A new and faster method of sorting by transpositions

Maxime Benoît-Gagné et Sylvie Hamel

Département d'Informatique et de Recherche Opérationnelle
Université de Montréal

Combinatorial Pattern Matching 2007

Outline

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1\pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1 \pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

Example

$$\pi \quad = \quad 6 \quad 3 \quad 1 \quad 2 \quad 4 \quad 5$$

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1 \pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

Example

$$\pi \quad = \quad 6 \quad 3 \quad \underbrace{1 \quad 2} \quad 4 \quad 5$$

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1\pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

Example

$$\pi \quad = \quad 6 \quad 3 \quad 1 \quad 2 \quad 4 \quad 5$$

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1 \pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

Example

$$\pi = 6\ 3\ 1\ 2\ 4\ 5$$

$$\rho \cdot \pi = 1\ 2\ 6\ 3\ 4\ 5$$

Sorting a permutation by transpositions

Definition

A **transposition** ρ is an operation, on a permutation $\pi = \pi_1\pi_2 \dots \pi_n$, that moves a block of contiguous elements and place it elsewhere.

The Problem

The **transposition distance problem** consists in finding the minimal number of transpositions needed to sequentially transform any permutation $\pi = \pi_1\pi_2 \dots \pi_n$ into the identity permutation $\mathrm{Id} = 12 \dots n$. This transposition distance is denoted $d(\pi)$.

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

A biological origin for the problem

2 DNA sequences containing the same n genes, in some different orders, are each represented by a permutation.

Hypothesis:

- Same set of genes
- No gene duplications

Estimate the evolutionary distance between two different organisms: finding out which evolutionary events are more probable to have modified one genome into another.

Genome Rearrangement

Definition

The **parsimonious transposition distance** between two species is the minimal number of transpositions needed to change one gene order into the other one.

Hypothesis

- The genomes have only one chromosome.
- There are only transpositions (so nor inversions nor inverse transpositions).
- The shortest sequence of rearrangements is the more probable.

Definition

The **parsimonious transposition distance** between two species is the minimal number of transpositions needed to change one gene order into the other one.

Hypothesis

- The genomes have only one chromosome.
- There are only transpositions (so nor inversions nor inverse transpositions).
- The shortest sequence of rearrangements is the more probable.

Definition

The **parsimonious transposition distance** between two species is the minimal number of transpositions needed to change one gene order into the other one.

Hypothesis

- The genomes have only one chromosome.
- There are only transpositions (so nor inversions nor inverse transpositions).
- The shortest sequence of rearrangements is the more probable.

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

Why are we interested in this problem?

Complexity:

- It is not known yet if this problem has polynomial complexity.
- Therefore, it is a nice theoretical computer science problem to explore

Beauty of the combinatorial problem

What more can be said....

The other approaches

The other approaches

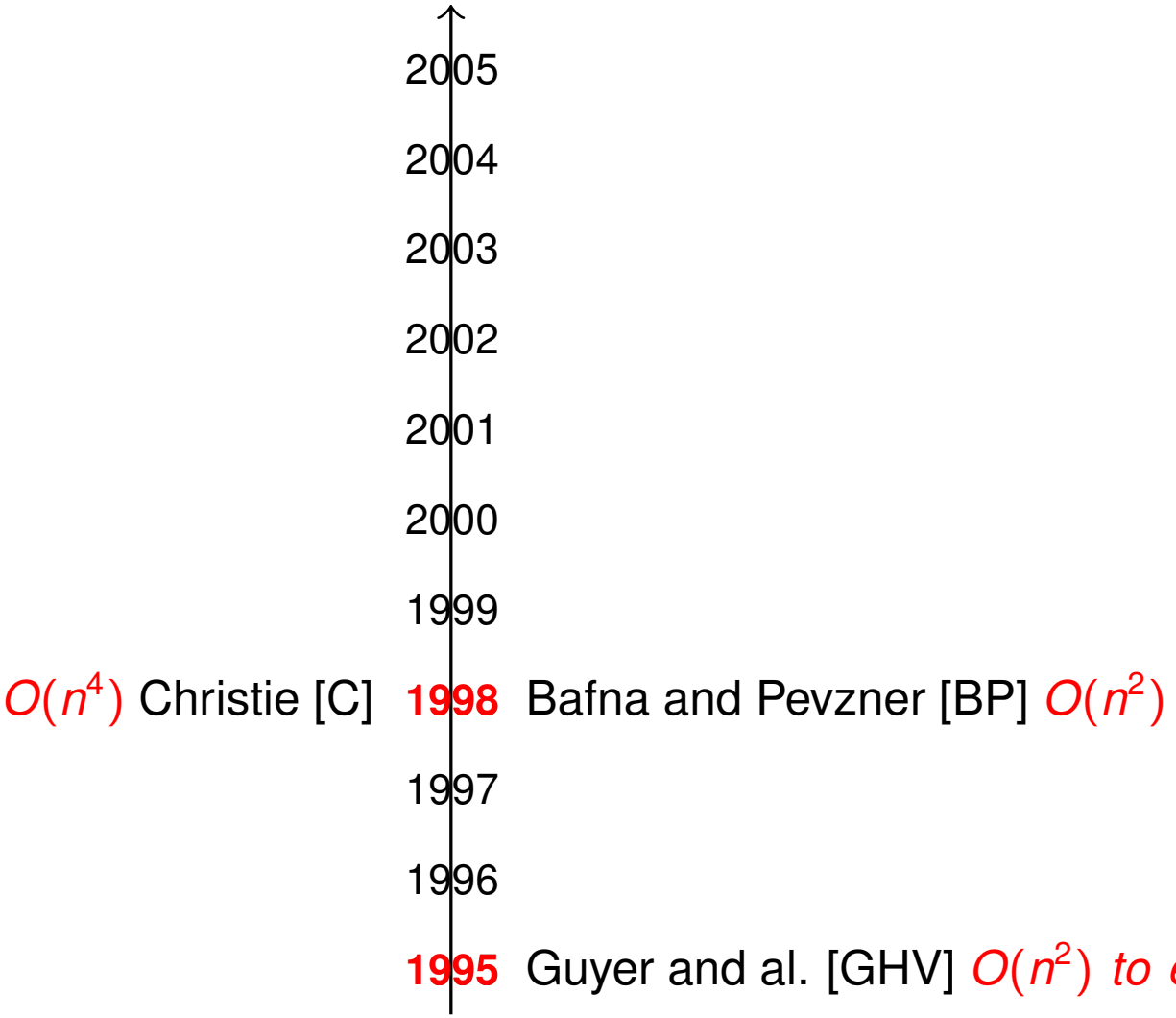
Guyer and al. [GHV] $O(n^2)$ *to exp.*

The other approaches

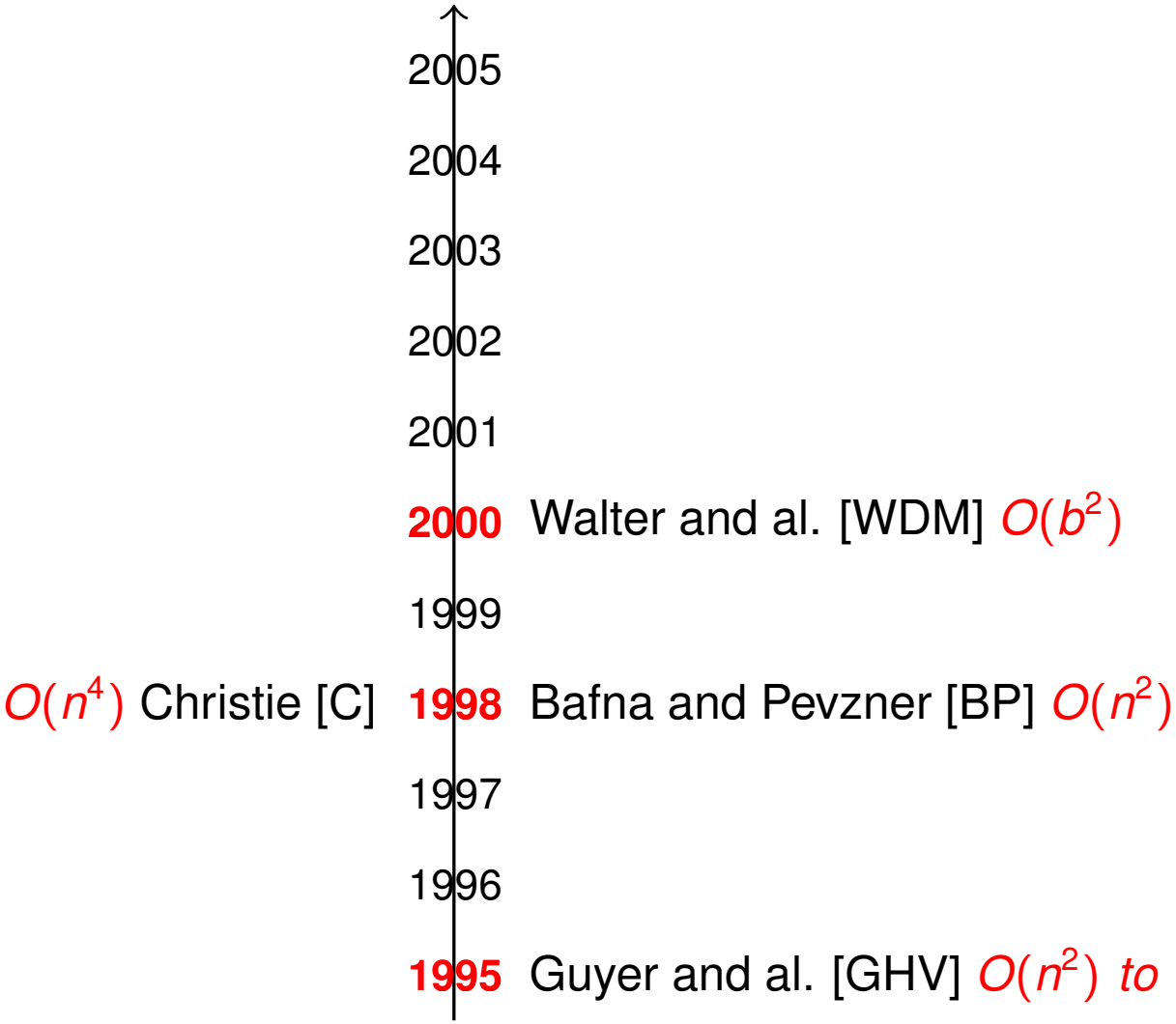
1998 Bafna and Pevzner [BP] $O(n^2)$

1995 Guyer and al. [GHV] $O(n^2)$ *to exp.*

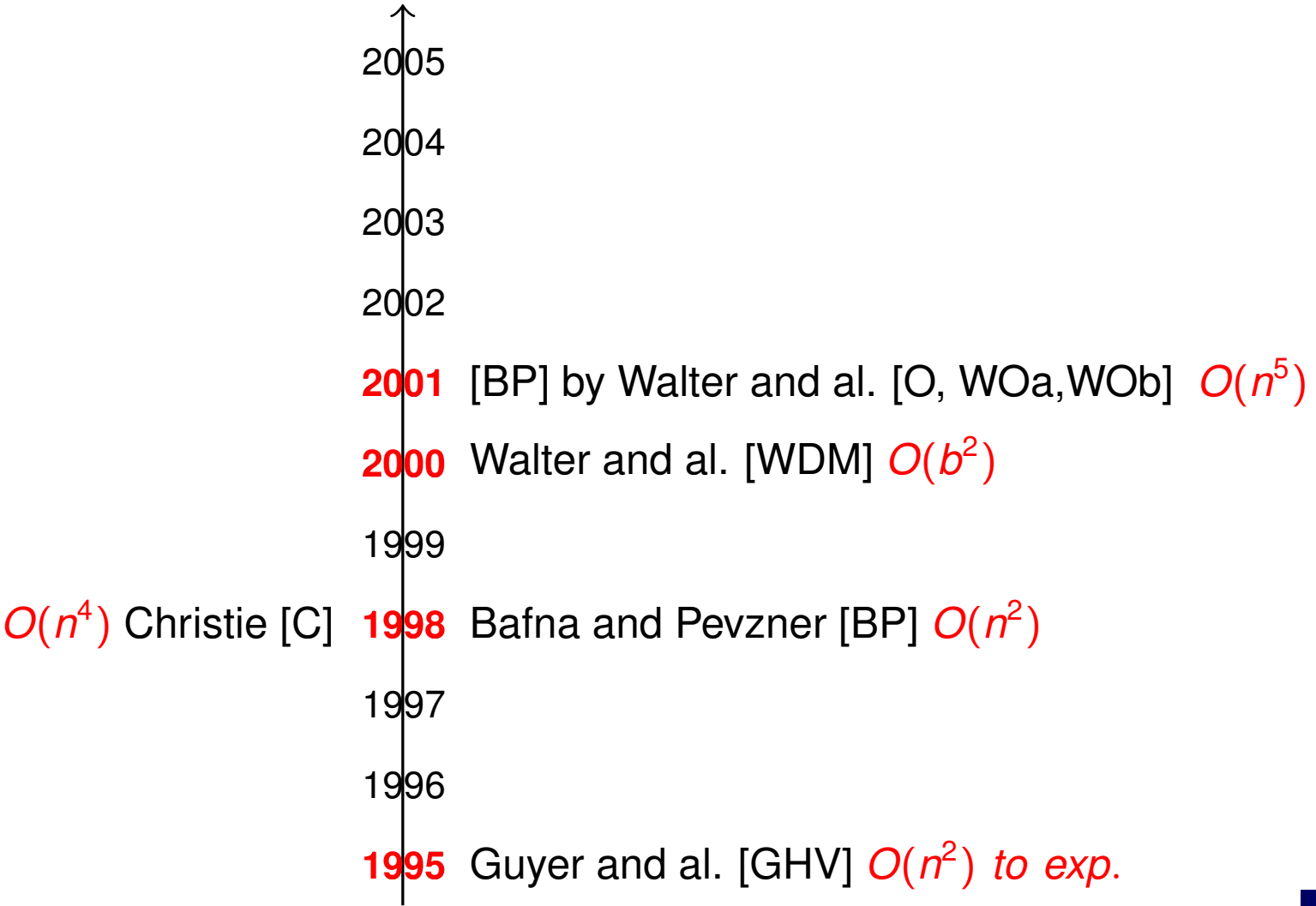
The other approaches



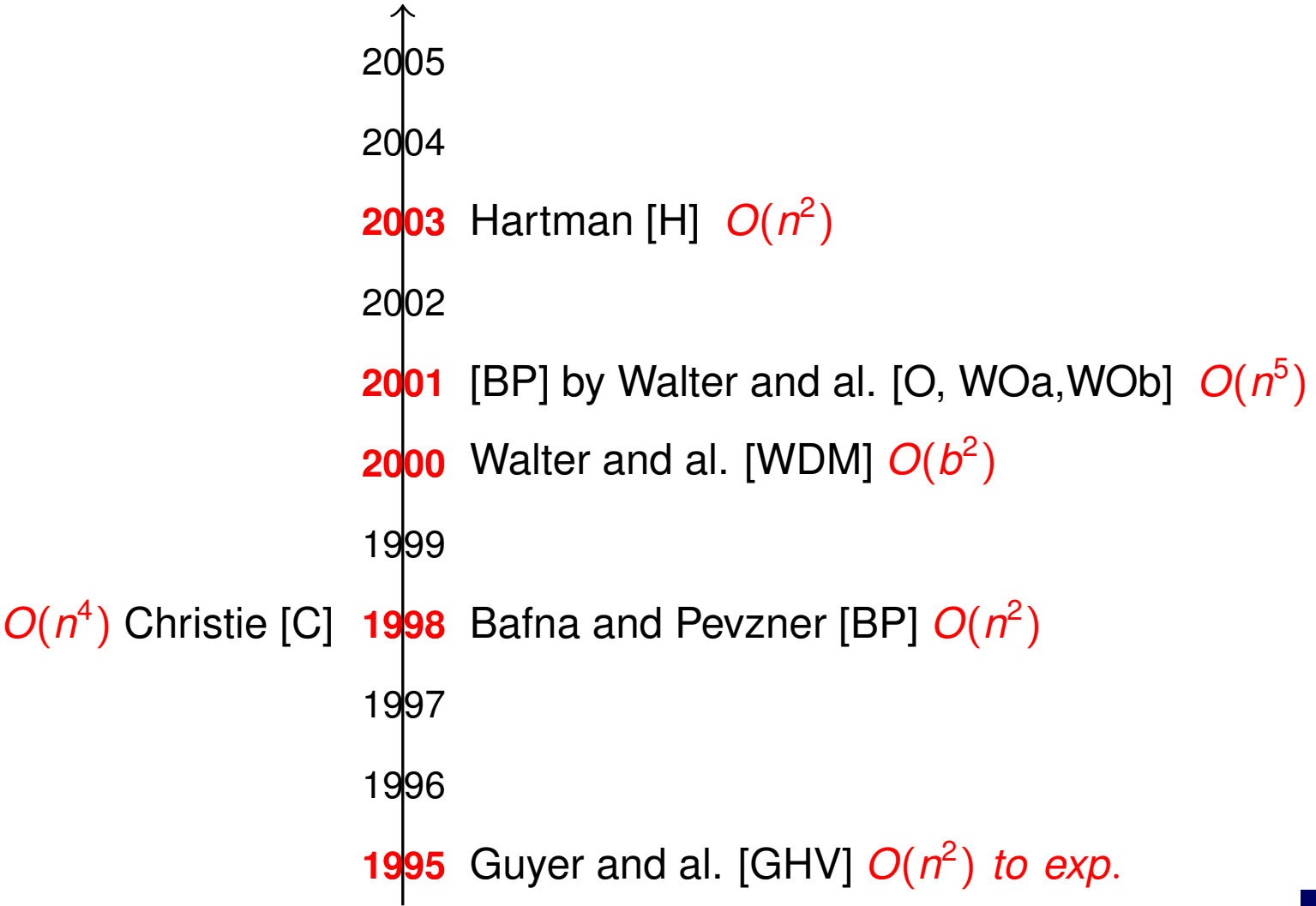
The other approaches



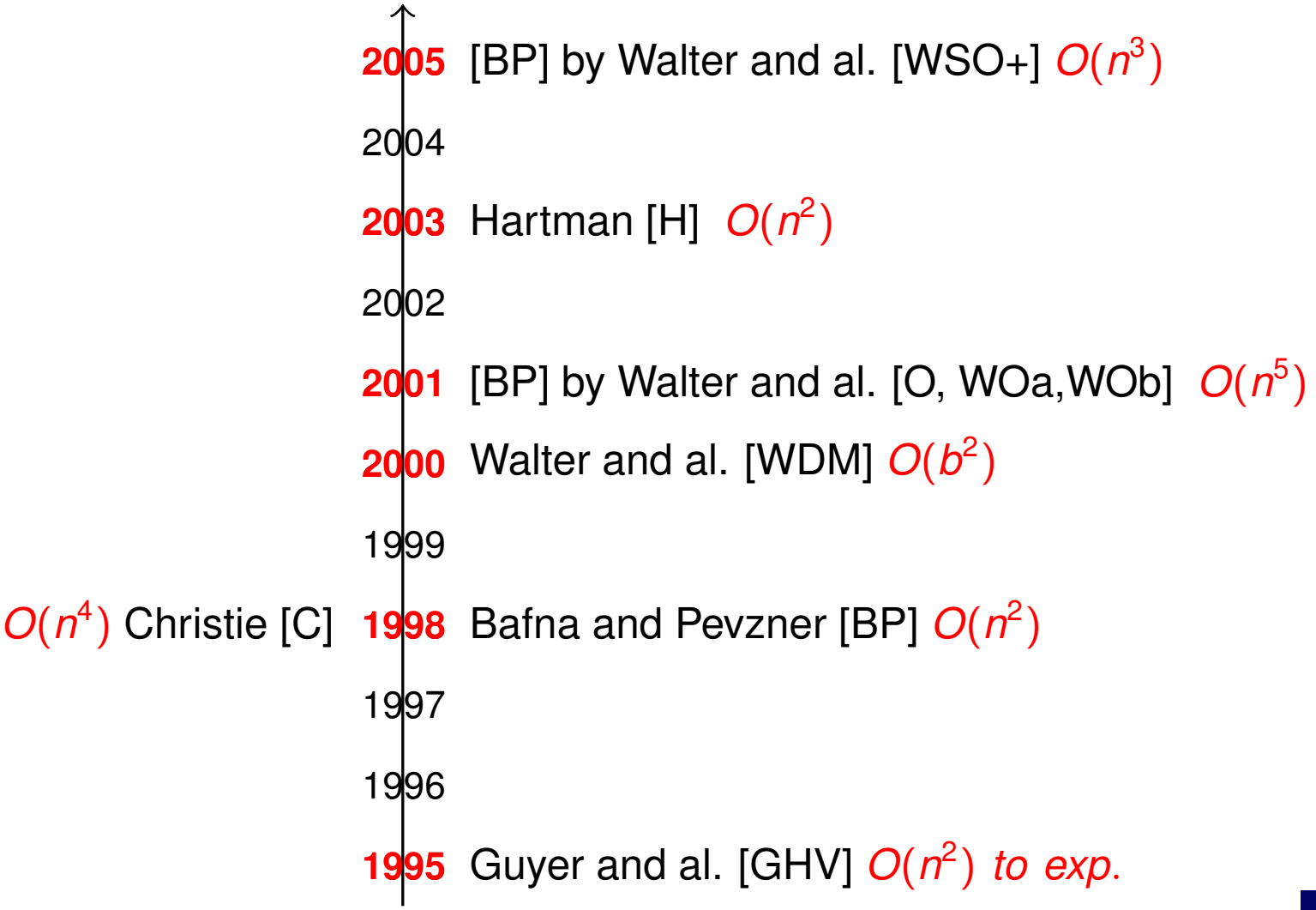
The other approaches



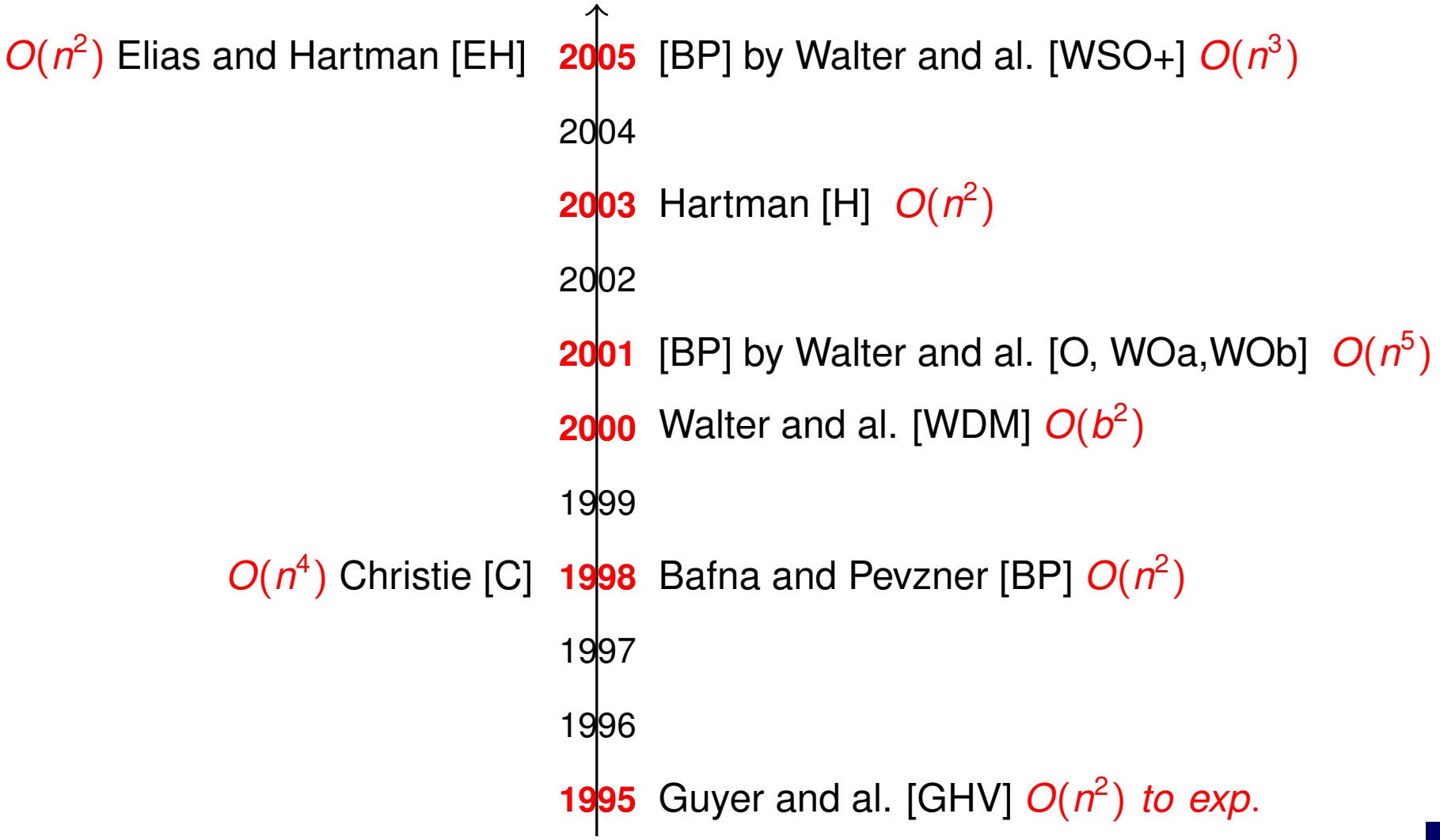
The other approaches



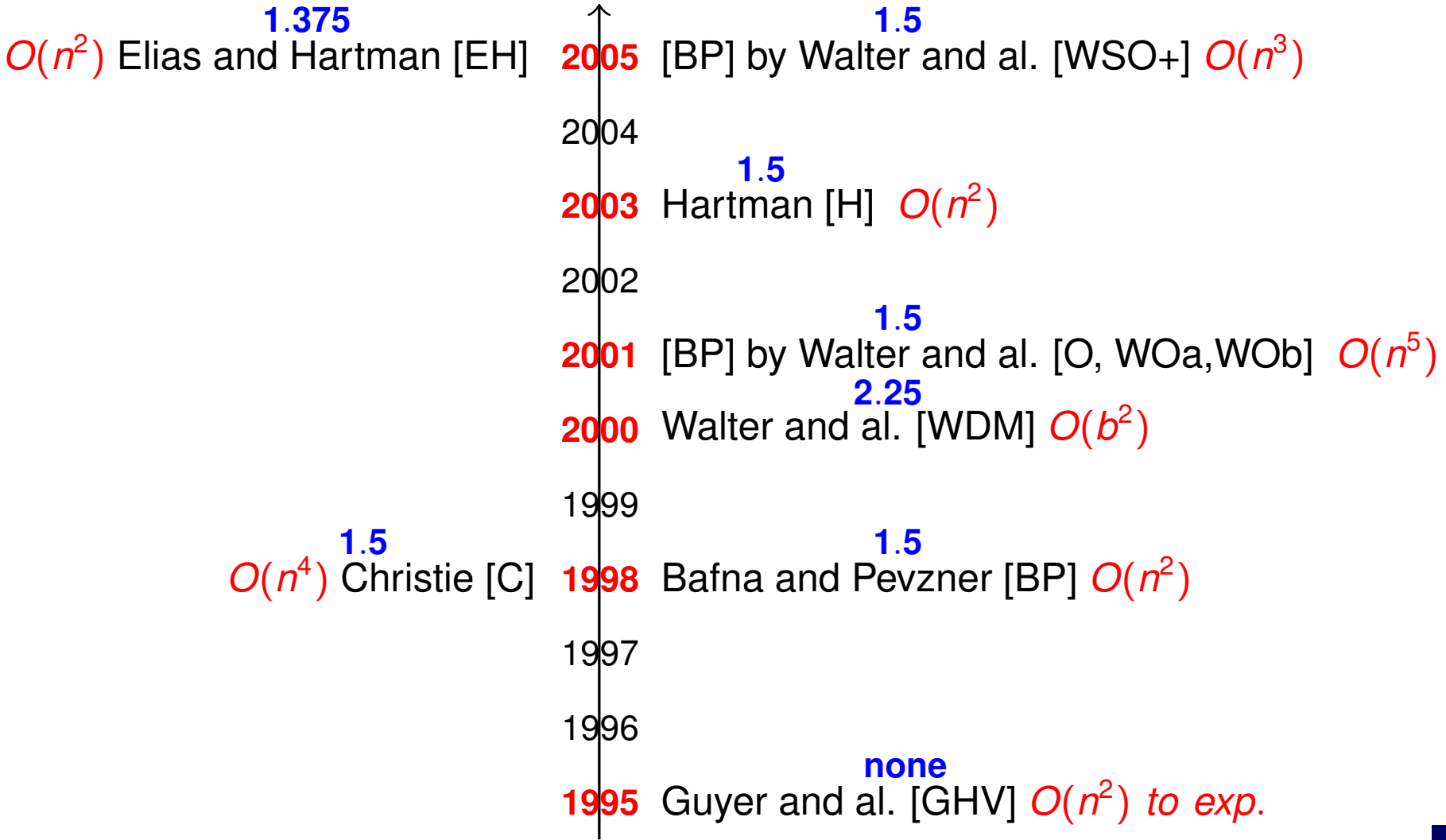
The other approaches



The other approaches



The other approaches



The cycle graph

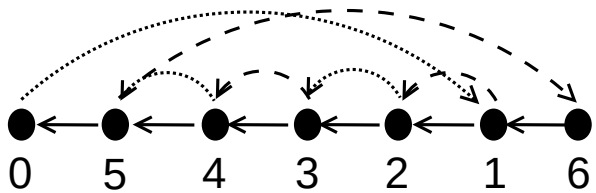
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

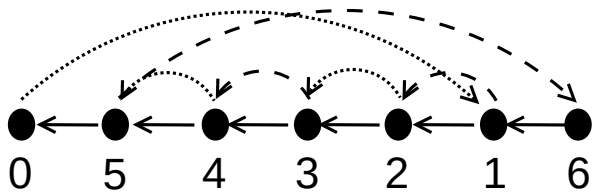
The cycle graph

Properties of cycles:

- oriented or not
- Crossing or non-interfering
- Short or long

BP2 [BP]
Approx. factor = 2

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

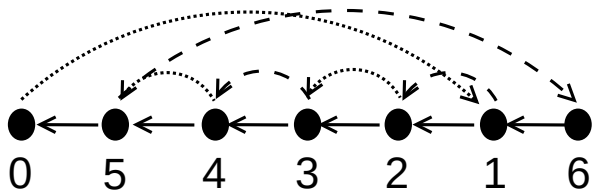
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

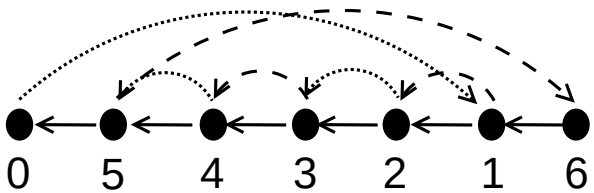
TransSort [BP]
Approx. factor = 1.5

The cycle graph

Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2



- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75

Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

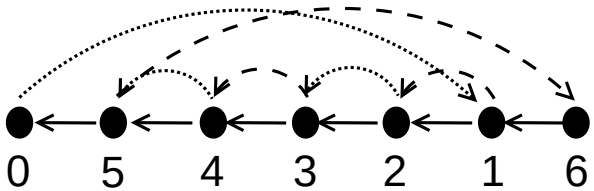
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

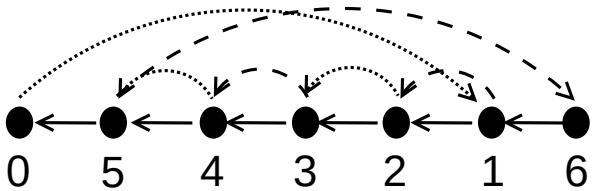
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

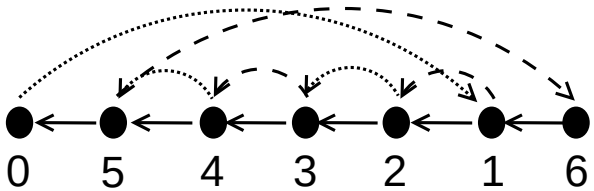
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

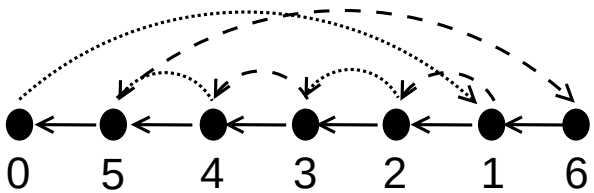
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

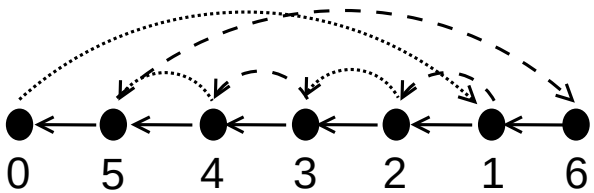
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

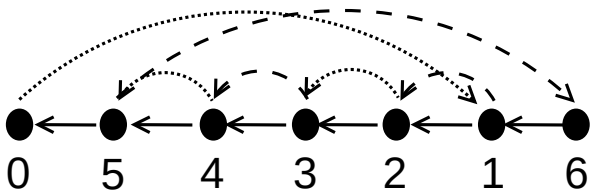
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

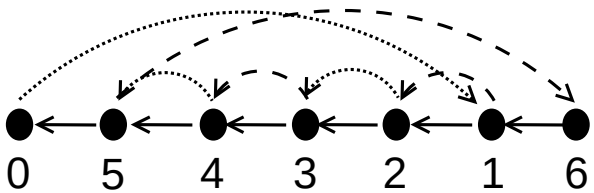
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

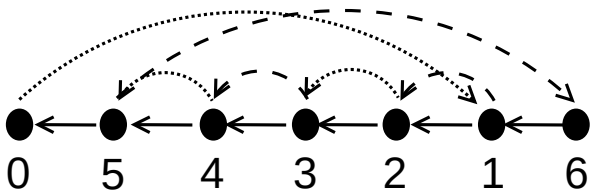
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

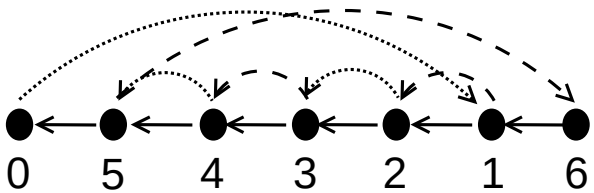
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

The cycle graph

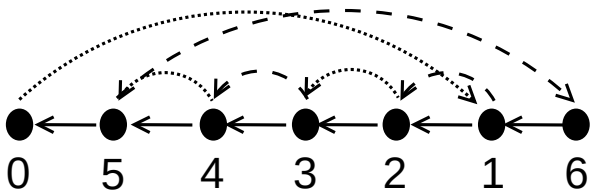
Properties of cycles:

- oriented or not

BP2 [BP]
Approx. factor = 2

- Crossing or non-interfering
- Short or long

TSort [BP]
Approx. factor = 1.75



Other properties:

- Odd or even
- Strongly oriented
- Strongly non-interfering
- Strongly-crossing

⋮

TransSort [BP]
Approx. factor = 1.5

Coding the permutation

For a given permutation π, we can compute two **codes**.

From these codes, we can derive an approximate transposition distance from the permutation.

Intuitive definitions of the codes

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Coding the permutation

For a given permutation π, we can compute two **codes**.

From these codes, we can derive an approximate transposition distance from the permutation.

Intuitive definitions of the codes

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Coding the permutation

For a given permutation π, we can compute two **codes**.

From these codes, we can derive an approximate transposition distance from the permutation.

Intuitive definitions of the codes

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Coding the permutation

For a given permutation π, we can compute two **codes**.

From these codes, we can derive an approximate transposition distance from the permutation.

Intuitive definitions of the codes

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Coding the permutation

For a given permutation π, we can compute two **codes**.

From these codes, we can derive an approximate transposition distance from the permutation.

Intuitive definitions of the codes

For a position *i* in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

For a position *i* in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Example of left and right codes

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position *i* in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$lc(\pi) =$

$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$

Intuitive definition

For a position *i* in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$\begin{array}{rcccccccc} lc(\pi) & = & 0 & 1 & & & & & \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 & \end{array} \qquad \begin{array}{rcccccc} \mathrm{Id} & = & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0\ 1$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0\ 1$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$lc(\pi) = 0\ 1\ 2$$

$$\pi = 6\ 3\ 2\ 1\ 4\ 5 \qquad \mathrm{Id} = 1\ 2\ 3\ 4\ 5\ 6$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Left code

$$\begin{array}{rcccccc} lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad \begin{array}{rcccccc} lc(\mathrm{Id}) & = & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathrm{Id} & = & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Intuitive definition

For a position i in a permutation π, the **left code** at this position is the number of elements bigger than π_i to its left.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Right code

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad \begin{array}{rcccccc} rc(\mathrm{Id}) & = & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathrm{Id} & = & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Intuitive definition

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Our goal

Raising the number of zeros in either the left or right code of π.

Example of left and right codes

Right code

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad \begin{array}{rcccccc} rc(\mathrm{Id}) & = & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathrm{Id} & = & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

Intuitive definition

For a position i in a permutation π, the **right code** at this position is the number of elements smaller than π_i to its right.

Our goal

Raising the number of zeros in either the left or right code of π.

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$rc(\pi)$	$=$	5	2	1	0	0	0
$lc(\pi)$	$=$	0	1	2	3	1	1
π	$=$	6	3	2	1	4	5

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad p(rc(\pi)) = 3$$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad p(rc(\pi)) = 3$$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5
\end{array}
\qquad
\begin{array}{l}
p(rc(\pi)) = 3 \\
p(lc(\pi)) = 4
\end{array}
$$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array} \qquad \begin{array}{l} p(rc(\pi)) = 3 \\ p(lc(\pi)) = 4 \end{array}$$

Definition

$p(\pi) = min\{p(lc(\pi)), p(rc(\pi))\}$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$p(\pi) = 3$$

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$$p(rc(\pi)) = 3$$

$$p(lc(\pi)) = 4$$

Definition

$p(\pi) = min\{p(lc(\pi)), p(rc(\pi))\}$

Plateau

Definition

Let us call **plateau** any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code c is denoted $p(c)$.

$$p(\pi) = 3$$

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$$p(rc(\pi)) = 3$$

$$p(lc(\pi)) = 4$$

Definition

$p(\pi) = min\{p(lc(\pi)), p(rc(\pi))\}$

$$p(\pi) = 0 \iff \pi = \mathrm{Id}$$

Removing a plateau

Lemma 1

Given a permutation $\pi = \pi_1 \dots \pi_n$, the leftmost plateau of $lc(\pi)$ can be removed by a transposition to the left without creating any new plateau in the code.

Example

$$\begin{array}{rcccccc} lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$$\downarrow \rho(2,3,1)$$

$$\begin{array}{cccccc} 0 & 0 & 2 & 3 & 1 & 1 \\ 3 & 6 & 2 & 1 & 4 & 5 \end{array}$$

Lemma 2

$d(\pi) \leq p(\pi)$

Removing a plateau

Lemma 1

Given a permutation $\pi = \pi_1 \dots \pi_n$, the leftmost plateau of $lc(\pi)$ can be removed by a transposition to the left without creating any new plateau in the code.

Example

$$\begin{array}{rcccccc} lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$\downarrow \rho(2,3,1)$

$$\begin{array}{cccccc} 0 & 0 & 2 & 3 & 1 & 1 \\ 3 & 6 & 2 & 1 & 4 & 5 \end{array}$$

Lemma 2

$d(\pi) \leq p(\pi)$

Removing a plateau

Lemma 1

Given a permutation $\pi = \pi_1 \dots \pi_n$, the leftmost plateau of $lc(\pi)$ can be removed by a transposition to the left without creating any new plateau in the code.

Example

$$\begin{array}{rcccccc} lc(\pi) & = & 0 & 1 & 2 & 3 & 1 & 1 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$\downarrow \rho(2,3,1)$

$$\begin{array}{cccccc} 0 & 0 & 2 & 3 & 1 & 1 \\ 3 & 6 & 2 & 1 & 4 & 5 \end{array}$$

Lemma 2

$d(\pi) \leq p(\pi)$

Algorithm EasySorting

Algorithm EasySorting
(input: π a permutation of $[n]$)

$lc(\pi) =$ left code of π
$rc(\pi) =$ right code of π
$lp(\pi) = p(lc(\pi))$
$rp(\pi) = p(rc(\pi))$
RETURN $p(\pi) = \min\{lp(\pi), rp(\pi)\}$

$rc(\pi)$	$=$	5	2	1	0	0	0
$lc(\pi)$	$=$	0	1	2	3	1	1
π	$=$	6	3	2	1	4	5

$rp(\pi) = 3$
$lp(\pi) = 4$
$p(\pi) = 3$

Algorithm EasySorting

Algorithm EasySorting
(input: π a permutation of $[n]$)

$lc(\pi) =$ left code of π
$rc(\pi) =$ right code of π
$lp(\pi) = p(lc(\pi))$
$rp(\pi) = p(rc(\pi))$
RETURN $p(\pi) = \min\{lp(\pi), rp(\pi)\}$

To list the $p(\pi)$ transpositions needed to sort π, while computing $p(lc(\pi))$ and $p(rc(\pi))$, we need only to record both ends (start and finish) of the plateaux considered as well as their code values.

Algorithm EasySorting

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

Algorithm EasySorting

$$\begin{array}{rcccccc} rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\ \pi & = & 6 & 3 & 2 & 1 & 4 & 5 \end{array}$$

$$\downarrow\rho(3,4,5)$$

Algorithm EasySorting

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5 \\
\multicolumn{7}{c}{\downarrow \rho(3,4,5)} \\
 & & 5 & 2 & 0 & 0 & 0 & 0 \\
 & & 6 & 3 & 1 & 2 & 4 & 5
\end{array}
$$

Algorithm EasySorting

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5
\end{array}
$$

$\downarrow \rho(3,4,5)$

$$
\begin{array}{cccccc}
5 & 2 & 0 & 0 & 0 & 0 \\
6 & 3 & 1 & 2 & 4 & 5
\end{array}
$$

$\downarrow \rho(2,3,5)$

Algorithm EasySorting

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5
\end{array}
$$

$\downarrow \rho(3,4,5)$

$$
\begin{array}{cccccc}
5 & 2 & 0 & 0 & 0 & 0 \\
6 & 3 & 1 & 2 & 4 & 5
\end{array}
$$

$\downarrow \rho(2,3,5)$

$$
\begin{array}{cccccc}
5 & 0 & 0 & 0 & 0 & 0 \\
6 & 1 & 2 & 3 & 4 & 5
\end{array}
$$

Algorithm EasySorting

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5
\end{array}
$$

$\downarrow \rho(3,4,5)$

$$
\begin{array}{cccccc}
5 & 2 & 0 & 0 & 0 & 0 \\
6 & 3 & 1 & 2 & 4 & 5
\end{array}
$$

$\downarrow \rho(2,3,5)$

$$
\begin{array}{cccccc}
5 & 0 & 0 & 0 & 0 & 0 \\
6 & 1 & 2 & 3 & 4 & 5
\end{array}
$$

$\downarrow \rho(1,2,7)$

Algorithm EasySorting

$$
\begin{array}{rcccccc}
rc(\pi) & = & 5 & 2 & 1 & 0 & 0 & 0 \\
\pi & = & 6 & 3 & 2 & 1 & 4 & 5 \\
\downarrow\rho(3,4,5) & & & & & & & \\
 & & 5 & 2 & 0 & 0 & 0 & 0 \\
 & & 6 & 3 & 1 & 2 & 4 & 5 \\
\downarrow\rho(2,3,5) & & & & & & & \\
 & & 5 & 0 & 0 & 0 & 0 & 0 \\
 & & 6 & 1 & 2 & 3 & 4 & 5 \\
\downarrow\rho(1,2,7) & & & & & & & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & 1 & 2 & 3 & 4 & 5 & 6
\end{array}
$$

Complexity and Approximation Ratio

Algorithm EasySorting (input: π a permutation of $[n]$)

$lc(\pi) =$ left code of π (easily computed in $\mathcal{O}(n^2)$)
$rc(\pi) =$ right code of π (easily computed in $\mathcal{O}(n^2)$)
$lp(\pi) = p(lc(\pi))$(easily computed in $\mathcal{O}(n)$)
$rp(\pi) = p(rc(\pi))$(easily computed in $\mathcal{O}(n)$)
RETURN $p(\pi) = \min\{lp(\pi), rp(\pi)\}$

The complexity of EasySorting is in $O(n^2)$.

The approximation ratio of EasySorting is 3.

Complexity and Approximation Ratio

Algorithm EasySorting (input: π a permutation of $[n]$)

$lc(\pi) =$ left code of π (easily computed in $\mathcal{O}(n^2)$)
$rc(\pi) =$ right code of π (easily computed in $\mathcal{O}(n^2)$)
$lp(\pi) = p(lc(\pi))$(easily computed in $\mathcal{O}(n)$)
$rp(\pi) = p(rc(\pi))$(easily computed in $\mathcal{O}(n)$)
RETURN $p(\pi) = \min\{lp(\pi), rp(\pi)\}$

The complexity of EasySorting is in $O(n^2)$.

The approximation ratio of EasySorting is 3.

Complexity and Approximation Ratio

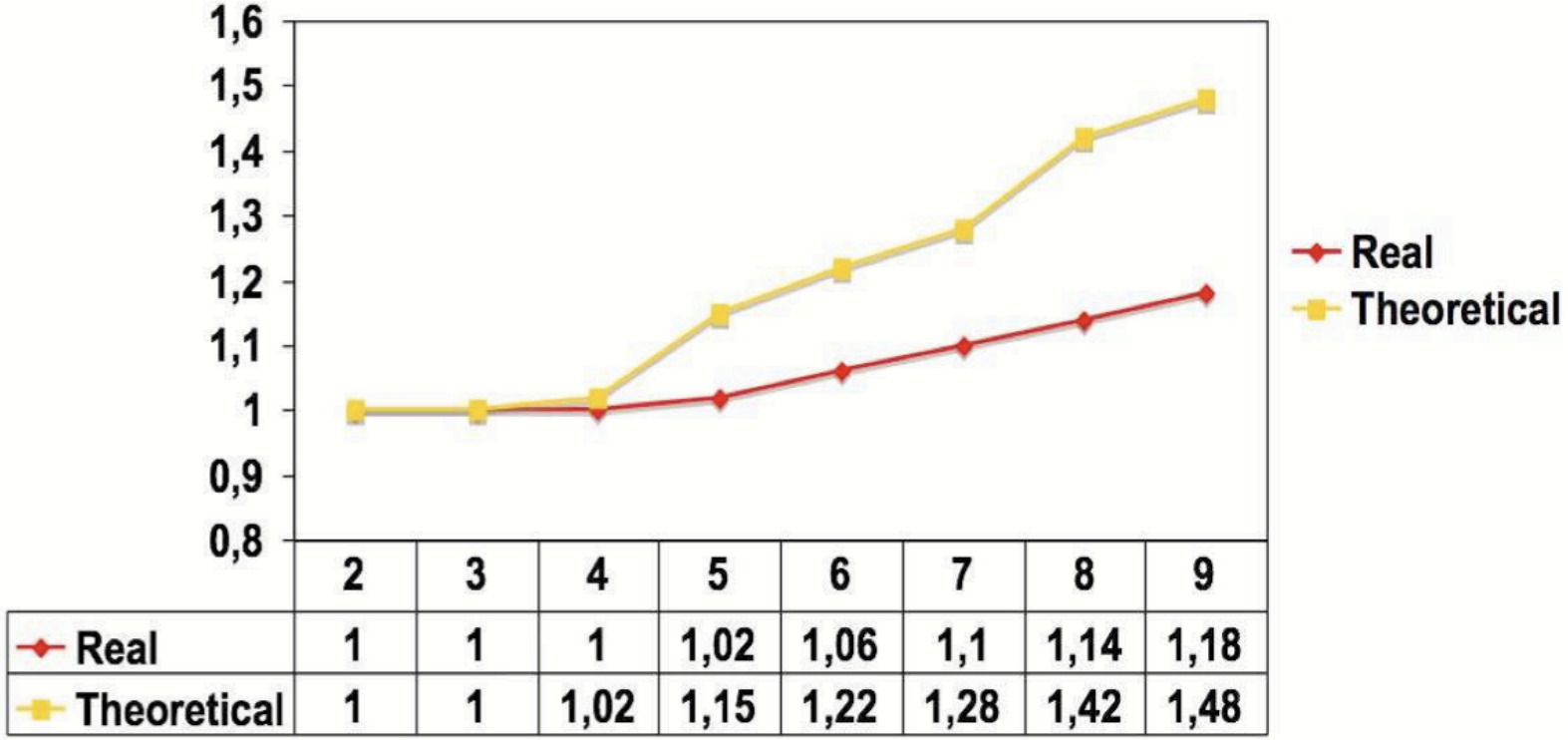
Algorithm EasySorting (input: π a permutation of $[n]$)

$lc(\pi) =$ left code of π (easily computed in $\mathcal{O}(n^2)$)
$rc(\pi) =$ right code of π (easily computed in $\mathcal{O}(n^2)$)
$lp(\pi) = p(lc(\pi))$(easily computed in $\mathcal{O}(n)$)
$rp(\pi) = p(rc(\pi))$(easily computed in $\mathcal{O}(n)$)
RETURN $p(\pi) = \min\{lp(\pi), rp(\pi)\}$

The complexity of EasySorting is in $O(n^2)$.

The approximation ratio of EasySorting is 3.

Complexity and Approximation Ratio



	2	3	4	5	6	7	8	9
Real	1	1	1	1,02	1,06	1,1	1,14	1,18
Theoretical	1	1	1,02	1,15	1,22	1,28	1,42	1,48

Experimental results

Comparison of the number of wrong calculations on small permutations

		EasySort			BP2
n	Total	$\neq d(\pi)$			$\neq d(\pi)$
2	2	0			0
3	6	0			0
4	24	0			1
5	120	6			7
6	720	108			86
7	5040	1423			792
8	40320	17577			9162
9	362880	211863			100332

Experimental results

Comparison of the mean of the transposition distances and computational time for large permutations

		EasySort		BP2	
n	sample size	distance	time	distance	time
10	1000	5,59	-	5,06	-
16	1000	10,63	-	8,50	-
32	1000	25,02	-	17,76	-
64	1000	55,45	-	36,09	0,00167
128	1000	117,96	0,00106	72,57	0,00484
256	1000	244,38	0,00255	145,18	0,02288
512	1000	498,91	0,00761	290,56	0,1160
1000	100	985,63	0,02874	567,53	1,299
5000	100	4982,05	0,5824	2840,89	144,8

Can we do better?

Can we do better?

EasySorting removes only 1 plateau by transposition.

Can we do better?

EasySorting removes only 1 plateau by transposition.

Can we remove more plateaux with each transposition?

Can we do better?

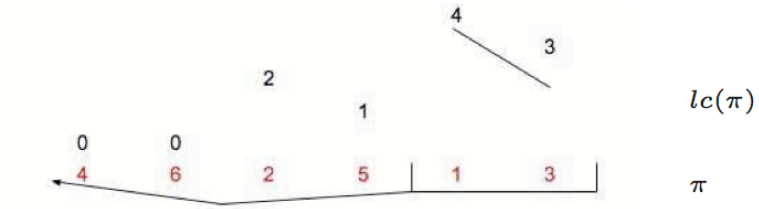
EasySorting removes only 1 plateau by transposition.

Can we remove more plateaux with each transposition?

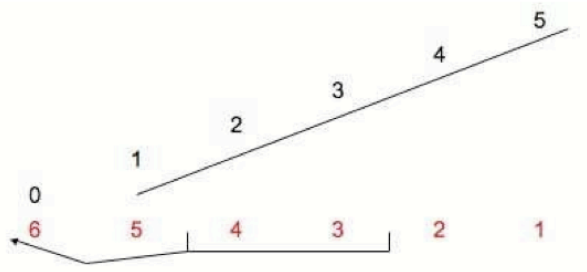
YES !!

Other entities

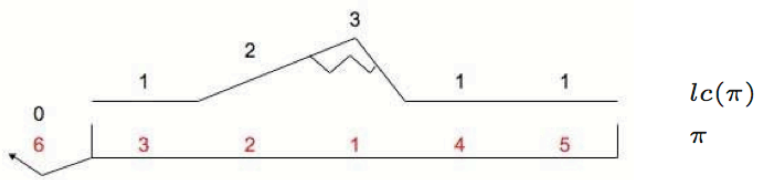
Some of the entities for the algorithms Mountain and MADFRP



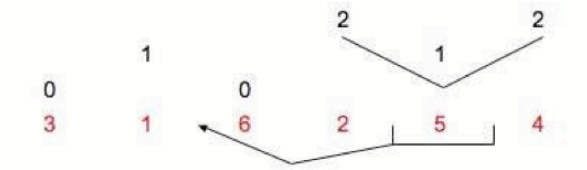
a) good fall



b) ascent



c) mountain



d) ditch

Experimental results

Comparison of the number of wrong calculations on small permutations

		EasySort	Mountain	MADFRP	BP2
n	Total	$\neq d(\pi)$	$\neq d(\pi)$	$\neq d(\pi)$	$\neq d(\pi)$
2	2	0	0	0	0
3	6	0	0	0	0
4	24	0	0	0	1
5	120	6	6	1	7
6	720	108	103	29	86
7	5040	1423	1314	484	792
8	40320	17577	15941	7416	9162
9	362880	211863	190528	102217	100332

Experimental results

Comparison of the mean of the transposition distances and computational time for large permutations

		Mountain		BP2	
n	sample size	distance	time	distance	time
10	1000	5,46	-	5,06	-
16	1000	10,11	-	8,50	-
32	1000	23,60	-	17,76	-
64	1000	52,50	-	36,09	0,00167
128	1000	113,29	0,00181	72,57	0,00484
256	1000	237,23	0,00529	145,18	0,02288
512	1000	488,57	0,0234	290,56	0,1160
1000	100	971,72	0,1152	567,53	1,299
5000	100	4958,75	4,448	2840,89	144,8

Advantages and drawbacks of our algorithms

Advantages of our algorithms

- Simplicity
- Quickness of execution

Drawbacks of our algorithms

- Bigger approximated transposition distances

Advantages and drawbacks of our algorithms

Advantages of our algorithms

- Simplicity
- Quickness of execution

Drawbacks of our algorithms

- Bigger approximated transposition distances

Advantages and drawbacks of our algorithms

Advantages of our algorithms

- Simplicity
- Quickness of execution

Drawbacks of our algorithms

- Bigger approximated transposition distances

Advantages and drawbacks of our algorithms

Advantages of our algorithms

- Simplicity
- Quickness of execution

Drawbacks of our algorithms

- Bigger approximated transposition distances

Advantages and drawbacks of our algorithms

Advantages of our algorithms

- Simplicity
- Quickness of execution

Drawbacks of our algorithms

- Bigger approximated transposition distances

Future Work

Future work and Open problems

Future Work

Future work and Open problems

- Treating the entities independently of their kind from the ends of the permutation to the center instead of treating the kinds of entities one by one.
- Implementing the entities (other than plateaux) with the left **and** right codes.

Future Work

Future work and Open problems

- Treating the entities independently of their kind from the ends of the permutation to the center instead of treating the kinds of entities one by one.
- Implementing the entities (other than plateaux) with the left **and** right codes.

General open problems about sorting by transpositions

Future Work

Future work and Open problems

- Treating the entities independently of their kind from the ends of the permutation to the center instead of treating the kinds of entities one by one.
- Implementing the entities (other than plateaux) with the left **and** right codes.

General open problems about sorting by transpositions

- Finding the class of complexity of this problem.
- Finding the transposition diameter in function of n.

Thank you !!

For Further Reading

- [BP] V. Bafna and P. A. Pevzner, Sorting by transpositions, *SIAM Journal on Discrete Maths*, 11(2): 224-240, 1998.

- [C] D. A. Christie, Genome rearrangements problems, PhD thesis, Glasgow University, 1998.

- [EH] I. Elias and T. Hartman, A 1.375-approximation algorithm for sorting by transpositions, Lecture Notes in Bioinformatics 3692, 204-215, 2005.

-[GHV] S. A. Guyer and L. S. Heath and J. P. C. Vergara, Subsequence and run heuristics for sorting by transpositions, *4th DIMACS International Algorithm Implementation Challenge*, 1995.

- [H] T. Hartman, A simpler 1.5-approximation algorithm for sorting by transpositions, Lecture Notes in Computer Science 2676, 156-169, 2003.

- [O] E. T. G. Oliveira, Implementations of algorithms for the problem of sorting by transpositions, Master's thesis, University of Brasilia, 2001.

- [WDM] M. E. M. T. Walter and Z. Dias and J. Meidanis, A new approach for approximating the transposition distance, *Proceedings of SPIRE*, 199-208, 1995.

- [WOa] M. E. M. T. Walter and E. T. G. de Oliveira, Extending the theory of Bafna and Pevzner for the problem of sorting by transpositions, *CNMAC)*, volume 3(1), 213-222, 2002.

- [WOb] M. E. M. T. Walter and E. T. G. de Oliveira, The problem of sorting by transpositions and the algorithm of Bafna and Pevzner, *XXII Seminário Integrado de Software e Hardware (SEMISH)*, 2002.

- [WSO+] M. E. M. T. Walter *et al.*, Improving the algorithm of Bafna and Pevzner for the problem of sorting by transpositions: a practical approach, *Journal of Discrete Algorithms*, 3: 342-361, 2005.