# RSK bumping trees and a fast RSK algorithm

Vasilii Duzhin[1], Artem Kuzmin[1], Nikolay Vassiliev[1,2]

[1]Saint Petersburg Electrotechnical University "LETI"

[2]Saint Petersburg Department of Steklov Mathematical Institute of Russian Academy of Sciences

Polynomial Computer Algebra '19
April 20, 2019


# Overview

1. Robinson-Schensted-Knuth algorithm
2. Bumping routes, bumping trees & bumping forest
3. A fast RSK algorithm
4. The comparison of efficiency of RSK algorithms

# Robinson-Schensted-Knuth (RSK) correspondence

**The Robinson-Schensted-Knuth** correspondence is a bijection between a set of permutations of integers and a set of pairs of Young tableaux of the same shape: insertion tableau $P$ and recording tableau $Q$.

For example, the permutation

$$(13, 2, 16, 4, 7, 9, 12, 1, 3, 20, 11, 6, 18, 14, 5, 19, 17, 10, 8, 15)$$

corresponds to the following pair of Young tableaux:

$P$ :

| | | | | |
|---|---|---|---|---|
| 15 | | | | |
| 14 | | | | |
| 10 | 17 | | | |
| 8 | 9 | 19 | | |
| 5 | 6 | 18 | | |
| 3 | 4 | 11 | 16 | 20 |
| 1 | 2 | 7 | 12 | 13 |

$Q$ :

| | | | | |
|---|---|---|---|---|
| 16 | | | | |
| 10 | | | | |
| 7 | 17 | | | |
| 6 | 13 | 20 | | |
| 5 | 11 | 14 | | |
| 3 | 4 | 9 | 18 | 19 |
| 1 | 2 | 8 | 12 | 15 |

# RSK algorithm

A permutation $(s_1, s_2, ..., s_n)$ generates a tableau $P$ consisting of columns $(P_1, P_2, ..., P_m)$ of heights $(h(P_1), h(P_2), ..., h(P_m))$ as follows.

$P = s_1$

**For** $s_i, i \in [2, m]$:

**If** $s_{i-1} > \max\limits_{1 \le k \le h(P_i)} \{P_{i,k}\}$, $s_i \longrightarrow$ the top of $P_i$. $i = i + 1$.

**Else** $k = \min\limits_{1 \le k \le h(P_i)} \{P_{i,k} > s_i\}$; $s_i \longrightarrow P_{i,k}$. $P_{i,k}$ is bumped to the column $P_{i+1}$ by the same principle.

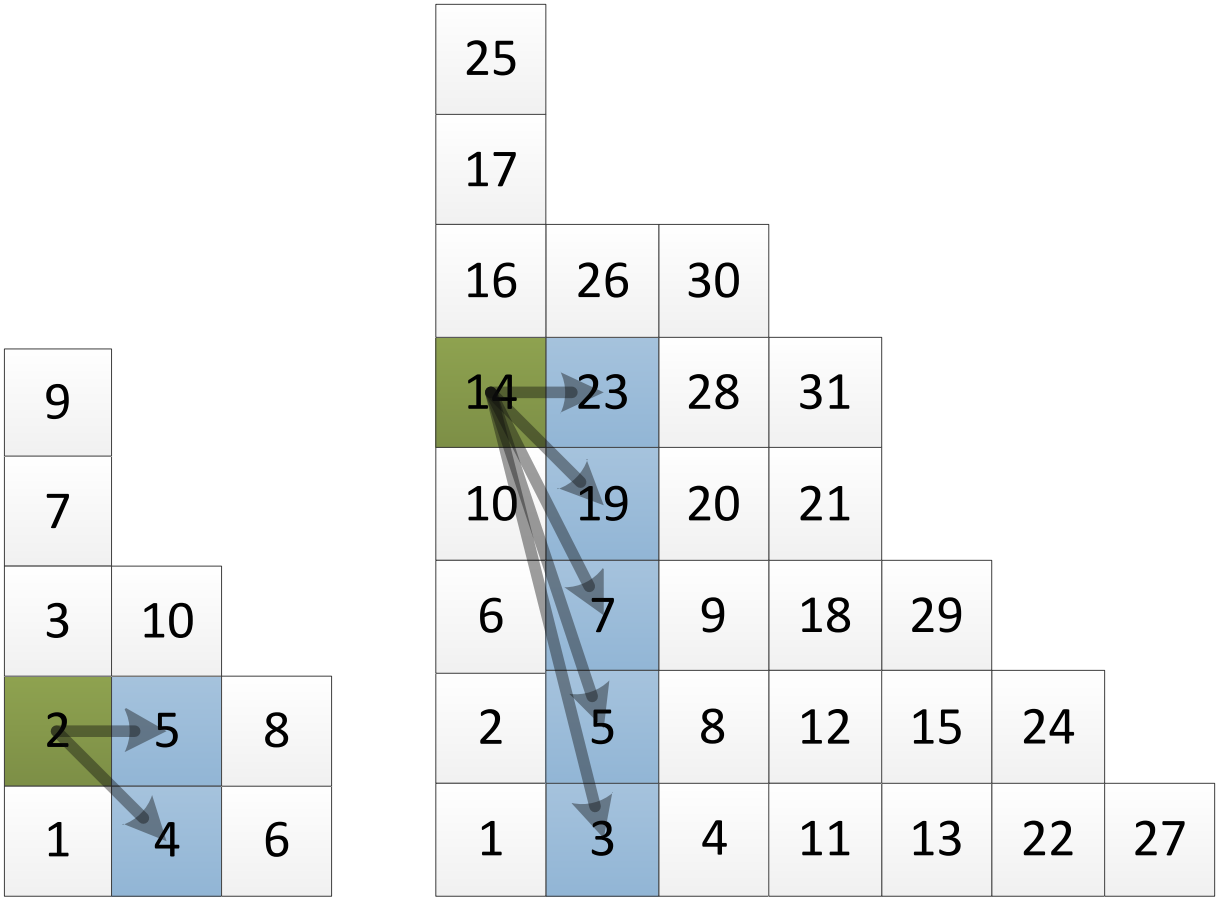
Tableau $Q$ records the changes of shapes of the intermediate tableaux $P$.

# RSK algorithm: an example

See RSKexample.pdf

# RSK: finding the nearest larger value

**A problem:** low efficiency of the standard algorithm (the complexity of search of position for bumping is proportional to the column heights).

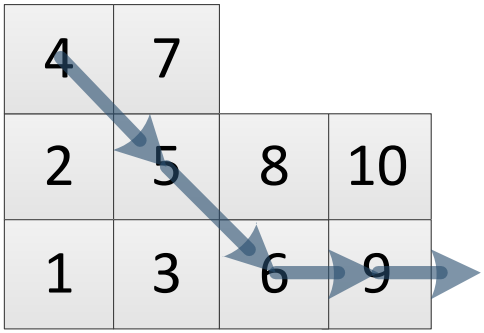


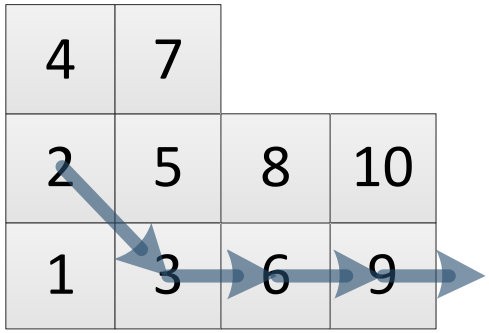
# Motivation

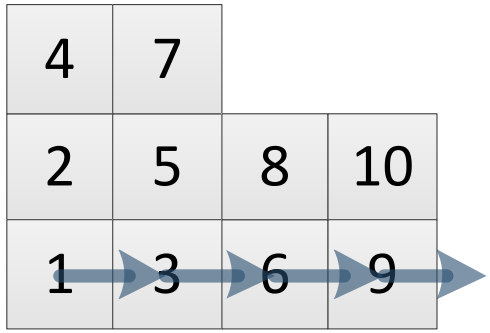
- There are various problems that need to be computed on a large number of very long permutations ($10^5$ numbers and more).
- The performance of the standard RSK is not enough to solve many computational tasks.
- The computational costs of the standard RSK are mainly caused by searching a position where the next number should be bumped in tableau $P$.
- The **goal** of this work is to implement a special variant of RSK which works significantly faster than the original algorithm.

# Bumping routes

**A bumping route** is a sequence of positions in insertion tableau where bumping occures during the RSK transformation.

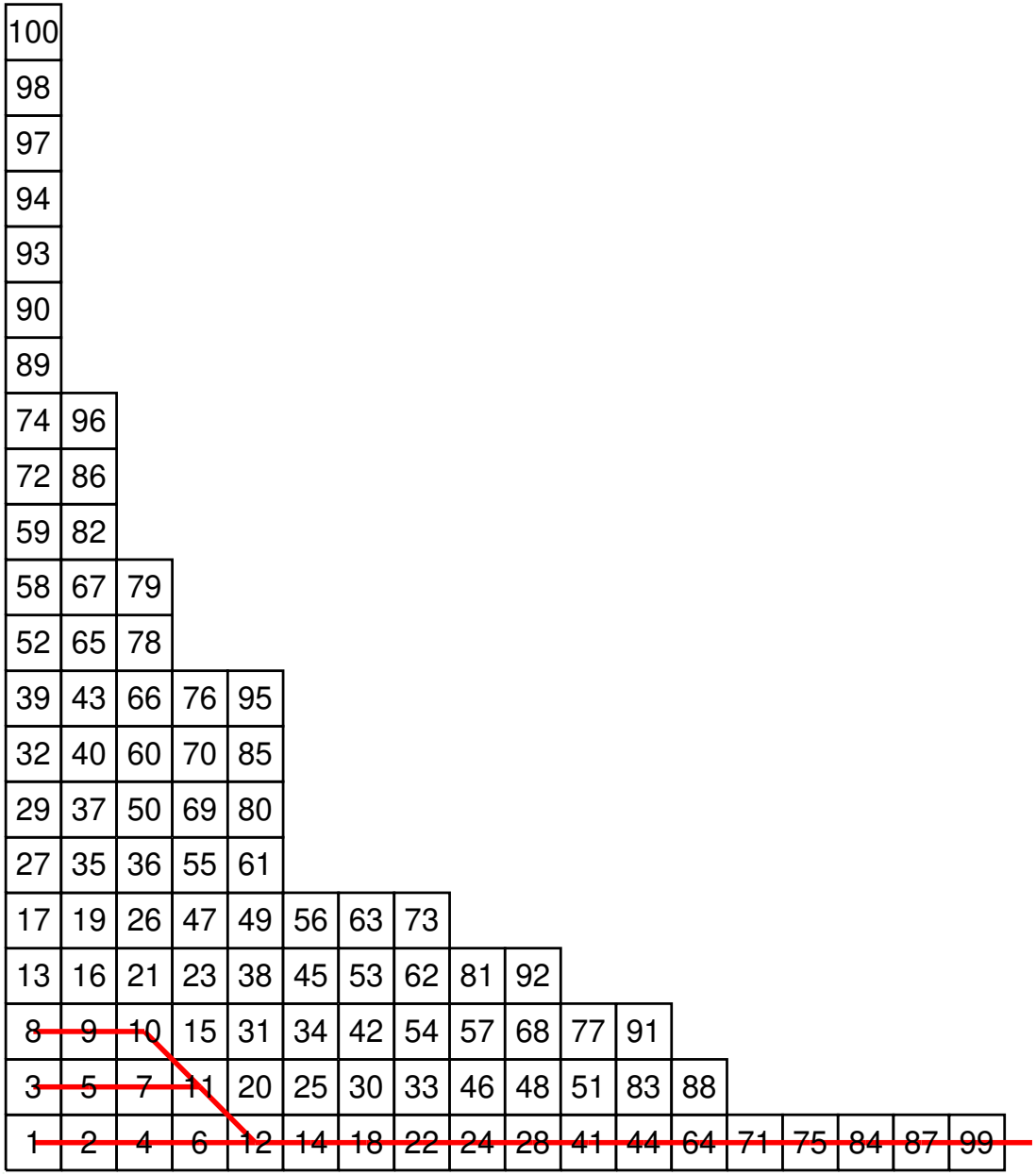


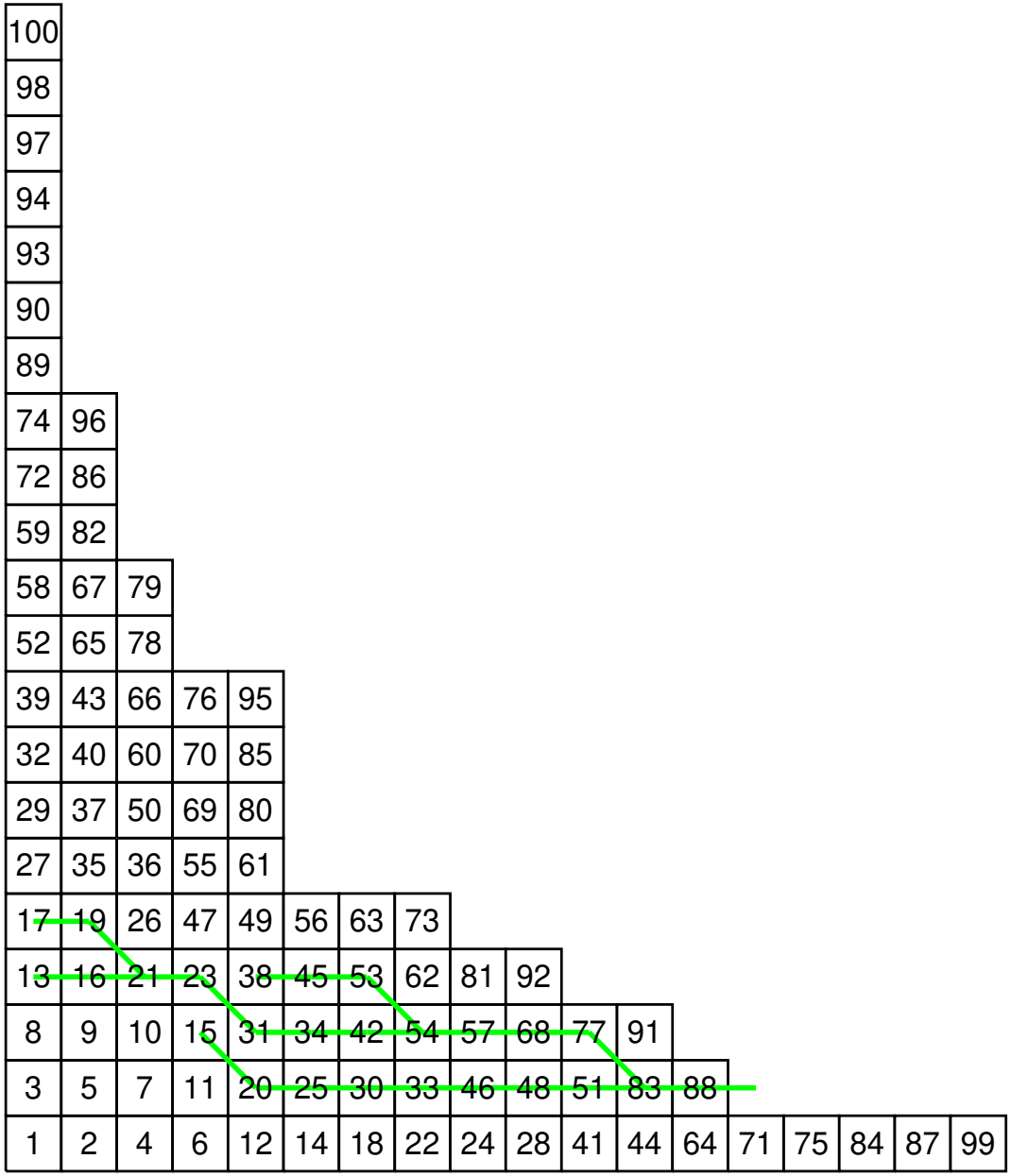

**The idea**: to remember all bumping routes of tableau in advance. In that case, we do not need to search a position where the next number should be bumped.

# Bumping trees



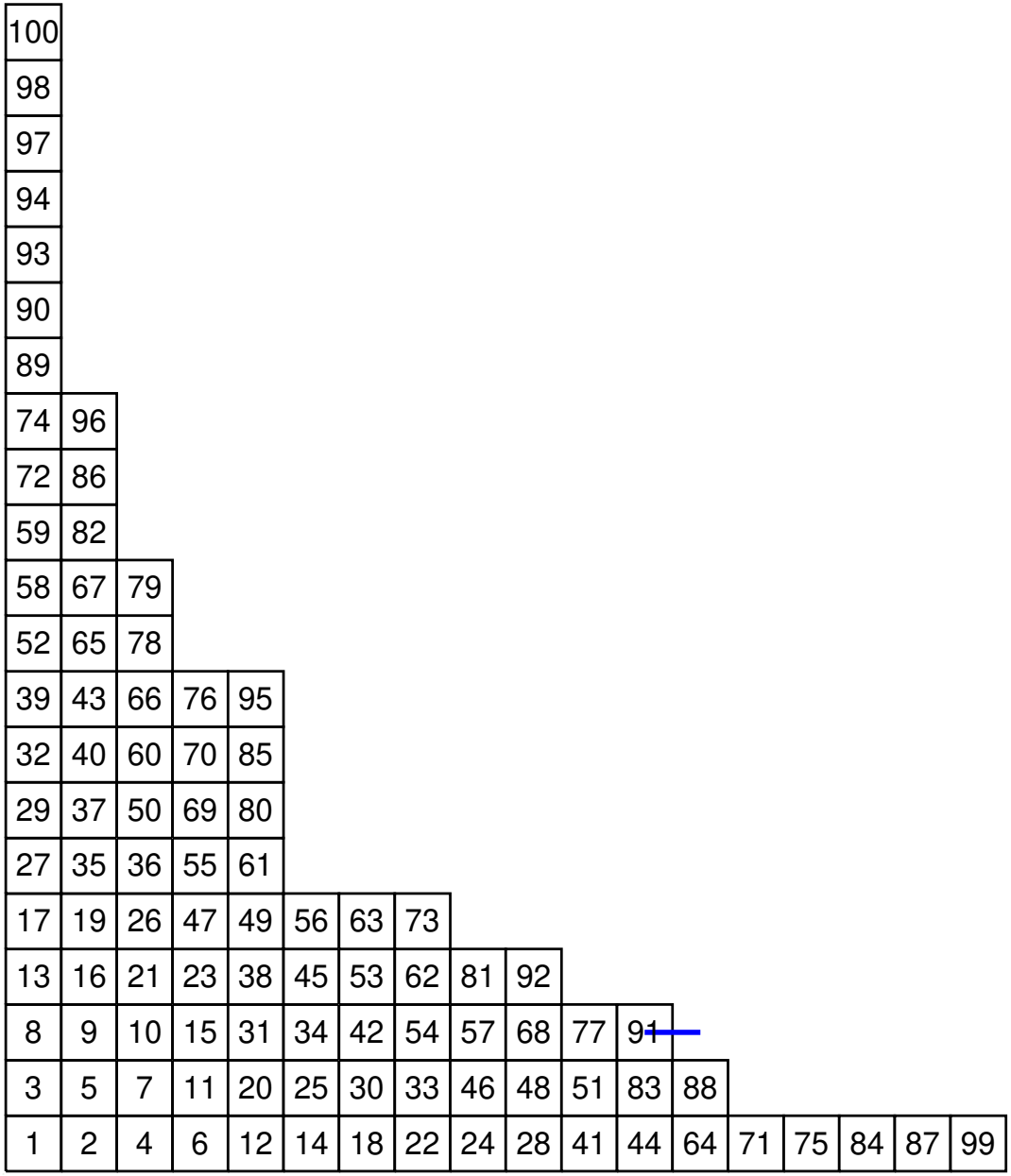

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



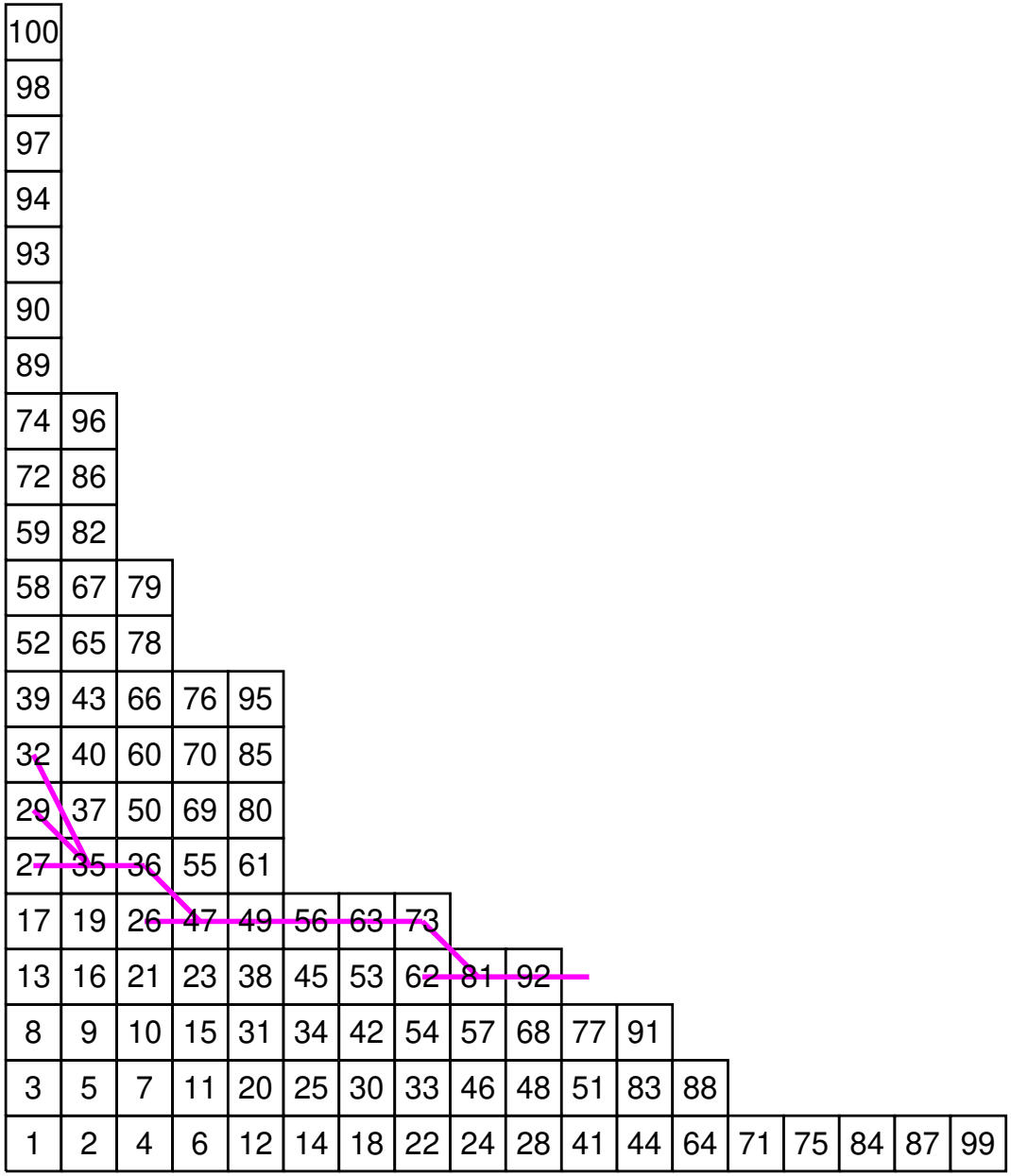

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



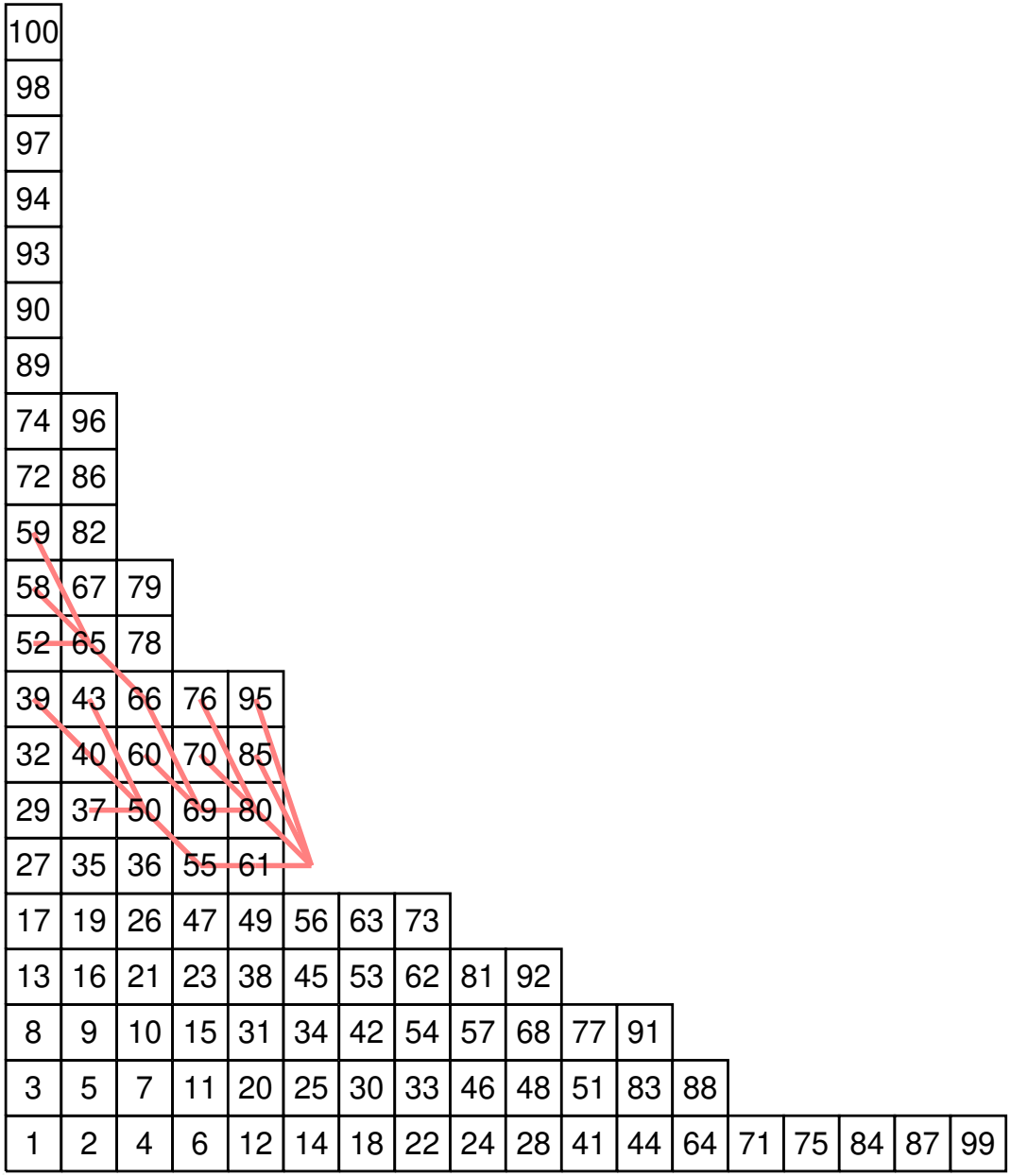

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



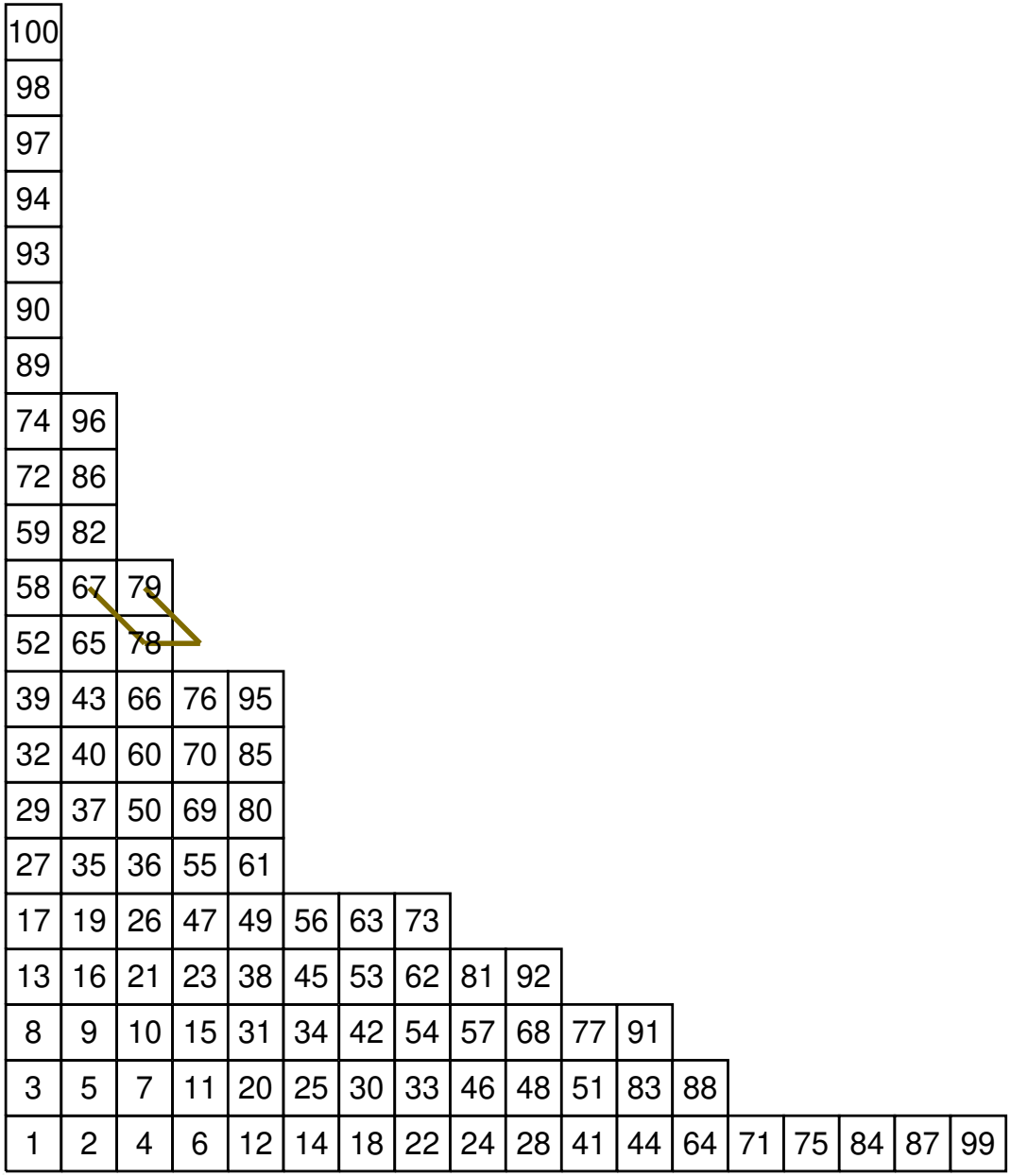

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



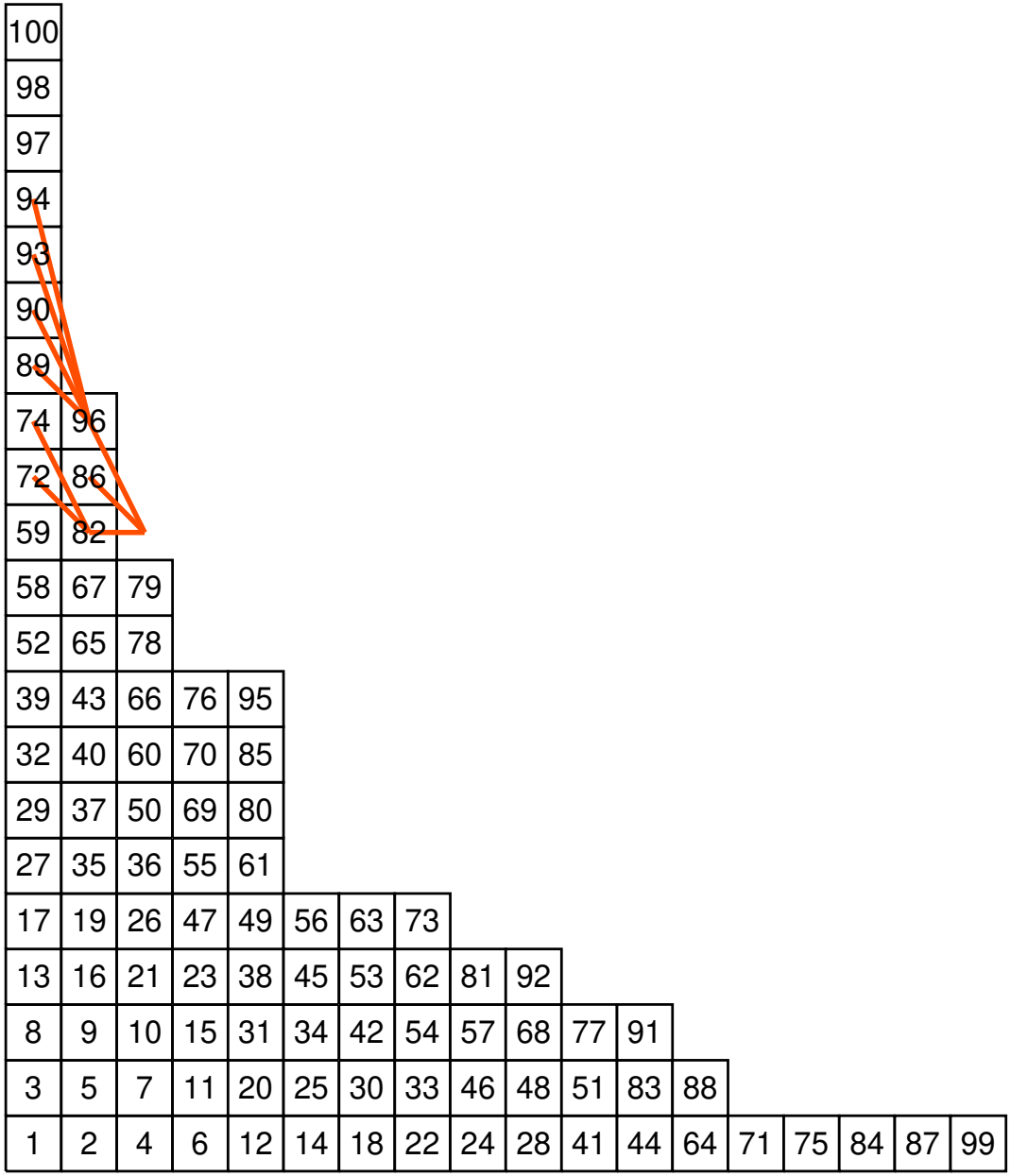

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



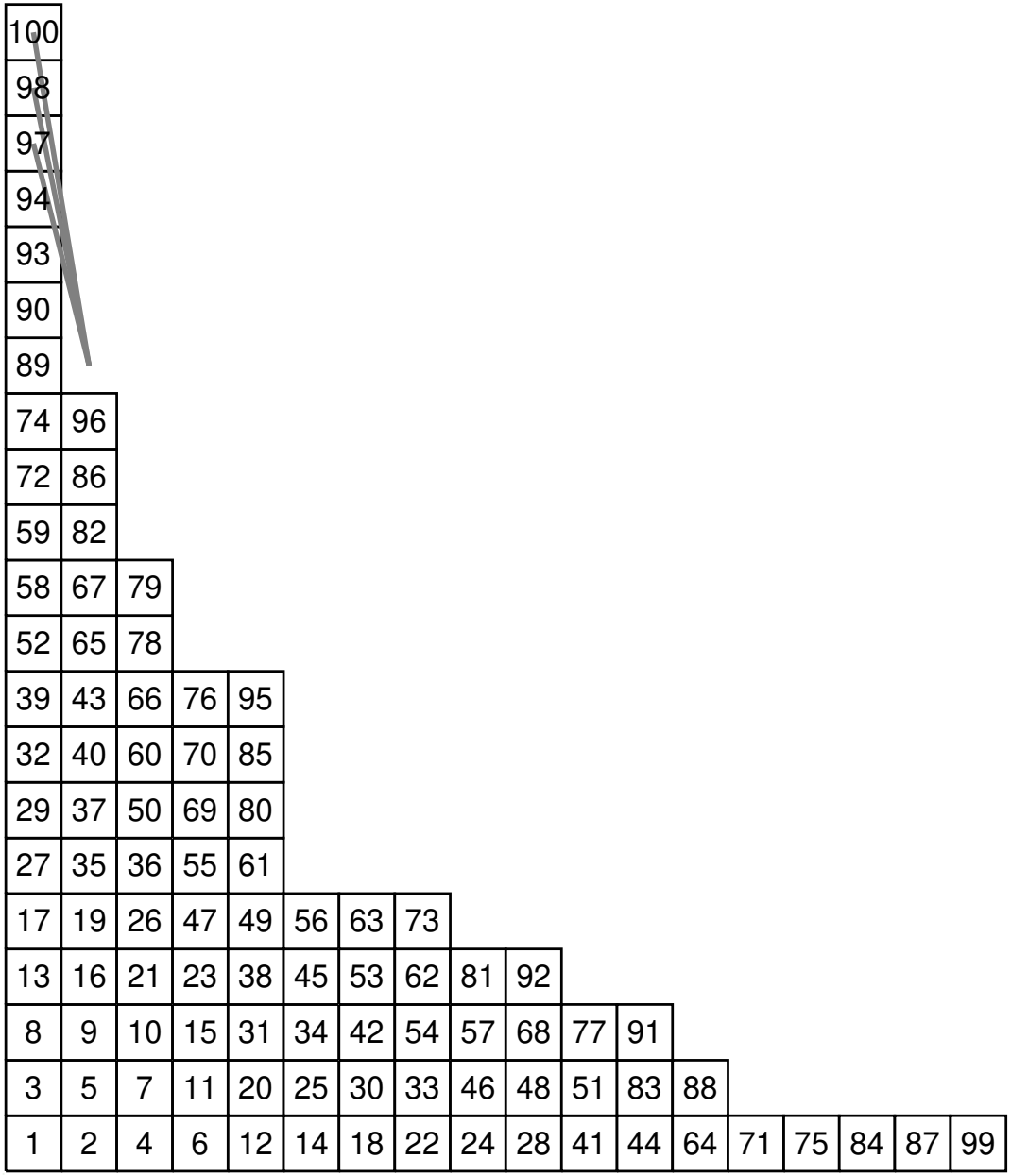

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



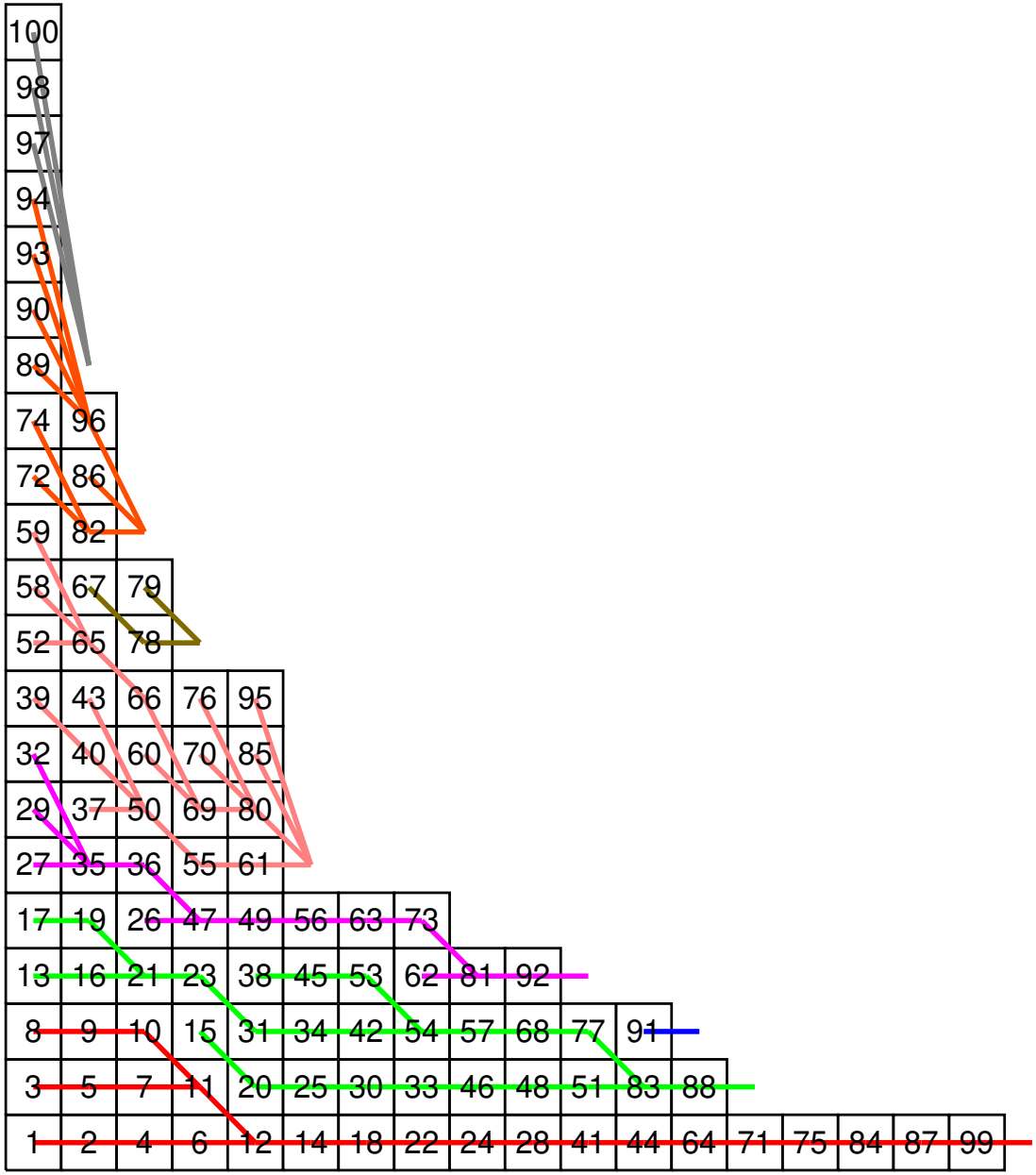

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping trees



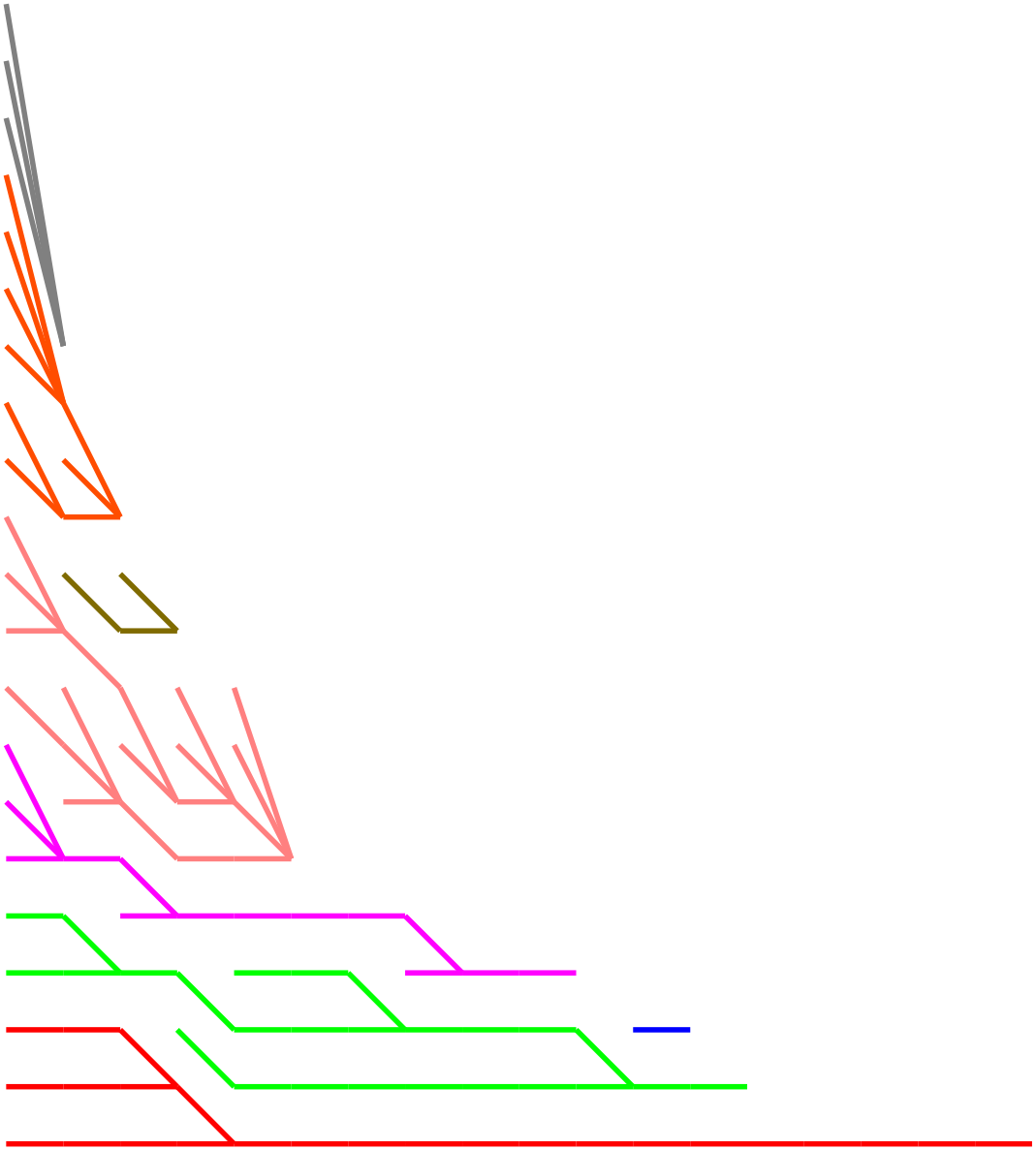

**A bumping tree** is a union of bumping routes converging to a same position where the shape of tableau can be changed.

# Bumping forest




**A bumping forest** is a union of all possible bumping routes of a tableau.

# Bumping forest



**A bumping forest** is a union of all possible bumping routes of a tableau.

# A fast RSK algorithm

Fast RSK is based on maintaining the structure of the bumping forest. The following actions are performed to insert a value $s_i$ in a tableau $P$:

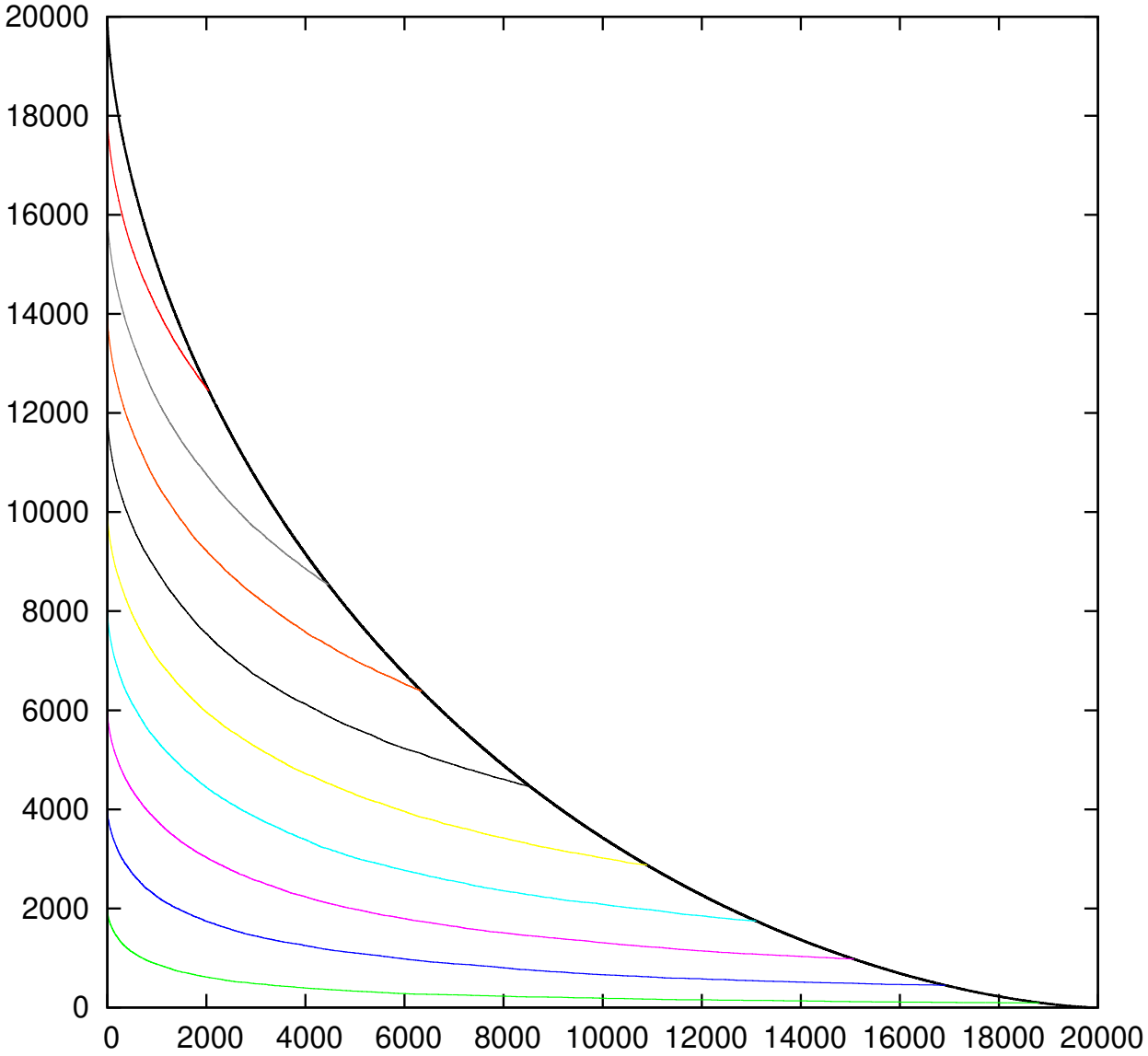
- Place $s_i$ in the first column (samely as in the standard algorithm).
- While the shape of tableau $P$ has not expanded,
  - Follow the link and write there the bumped value of $s'$. Put the old value into the buffer $b$.
  - Consequently compare $b$ with the values in the previous column, starting at $s_i$ and upper.
  - Check the references for all values $< b$.

Tableau $Q$ is constructed as in the standard algorithm.

# A fast RSK algorithm: an example

See FastRSK.avi

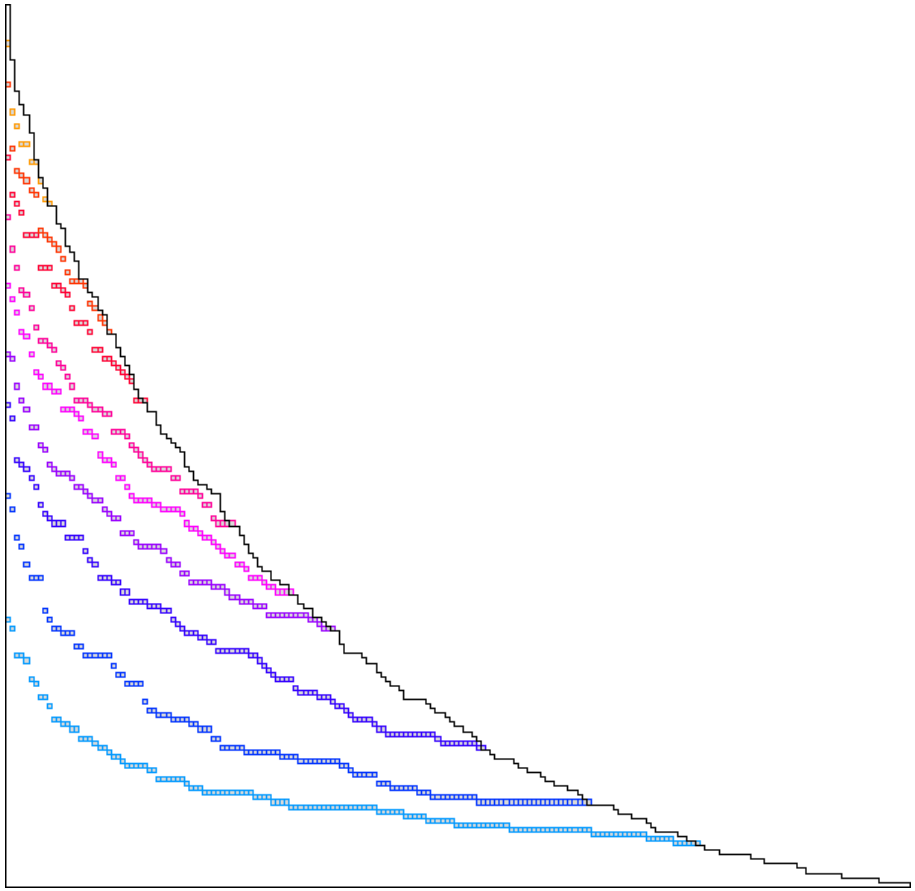
# Bumping routes



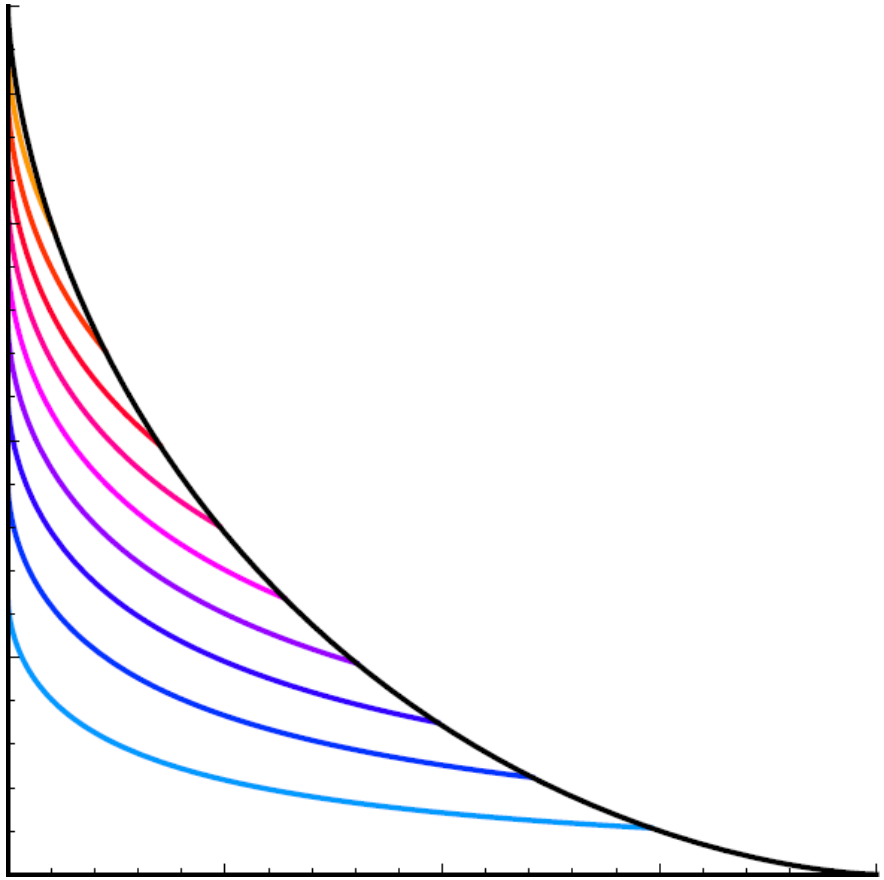

Some bumping routes of the random Young tableau of size $10^8$.

# Limit shapes of bumping routes

The asymptotic behaviour of limit shapes of bumping routes was studied in [Romik, Śniady '14].



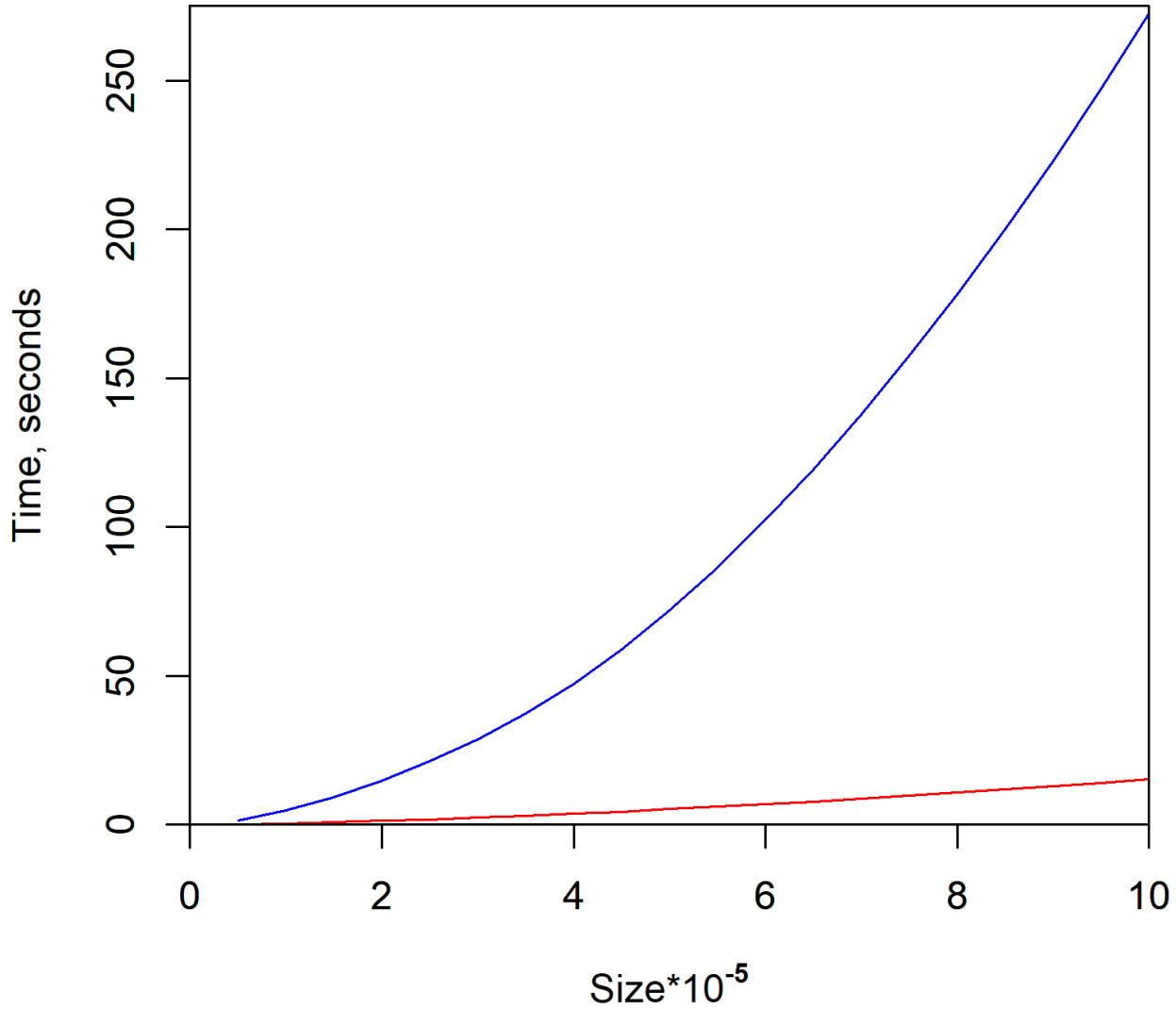
Bumping routes ($n = 10^4$)



Limit shapes

# The comparison of efficiency of RSK algorithms

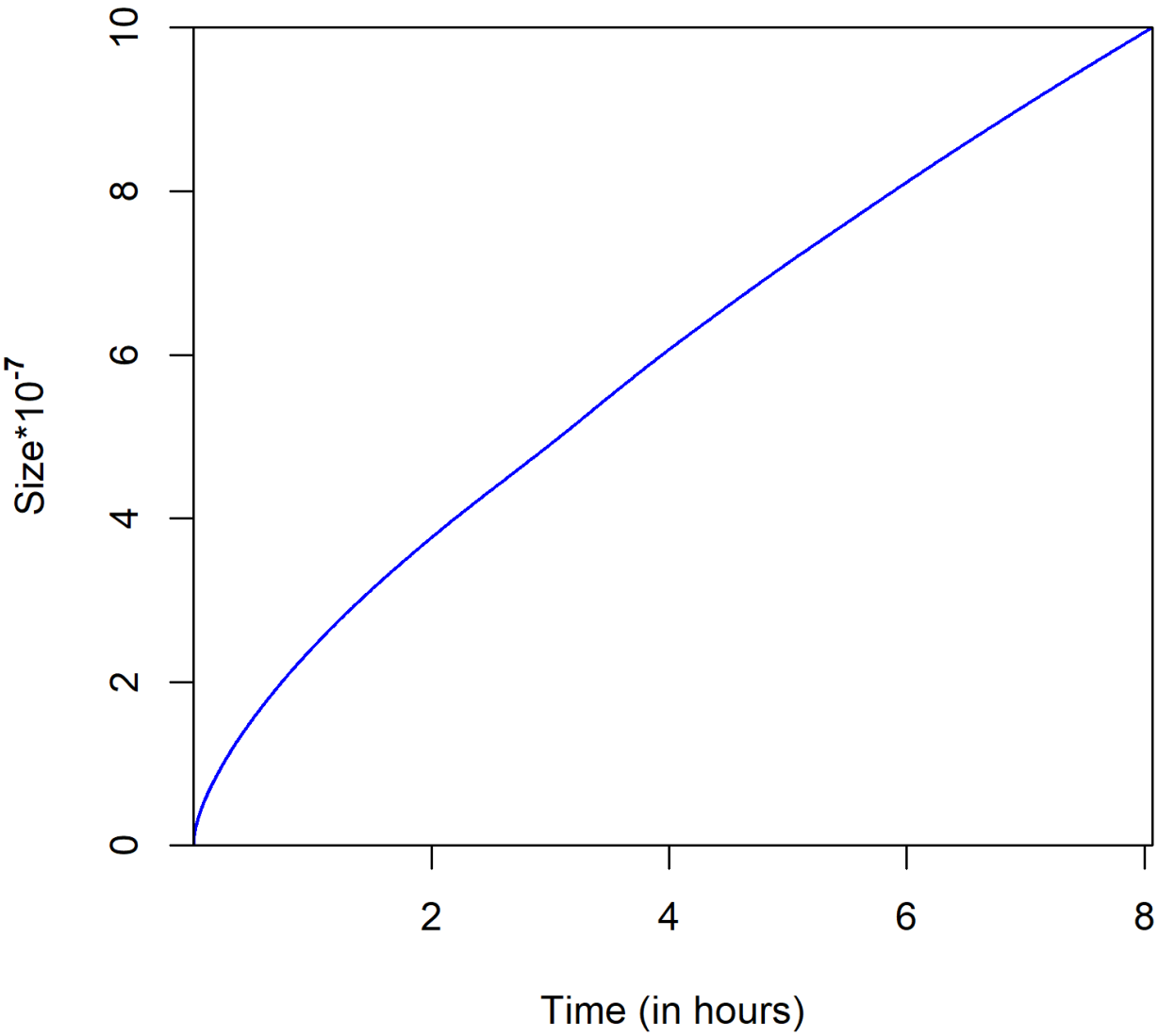
Blue curve is the running time of standard algorithm,
Red curve is the running time of fast algorithm.



# The comparison of efficiency of RSK algorithms

| Permutation size | Elapsed time by standard RSK (in sec) | Elapsed time by fast RSK (in sec) |
|---|---|---|
| 100.000 | 4.78 | 0.49 |
| 200.000 | 14.90 | 1.32 |
| 300.000 | 28.69 | 2.41 |
| 400.000 | 47.34 | 3.70 |
| 500.000 | 72.03 | 5.26 |
| 600.000 | 102.37 | 6.93 |
| 700.000 | 137.873 | 8.78 |
| 800.000 | 178.28 | 10.79 |
| 900.000 | 223.23 | 12.96 |
| 1.000.000 | 272.63 | 15.35 |

# The speed of the fast RSK algorithm



# Fast RSK: pros and cons

+

1. No need to search the places for bumping.
2. A bumping forest of an insertion tableau is provided.

−

1. Need to maintain the structure of a bumping forest.
2. A bumping forest consumes extra memory.

# Future plans

- Implementation of the fast inverse RSK algorithm.
- Theoretical estimation of speed of fast algorithms.
- Massive computations using developed algorithms.

Thanks for your attention!