find preimages of avoidance classes under the stack-sort operator
- Given a classical pattern p (the target), the algorithm finds a set M of marked mesh patterns such that

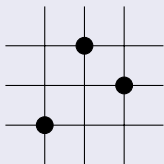
$$S^{-1}(\text{Av}(p)) = \text{Av}(M)$$

- The algorithm proceeds in two steps
 - 1 Generate possible classical pattern *candidates*
 - 2 Add appropriate shadings and markings to each candidate, if possible

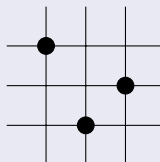
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



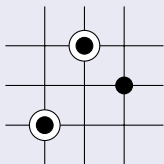
Candidate



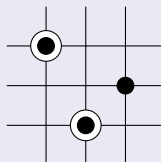
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



Candidate

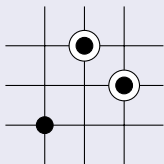


Inversions can become non-inversions

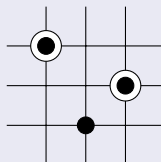
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



Candidate

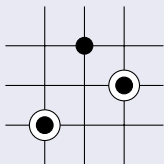


Inversions can stay inverted

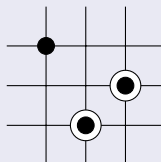
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



Candidate

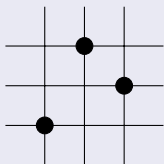


Non-inversions must remain non-inversions

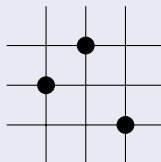
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



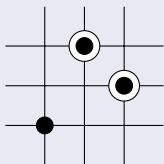
Not a candidate



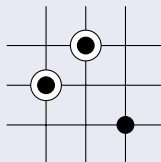
Candidates

Identify patterns that can possibly become the target pattern after stack sorting

Target



Not a candidate

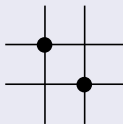


Non-inversions cannot become inversions

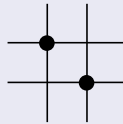
Shading and marking

- Inversion in target and candidate

Target



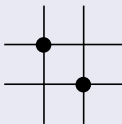
Candidate



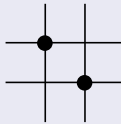
Shading and marking

- Inversion in target and candidate
- Inversion must be maintained

Target



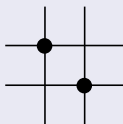
Candidate



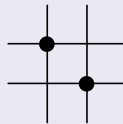
Shading and marking

- Inversion in target and candidate
- Inversion must be maintained
- There must be an element element in the candidate that pops the first element in the inversion off the stack before the second one appears

Target




Candidate




Marked mesh patterns

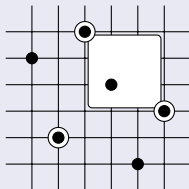
Marked mesh patterns (Úlfarsson, 2011) allow finer control of the number of elements in a region.

The pattern  occurs in the permutation 526413

Marked mesh patterns

Marked mesh patterns (Úlfarsson, 2011) allow finer control of the number of elements in a region.


The pattern  occurs in the permutation 526413

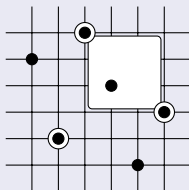


This is an occurrence

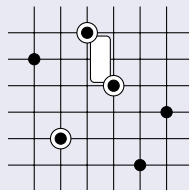
Marked mesh patterns

Marked mesh patterns (Úlfarsson, 2011) allow finer control of the number of elements in a region.

The pattern  occurs in the permutation 526413



This is an occurrence

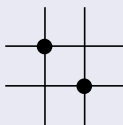


This is not an occurrence

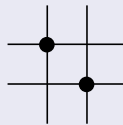
Shading and marking

- Inversion in target and candidate
- Inversion must be maintained
- There must be an element element in the candidate that pops the first element in the inversion off the stack before the second one appears

Target



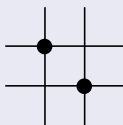
Candidate



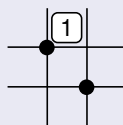
Shading and marking

- Inversion in target and candidate
- Inversion must be maintained
- There must be an element element in the candidate that pops the first element in the inversion off the stack before the second one appears

Target



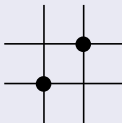
Candidate



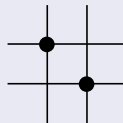
Shading and marking

- Non-inversion in target and inversion in candidate

Target



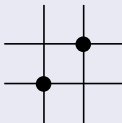
Candidate



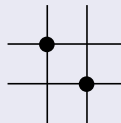
Shading and marking

- Non-inversion in target and inversion in candidate
- Inversion must be fixed

Target



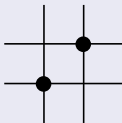
Candidate



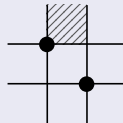
Shading and marking

- Non-inversion in target and inversion in candidate
- Inversion must be fixed
- The second element of the inversion must land on top of the first one in the stack

Target



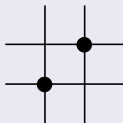
Candidate



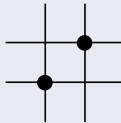
Shading and marking

- Non-inversion in target and candidate
- Nothing needs to be done – stack sort doesn't make things worse

Target

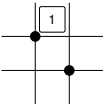
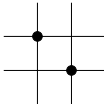
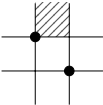
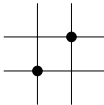


Candidate



The second step of the algorithm

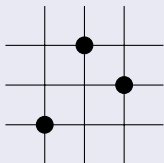
- We need only consider inversions in candidates
- Apply shadings and markings according to *meta patterns*

Meta pattern	Target
	
	

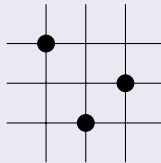
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



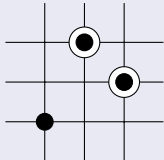
Candidate



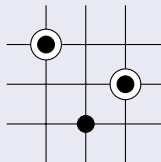
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



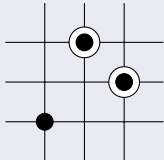
Candidate



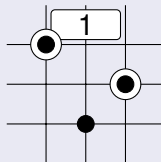
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



Candidate



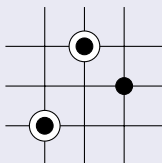
Inversion must be maintained. Apply



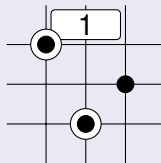
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



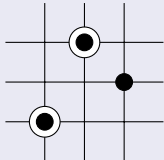
Candidate



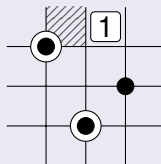
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



Candidate



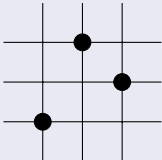
Inversion must be fixed. Apply



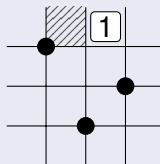
Shading and marking

Try to add shadings and markings to the candidate so that it becomes the target after stack sorting

Target



Candidate

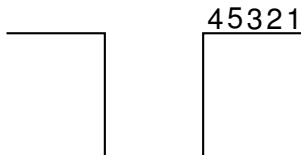


Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

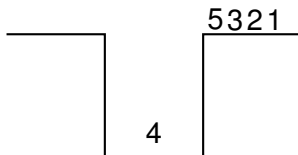
An example

Stack-sort of depth 3 applied to 45321



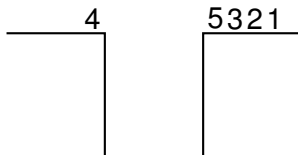
An example

Stack-sort of depth 3 applied to 45321



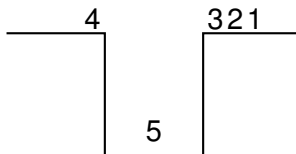
An example

Stack-sort of depth 3 applied to 45321



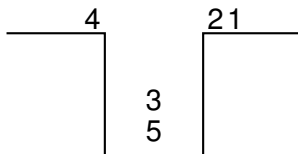
An example

Stack-sort of depth 3 applied to 45321



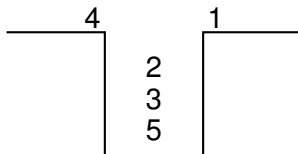
An example

Stack-sort of depth 3 applied to 45321



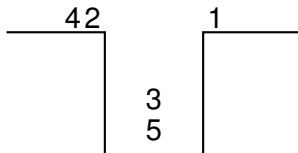
An example

Stack-sort of depth 3 applied to 45321



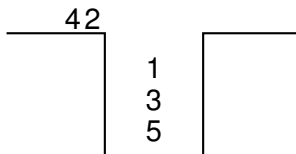
An example

Stack-sort of depth 3 applied to 45321



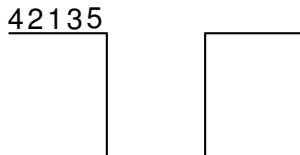
An example

Stack-sort of depth 3 applied to 45321



An example

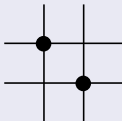
Stack-sort of depth 3 applied to 45321



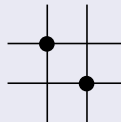
Shading and marking

Inversion in target and candidate

Target



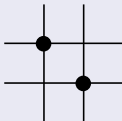
Candidate



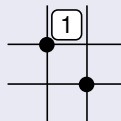
Shading and marking

Inversion in target and candidate

Target



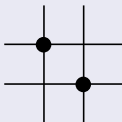
Candidate



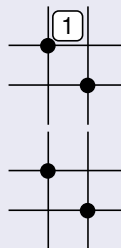
Shading and marking

Inversion in target and candidate

Target



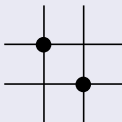
Candidate



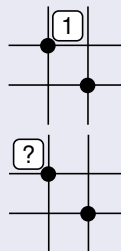
Shading and marking

Inversion in target and candidate

Target



Candidate



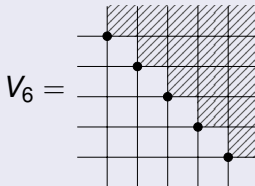
Filling the stack

- Need a sequence of decreasing elements
- The stack cannot be popped until last element has arrived

Filling the stack


- Need a sequence of decreasing elements
- The stack cannot be popped until last element has arrived

Example




Decorated patterns

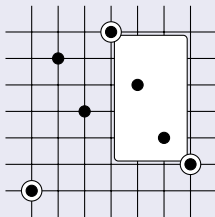
Decorated patterns (Úlfarsson, 2012) allow us to forbid occurrences of patterns inside boxes.

The pattern  occurs in the permutation 1647532

Decorated patterns

Decorated patterns (Úlfarsson, 2012) allow us to forbid occurrences of patterns inside boxes.


The pattern  occurs in the permutation 1647532

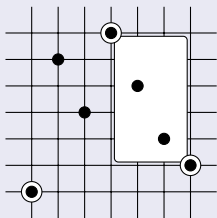


This is an occurrence

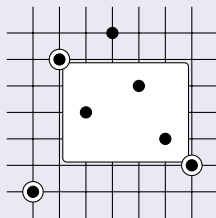
Decorated patterns

Decorated patterns (Úlfarsson, 2012) allow us to forbid occurrences of patterns inside boxes.

The pattern  occurs in the permutation 1647532




This is an occurrence



This is not an occurrence


Decorated patterns

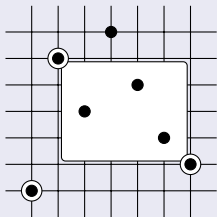
Decorated patterns also allow us to require occurrences of patterns inside boxes.

The pattern  occurs in the permutation 1647532

Decorated patterns

Decorated patterns also allow us to require occurrences of patterns inside boxes.


The pattern  occurs in the permutation 1647532

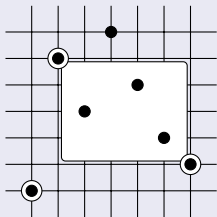


This is an occurrence

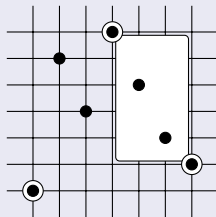
Decorated patterns

Decorated patterns also allow us to require occurrences of patterns inside boxes.

The pattern  occurs in the permutation 1647532



This is an occurrence

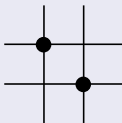


This is not an occurrence

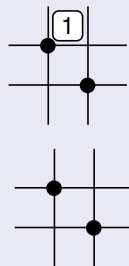
Shading and marking

Inversion in target and candidate

Target



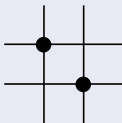
Candidate



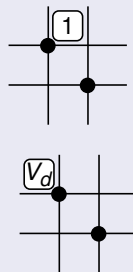
Shading and marking

Inversion in target and candidate

Target



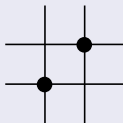
Candidate



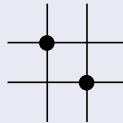
Shading and marking

Non-inversion in target and inversion in candidate

Target



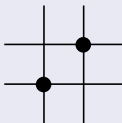
Candidate



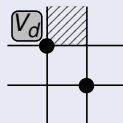
Shading and marking

Non-inversion in target and inversion in candidate

Target



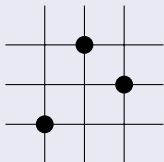
Candidate



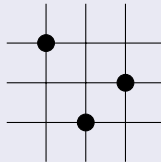
Example – Case 1

Apply meta patterns to inversions

Target



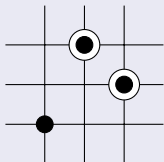
Candidate



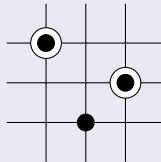
Example – Case 1

Apply meta patterns to inversions

Target



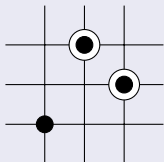
Candidate



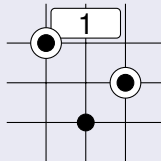
Example – Case 1

Apply meta patterns to inversions

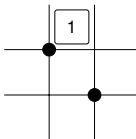
Target



Candidate



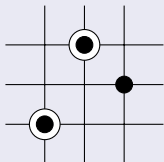
Inversion maintained – apply



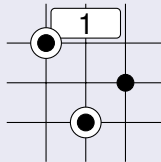
Example – Case 1

Apply meta patterns to inversions

Target



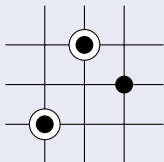
Candidate



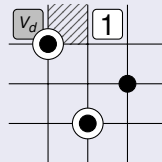
Example – Case 1

Apply meta patterns to inversions

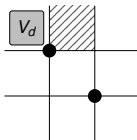
Target



Candidate



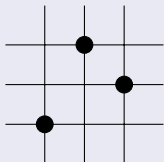
Inversion fixed – apply



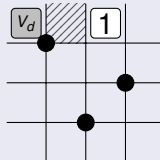
Example – Case 1

Apply meta patterns to inversions

Target



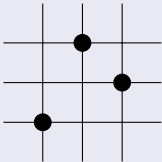
Candidate



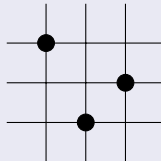
Example – Case 2

Apply meta patterns to inversions

Target



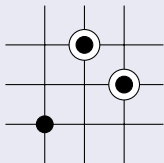
Candidate



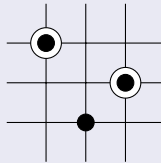
Example – Case 2

Apply meta patterns to inversions

Target



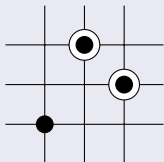
Candidate



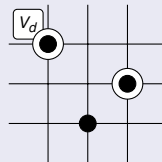
Example – Case 2

Apply meta patterns to inversions

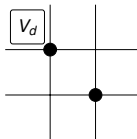
Target



Candidate



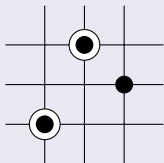
Inversion maintained – apply



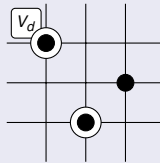
Example – Case 2

Apply meta patterns to inversions

Target



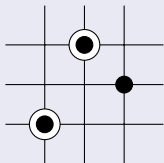
Candidate



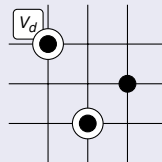
Example – Case 2

Apply meta patterns to inversions

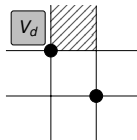
Target



Candidate



Inversion fixed – cannot apply

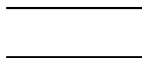


Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 **Preimage of queue sort**
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

An example

Queue-sorting 235416



235416

An example

Queue-sorting 235416

2

35416

An example

Queue-sorting 235416

23

5416

An example

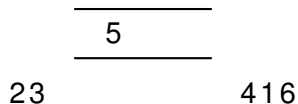
Queue-sorting 235416

235

416

An example

Queue-sorting 235416



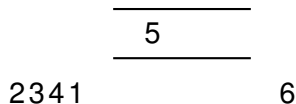
An example

Queue-sorting 235416

	<div><div>5</div></div>	
234		16

An example

Queue-sorting 235416



An example

Queue-sorting 235416

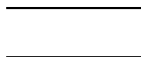
56

2341

An example

Queue-sorting 235416

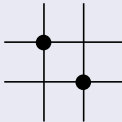
234156



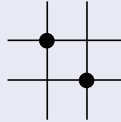
Shading and marking

Inversion in target and candidate

Target



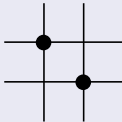
Candidate



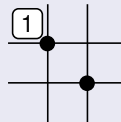
Shading and marking

Inversion in target and candidate

Target



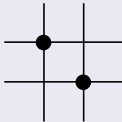
Candidate



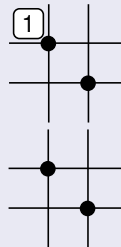
Shading and marking

Inversion in target and candidate

Target



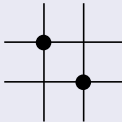
Candidate



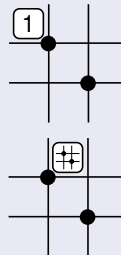
Shading and marking

Inversion in target and candidate

Target



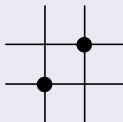
Candidate



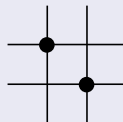
Shading and marking

Non-inversion in target and inversion in candidate

Target



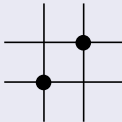
Candidate



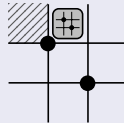
Shading and marking

Non-inversion in target and inversion in candidate

Target



Candidate



A linear time algorithm for avoidance

Theorem (✓ & Úlfarsson, 2012)

A permutation π avoids 4312 if and only if

$$(S \circ r \circ c \circ Q)(\pi) = \text{id}$$

A linear time algorithm for avoidance

Theorem (✓ & Úlfarsson, 2012)

A permutation π avoids 4312 if and only if

$$(S \circ r \circ c \circ Q)(\pi) = \text{id}$$

This theorem gives a linear time algorithm to check if 4312 (or any of its symmetries) occurs in a permutation.

Outline

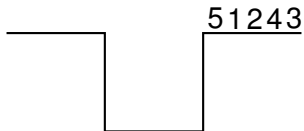
- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

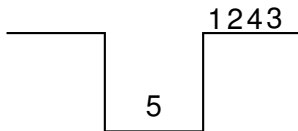
Another example

Pop-stack sort applied to 51243



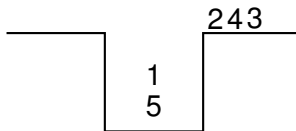
Another example

Pop-stack sort applied to 51243



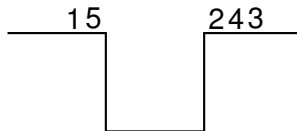
Another example

Pop-stack sort applied to 51243



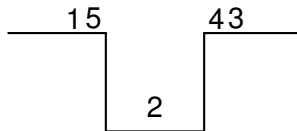
Another example

Pop-stack sort applied to 51243



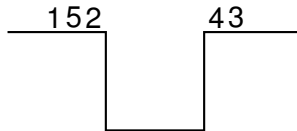
Another example

Pop-stack sort applied to 51243



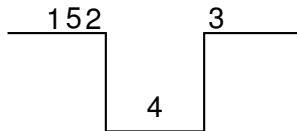
Another example

Pop-stack sort applied to 51243



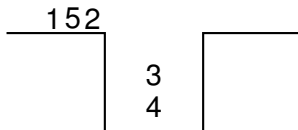
Another example

Pop-stack sort applied to 51243



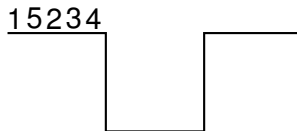
Another example

Pop-stack sort applied to 51243

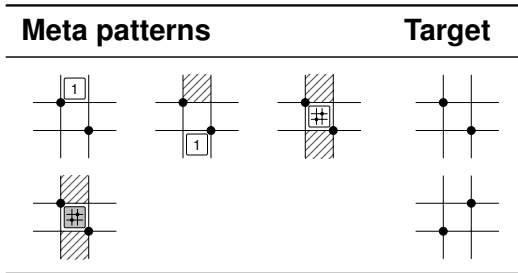


Another example

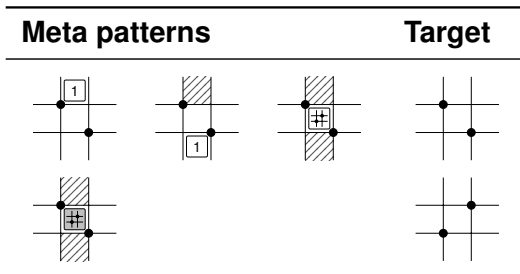
Pop-stack sort applied to 51243



Algorithm



Algorithm



Corollary

The pop-stack-sortable permutations are $\text{Av}(231, 312)$.

Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - **Insertion sort**
 - Pancake sort

Example

13524

Example

Locate first step down

13524

Example

Locate first step down

13524

Example

Move it to its proper place to the left

13524

Example

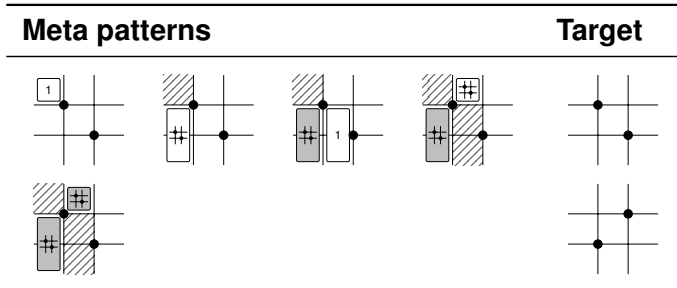
Move it to its proper place to the left

12354

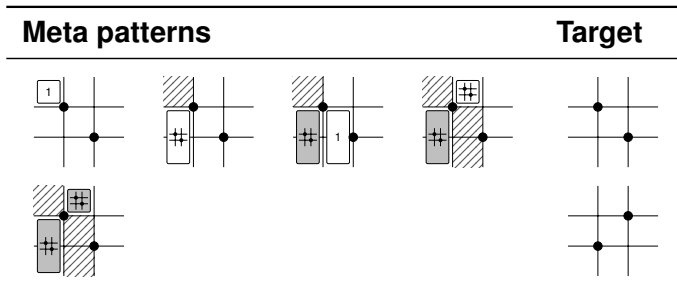
Example

12354

Algorithm



Algorithm



Corollary

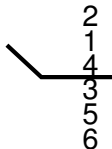
The insertion-sortable permutations are $Av(312, 321, 2143)$.

Outline

- 1 Introduction
 - Permutations and Patterns
 - Stack Sort
- 2 Preimage of stack sort
 - Stack of limited depth
- 3 Preimage of queue sort
- 4 Preimage of other sorting operators
 - Pop-stack sort
 - Insertion sort
 - Pancake sort

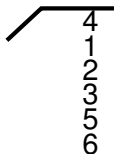
Another example

Pancake sort applied to 214356



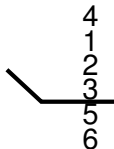
Another example

Pancake sort applied to 214356



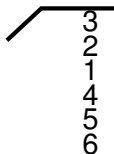
Another example

Pancake sort applied to 214356



Another example

Pancake sort applied to 214356



Another example

Pancake sort applied to 214356

3
2
1
4
5
6

Algorithm

- Things get a little complicated
- Non-inversions can become inversions
- We need 17 meta patterns to describe the algorithm

Algorithm

- Things get a little complicated
- Non-inversions can become inversions
- We need 17 meta patterns to describe the algorithm

Corollary

The pancake-sortable permutations are $Av(132, 312, 3241)$.

