Received 9/78; revised 6/79; accepted 1/80

References

1. Aho, AV., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Comm. ACM 18,* 9 (Sept. 1975), 509–517.
3. Bentley, J.L. Divide and conquer algorithms for closest point problems in multidimensional space. Unpublished Ph.D. dissertation, Univ. of North Carolina, Chapel Hill, N.C., 1976.
4. Bentley, J.L. Decomposable searching problems. *Inform. Proc. Letters 8,* 5 (June 1979), 244–251.
5. Bentley, J.L., and Friedman, J.H. Algorithms and data structures for range searching. *Comptng. Surv. 11,* 4 (Dec. 1979), 397–409.
6. Bentley, J.L., Kung, H.T., Schkolnick, M., and Thompson, C.D. On the average number of maxima in a set of vectors and applications. *J. ACM 25,* 4 (Oct. 1978), 536–543.
7. Bentley, J.L., and Maurer, H.A. Efficient worst-case data structures for range searching. To appear in *Acta Informatica* (1980).
8. Bentley, J.L., and Shamos, M.I. Divide and conquer in multidimensional space. In Proc. ACM Symp. Theory of Comptng., May 1976, pp. 220–230.
9. Bentley, J.L., and Shamos, M.I. A problem in multivariate statistics: Algorithm, data structure, and applications. In Proc. 15th Allerton Conf. Communication, Control, and Comptng., Sept. 1977, pp. 193–201.
10. Blum, M., et al. Time bounds for selection. *J. Comptr. Syst. Sci. 7,* 4 (Aug. 1972), 448–461.
11. Dobkin, D., and Lipton, R.J. Multidimensional search problems. *SIAM J. Comptng. 5,* 2 (June 1976), 181–186.
12. Fredman, M. A near optimal data structure for a type of range query problem. In Proc. 11th ACM Symp. Theory of Comptng., April 1979, pp. 62–66.
13. Fredman, M., and Weide, B.W. On the complexity of computing the measure of $\cup [a_i, b_i]$. *Comm. ACM 21,* 7 (July 1978), 540–544.
14. Friedman, J.H. A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Comptrs. C-26,* 4 (April 1977), 404–408.
15. Friedman, J. H. A nested partitioning algorithm for numerical multiple integration. Rep. SLAC-PUB-2006, Stanford Linear Accelerator Ctr., 1978.
16. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
17. Kung, H.T., Luccio, F., and Preparata, F.P. On finding the maxima of a set of vectors. *J. ACM 22,* 4 (Oct. 1975), 469–476.
18. Lee, D.T., and Wong, C.K. Qintary trees: A file structure for multidimensional database systems. To appear in *ACM Trans. Database Syst.*
19. Lipton, R., and Tarjan, R.E. Applications of a planar separator theorem. In Proc. 18th Symp. Foundations of Comptr. Sci., Oct. 1977, pp. 162–170.
20. Lueker, G. A data structure for orthogonal range queries. In Proc. 19th Symp. Foundations of Comptr. Sci., Oct. 1978, pp. 28–34.
21. Monier, L. Combinatorial solutions of multidimensional divide-and-conquer recurrences. To appear in the *J. of Algorithms.*
22. Murray, J.A. *Lieutenant, Police Department—The Complete Study Guide for Scoring High* (4th ed.). Arco, New York, 1966, p. 184, question 3.
23. Reddy, D.R., and Rubin, S. Representation of three-dimensional objects. Carnegie-Mellon Comptr. Sci. Rep. CMU-CS-78-113, Carnegie-Mellon Univ., Pittsburgh, Pa., 1978.
24. Saxe, J.B. On the number of range queries in k-space. *Discrete Appl. Math. 1,* 3 (Nov. 1979), 217–225.
25. Shamos, M.I. Computational geometry. Unpublished Ph.D. dissertation, Yale Univ., New Haven, Conn., 1978.
26. Shamos, M.I. Geometric complexity. In Proc. 7th ACM Symp. Theory of Comptng., May 1975, pp. 224–233.
27. Weide, B. A survey of analysis techniques for discrete algorithms. *Comptng. Surv. 9,* 4 (Dec. 1977), 291–313.
28. Willard, D.E. New data structures for orthogonal queries. Harvard Aiken Comptr. Lab. Rep., Cambridge, Mass., 1978.
29. Yao, F.F. On finding the maximal elements in a set of planar vectors. Rep. UIUCDCS-R-74-667, Comptr. Sci. Dept., Univ. of Illinois, Urbana, July 1974.

Programming Techniques

R. Rivest Editor

A Unifying Look at Data Structures

Jean Vuillemin
University of Paris–South

Examples of fruitful interaction between geometrical combinatorics and the design and analysis of algorithms are presented. A demonstration is given of the way in which a simple geometrical construction yields new and efficient algorithms for various searching and list manipulation problems.

Key Words and Phrases: data structures, dictionaries, linear list, search, merge, permutations, analysis of algorithms

CR Categories: 4.34, 5.24, 5.25, 5.32, 8.1

1. Introduction

Whenever two combinatorial structures are counted by the same number, there exist bijections (one–one mappings) between the two structures. One goal of geometrical combinatorics (see, for example, Foata and Schutzenberger [7]) is to explicitly construct such bijections. This is bringing the field very close to computer science: One can regard combinatorial representations of remarkable numbers as equivalent data structures; explicit bijections between such representations provide coding and decoding algorithms between the structures. Earlier investigations along these lines are reported in Françon et al. [10] and Flajolet et al. [6].

This paper should be regarded as an introduction to using methods of geometrical combinatorics in the field of algorithm design and analysis. For this purpose, we consider representation of $n!$ as a running example and demonstrate how we are led to discovering new and efficient data structures and algorithms for solving various data manipulation problems.

This work was supported by the National Center for Scientific Research (CNRS), Paris, under Grant 3941.

Author's address: J. Vuillemin, Laboratory for Information Research, Building 490, University of Paris–South, 91405 Orsay, France.

In Section 2 we review representations of $n!$ by permutations, complete $n \times n$ diagrams (rook diagrams), and subdiagonal tables (Lehmer code, inversion table), and we follow the various appearances of Stirling numbers in these representations.

In Section 3 we study the tree representations of $n!$, showing the combinatorial significance of the pair of inverse algorithms CUT and CONCATENATE.

In Section 4 we put the tree representations to an original use for solving two-dimensional searching problems. We also present a new algorithm for merging binary search trees.

In Section 5 we apply the basic construction to representing linear lists. The conceptual simplicity of the algorithms involved makes them easy to program, and their average execution time is faster than that of any linear list representation known to the author.

2. Classical Representations of Factorial

2.1 Permutations and Sequences

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. A *permutation* is a *bijection* $\sigma:[n] \to [n]$; we write $\sigma \in S_n$, where S_n is the symmetric group over n objects. A permutation $\sigma \in S_n$ can be represented by a word $\sigma(1), \sigma(2), \ldots, \sigma(n)$ of length n over $[n]^*$; the bijective property of σ is expressed by $i \neq j \Rightarrow \sigma(i) \neq \sigma(j)$ for $1 \leq i, j \leq n$ (see Figure 1).

Permutations appear naturally in the analysis of algorithms (decision trees) which can only perform comparisons on their inputs. Such algorithms have exactly the same behavior on sequences of inputs sharing the same relative ordering. To be precise, let $S = (s_1, \ldots, s_n)$ and $S' = (s'_1, \ldots, s'_n)$ (with $s_i, s'_i \in \mathbb{R}$ for $1 \leq i \leq n$) be two sequences of length n over a totally ordered set $\mathbb{R}$. We say that S and S' are *order equivalent* if $s_i < s_j \Leftrightarrow s'_i < s'_j$ for $1 \leq i \neq j \leq n$ (see Figure 2).

Permutations can thus be regarded as equivalence classes of sequences under order equivalence.

Let $S \in \mathbb{R}^n$ be a sequence of length $|S| = n$. We define the *left-to-right minima* of S as the subsequence $\mathrm{LRMIN}(S) = (s_{i_1}, s_{i_2}, \ldots, s_{i_k})$ made of elements $s_i \in S$ such that $s_i < s_j$ for $1 \leq j < i$ (see Figure 3).

Going from right to left and changing the order, we can define the four sequences LRMIN, RLMIN, LRMAX, and RLMAX. We also consider LRM = LRMIN ∪ LRMAX, RLM = RLMIN ∪ RLMAX, MIN = LRMIN ∪ RLMIN, and MAX = LRMAX ∪ RLMIN.

An explicit bijection of Foata and Schutzenberger [7] shows that the number of permutations $\sigma \in S_n$ such that $|\mathrm{LRMIN}(\sigma)| = k$ is equal to the number of permutations $\sigma \in S_n$ having k cycles.

2.2 Cartesian Representations

A *complete n-diagram* is a set $D = \{\langle x_i, y_i \rangle \mid 1 \leq i \leq n\}$, where $\{x_i \mid 1 \leq i \leq n\} = \{y_j \mid 1 \leq j \leq n\} = [n]$ (see Figure 4).

Fig. 1. A permutation $\sigma \in S_8$.

σ = (5 7 3 9 6 1 4 2 8)

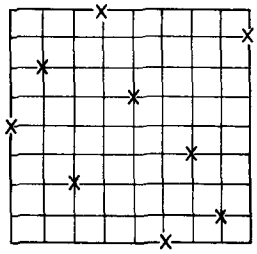
Fig. 2. Two order equivalent sequences S and S' and the permutation σ representing their equivalence class.

σ = (5 7 3 9 6 1 4 2 8)
S = (22, 48, 13, 97, 35, 3, 17, 5, 53)
S' = (1.7, 2.8, −0.3, 5.1, 2.3, −2.1, 0.4, −1.2, 3.2)

Fig. 3.

S = (22 48 13 97 35 3 17 5 53)
LRMIN(S) = (22 13 3)
RLMIN(S) = (3 5 53)
LRM(S) = (22 48 13 97 3)
MIN(S) = (22 13 3 5 53)
LRMAX(S) = (22 48 97)
RLMAX(S) = (97 53)
RLM(S) = (97 3 5 53)
MAX(S) = (22 48 97 53)

Fig. 4. The complete n-diagram associated with (5 7 3 9 6 1 4 2 8).



It is convenient to represent n-diagrams on an $n \times n$ plane grid with one point on each line and each column.

With each permutation $\sigma \in S_n$, we can associate the diagram $D_\sigma = \{\langle i, \sigma(i) \rangle \mid 1 \leq i \leq n\}$; conversely, with each diagram $D = \{\langle x_i, y_i \rangle \mid 1 \leq i \leq n\}$ we can associate the permutation $\sigma \in S_n$, where $\sigma(x_i) = y_i$. This establishes a natural bijection between permutations S_n and complete n-diagrams D_n.

The plane representation of complete diagrams possesses eight natural symmetries, which correspond to the following bijections $S_n \to S_n$, defined, for $\sigma = (\sigma(1), \ldots, \sigma(n))$, by

(1) $\tilde{\sigma} = (\sigma(n), \ldots, \sigma(1))$
(2) $-\sigma = (n + 1 - \sigma(1), \ldots, n + 1 - \sigma(n))$
(3) $\sigma^{-1} = (\sigma^{-1}(1), \ldots, \sigma^{-1}(n))$.

and their compositions (see Figure 5).

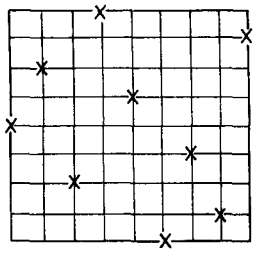
Again, n-diagrams appear in the analysis of algorithms as equivalence classes of sets of points in the plane: Two sets of points $P = \{\langle x_i, y_j \rangle \mid 1 \leq i \leq n, x_i, y_i \in R\}$ and P' are *order equivalent* if, for all $1 \leq i, j \leq n$, $(x_i < x_j$ iff $x'_i < x'_j)$ and $(y_i < y_j$ iff $y'_i < y'_j)$.

With each point $p \in D$ of an n-diagram D, we associate the number $\mathrm{LL}(p) = |\{\langle x_i, y_i \rangle \mid x_i < x, y_i < y, \langle x_i, y_i \rangle \in D, \langle x, y \rangle = p \in D\}|$ of points in D located in the *lower left* rectangle under and to the left of p; the

Fig. 5. A permutation and its "natural" symmetries.

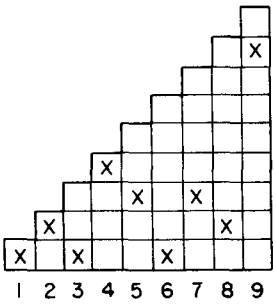
$\sigma = (5\ 7\ 3\ 9\ 6\ 1\ 4\ 2\ 8)$ $\quad \sigma^{-1} = (6\ 8\ 3\ 7\ 1\ 5\ 2\ 9\ 4)$
$\tilde{\sigma} = (8\ 2\ 4\ 1\ 6\ 9\ 3\ 7\ 5)$ $\quad \tilde{\sigma}^{-1} = (4\ 9\ 2\ 5\ 1\ 7\ 3\ 8\ 6)$
$-\sigma = (5\ 3\ 7\ 1\ 4\ 9\ 6\ 8\ 2)$ $\quad \sigma^{-1} = (4\ 2\ 7\ 3\ 9\ 5\ 8\ 1\ 6)$
$-\tilde{\sigma} = (2\ 8\ 6\ 9\ 4\ 1\ 7\ 3\ 5)$ $\quad \tilde{\sigma}^{-1} = (6\ 1\ 8\ 5\ 9\ 3\ 7\ 2\ 4)$

Fig. 6. The n-diagram associated with $\sigma = (5\ 7\ 3\ 9\ 6\ 1\ 4\ 2\ 8)$ and the four inversion tables LL_σ, LR_σ, UL_σ, and UR_σ.



$LL_\sigma = (0, 1, 0, 3, 2, 0, 2, 1, 7)$
$UL_\sigma = (0, 0, 2, 0, 2, 5, 4, 6, 1)$
$LR_\sigma = (4, 5, 2, 5, 3, 0, 1, 0, 0)$
$UR_\sigma = (4, 2, 4, 0, 1, 3, 1, 1, 0)$

Fig. 7. The subdiagonal table LL(1), ... , LL(9) associated with (5 7 3 9 6 1 4 2 8).



numbers UL, UR, and LR are defined in a symmetrical fashion. If we consider the permutation associated with D, the numbers LL, UL, UR, and LR represent the four natural inversion tables (also called Lehmer code by Knuth [13]) of the permutation associated with D (see Figure 6).

2.3 Subdiagonal Tables

The correspondence between n-diagrams D and the sequence (LL(1), ... , LL(n)) is a one–one mapping.

With each n-diagram, thus permutation, we associate a sequence of numbers LL(1), ... , LL(n) such that $0 \leq LL(i) < i$ for $1 \leq i \leq n$. Such sequences of numbers are called *subdiagonal tables* (see Figure 7).

Note that in this correspondence, left-to-right minima are mapped into the bottom line ($LL(i) = 0$), and left-to-right maxima into the top line ($LL(i) = i - 1$) of the subdiagonal table.

Subdiagonal tables play an important role in computer science: They are at the heart of interesting sorting algorithms (see Knuth [13]), and they are used explicitly or implicitly in all the algorithms known to the present author for enumerating permutations or generating random permutations. They are also called *inversions*.

They are important for counting purposes since a random subdiagonal table can be regarded as the *product of independent random variables* $v_1, \ldots, v_n$ *with* $0 \leq v_i < i$. The probability that $v_i = 0$ is $1/i$, so the average number of zero elements in a subdiagonal table is $1 + 1/2 + \cdots + 1/n = H_n$, the nth harmonic number. The distribution of this parameter, i.e., the number of subdiagonal tables of size n having k zeros, *is the classical* (see Comtet [3]) *Stirling number of first kind* $s_{n,k}$. The contribution of the ith variable v_i to the enumerating polynomial $\sum_k s_{n,k} x^k$ is $(x + i - 1)$ so we have

$$\sum_k s_{n,k} x^k = x(x+1) \ldots (x + n - 1), \qquad \text{for } n \geq 1. \quad (1)$$

An immediate consequence of the definition is

$$s_{n,k} = (n-1)s_{n-1,k} + s_{n-1,k-1}, \qquad \text{for } 1 \leq k \leq n,$$
$$s_{n,0} = 0, \quad s_{n,n} = 1, \quad s_{n,1} = (n-1)!. \quad (2)$$

Putting everything together gives the following proposition.

PROPOSITION 1. *Stirling numbers of first kind* $s_{n,k}$ *(whose average value is* $(1/n!) \sum_k k s_{n,k} = H_n$*) count:*

(*i*) *permutations* $\sigma \in S_n$ *such that* $|LRMIN(\sigma)| = k$, *or* $|RLMIN(\sigma)| = k$, *or* $|LRMAX(\sigma)| = k$, *or* $|RLMAX(\sigma)| = k$;
(*ii*) *permutations* $\sigma \in S_n$ *having* k *cycles*;
(*iii*) *complete* n*-diagrams* $\delta \in D_n$ *such that* $|\{p \in \delta | LL(p) = \emptyset\}| = k$, *and similarly for LR, UL, and UR*;
(*iv*) *subdiagonal tables* $t \in SD_n$ *such that* $|\{i | 1 \leq i \leq n, t(i) = 0\}| = k$.

From the preceding definitions and correspondences, the reader will easily construct explicit one–one mappings between the above structures, counted by $s_{n,k}$.

Going back to subdiagonal tables, we see that the probability that $v_i = 0$ or $v_i = i - 1$ is 1 for $i = 1$ and $2/i$ for $i > 1$. *The average number of elements on the bottom or top line of a random subdiagonal table of size n is thus* $2H_n - 1$.

Let $t_{n,k}$ be the number of tables $t \in SD_n$ having k elements on either the top or the bottom line, i.e., $k = |\{i | 1 \leq i \leq n, t(i) = 0 \text{ or } t(i) = i - 1\}|$. The enumerating polynomial of $t_{n,k}$ is directly

$$\sum_k t_{n,k} x^k = x(2x) \ldots (2x - i - 2) \ldots (2x - n - 2)$$

which, using (1), yields

$$t_{n,k} = 2^{k-1} s_{n-1,k-1}.$$

We can thus state Proposition 2.

PROPOSITION 2. *The numbers* $t_{n,k} = 2^{k-1} s_{n-1,k-1}$ *(whose average value is* $(1/n!) \sum_k k t_{n,k} = 2H_n - 1$*) count:*

(*i*) *permutations* $\sigma \in S_n$ *such that* $|MIN(\sigma)| = k$, *or* $|MAX(\sigma)| = k$, *or* $|LRM(\sigma)| = k$, *or* $|RLM(\sigma)| = k$;
(*ii*) *complete* n*-diagrams* $\delta \in D_n$ *such that* $|\{p \in j | LL(p) = \emptyset \text{ or } UL(p) = \emptyset\}| = k$, *etc.*;
(*iii*) *subdiagonal tables* $t \in SD_n$ *such that* $|\{i | 1 \leq i \leq n, t(i) = 0 \text{ or } t(i) = i - 1\}|$.

3. Tree Representation of Point Sets and Sequences

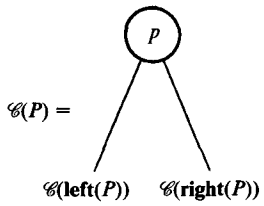
3.1 Representing a Plane Point Set by a Binary Tree

Let $P = \{p_1, \ldots, p_n\}$ be a finite set of points in the plane, represented in some Cartesian coordinate system so that $p_i = \langle x_i, y_i \rangle$ with $x_i, y_i \in \mathbb{R}$ for $1 \leq i \leq n$.

With such a plane point set P, we associate a labeled binary tree, called the *Cartesian tree* $\mathscr{C}(P)$ *of* P by the rules:

(1) If $P = \emptyset$, then $\mathscr{C}(P) = \emptyset$, the empty tree.

(2) If $P \neq \emptyset$, let $p = p_i = \langle x_i, y_i \rangle \in P$ be the point of p having the *least y-coordinate*, so that $1 \leq j \neq i \leq |P| \Rightarrow y_j > y_i$ for $\langle x_j, y_j \rangle \in P$; let **left**$(P)$ represent the points in P whose x-coordinate is less than x_i, and **right**(P) its complement in $P \backslash \{p\}$, so that **left**$(P) = \{\langle x, y \rangle \in P | x < x_i\}$, and **right**$(P) = \{\langle x, y \rangle \in P | x > x_i\}$. The tree $\mathscr{C}(P)$ is defined recursively by the rule



To avoid cumbersome special cases, the above definition has assumed that different points in P have different x and y coordinates; should this not be the case, ties can be broken in arbitrary ways to make this definition general (see Figure 8).

Let the x *part of* $\mathscr{C}(P)$ be the binary tree formed from $\mathscr{C}(P)$ by only keeping for each node $p = \langle x, y \rangle$ in $\mathscr{C}(P)$ the x value of p; the y *part of* $\mathscr{C}(P)$ is defined in a similar way, and we can regard $\mathscr{C}(P)$ as constructed by a direct Cartesian product of its x and y parts, which are two labeled trees of the same shape (see Figure 9).

The characteristic properties of such labelings are:

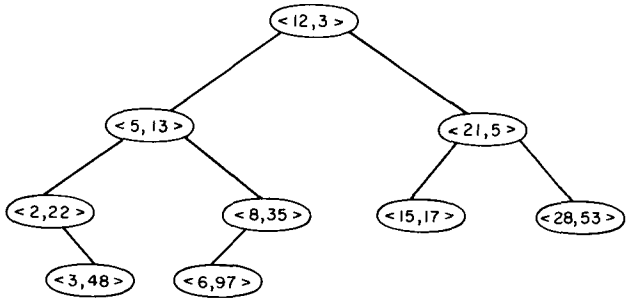
(1) The x part of $\mathscr{C}(P)$ is a *binary search tree*, i.e., the x_is increase as we traverse $\mathscr{C}(P)$ in symmetric order (left subtree; root; right subtree; see Knuth [13]).

(2) The y part of $\mathscr{C}(P)$ is a *binary tournament*, i.e., the y_is increase as we follow any path from the root to a leaf in $\mathscr{C}(P)$.

3.2 Representing a Sequence by a Binary Tree

Let $S = \langle s_1, \ldots, s_n \rangle \in \mathbb{R}^n$ be a sequence of length n. With such a sequence, we associate the point set $P = \{\langle 1, s_1 \rangle, \ldots, \langle n, s_n \rangle\}$ and consider the tree $\mathscr{C}(P)$ (see Figure 10).

Fig. 8. The Cartesian tree $\mathscr{C}(P)$ associated with the point set $P = \{\langle 2, 22 \rangle, \langle 3, 48 \rangle, \langle 5, 13 \rangle, \langle 6, 97 \rangle, \langle 8, 35 \rangle, \langle 12, 3 \rangle, \langle 15, 17 \rangle, \langle 21, 5 \rangle, \langle 28, 53 \rangle\}$.



The x part of such a tree is simply a binary tree labeled by 1, 2, ..., n in symmetric order. Since x and y parts have the same underlying binary tree, it is sufficient to know the y part in order to reconstruct the entire structure.

We have thus exhibited a one–one mapping between sequences $S \in \mathbb{R}^n$ and binary tournaments of size n with labels in $\mathbb{R}$. This bijection $S \rightarrow \mathscr{T}(S)$ can be defined directly, without invoking point sets:

(1) To the empty sequence $\Lambda (n = 0)$ is associated the empty tree $\emptyset$; thus $\mathscr{T}(\Lambda) = \emptyset$.

(2) For $n > 0$, let i be such that $s_i = \min_{1 \leq j \leq n} \{s_j\}$; we define $\mathscr{T}(S) =$

where $L = \mathscr{T}(\langle s_1, \ldots, s_{i-1} \rangle)$ and $R = \mathscr{T}(\langle s_{i+1}, \ldots, s_n \rangle)$.

We call $\mathscr{T}(S)$ *the tournament representation* of sequence $S \in \mathbb{R}^n$; conversely, with any binary tournament T, we associate the sequence $\mathscr{T}^{-1}(T)$ of its labels, taken in symmetric order.

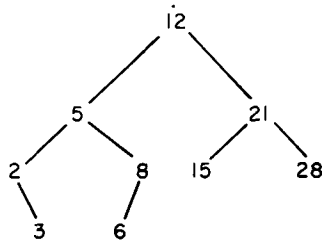
3.3 Combinatorial Properties of Binary Tournaments

We let T_n denote the set of tournament representations of all permutations $\sigma \in S_n$, considered as elements of $[n]^n$ (see Figure 11).

Correspondence $\mathscr{T}: S_n \rightarrow T_n$ being one–one implies that $|\mathscr{T}_n| = n!$.

To express combinatorial properties of this corre-

Fig. 9. The x and y Parts of the Cartesian Tree of Figure 8.



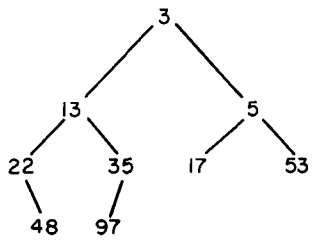
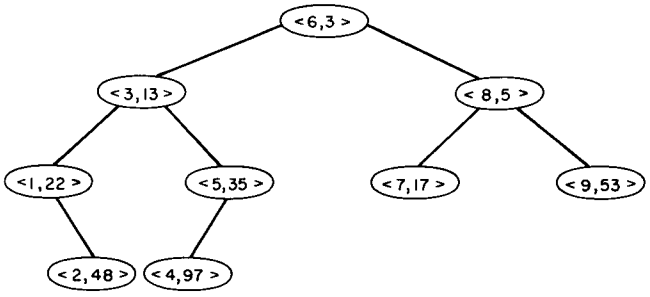


Fig. 10. The Cartesian tree associated with the sequence ⟨22, 48, 13, 97, 35, 3, 17, 5, 53⟩. The y part of this tree is given in Figure 9.



spondence, we define the *left branch* LB (T) of a labeled binary tree T as the sequence:

(1) LB ($\emptyset$) = Λ for the empty tree;
(2) LB (T) = $v(T)\cdot$LB($l(T)$), where $v(T)$ is the label of the root of $T \neq \emptyset$ and $l(T)$ is the left subtree of T.

We define the *right branch* RB (T) in a symmetrical manner, and we let B (T) = LB (T) $\cup$ RB (T) denote the *branches* of T.

PROPOSITION 3. *Stirling numbers of first kind count the binary tournaments $T \in T_n$ of size n having a left branch $|LB(T)| = k$ or a right branch $|RB(T)| = k$. The left branch and right branch of a random $T \in T_n$ has length $H_n = 1 + 1/2 + \cdots + 1/n$. Numbers $t_{n,k} = 2^{k-1}s_{n-1,k-1}$ count the binary tournaments $T \in T_n$ such that $|B(T)| = |LB(T)| + |RB(T)| - 1 = k$.*

PROOF. Correspondence $\mathcal{T}$ maps LRMIN(S) into LB (T) and RLMIN(S) into RB (T). The result thus follows from Propositions 1 and 2. □

3.4 Cutting a Cartesian Tree

Given the Cartesian tree $\mathcal{C}(P)$ representing a set of points $P = \{p_1, \ldots, p_n\}$, we wish to cut it in two parts $\mathcal{C}(P_{<c})$ and $\mathcal{C}(P_{\geq c})$; here $c \in \mathbb{R}$ is given, $P_{<c} = \{p \in P | p = \langle x, y\rangle, x < c\}$ is the subset of P whose x-coordinate is $<c$ and $P_{\geq c}$ is the complement (see Figure 12).

The cut operation is described by Algorithm 1 below; if T is a Cartesian tree and $c \in \mathbb{R}$ a real number, the call (L, R) $\leftarrow$ CUT(T, c) creates two Cartesian trees $L = \mathcal{C}(P_{<c})$ and $R = \mathcal{C}(P_{\geq c})$, where $P = \mathcal{C}^{-1}(T)$ is the plane set represented by T.

Algorithm 1 (Cut of a Cartesian Tree)

```
proc(L, R) ← CUT(CT, c)
  Cartesian tree L, R, CT; real c; point p;
  if CT = ∅ then (L, R) ← (∅, ∅)
          else p ← v(CT);
            if c < x(p) then (L, R) ← CUT(l(CT), c);
                             R ← ⟨R, p, r(T)⟩
                        else (L, R) ← CUT(r(CT), c);
                             L ← ⟨l(T), p, L⟩
            fi
  fi
fproc CUT.
```

In this algorithm, we introduce the data type **point**; a **point** p is a pair $\langle a, b\rangle \in \mathbb{R}^2$ of its x coordinate $x(p) = a$ and its y coordinate $y(p) = b$. The data type **Cartesian tree** designates binary trees labeled by points so as to form binary search trees on their x labels and binary tournaments on their y labels. If CT is a Cartesian tree, $v(CT)$ is the point labeling the root of CT; the left and right subtrees of CT are $l(CT)$ and $r(CT)$. Cartesian trees are thus defined by the rules:

(1) The empty tree $\emptyset$ is a Cartesian tree.
(2) If p is a point, L and R are Cartesian trees, then $\langle L, p, R\rangle = CT$ is a Cartesian tree such that $v(CT) = p$, $l(CT) = L$ and $r(CT) = R$ provided that
  (a) $\forall q \in L: x(q) < x(p)$, $\forall q \in R: x(q) > x(p)$ (binary search tree in x);
  (b) $L \neq \emptyset \Rightarrow y(v(L)) > y(p)$, $R \neq \emptyset \Rightarrow y(v(R)) > y(p)$ (binary tournament in y).

The algorithm CUT(CT, c) need only examine a path, **path**(1, c), in CT formed of the points $p \in CT$ for which a comparison $c \,?\, x(p)$ is performed; in the resulting trees L and R, the points thus examined are found to form RB (L) union LB (R). This simple observation leads to an analysis of the number of comparisons in the CUT algorithm and to the definition of an interesting combinatorial correspondence.

PROPOSITION 4. *The average number of comparisons $c \,?\, x(p)$ in CUT(CT, c), where $|CT| = (n-1)$ is $2H_n - 2$; the probability that this number of comparisons is exactly k equals $1/n!\cdot t_{n,k+2} = 2^k/n!\cdot s_{n-1,k}$. There is a one–one mapping $\theta: T_n \rightarrow T_n$ between binary tournaments transforming the path,* **path**(1, n, T), *from the root to the point labeled n in $T \in T_n$ into the branches B ($\theta(T)$) of $\theta(T) \in T_n$: |**path**(1, n, T)| = |B ($\theta(T)$)|.*

Fig. 11. The 6 = 3! Binary Tournaments T_3.

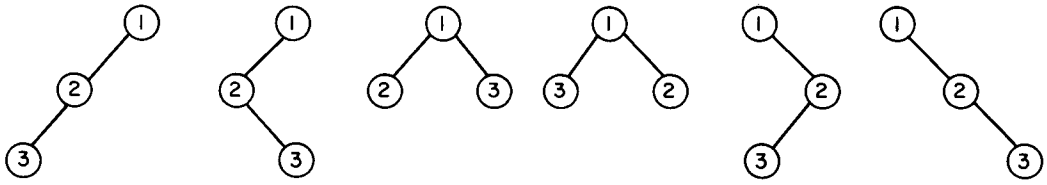
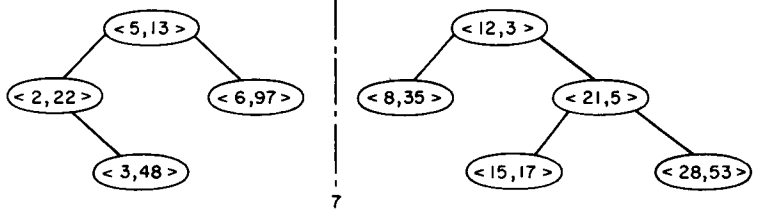


Fig. 12. The Cartesian Tree of Figure 8 Cut at $c = 7$.



PROOF. Let $T \in T_n$ be a binary tournament, with $T = \mathcal{T}(\sigma)$ where $\sigma = (\sigma(1), \ldots, \sigma(n))$ is the permutation obtained by taking the labels of T in symmetric order. Let $i = \sigma^{-1}(n)$ be the index of the node labeled n in that traversal. We also consider the Cartesian tree $\mathcal{C}(P)$ associated with σ by $P = \{\langle i, \sigma(i)\rangle \mid 1 \leq i \leq n\}$ as in Section 3.2. Note that $\mathcal{C}(P)$ can be formed from T by adding, as an x coordinate to every node in T, its index in symmetric order. We proceed to form $(L, R) \leftarrow \text{CUT}(\mathcal{C}(P), i)$.

The path **path**$(1, n, T)$ is now in correspondence with RB (L) union LB (R). Construct L' and R' by removing (either from L or from R) the node whose label is $\langle i, n\rangle$, remove the x part of all labels, and increase the y labels by 1. We are left with two binary tournaments L' and R', whose labels are all different, and labels $(L' \cup R') = \{2, 3, \ldots, n\}$. The resulting binary tournament $\theta(T) = \langle R', 1, L'\rangle \in T_n$ is indeed such that

$$|\text{B}(\theta(T)| = 1 + |\text{LB}(R')| + |\text{RB}(L')| = |\text{RB}(L)| + |\text{LB}(R)| = |\textbf{path}(1, n, T)|.$$

Note that the associated permutation $\theta(\sigma)$ can be described by the simple rule

$$\theta(\sigma) = (\sigma(i+1) + 1, \cdots, \sigma(n) + 1, 1, \sigma(1) + 1, \cdots, \sigma(i-1) + 1)$$

where $i = \sigma^{-1}(n)$ is the index of the largest element in σ.

The analysis of Algorithm 1 is performed under the hypothesis that the $n!$ order equivalence classes of Cartesian trees $(P \cup \langle i, +\infty\rangle)$, where $|P| = n - 1$ are equally likely. The number of comparisons c ? $x(p)$ is equal to $|\textbf{path}(1, n, T)|$, where T is the order equivalence class of $(P \cup \langle c, +\infty\rangle)$, and the rest of the proposition follows from bijection θ and Proposition 3 (see Figure 13). □

Algorithm 1 and the following algorithms are described recursively; it is routine work for a trained computer scientist to produce efficient nonrecursive versions of these algorithms.

3.5 Concatenation of Cartesian Trees

The preceding section establishes that CUT(P, c) is a one–one mapping between Cartesian trees P, value c, and pairs (L, R) = CUT(P, c) of Cartesian trees such that the x values of labels in L are $<c$, and they are $>c$ in R. Given such a pair (L, R), the converse of CUT is the *concatenation* of Cartesian trees, described by *Algorithm 2.*

Algorithm 2 (Concatenation of Cartesian trees L and R, such that $p \in L, q \in R \Rightarrow x(p) < x(q)$)

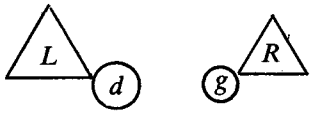
proc $CT \leftarrow$ CONS(L, R)
  Cartesian tree CT, L, R; **comment** $\forall p \in L, q \in R: x(p) < x(q)$;
  if $L = \varnothing$ **then** $CT \leftarrow R$
  elsif $R = \varnothing$ **then** $CT \leftarrow L$
  elsif $y(v(L)) < y(v(R))$ **then** $CT \leftarrow \langle l(T), v(L), \text{CONS}(r(L), R)\rangle$
  elsif $y(v(L)) > y(v(R))$ **then** $CT \leftarrow \langle \text{CONS}(L, l(R)), v(R), r(R)\rangle$
  fi
fproc CONS.

This algorithm is inverse to CUT in the sense that, performing $(L, R) \leftarrow \text{CUT}(T_1, c)$, $T_2 \leftarrow \text{CONS}(L, R)$ is equivalent to $T_2 \leftarrow T_1$. Precisely, it *merges* the two ordered sequences RB (L) and LB (L) into the ordered sequence **path**$(1, c, CT)$ where $c =$ **max**(RB (L), LB (R)). Although CUT(CT, c) and CONS(L, R) are mathematically inverse in this strong sense, the number of operations performed by both algorithms is *not* the same.

The reason is that the number C of comparisons performed in merging two ordered sequences $S = (s_1, \ldots, s_p)$ with $s_i < s_{i+1}$ for $1 \leq i < p$ and $R = (r_1, \ldots, r_q)$ with $r_j < r_{j+1}$ for $1 \leq j < q$ *is not equal* to $p + q = |R| + |S|$, but rather to $|R| + |S| - |H(R, S)|$ where $H(R, S) = \{r \in R \mid r > s_p\} \cup \{s \in S \mid s > r_q\}$ is the set of elements of R (and S) greater than all elements in S (and R).

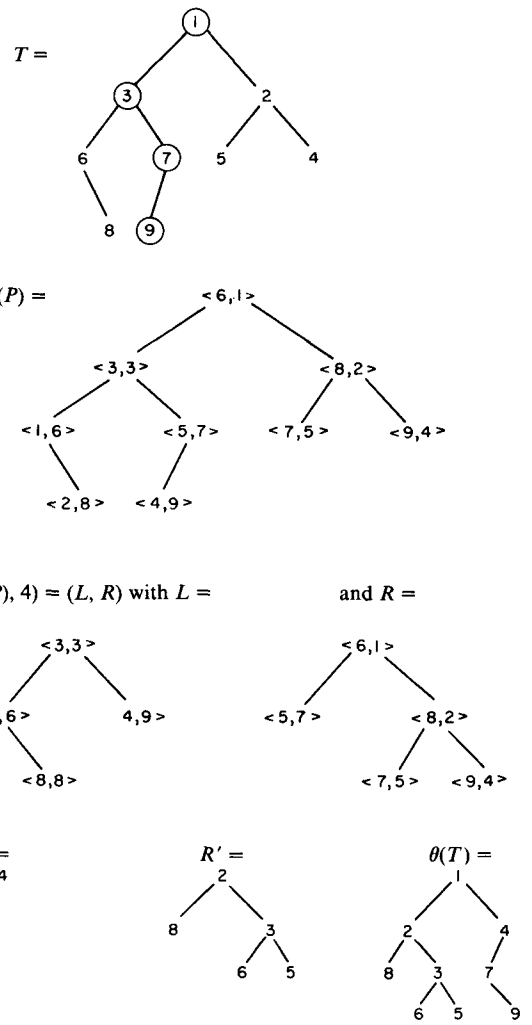
PROPOSITION 5. *The average number of comparisons $C(L, R)$ of the type $y(v(L))$? $y(v(R))$ in Algorithm 2 is equal to $H_{n+1} + H_{m+1} - 2$, where $n = |L|$, $m = |R|$.*

PROOF. Let $H(L, R)$ represent the union of points in RB (L) having a y coordinate greater than g together with points in LB (R) greater than d; here g(resp. d) is the point in LB (R) (resp. RB (R)) having the largest y coordinate. If $y(d) < y(g)$, we have $H(L, R) \subseteq$ LB (R), else $H(L, R) \subseteq$ RB (L). In both cases, $C(L, R) = |\text{RB}(L)| + |\text{LB}(R)| - |H(L, R)|$.



To find another expression of $C(L, R)$, we remark that, if $d < g$, then $C(L, R) = |\text{RB}(L)| + |\text{LB}(\text{CONS}(d, R))| - 1$; reestablishing symmetry yields the formula $C(L, R) = |\text{RB}(\text{CONS}(L, g))| + |\text{LB}(\text{CONS}(d, R))| - 2$. The average value $H_{n+1} + H_{m+1} - 2$ follows from the

Fig. 13. The steps of bijection $T \rightarrow \theta(T)$ in the proof of Proposition 4; here **path**(1, n, T) = (1, 3, 7, 9) and B ($\theta(T)$) = (8, 2, 1, 4).



assumption that all $\binom{n+m}{r}$ resulting order equivalence trees are equally likely, and we apply Proposition 3. □

The preceding analysis of the number of comparisons $C(L, R)$ in the algorithm CONS(L, R) illustrates an interesting point: This quantity has been analyzed *geometrically*, i.e., it has been shown to correspond to some intrinsic (*static*) parameter, expressed in terms of L, R, and CT = CONS(L, R). A more computational and naive approach to the analysis of this parameter can easily lead to quite complicated formulas, as well as yield a weaker result. We make other uses of the method of geometric analysis of algorithms throughout this paper.

4. Applications to Searching

4.1 Two-Dimensional Searching

Cartesian trees (Section 3.1) are a natural data structure for representing plane sets and performing various kind of searches. Natural procedures for MERGE, SEARCH, SEARCHRANGE, and EXTRACT can be designed and geometrically analyzed.

Here, we limit ourselves to describing and analyzing the insertion procedure INSERT(CT, p) which adds point p to the Cartesian tree CT. If p has a y value $y(p) < y(v(CT))$ less than that of the root of CT, INSERT merely cuts CT at $x(p)$ to yield (L, R) = CUT(CT, $x(p)$); the result S of insert is then $S = \langle L, p, R\rangle$. If $y(p) > y(v(CT))$, we search for the largest subtree Q of CT, with root on **path**(1, $x(p)$), CT) for which $y(p) < y(v(Q))$, in which case we insert p in Q as described in the previous case.

Algorithm 3 (Insertion of p in the Cartesian tree CT)

proc $S \leftarrow$ INSERT(CT, p)
  Cartesian tree S, CT, L, R; **point** p;
  if $CT = \emptyset$ **then** $S \leftarrow \langle\emptyset, p, \emptyset\rangle$
  elsif $y(p) < y(v(CT))$ **then** (L, R) $\leftarrow$ CUT(CT, $x(p)$); $S \leftarrow \langle L, p, R\rangle$
  elsif $x(p) < x(v(CT))$ **then** $S \leftarrow \langle$INSERT($l(CT)$, p), $v(CT)$, $r(CT)\rangle$
  else $S \leftarrow \langle l(CT), v(CT),$ INSERT($r(CT)$, p)$\rangle$
  fi
fproc INSERT.

This algorithm involves $C_y(CT, p)$ comparisons between y coordinates $y(p)$? $y(v(CT))$ and $C_x(CT, p)$ comparisons between x coordinates, either of the type $x(p)$? $x(v(CT))$ or occurring in the call to CUT(CT, $x(p)$). Comparisons C_y are equal to the depth of p in the result; using Proposition 4 and summing yields $2(1 + 1/n)\, H_n - 4$.

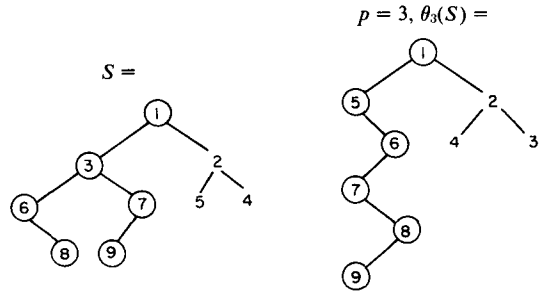
PROPOSITION 6. *The average number of x comparisons C_x in Algorithm 3 INSERT is $2H_n - 1$, where $n = |S|$ is the size of the result. The probability that $C_x = k$ is $t_{n,k}$.*

PROOF. The assumptions in this analysis are that S is a random element of T_n and that p can be any of the n element of S, with equal probability. Let T_p be the subtree of root p in S. Our parameter C_x is equal to $C_x = |\mathbf{path}(1, p, S)| + |\mathrm{RB}(l(T_p))| + |\mathrm{LB}(r(T_p))|$ (see Figure 14).

In order to analyze this parameter, we generalize bijection θ of Proposition 4 and construct a bijection θ_p which exchanges C_x with **path**(1, n, $\theta_p(S)$). Bijection θ is the special case $p = 1$.

The easiest way to describe bijection θ_p is to consider the permutation $\sigma_s = \mathcal{T}^{-1}(S)$ constructed by visiting S

Fig. 14. An example of application of bijection θ_3.

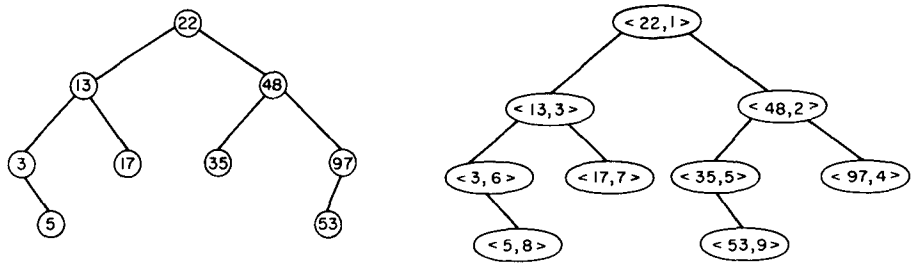


path(1, 3, S) = (1, 3)
LB ($r(T_3)$) = (7, 9)
RB ($l(T_3)$) = (6, 8)

$C_x(S, 3) = 6$

path(1, 9, $\theta_3(S)$) = (1, 5, 6, 7, 8, 9)

Fig. 15. Binary Search Tree and Binary Search Tree with Arrival Time Associated with $S = \langle 22, 48, 13, 97, 35, 3, 17, 5, 53 \rangle$.



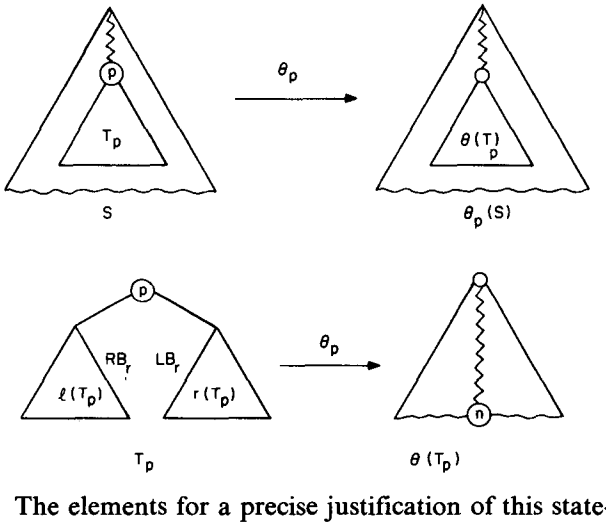
BST (S) BSTWAT (S)

in symmetric order; in the example of Figure 3, σ_s = (6 8 3 9 7 1 5 2 4). Let $\sigma_s = (\sigma(1), \ldots, \sigma(n))$ and define $\sigma' = \theta_p(\sigma_s) = (\sigma'(1), \ldots, \sigma'(n))$ by the rules:

(1) $\sigma'(i) = \sigma(i)$ iff $\sigma(i) < p$.
(2) $\sigma'(i) = \sigma(i) - 1$ iff $\sigma(i) > p$.
(3) $\sigma'(i) = n$ iff $\sigma(i) = p$.

We then define $\theta_p(S) = \mathcal{T}(\theta_p(\sigma_s))$ as the binary tournament associated with $\sigma' = \theta_p(\sigma_s)$.

The binary tournament $\theta_p(S)$ has the same shape as S, except for the subtree T_p in which the two branches RB($l(T_p)$) and LB($r(T_p)$) are mapped into **path**(root, n, $\theta_p(T_p)$).



The elements for a precise justification of this statement are given in the proof of Proposition 4.

Since parameter S_x has the same distribution as **path**(1, n, $\theta_p(S)$), the result follows from Proposition 3. □

4.2 Binary Search Trees

Let $S = \langle s_1, \ldots, s_n \rangle$ be an ordered sequence $S \in R^n$. With such a sequence can be associated the Cartesian tree $\mathcal{C}(P)$ where $P = \{\langle s_i, i \rangle \mid 1 \leq i \leq n\}$, which we call *binary search tree with arrival time* (BSTWAT) of S. The x part of $\mathcal{C}(P)$ is the usual *binary search tree* associated with S (see, for example, Knuth [13]); the y part of $\mathcal{C}(P)$ specifies the rank of each element in sequence S (see Figure 15).

Traditional algorithms SEARCH, INSERT, and EXTRACT are described and analyzed by Knuth [13]; a complete geometrical treatment of these analyses is provided by Françon [9]. In this paper, we limit ourselves to presenting an algorithm for MERGING two binary search trees, which is apparently original.

The algorithm for merging the two binary search trees G and D considers the label $x = v(D)$ at the root of D; using CUT, it produces $(L, R) = \text{CUT}(G, x)$ and proceeds recursively to merge L with $l(D)$, then R with $r(D)$. A precise description of MERGE is given in Algorithm 4.

Algorithm 4 (An algorithm for merging the two binary search trees G and D, producing the binary search tree S for result)

```
proc S ← MERGE(G, D)
  binary search tree S, G, D, L, R;
  if D = ∅ then S ← G
          else (L, R) ← CUT(G, v(D));
               S ← ⟨MERGE(L, l(D)), v(D), MERGE(R, r(D))⟩
  fi
fproc MERGE.
```

In this algorithm, comparisons occur during the call to CUT(G, $v(D)$).

PROPOSITION 7. *If G and D are independent random binary search trees of respective sizes n and m, the number of comparisons in MERGE(G, D) is equal to* $2(n + m + 1)H_{n+m} - 2(n + 1)H_n - 2(m + 1)H_m$.

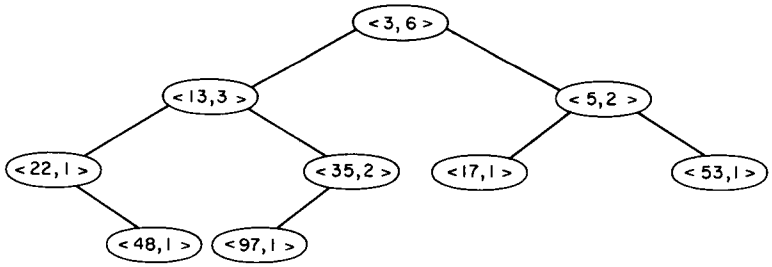
PROOF. To carry out the analysis, we consider two BSWATs, G' and D', having the same x part as G and D, respectively. The y labels in D' are $\{1, 2, \ldots, m\}$ and those in G' are $\{m + 1, \ldots, m + n\}$. We can look at MERGE as a succession of insertions (Algorithm 3) of the elements $p_1, \ldots, p_m$ of D', in order of increasing x-value into $G'_0 = G_0, \ldots, G'_i = G'_{i-1} + p_i, \ldots$.

During the insertion of the ith element, we only need to count the number of comparisons X_i of p_i with elements located *below* in the resulting tree G'_i. By Proposition 6, we know that **path**(1, p_i, G'_i) + X_i = **path**(1, n, G'_i), which combined with Proposition 4 yields $X_i = 2H_{n+i} - 2H_i$.

The result follows by summing, and by use of the identity $\sum_{1 \leq i \leq k} H_i = (k + 1)H_k - k$. □

This MERGE(G, D) procedure reduces to the classical INSERT (Knuth [13]) *at the leaves* in the case $|G| = 1$; in the case $|D| = 1$, we obtain the procedure of *insertion at the root* discovered by Stephenson [15]. When G and D are roughly of equal size, say $G = O(n)$ and

Fig. 16. The Sequence $S = \langle 22, 48, 13, 97, 35, 3, 17, 5, 53\rangle$ and Its Position Tournament Representation $\mathscr{P}(S)$.



$D = O(n)$, then Proposition 7 shows that MERGE is performed in *average linear time* $O(n)$. Note that this program, unlike the other ones presented in this paper, requires the use of a stack in its nonrecursive description.

5. Applications to Linear Lists

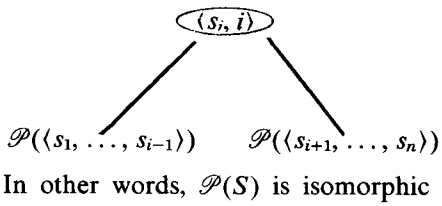
A data structure for representing *linear lists* allows one to manipulate sequences (or words) $\langle s_1, \ldots, s_n \rangle \in \mathbb{R}^*$, where the primitive operations allowed are

(1) SEARCH, INSERT, or DELETE the kth element of a list;
(2) CONCATENATE two lists or CUT a list in kth position.

We propose here to use a data structure derived from Cartesian trees, allowing very simple algorithms for each of these primitives to have an average execution time of $O(\log n)$, where n is the size of the lists manipulated.

To represent the list $S = \langle s_1, \ldots, s_n\rangle$, we use the *position tournament* $\mathscr{P}(S)$ defined by the rules:

(1) $\mathscr{P}(\Lambda) = \varnothing$.
(2) If $s_i = \min_{1\le j\le n}\{s_j\}$, then $\mathscr{P}(s_1, \ldots, s_n) =$



In other words, $\mathscr{P}(S)$ is isomorphic to the binary tournament $\mathscr{T}(S)$ of S (see Section 3.2), except that each label p_i in $\mathscr{P}(S)$ is formed by the pair $p_i = \langle x_i, r_i\rangle$, where x_i is the value of the ith element of S and $r_i - 1$ is the size of the left subtree $l(T_i)$ of root p_i in $\mathscr{P}(S)$.

If $p = \langle x, r\rangle$ is a label in $\mathscr{P}(S)$, we call $\mathbf{rank}(p) = r$ the size of its left subtree increased by 1 (see Figure 16).

Searching for the kth element in such a structure is isomorphic to searching a binary search tree: First consider the rank r of the root; if $r > k$, we search the kth element of the left subtree; if $r = k$, the root is the answer; if $r < k$, we search the $(k - r)$th element of the right subtree (see Algorithm 5).

Algorithm 5 (Searching for the kth element of position tree P)

```
proc X ← SEARCH(P, K)
  real X; position tournament P; integer K;
  comment 1 ≤ K ≤ |P|;
  if K = rank(v(P)) then X ← x(v(P))
  elsif K < rank(v(P)) then X ← SEARCH(l(P), K)
  else X ← SEARCH(r(P), K − rank(v(P)))
  fi
fproc SEARCH.
```

Analysis of this algorithm is a straightforward consequence of the previous results.

PROPOSITION 8. *The number C of comparisons in executing Algorithm 5, $X \leftarrow SEARCH(P, K)$ where $|P| = n$ is as follows*:

(*i*) *if* $k = 1$, *then* $\mathbf{average}(C) = H_n$ *and* $\mathbf{prob}(C = k) = 1/n!\cdot s_{n,k}$;
(*ii*) *if X is the largest x value in P, then* $\mathbf{average}(C) = 2H_n - 1$ *and* $\mathbf{prob}(C = k) = 1/n!\cdot t_{n,k} = 2^k/n!\cdot s_{n-1,k-1}$;
(*iii*) *the average value of C in searching for all values* $1 \le k \le n$ *is* $2(1 + 1/n)H_n - 3$.

PROOF. Conditions (i) and (ii) are mere rephrasing of Propositions 3 and 4. Note that the analysis can be refined in case (ii): the number of comparisons for which $K < \mathbf{rank}(v(P))$, i.e., the number i of executions of instruction $X \leftarrow \text{SEARCH}(r(P), K - \mathbf{rank}(v(P)))$ has $\binom{k}{i}s_{n-1,k-1}$ for distribution.

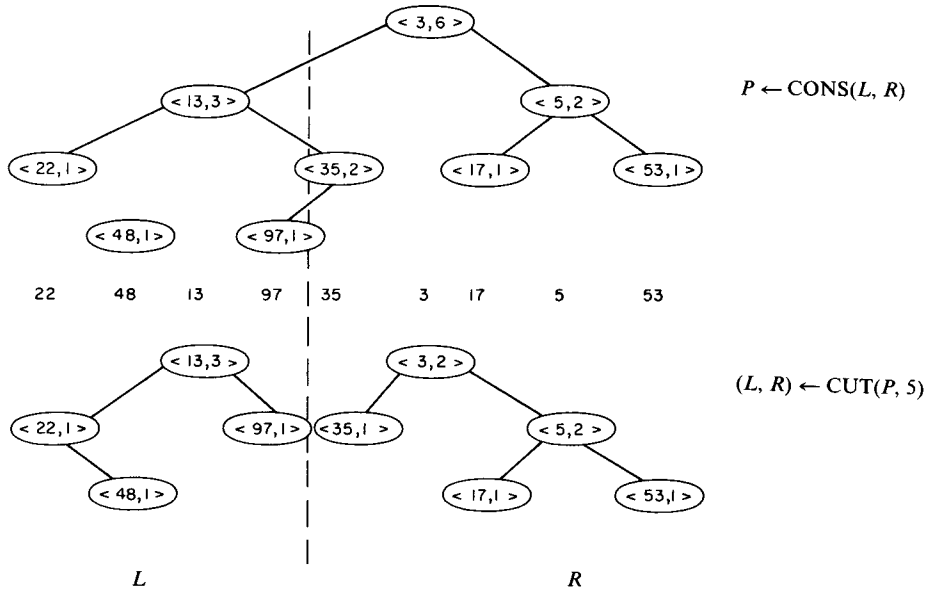
To compute the average value of C in case (iii), we note that it is equivalent to search P for each of its x values; by Proposition 4, searching for the jth value involves $2H_j - 1$ comparisons on the average, thus the result $(1/n)\sum_{1\le j\le n}(2H_j - 1)$. □

Algorithms for cutting a position tournament P in position K, with $0 \le K \le |P|$, and the inverse operation of concatenation, are straightforward adaptations (homomorphisms to be precise) of CUT and CONS Algorithms 1 and 2. Figure 17 illustrates these algorithms on an example.

Similarly, *insertion* in a position tournament is a direct adaptation of Algorithm 3, the analysis of Proposition 6 remaining valid.

Algorithm 6 gives a complete description of $(X, Q) \leftarrow \text{EXTRACT}(P, K)$ which extracts the kth element $X(1 \le k \le |P|)$ of position tournament P and yields the position tournament Q for result.

Fig. 17. Operations CUT and CONS on Position Tournaments.



Algorithm 6 (Extraction of the kth element of position tournament P)

proc $(X, Q) \leftarrow$ EXTRACT(P, K)
  real X; **position tournament** P, Q, S; **integer** K;
  comment $1 \leq K \leq |P|$;
  if $K =$ **rank**$(v(P))$ **then** $X \leftarrow x(v(P))$; $Q \leftarrow$ CONS$(l(P), r(P))$
  elsif $K <$ **rank**$(v(P))$ **then** $(X, S) \leftarrow$ EXTRACT$(l(P), K)$;
    $Q \leftarrow \langle S, \langle x(v(P)),$ **rank**$(v(P)) - 1\rangle, r(P)\rangle$
  elsif $K >$ **rank**$(v(P))$ **then** $(X, S) \leftarrow$ EXTRACT$(r(P), K -$ **rank**$(v(P)))$;
    $Q \leftarrow \langle l(P), v(P), S\rangle$
  fi
fproc EXTRACT.

This algorithm performs two types of comparisons: Rank comparisons K ? **rank**$(v(P))$ are analyzed in Proposition 8; as for x comparisons that take place during recursive calls to CONS$(l(P), r(P))$, an analysis by Françon et al. [10] shows that the average value of this parameter is $1 - (2H_n/n) + 1/n$, where $n = |P|$.

6. Other Applications and Conclusions

Other interesting algorithms use binary tournaments as their underlying combinatorial structure. The most widely used is the sorting algorithm Quicksort of Hoare [12], which has been completely analyzed by Sedgewick [14]. The key to Quicksort is a partitioning algorithm, whose successive applications implicitly constructs a *binary search tree;* the combinatorial tools shown in this paper can be readily applied to confirm the analysis of Sedgewick [14].

Another application has been discovered by Françon et al. [10] who propose to represent *priority queues* (see Knuth [13] for a definition) by binary tournaments. The key to their result is a data structure, *the pagoda*, which is an upside-down representation of binary tournaments; the discovery of pagodas arises from a careful inspection of the CONS Algorithm 2, showing that a *bottom-up* merging of the sorted sequences RB(L) and LB(R) is more efficient than the straightforward *top-down* method of this paper.

To conclude, we note that a careful use of the *same* combinatorial structure, namely, binary tournaments, leads to a wide variety of algorithms and data structures, for sorting (quicksort), representing linear lists (position tournaments), and priority queues (pagodas). These algorithms have a definite *practical significance*, since each of them provides the *fastest average time solution known* to its specific problem.

Our contention here is that a close examination of the underlying combinatorial structure (in our case permutations) brings up interesting new algorithms and data structures; it also brings a surprising amount of unity to a field which we think badly needs it. One worthwhile goal in that respect would be to apply the same kind of work to another family of algorithms, including digital search tree, h-code, and radix sorting (as described and analyzed by Knuth [13]) and to bring to light the underlying combinatorial structure common to so many of these algorithms. Despite the enormous number of interesting solutions known for solving basic data manipulation problems, we believe that most of these solutions ultimately rest on a relatively small number of different combinatorial structures. A progress in our understanding of these questions should drastically affect the way in which we discover and explain the fundamental algorithms, as catalogued by Knuth [13] and Aho et al. [1].

7. Bibliographical Note

An account of methods in combinatorial geometry is given by Foata and Schutzenberger [7], which has had great influence on the present work.

The classical results of Section 2 of this paper can be found, for example, in Comtet [3]. Construction of the

binary tournament (Section 3) associated with a permutation is used by Foata and Schutzenberger [7] and Foata and Strehl [8]. Various algorithmic uses of the construct can also be found in Burge [4] and Viennot [16].

Binary search trees are discussed in particular by Knuth [13] and Françon [9].

The analysis of position tournament algorithms is implicit in Françon et al. [10], where application to priority queues is discussed. Balanced tree representations of linear lists are described by Aho et al. [1], Guibas et al. [11], and Brown and Tarjan [2].

Finally, a much more comprehensive treatment of combinatorial methods in algorithm design and analysis is attempted by Flajolet et al. [5].

Acknowledgments. The results presented here are aspects of more general joint research with P. Flajolet, J. Françon, and G. Viennot. The main ideas presented in this paper all evolved from this joint work.

Received 4/79; revised 9/79; accepted 12/79

References

1. Aho, A.V., Hopcroft, J., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. Brown, M.R., and Tarjan, R.E. A representation for linear lists with movable fingers. In Proc. 10th Ann. ACM Symp. Theory of Comptg., 1978, pp. 19–29.
3. Comtet, L. *Advanced Combinatorics.* D. Reided Pub. Co., Boston, Mass., 1974.
4. Burge, W.H. An analysis of a tree sorting method and some properties of a set of trees. In Proc. 1st USA–Japan Comptr. Conf., 1972, pp. 372–378.
5. Flajolet, P., Françon, J., Viennot, G., and Vuillemin, J. Algorithmique et combinatoire des arbres et permutations. To appear (1981).
6. Flajolet, P., Françon, J., and Vuillemin, J. Sequence of operation analysis for dynamic data structures. *J. Algorithms.* To appear (1980).
7. Foata, D., and Schutzenberger, M.P. Theorie géométrique des polynômes Euleriens. *Lecture Notes in Mathematics, No. 138,* Springer-Verlag, Berlin, 1970.
8. Foata, D., and Strehl, V. Rearrangements of the symmetric group and enumerative properties of the tangent and secant numbers. *Math. Zamet. 137* (1974), 256–264.
9. Françon, J. Arbres binaires de recherche, propriétés combinatoires et applications. *RAIRO Informatique Théorique 10* (1976), 35–50.
10. Françon, J., Viennot, G., and Vuillemin, J. Description et analyse d'une représentation performante des files de priorité. Rep. 12, Lab. d'Informatique, Orsay, France; also in Proc. 19th Ann. ACM Symp. on Foundations of Comptr. Sci., 1978, pp. 1–7.
11. Guibas, L., Creight, E.M.M., Plass, M.F., and Roberts, J.R. A new representation for linear lists. In Proc. 9th Ann. ACM Symp. Theory of Comptg., 1977, pp. 49–60.
12. Hoare, C.A.R. Quicksort. *Comptr. J. 5,* 1 (1962), 10–15.
13. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
14. Sedgewick, R. Quicksort. Rep. STAN-CS-75-492, Dept. Comptr. Sci., Stanford Univ., Stanford, Calif., May 1975.
15. Stephenson, C.J. A method for constructing binary search trees by making insertions at the root. Rep. RC 6298, IBM Thomas J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
16. Viennot, G. Quelques algorithmes de permutations. Astérique 38–39, Societé Mathématique de France, 1976, pp. 275–293.

Professional Activities Calendar of Events

ACM's calendar policy is to list open computer science meetings that are held on a not-for-profit basis. Not included in the calendar are educational seminars, institutes, and courses. Submittals should be substantiated with name of the sponsoring organization, fee schedule, and chairman's name and full address.

One telephone number contact for those interested in attending a meeting will be given when a number is specified for this purpose.

All requests for ACM sponsorship or cooperation should be addressed to Chairman, Conferences and Symposia Committee, Seymour J. Wolfson, 643 MacKenzie Hall, Wayne State University, Detroit, MI 48202, with a copy to Louis Fiora, Conference Coordinator, ACM Headquarters, 1133 Avenue of the Americas, New York, NY 10036; 212 265-6300. For European events, a copy of the request should also be sent to the European Representative. Technical Meeting Request Forms for this purpose can be obtained from ACM Headquarters or from the European Regional Representative. Lead time should include 2 months (3 months if for Europe) for processing of the request, plus the necessary months (minimum 3) for any publicity to appear in *Communications.*

■ This symbol indicates that the Conferences and Symposia Committee has given its approval for ACM sponsorship or cooperation.

In this issue the calendar is given to November 1981. New Listings are shown first; they will appear next month as Previous Listings.

NEW LISTINGS

15–17 April 1980

■ **Annual Conference on Computer Graphics,** Detroit, Mich. Sponsor: Engineering Society of Detroit in cooperation with ACM. Conf. chm: Fred Langhorst, Transportation Systems Center, General Motors Technical Center, Warren, MI 48090; 313 575-8311.

29-30 April 1980

Sixth Illinois Conference on Medical Information Systems, Champaign, Ill. Sponsors: University of Illinois, Regional Health Resource Center, Society for Advanced Medical Systems, Society for Computer Medicine, University of Missouri Health Care Technology Center. Contact: Saundra Wheeler, 1408 W. University, Urbana, IL 61801.

30 April 1980

Capacity Planning and Shop Floor Control, Syracuse, N.Y. Sponsors: ACM Syracuse Chapter, American Production and Inventory Control Society Syracuse Chapter. Contact: Michael Busse, Anaren Microwave Inc., 185 Ainsley Dr., Syracuse, NY 13205 or Hamilton Armstrong, Carrier Corp., Box 4895, Syracuse, NY 13221.

6–10 May 1980

Canadian Association for Information Science Annual Conference, Toronto, Ont., Canada. Sponsor: CAIS. Contact: Ilse Cockburn, 36 Brookdale Ave., Toronto, Ontario, Canada M5M 1P3.

15–16 May 1980

NYU Symposium on Distributed Processing Practice, New York City. Sponsor: Graduate School of Business Administration of New York University. Contact: CAIS Dept., 700 Merrill Hall, 90 Trinity Place, New York, NY 10006; 212 285-6120.

3–6 June 1980

International Conference on Boundary and Interior Layers—Computational and Asymptotic Methods (BAIL I), Dublin, Ireland. Sponsor: Numerical Analysis Group. Contact: BAIL I Conference, 39 Trinity College, Dublin 2, Ireland.

16–19 June 1980

Thirteenth Annual Conference of Association of Small Computer Users in Education, University of Tennessee, Martin. Sponsor: ASCUE. Contact: James Westmoreland, Computer Center, University of Tennessee, Martin, TN 38238; 901 587-7891.

26–27 June 1980

■ **ACM SIGPCR Seventeenth Annual Computer Personnel Research Conference,** Boca Raton, Fla. Sponsor: ACM SIGPCR. Conf. chm: Elias M. Awad, College of Business and Organizational Sciences, Florida International University, Miami, FL 33199; 305 552-2791.

30 June 1980

Panel on Software Metrics, Washington, D.C. Sponsors: Yale University and Office of Naval Research. Contact: ONR Software Metrics Panel, Attn. Alan J. Perlis, Computer Science Dept., Yale University, 2158 Yale Station, New Haven, CT 06520.

19–21 August 1980

■ **National Artificial Intelligence Conference,** Palo Alto, Calif. Sponsor: American Association for Artificial Intelligence in cooperation with ACM SIGART. Conf. chm: J. Marty Tenenbaum, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025; 415 326-6200 x4167.

3–5 September 1980

19th Annual Lake Arrowhead Workshop on Office Information Systems, near Los Angeles, Calif. Sponsor: IEEE-CS. Workshop co-chm: Clarence A. Ellis and Gary J. Nutt, Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304.

15–17 October 1980

V ICCRE, Fifth International Conference on Computers in Chemical Research and Education, Toyohashi, Japan (a post congress symposium of Seventh International CODATA Conference, Kyoto, Oct. 8-11). Contact: S. Sasaki, School of Materials Science, Toyohashi University of Technology, Tempaku, Toyohashi, Japan 440.

2–5 November 1980

Fourth Annual Symposium on Computer Applications in Medical Care, Washington, D.C. Sponsor: The George Washington University Medical Center. Prog. chm: Joseph T. O'Neill, National Center for Health Services Research, Center Building, Room 8-30 #1, 3700 East-West Highway, Hyattsville, MD 20782; 301 436-8946.

30 November–2 December 1980

■ **Micro 13—13th Annual Workshop on Microprogramming,** Colorado Springs, Colo. Sponsors: ACM SIGMICRO, IEEE-CS. Conf. chm: G.R. Johnson, Dept. of Engineering Science, Colorado State University, Fort Collins, CO 80523; 303 491-7585.

3–8 December 1980

■ **1980 Winter Simulation Conference,** Orlando, Fla. Sponsors: ACM SIGSIM, ORSA, TIMS, AIEE, SCS, U.S. Dept. of Energy. Conf. chm: Paul F. Roth, U.S. Dept. of Energy, Mail Stop 4530, 12th and Penn, Washington, DC 20461; 202 633-9629.

3–5 February 1981

■ **Fifth Berkeley Workshop on Distributed Data Management and Computer Networks,** Emeryville, Calif. Sponsor: Lawrence Berkeley Laboratory in cooperation with ACM. Conf. chm: Rowland R. Johnson, Lawrence Berkeley Laboratory, University of California, Berkeley, CA 94720; 415 486-6321.

19–22 April 1981

■ **Computing for Development,** Bangkok, Thailand. Sponsors: Carl Duisberg Gesellschaft, Asian Institute of Technology in cooperation with ACM SIGBDP, SIGCAS, SIGMOD, SIGSIM. Conf. chm: M. Nawaz Sharif, Div. of Computer Applications, Asian Institute of Technology, P.O. Box 2754, Bangkok, Thailand.

(*Calendar continued on p. 242*)