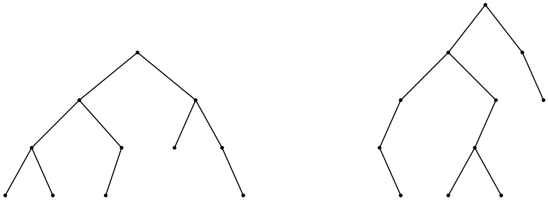
Generating Tree Permutations

Constant-memory Iterative Generation of Special Strings Representing Binary Trees

Sebastian Smyczyński

Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Toruń, Poland

Interesting combinatorical problem



Enumerate all possible shapes of binary trees with the given number of nodes.

Natural order

Natural order of binary trees

The natural order of binary trees follows the recursive definition:
We say that $T_1 \prec T_2$ if:

- $|T_1| < |T_2|$, or
- $|T_1| = |T_2|$ and $left(T_1) \prec left(T_2)$, lub
- $|T_1| = |T_2|$ and $left(T_1) = left(T_2)$ and $right(T_1) \prec right(T_2)$,

Natural order

Natural order of binary trees

The natural order of binary trees follows the recursive definition:
We say that $T_1 \prec T_2$ if:

- $|T_1| < |T_2|$, or
- $|T_1| = |T_2|$ and $left(T_1) \prec left(T_2)$, lub
- $|T_1| = |T_2|$ and $left(T_1) = left(T_2)$ and $right(T_1) \prec right(T_2)$,

- The size matters - the smaller trees precedes the larger ones.

Natural order

Natural order of binary trees

The natural order of binary trees follows the recursive definition:
We say that $T_1 \prec T_2$ if:

$|T_1| < |T_2|$, or

$|T_1| = |T_2|$ and $left(T_1) \prec left(T_2)$, lub

$|T_1| = |T_2|$ and $left(T_1) = left(T_2)$ and $right(T_1) \prec right(T_2)$,

- The size matters - the smaller trees precedes the larger ones.
- It is defined in natural for binary trees recursive way.

Natural order

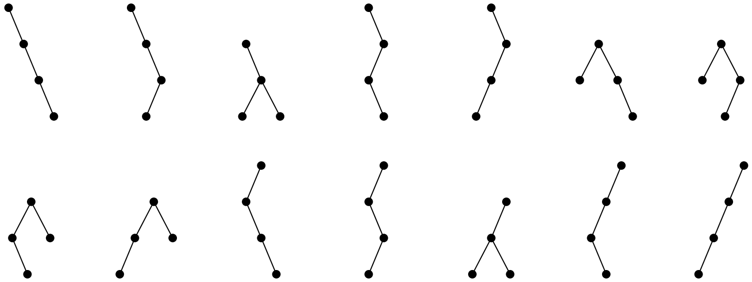
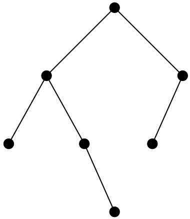


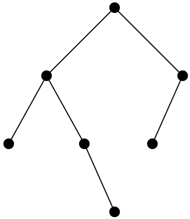
Figure: All shapes of binary trees with 4 nodes listed in their natural order

Tree permutations

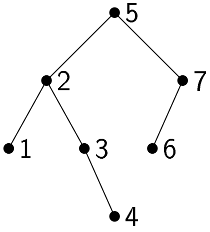


Tree permutations

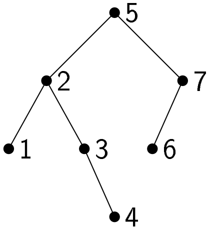
- We can represent the tree T with n nodes uniquely as a sequence of the integer numbers $1, 2, \ldots, n$,



- We can represent the tree T with n nodes uniquely as a sequence of the integer numbers $1, 2, \ldots, n$,
- first labeling the nodes with their position's number as they appear in the preorder travelsal of the tree,



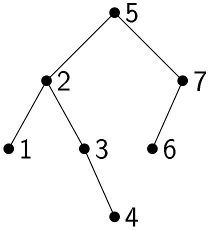
- We can represent the tree T with n nodes uniquely as a sequence of the integer numbers $1, 2, \ldots, n$,
- first labeling the nodes with their position's number as they appear in the preorder travelsal of the tree,
- then listing those labels as they appear in the preorder traversal of the tree.



5 2 1 3 4 7 6

Tree permutations

- We can represent the tree T with n nodes uniquely as a sequence of the integer numbers $1, 2, \ldots, n$,
- first labeling the nodes with their position's number as they appear in the preorder travelsal of the tree,
- then listing those labels as they appear in the preorder traversal of the tree.
- We shall call the resulting permutation $p = p_1, p_2, \ldots, p_n$ tree permutation of the T.



5 2 1 3 4 7 6

Tree permutations and the natural order

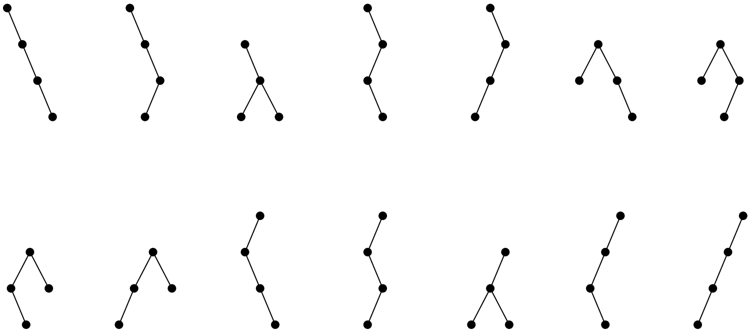


Figure: All shapes of binary trees with 4 nodes listed in their natural order

Tree permutations and the natural order

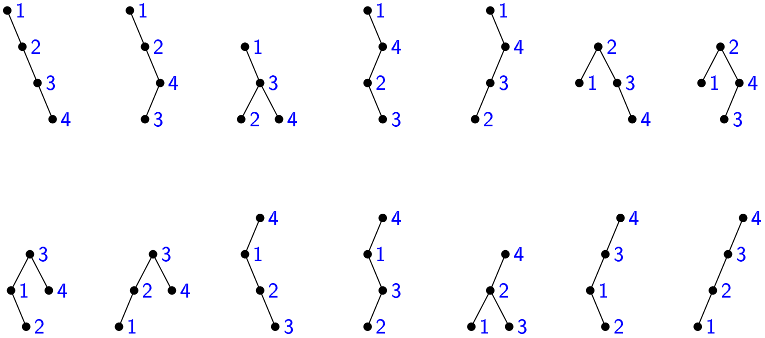


Figure: All shapes of binary trees with 4 nodes listed in their natural order

Tree permutations and the natural order

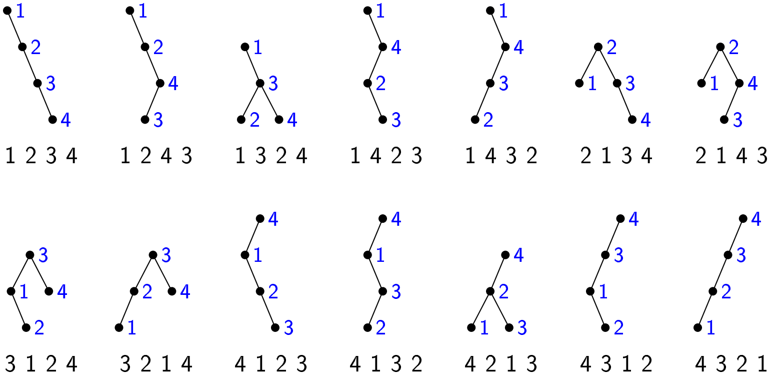


Figure: All shapes of binary trees with 4 nodes listed in their natural order

1234
1243
1234
1243
1324
1423
1432
2134
2143
3124
3214
4123
4132
4213
4312
4321

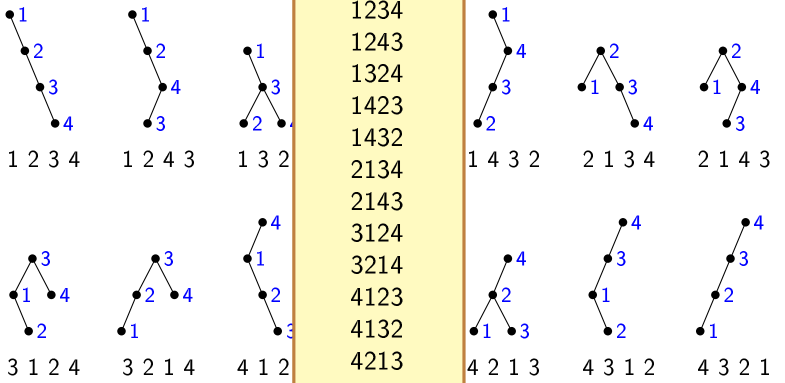


Figure: All shapes of binary trees with 4 nodes listed in their natural order

$$\forall p \in \mathscr{T}_n \quad \exists p' \in \mathscr{T}_{p_1 - 1} \quad \exists p'' \in \mathscr{T}_{n - p_1}$$
$$p = p_1 \; p' \; (p'' \oplus p_1)$$

$$p = 8 \quad \overbrace{4 \quad 1 \quad 3 \quad 2 \quad 6 \quad 5 \quad 7}^{p'} \quad \overbrace{12 \quad 11 \quad 9 \quad 10}^{\overbrace{4 \quad 3 \quad 1 \quad 2}^{p''} \oplus 8}$$

Climbing property of tree permutations

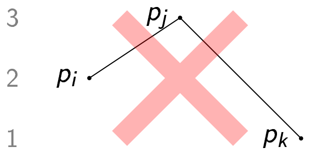
Lemma

A permutation $p = p_1, p_2, \ldots, p_n$ of the integer numbers $1, 2, \ldots, n$ is a tree permutation iff it is 231-avoiding - there are no such indices $i < j < k$ such that $p_k < p_i < p_j$.

Climbing property of tree permutations

Lemma

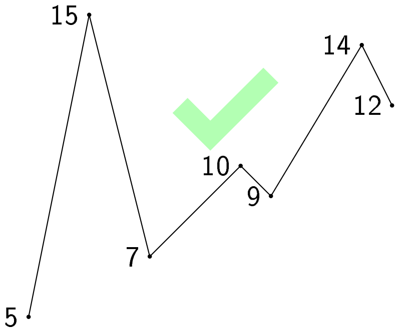
A permutation $p = p_1, p_2, \ldots, p_n$ of the integer numbers $1, 2, \ldots, n$ is a tree permutation iff it is 231-avoiding - there are no such indices $i < j < k$ such that $p_k < p_i < p_j$.



Climbing property of tree permutations

Lemma

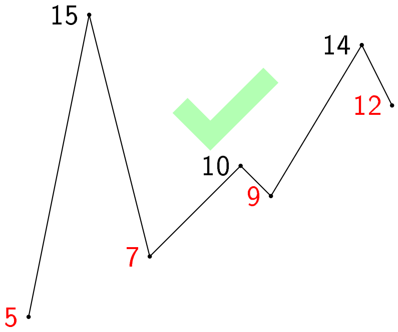
A permutation $p = p_1, p_2, \ldots, p_n$ *of the integer numbers* $1, 2, \ldots, n$ *is a tree permutation iff it is* 231*-avoiding - there are no such indices* $i < j < k$ *such that* $p_k < p_i < p_j$.



Climbing property of tree permutations

Lemma

A permutation $p = p_1, p_2, \ldots, p_n$ of the integer numbers $1, 2, \ldots, n$ is a tree permutation iff it is 231-avoiding - there are no such indices $i < j < k$ such that $p_k < p_i < p_j$.



Definition

Let $p = p_1, p_2, \ldots, p_n$ be a tree permutation. The suffix of p which makes p different from its successor in the lexicographic order is called working suffix.

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

Working suffix properties

Definition

Let $p = p_1, p_2, \ldots, p_n$ be a tree permutation. The suffix of p which makes p different from its successor in the lexicographic order is called working suffix.

Lemma

Let p be a tree permutation and let i be the index of the first position of its working suffix. If tree permutation q is the successor of p in the lexicographic order, then $q_i = p_i + 1$.

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

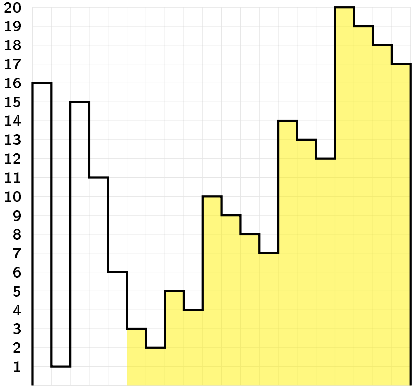
Working suffix properties

Lemma

Let p be a tree permutation and let i be the index of the first position of its working suffix, then there exist no such indices j, k such that $i < j < k$ and $p_k = p_j + 1$.

Lemma

Let $p = p_1, p_2, \ldots, p_n$ be a tree permutation and i be the starting index of its working suffix. For any index $i \leq k < n$, $p_k > p_k + 1$ implies that $p_k = p_k + 1 + 1$.



The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

1 2 3 4

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

1 2 3 4

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

1	2	3	4
1	2	4	3

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

1	2	3	4
1	2	4	3

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

1	2	3	4
1	2	4	3

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

1	2	3	4
1	2	4	3

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 … n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 … n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 ... n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

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

The algorithm

Generating tree permutations in lexicographic order:

- Start with permutation 1 2 … n.
- Find the first pair of indices $i < j$ starting from the end of the permutation such that $p_j = p_i + 1$.
- Swap the elements at found positions.
- Sort all but the first elements of the working suffix in ascending order.

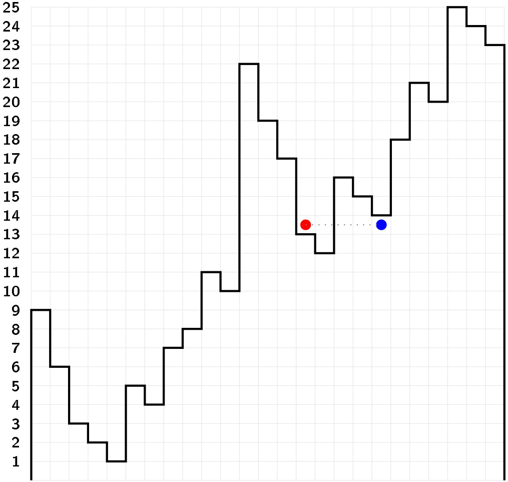
1	2	3	4
1	2	4	3
1	3	2	4
1	4	2	3
1	4	3	2
2	1	3	4
2	1	4	3
3	1	2	4
3	2	1	4
4	1	2	3
4	1	3	2
4	2	1	3
4	3	1	2
4	3	2	1

Optimalization of the searching step

9, 6, 3, 2, 1, 5, 4, 7, 8, 11, 10, 22, 19, 17, 13, 12, 16, 15, 14, 18, 21, 20, 25, 24, 23

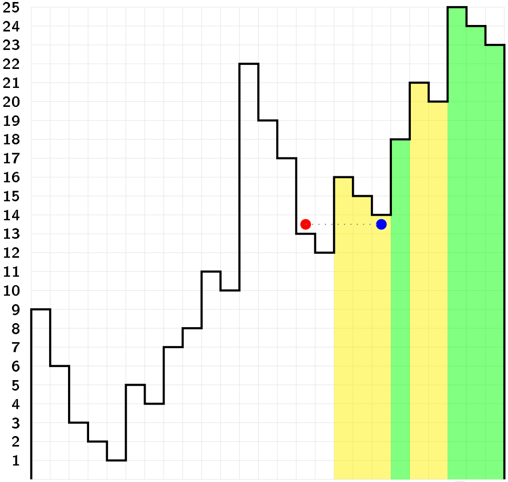
Optimalization of the searching step

9, 6, 3, 2, 1, 5, 4, 7, 8, 11, 10, 22, 19, 17, 13, 12, 16, 15, 14, 18, 21, 20, 25, 24, 23



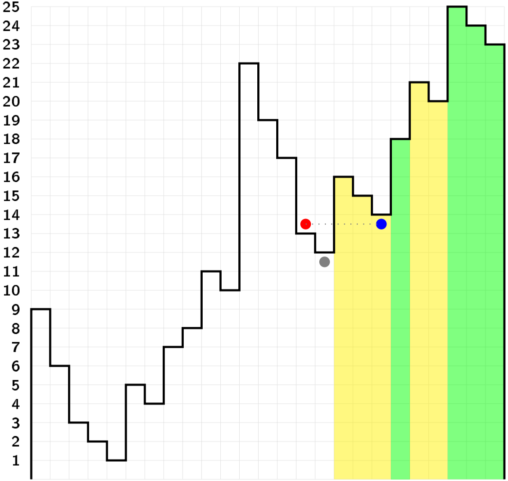
Optimalization of the searching step

9, 6, 3, 2, 1, 5, 4, 7, 8, 11, 10, 22, 19, 17, 13, 12, 16, 15, 14, 18, 21, 20, 25, 24, 23



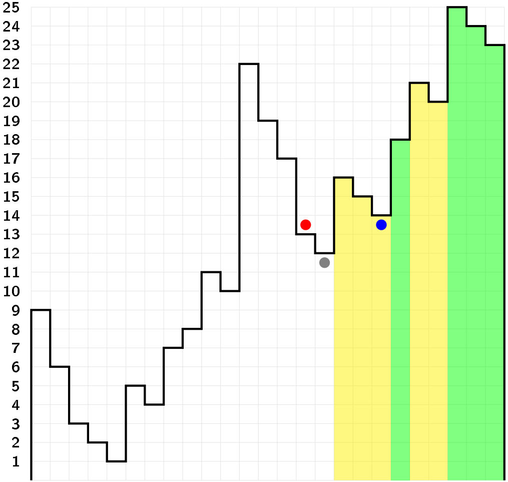
Optimalization of the searching step

9,6,3,2,1,5,4,7,8,11,10,22,19,17,13,12,16,15,14,18,21,20,25,24,23



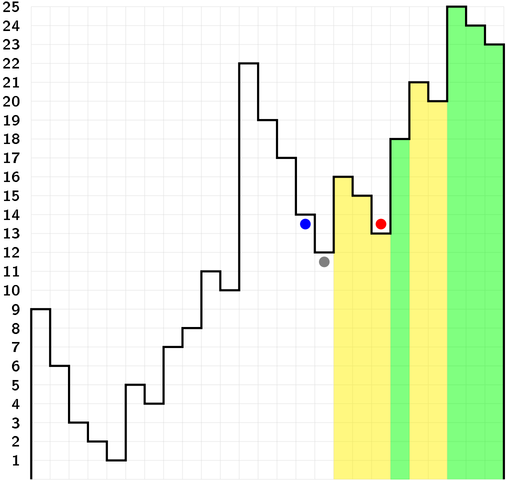
Optimalization of the sorting step

9, 6, 3, 2, 1, 5, 4, 7, 8, 11, 10, 22, 19, 17, 13, 12, 16, 15, 14, 18, 21, 20, 25, 24, 23



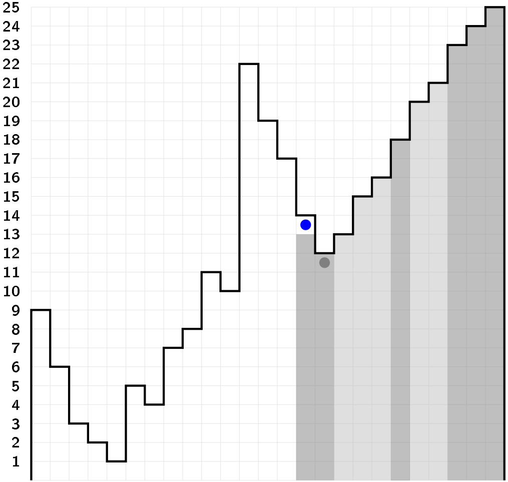
Optimalization of the sorting step

9, 6, 3, 2, 1, 5, 4, 7, 8, 11, 10, 22, 19, 17, 14, 12, 16, 15, 13, 18, 21, 20, 25, 24, 23



Optimalization of the sorting step

9,6,3,2,1,5,4,7,8,11,10,22,19,17,14,12,13,15,16,18,20,21,23,24,25



Time Complexity

Knowing that the number of binary trees with n nodes is is given by the Catalan number

$$C_n = \binom{2n}{n}/(n+1).$$

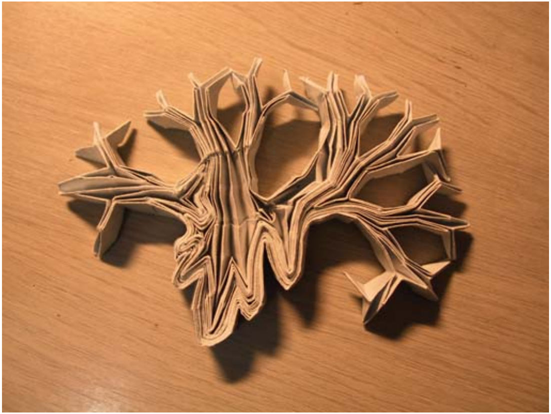
and using the recursive property of tree permutations we can give the formula for the summarized length of all working suffixes of tree permutations for given number of nodes n

$$W_n = \sum_{i=1}^{n}\Big(C_{i-1}W_{n-i} + W_{i-1} + (n-i)(C_{i-1}-1)\Big) + n(n-1).$$

Solving this recurrence we obtain $W_n = C_{n+1} - n - 1$.
Since $C_{n+1} = \frac{2(2n+1)}{n+2}C_n$, therefore the average-case time-complexity of the algorithm is constant $O(\frac{W_n}{C_n}) = O(1)$.

Thank you!

Binary origami tree



http://www.tsg.ne.jp/TT/origami/gallery.html