# Three Simple and Efficient Algorithms for Jordan Sorting

Eduard Sojka
Department of Computer Science, Technical University of Ostrava,
tr.17.listopadu, 708 33 Ostrava-Poruba, Czech Republic
eduard.sojka@vsb.cz

In this work, three simple and efficient algorithms for Jordan sorting are presented. The results of the analysis of their time complexity as well as the results of testing the algorithms are given.

**Problem** (*Jordan sorting*). Given N intersection points of a Jordan curve with the x-axis in the order in which they occur along the curve, sort these points into the order in which they occur along the x-axis (Fig. 1a).

In the algebraic decision-tree model of computation, the worst-case time complexity of general sorting is θ(N log N). Contrary to general sorting, the time complexity of Jordan sorting is only θ(N). Hoffman et al. [Hoffman86] proposed an O(N) worst-case time algorithm for solving the Jordan sorting problem and polygon clipping. In their algorithm, they used a sophisticated data structure, the *level-linked search tree*. Later, Fung et al. [Fung90] devised another similar O(N) time algorithm. In this algorithm, they replaced the level linked search tree with the *heterogeneous finger tree*. Although their algorithm is somewhat simpler than that proposed by Hoffman, it is still rather complicated.

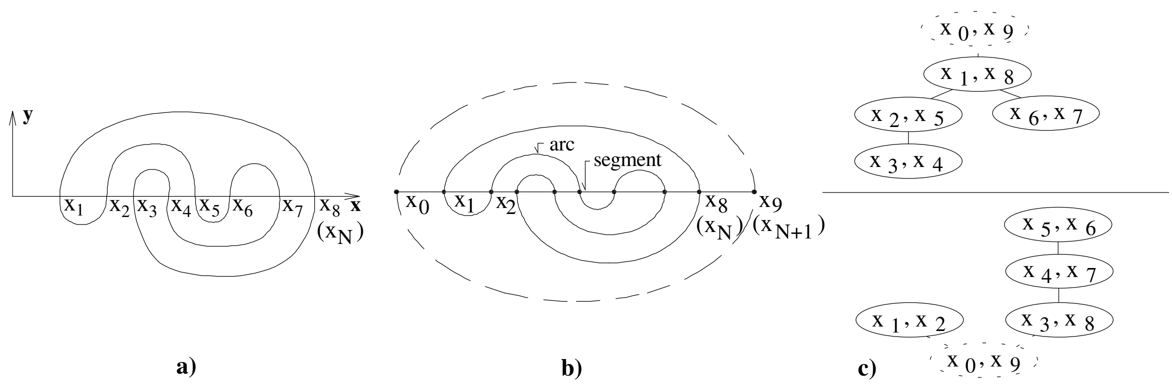


***Figure 1.** **a)** Cutting a Jordan curve by the x-axis; $x_8,x_1,x_2,x_5,x_6,x_7,x_4,x_3$ is an example of the input sequence for Jordan sorting. **b)** The map $\mathscr{m}$ corresponding to the final stage of sorting. **c)** The Hasse diagrams of the relation of enclosing are trees. The dotted nodes correspond to the additional arcs connecting the additional points $x_0$ and $x_{N+1}$.*

The algorithms presented in this work are simple. They are based on successively constructing a planar map $\mathscr{m}$ that corresponds to the given input (Fig. 1b). The computation starts with an initial map $\mathscr{m}^1$ (Fig. 2). The algorithms then read the intersection points from the input one by one and find the segment of the x-axis to which the new intersection point falls. Once this segment is found, the map is updated. During the computation, a sequence $\mathscr{m}^1,\mathscr{m}^2,\ldots,\mathscr{m}^N,\mathscr{m}$ of the maps is thus constructed (Fig. 2). Once the map $\mathscr{m}$ is found, the ordered sequence of points is read from this map. In the algorithms, the working data structure is the doubly linked list representing the map. Only Algorithm 2 uses also the binary search tree. The algorithms differ in the method which is used to find the segment containing the new intersection point. In the rest of this abstract, X will denote the intersection point that is just being processed, and PrevX will denote the point that was processed immediately before X.

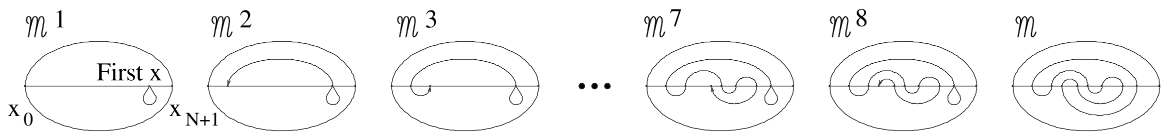


***Figure 2.*** *An example of the sequence of the maps that are constructed during the computation.*

**A Simple O(N log N) Algorithm (Algorithm 1)**

The step finding the segment into which the new intersection point X falls proceeds as follows: Beginning with PredX, go sequentially and simultaneously in both directions along the boundary of the face containing the arc PredX,X of the curve (Fig. 3a) and test the segments of the x-axis lying on this boundary whether they contain X. The order in which the segments are tested is important for the time complexity of the algorithm. The time complexity of this algorithm is analysed. Although its runnig time is O($N\log_2 N$) in the worst case, it is shown that certain properties make the algorithm useful for practical use: (a) The algorithm is extremely easy to implement. (b) The worst running time is achieved only if the input sequence is generated according to certain rules. Any deviation from these rules causes that the running time decreases. The “simpler“ the input sequence is, the shorter is the running time. (c) For a wide class of randomly generated Jordan sequences, the algorithm runs in linear expected time. (d) The algorithm is always better than the standard sorting algorithms (e.g., Quicksort) if they are used to sort the Jordan sequences. The corresponding assertions are presented in the form of theorems (with proofs). For example:

**Theorem 1.** Algorithm 1 requires no more than $N(4+\log_2 N)$ tests deciding whether x lies inside the segment.

**Theorem 2.** Consider the pair of Hasse diagrams corresponding to a given input of size N (Fig. 1c). Suppose that in total, the diagrams have Q inner nodes (including the roots of both trees) and, therefore, N-Q+2 leaves. Let $n_i$ denote the i-th inner node, let $d_i$ be the degree of this node (i.e., the number of its child nodes), let $L_i$ denote the number of leaves of the sub-tree whose root lies in $n_i$, and let $h_j$ denote the depth of the j-th leaf in the Hasse diagram. For the total number of tests, denoted by T, deciding whether x lies inside a segment the following inequalities hold

a) $T \le 2N + 2\sum_{i=1}^{Q} L_i$, b) $T \le 2N + 2\sum_{i=1}^{N-Q+2} h_j$.

**An O(N log log N) Worst-Case Time Algorithm (Algorithm 2)**

This algorithm improves the previous one. The method of reducing the running time consists in the following: During the computation, the intersection points are organised in groups (Fig 3b). Each group contains no more than C consecutive (along the x-axis) points. The first intersection points of the groups are organised in a height-balanced binary search tree. If a group containing more than C points occurs during the computation, then one half of points are split off from this group, and a new group is created for them. The first intersection point of the new group is then inserted into the search tree. We put $C=\lambda\lceil \log_2 N\rceil$ where $\lambda$ is a constant. The process of searching for the segment containing the new intersection point X proceeds as follows. Beginning with PredX, the segments lying on the boundary of the face containing the arc PredX,X are tested sequentially and simultaneously in both directions (Fig. 3a). If the segment containing X is found, the process stops. The process also stops if C segments were tested and the segment containing X still was not found. In this case, the binary search tree is used to find the group containing X. Inside the group, the sequential search is used to find the segment containing X.

**Theorem 3.** Algorithm 2 runs in $O(N \log_2 \log_2 N)$ worst-case time and requires O(N) space.

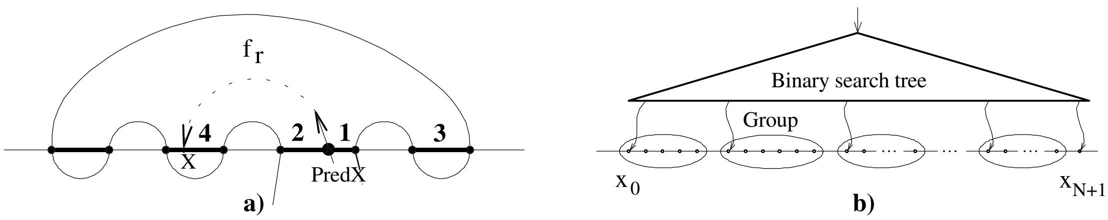


***Figure 3.*** ***a)*** *On the order in which the segments are tested.* ***b)*** *In Algorithm 2, the intersection points are organised in groups. The first points of groups are organised in a binary search tree.*

### Nearly Optimal Algorithm (Algorithm 3)

This algorithm solves the Jordan sorting problem in $O(N\psi(N))$ worst-case time where $\psi(N)$ is an extremely slowly growing function. For example, $\psi(2^3)=2$, $\psi(2^9)=3$, $\psi(2^{263})=4$,.... The technique of reducing the time complexity generalises the technique used in Algorithm 2. Whenever a face whose boundary contains more than a certain number of segments occurs, it is split into the two faces by an additional arc (additional arc is not a part of the curve).

### Implementation and Experimental Results

We have implemented and tested all the algorithms proposed in this paper. The time complexity of the algorithms was measured by the number of the tests that were needed to sort a given input sequence and by the real running time on a computer. The sequences leading to the worst running times of the algorithms and randomly generated sequences were used for testing. The tests confirmed the results of the theoretical analysis. In addition, the experimental results were compared with the corresponding values for QuickSort. In all cases, even Algorithm 1 gave significantly better results than QuickSort.

### References

Fung,K.Y., Nicholl,T.M., Tarjan,R.E., Van Wyk,C.J. 1990 Simplified linear-time Jordan sorting and polygon clipping, *Information Processing Letters*, Vol. 35, pp 85-92.

Hoffman,K., Mehlhorn,K., Rosenstiehl,P., Tarjan,R.E. 1986 Sorting Jordan sequences using level-linked search trees, *Inform. and Control*, Vol. 68, pp 170-184.