# Synergistic Solutions for Merging and Computing Planar Convex Hulls

**Jérémy Barbay**[1] **and Carlos Ochoa**[1]

1 **Departamento de Ciencias de la Computación, Universidad de Chile, Chile**
jeremy@barbay.cl, cochoa@dcc.uchile.cl

## Abstract

The analysis of algorithms beyond the worst case complexity over instances of fixed input size considers the input order or the input structure, but rarely both at the same time. Barbay et al. (2017) described "synergistic" solutions on multisets, which take advantage of some notions of input order and input structure, such as to asymptotically outperform any comparable solution which takes advantage only of one of those features. We consider the extension of their results to the computation of the CONVEX HULL of a set of planar points. After revisiting and improving previous approaches, taking advantage only of the input order or of the input structure, we describe synergistic solutions taking optimally advantage of various notions of input order and input structure in the plane. As intermediate results, we describe and analyze the first adaptive algorithm for merging a set of CONVEX HULLS.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems (Geometrical problems and computations); H.3.3 Information Search and Retrieval (Search and Selection process).

**Keywords and phrases** Convex Hull, Multivariate Analysis, Synergistic Solution.

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2017.46

## 1 Introduction

One way to close the gap between practical performance and the worst case complexity over instances of fixed input size is to refine the latter, considering smaller classes of instances defined via difficulty measures. Such difficulty measures can be seen along two axis: some depend on the *structure* of the input, such as the repetitions in a multiset [13], or the relative positions of the input points (in the plane [1] or in higher dimensions [9]); while some others depend on the *order* in which the input is given (such as for permutations [12], but also for points in the plane [2, 11]).

Barbay et al. [4] described various "synergistic" solutions on multisets, which take advantage of some notions of input structure (i.e., the repetition of the elements in the multiset) and of input order (i.e., the number and sizes of the runs in the multiset), both at the same time. Each of their solutions asymptotically outperforms other solutions taking advantage of a subset of these features.

In the context of the computation of the CONVEX HULL, various refinements of the worst case complexity over instances of fixed input size have been known for the last 30 years. In 1986, Kirkpatrick and Seidel described an algorithm optimal in the worst case over instances of fixed input size $n$ and output size $h$, for the computation of the CONVEX HULL in the plane [10]. They showed that the time complexity of this algorithm is within $O(n(1 + \log h))$: such a result can be classified as focused on the **input structure**, as $h$ depends only on the relative positions of the points, and is independent of the order in which the points are given. Afshani et al. [1] further refined this result in 2017, to optimally take advantage of any measure of input structure in 2 and 3 dimensions (i.e., taking advantage of the relative positions of the points while ignoring the input order). They showed that the time complexity of the algorithm described by Kirkpatrick and Seidel [10] is within $O(n(1+\mathcal{H}(n_1,\ldots,n_h))) \subseteq O(n(1+\log h)) \subseteq O(n\log n)$, where $n_1,\ldots,n_h$ (such that $\sum_{i=1}^{h} n_i = n$) are the sizes of a partition of the input points by enclosing triangles, such that every triangle is completely below the UPPER HULL of the points, with the minimum possible value for $\mathcal{H}(n_1,\ldots,n_h) = \sum_{i=1}^{h} \frac{n_i}{n}\log\frac{n}{n_i} \leq \log h$. Following a distinct approach, Levcopoulos et al. [11] described in 2002 an algorithm to compute the CONVEX HULL in the plane in time within $O(n(1+\log\kappa))$, where $\kappa$ is the minimal number of simple subchains into which the input sequence of $n$ points can be partitioned: such a result can be classified as one that focuses on the **input order**, as $\kappa$ depends on the order in which the points are given. Yet no algorithm (beyond a trivial dovetailing combination of the solutions described above) is known to take advantage of both the **input structure** and the **input order** at the same time for the computation of the CONVEX HULL, in the plane or in higher dimension, nor for any other problem than SORTING MULTISETS.

**Hypothesis.** It seems reasonable to expect that Barbay et al.'s synergistic results [4] on SORTING MULTISETS should generalize to similar problems in higher dimension, such as the computation of the CONVEX HULL of a set of points in the plane; and that such generalizations would find numerous practical applications: the largest point data sets are often conglomerates of smaller ones, which induce some structure and order. Yet this problem presents new difficulties on its own: (1) while the results on multisets [4] are strongly based on a variant of Demaine et al.'s instance optimal algorithm for MERGING MULTISETS [7], to this date the closest known result for MERGING CONVEX HULLS [3] (from 2008) is not adaptive to the size of the output, and hence not adaptive to the **input structure**; furthermore (2) whereas many **input order** adaptive results are known for SORTING MULTISETS (with two surveys in 1992 on the topic [8, 12], and some additional results [15] since then), it seems that only a few are known for the computation of the CONVEX HULL [2, 11].

**Our Results.** We confirm the hypothesis by (i) improving previous techniques to analyze the computation of CONVEX HULLS in function of the input order (i.e., considering not only the minimal number of simple chains into which the input sequence of points can be partitioned, but also the distribution of their sizes) (Theorem 1 of Section 2 and Theorem 12 of Section 4); (ii) presenting a new solution for MERGING CONVEX HULLS (Theorems 9 and 11 of Section 3) in the plane; and (iii) synthesizing all those results into a synergistic algorithm to compute the CONVEX HULL of a set of planar points (Theorem 13 of Section 4). The merging convex hull algorithm (described in Section 3) takes advantage of the number of convex hull sequences and of the fact that the points in the convex hull sequences are given in sorted order. As far as we know, this is the first adaptive algorithm that computes the union of a set of convex hulls while taking optimally advantage of the input structure (i.e., the relative position of the points). The synergistic algorithm that computes the convex hull (described in Section 4) outperforms the algorithms described by Kirkpatrick and Seidel [10] and by Levcopoulos et al. [11], by taking advantage of both the order (i.e., the number and distribution of the sizes of the simple chains into which the sequence of points can be partitioned) and the structure (i.e., the output size and the relative positions of the points) in the input, in a synergistic way. We conclude in Section 5 with a description of immediate corollaries for the MAXIMA problem, and a partial list of issues left open for improvement.

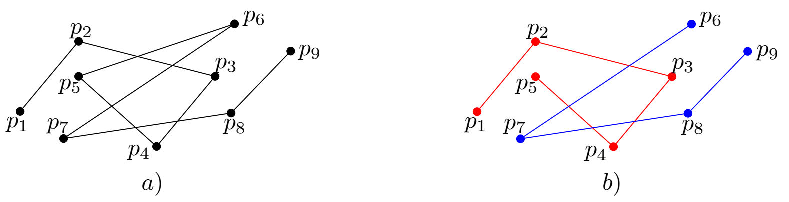

**Figure 1** a) A polygonal chain $\mathcal{P}$ specified by a sequence of 9 points. b) The decomposition of $\mathcal{P}$ into two simple subchains.

## 2 Input Order Adaptive Computation of the Convex Hull

A *polygonal chain* is a curve specified by a sequence of points $p_1, \dots, p_n$. The curve itself consists of the line segments connecting the pairs of consecutive points. A polygonal chain is *simple* if it does not have a self-intersection. (See Figure 1 for examples of these definitions.) Levcopoulos et al. [11] described an algorithm that computes the Convex Hull of $n$ planar points, taking advantage of the minimal number of simple subchains into which the polygonal chain specified by the input sequence of points can be partitioned. The algorithm tests if the polygonal chain $\mathcal{P}$ is simple: if $\mathcal{P}$ is simple, it computes the convex hull of $\mathcal{P}$ in linear time. Otherwise, if $\mathcal{P}$ is not simple, it partitions $\mathcal{P}$ into two subchains, whose sizes differ at most by one; recurses on each of them; and merges the resulting convex hulls. They measured the complexity of this algorithm in terms of the minimal number of simple subchains $\kappa$ into which the polygonal chain $\mathcal{P}$ can be partitioned. Let $t(n,\kappa)$ be the worst-case time complexity taken by this algorithm for a polygonal chain of $n$ points that can be partitioned into $\kappa$ simple subchains. They showed that $t(n,\kappa)$ satisfies the following recursion relation: $t(n,\kappa) \leq t(\lceil \frac{n}{2} \rceil, \kappa_1) + t(\lfloor \frac{n}{2} \rfloor, \kappa_2), \kappa_1 + \kappa_2 \leq \kappa + 1$. The solution to this recursion gives $t(n,\kappa) \in O(n(1+\log\kappa)) \subseteq O(n\log n)$. We describe next a refined analysis which takes into account the relative imbalance between the sizes of the simple subchains.

▶ **Theorem 1.** *Given a sequence $\mathcal{P}$ of $n$ planar points that can be partitioned into $\kappa$ simple subchains of respective sizes $r_1, \dots, r_\kappa$, Levcopoulos et al.'s algorithm [11] computes the convex hull of $\mathcal{P}$ in time within $O(n(1 + \mathcal{H}(r_1, \dots, r_\kappa))) \subseteq O(n(1+\log\kappa)) \subseteq O(n\log n)$, where $\mathcal{H}(r_1, \dots, r_\kappa) = \sum_{i=1}^{\kappa} \frac{r_i}{n} \log \frac{n}{r_i} \leq \log\kappa$. This time complexity is worst-case optimal over instances of $n$ points that can be partitioned into $\kappa$ simple subchains of sizes $r_1, \dots, r_\kappa$.*

**Proof.** Fix the subchain $c_i$ of size $r_i$. In the worst case, the algorithm considers the $r_i$ points of $c_i$ for the simplicity test and the merging process, in all the levels of the recursion tree from the first level to the level $\lceil \log \frac{n}{r_i} \rceil + 1$. In the next level, one of the nodes of the recursion tree fits completely inside $c_i$ and therefore it becomes a leaf. Hence, at least $\frac{r_i}{4}$ points from $c_i$ are discarded for the following iterations. In all of the following levels, the number of operations of the algorithm involving points from $c_i$ is bounded by the size of the subchains in those levels. The sum of the numbers of such operations is therefore within $O(r_i)$. As a result, the number of operations performed by the algorithm involving points from $c_i$ is within $O(r_i \log \frac{n}{r_i} + r_i)$. In total, the time complexity of the algorithm is within $O(n + \sum_{i=1}^{\kappa} r_i \log \frac{n}{r_i}) = O(n(1 + \mathcal{H}(r_1, \dots, r_\kappa))) \subseteq O(n(1+\log\kappa)) \subseteq O(n\log n)$. We defer the proof of the corresponding tight lower bound to the Appendix A. ◀

We define in the following section an algorithm for Merging Upper Hulls. This algorithm is the keystone of the synergistic algorithm that computes the Convex Hull of a set of planar points that we present in Section 4.

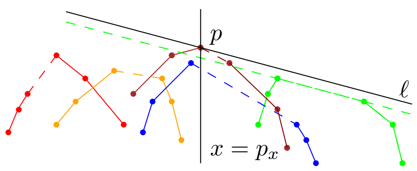

**Figure 2** An instance of the Merging Upper Hulls problem. The middle edge of each upper hull sequence is marked by dashed segments. The straight line $\ell$ is the supporting line of slope $\mu$, where $\mu$ is the median of the slopes of the middle edges. The line $\ell$ passes through the "pivot" point $p$. The algorithm `Quick Union Hull` partitions the upper hull sequences by the vertical line $x = p_x$.

## 3 Union of Upper Hulls

The computation of Convex Hulls reduces to the computation of Upper Hulls [10]. We describe the algorithm `Quick Union Hull`, which computes the Upper Hull of the union of upper hull sequences in the plane, assuming that in each upper hull sequence the points are given in sorted order by their $x$-coordinates. This algorithm is inspired by the algorithm `Simplified Ultimate Planar Convex Hull` described by Chan et al. [6], and by the algorithm `Quick Synergy Sort` described by Barbay et al. [4]. It takes advantage of the number of upper hull sequences, and of the fact that the points in the upper hull sequences are given in sorted order. It uses a divide-and-conquer technique to take advantage of the relative positions of the points.

### 3.1 Description of the algorithm Quick Union Hull

The algorithm `Quick Union Hull` identifies a point of the output, and uses this point as a "pivot" in the divide-and-conquer approach. To identify this point, it computes the median $\mu$ of the slopes of the middle edges of the upper hull sequences, in time linear in the number of upper hull sequences. Its pseudocode is described in Algorithm 1. It computes the point $p$ that has a supporting line $\ell$ (i.e., a straight line that leaves all the points in the same half-plane) of slope $\mu$. The point $p$ is identified by performing doubling searches [5] for the value $\mu$ in the list of slopes of the edges of the upper hull sequences. Note that $p$ is the extreme point in the direction orthogonal to the line $\ell$. The algorithm then partitions the points in the upper hull sequences by the vertical line $x = p_x$, by performing doubling searches for the value $p_x$ in the $x$-coordinates of the points in the upper hull sequences. (See Figure 2 for a graphical representation of these steps.) It computes the two tangents of $p$ with all the upper hull sequences: the one to the left, and the one to the right of $p$. For each tangent computed between $p$ and the upper hull sequence $\mathcal{V}$, the algorithm discards the points in $\mathcal{V}$ below this tangent. It computes all the tangents via doubling searches. Let $\mathcal{U}$ denote the upper hull sequence that contains $p$. In $\mathcal{U}$, the algorithm identifies a block $\mathcal{B}$ of consecutive points that forms part of the Upper Hull of the union ($p$ is contained in $\mathcal{B}$), and outputs all the points in $\mathcal{B}$. (In the next paragraph, we describe the details of this step, which is key to the optimality of the algorithm.) It also discards all points that lie underneath the lines that joins $p$ with the leftmost point and the rightmost point in the set of upper hull sequences, via doubling searches. It then recurses on the non-discarded points to the left of $p$ and to the right of $p$. All these steps take advantage of the facts that the points in the upper hull sequences are in sorted order, and that the slopes of the edges of the upper hull sequences are monotonically decreasing from left to right. The doubling searches allow the algorithm to output or to discard blocks of consecutive points of size $s$ in time within $O(\log s)$.

**Algorithm 1** `Quick Union Hull`

```
Input: A set $\mathcal{U}_1, \dots, \mathcal{U}_\rho$ of $\rho$ upper hull sequences
Output: The UPPER HULL of the union of $\mathcal{U}_1, \dots, \mathcal{U}_\rho$
1: Compute the median $\mu$ of the slopes of the middle edges of the upper hull sequences;
2: Find the point $p$ that has a supporting line of slope $\mu$, note $j \in [1..\rho]$ the index of the
   upper hull sequence containing $p$;
3: Partition the upper hull sequences by the line $x = p_x$;
4: Find the two tangents of $p$ with all upper hull sequences: the one to the left and the one
   to the right of $p$, and discard the points below these tangents;
5: Output the block $\mathcal{B}$ in $\mathcal{U}_j$ that contains $p$ and forms part of the union;
6: Discard all points that lie underneath the lines that joins $p$ with the leftmost point and
   the rightmost point of the set of upper hull sequences;
7: Recurse on the non-discarded points to the left and to the right of $p$.
```

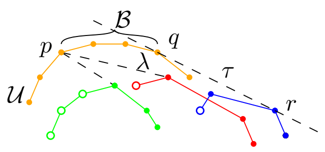

**Figure 3** The state of the `Quick Union Hull` algorithm during an execution of the step 5. The upper hull sequence $\mathcal{U}$ contains the point $p$. $\lambda$ marks the tangent of maximum slope between $p$ and the upper hull sequences to the right of $p$. $\tau$ marks the tangent of maximum slope between $\mathcal{U}$ and the upper hull sequences to the right of $p$. The points $q$ and $r$ lie in $\tau$. $\mathcal{B}$ marks the block of consecutive points in $\mathcal{U}$ that forms part of the UPPER HULL of the union.

We describe in more detail the step of the algorithm that identifies the block $\mathcal{B}$ of consecutive points in the upper hull sequence $\mathcal{U}$ that contains $p$. We describe only how to identify the portion of $\mathcal{B}$ to the right of $p$, as the left counterpart is symmetric. Let $\lambda$ be the tangent of maximum slope between $p$ and the upper hull sequences to the right of $p$ (i.e., the tangent of maximum slope among those computed in the previous step of the algorithm). $\lambda$ is a separating line between the block $\mathcal{B}$ and the other upper hull sequences. Let $\tau$ be the tangent of maximum slope between $\mathcal{U}$ and the upper hull sequences to the right of $p$. In the upper hull sequences, let $q$ and $r$ be the points that lie in $\tau$, such that $q \in \mathcal{U}$. The point $q$ is the rightmost point in $\mathcal{B}$, and so that it determines the end of the right portion of $\mathcal{B}$ (see Figure 3 for a graphical representation of these definitions). Given two upper hull sequences $\mathcal{U}$ and $\mathcal{U}_k$ separated by a vertical line, Barbay and Chen [3] described an algorithm that computes the common tangent between $\mathcal{U}$ and $\mathcal{U}_k$, in time within $O(\log a + \log b)$, where $a$ and $b$ are the positions of the points that lie in the common tangent in the list of points of $\mathcal{U}$ and $\mathcal{U}_k$, respectively. At each step this algorithm considers two points: one from $\mathcal{U}$ and the other from $\mathcal{U}_k$, and in at least one upper hull sequence, it can certify if the tangent is to the right or to the left of the point considered. A minor variant manages the case where the separating line is not vertical: as the first step, in each upper hull sequence, the algorithm computes the supporting line of slope equal to the slope of the separating line, by performing doubling searches. To compute $q$, the algorithm `Quick Union Hull` executes in parallel several instances of the algorithm described by Barbay and Chen [3], between $\mathcal{U}$ and the upper hull sequences to the right of $p$, always considering the same point $u \in \mathcal{U}$, where $\lambda$ is the separating line (similarly to how Demaine et al.'s algorithm [7] uses parallel doubling searches to compute the intersection of sorted sets). Once all parallel decisions about the

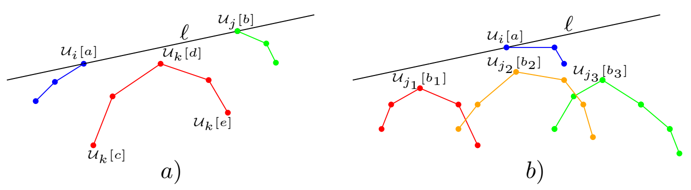

**Figure 4** Example of arguments: a) an Elementary Eliminator Argument formed by 3 blocks and b) an Elementary Convex Argument formed by 4 blocks. In both arguments, note that the blocks of $\mathcal{U}_i$ end or start in $\mathcal{U}_i[a]$ because the algorithm only needs to specify the position $a$ inside $\mathcal{U}_i$.

point $u \in \mathcal{U}$ are reached, the upper hull sequences can be divided into two sets: (i) those whose tangents are to the left of $u$ and (ii) those whose tangents are to the right of $u$. If the set (i) is not empty, then the algorithm stops the parallel computation of tangents in the set (ii). For each upper hull sequence $\mathcal{U}_k$ in the set (ii), the algorithm discards the points to the left of the penultimate point considered in $\mathcal{U}_k$ (i.e., the algorithm can certify that these points lie underneath the tangent between $\mathcal{U}$ and $\mathcal{U}_k$). The step continues until there is just one instance running, and computes the tangent $\tau$ in this instance.

In the upper hull sequences, the algorithm identifies blocks of consecutive points that form part of the output, in time logarithmic in the number of points of the block. At the same time, it also discards blocks of consecutive points that lie underneath the UPPER HULL of the union, in time logarithmic in the number of points of the block. These facts together with the divide-and-conquer approach are the keys to the optimality of the algorithm.

## 3.2 Analysis of the Quick Union Hull Algorithm

Each algorithm for MERGING UPPER HULLS needs to certify that some blocks of consecutive points of the upper hull sequences can not participate in the UPPER HULL of the union, and that some other blocks of the upper hull sequences are indeed in the UPPER HULL of the union. In the following, we formalize the notion of a *certificate*, which can be used to check the correctness of the output in less time than to recompute the output itself. This notion of certificate yields a measure of the difficulty of an instance (short certificates characterize "easy" instances, while "long" certificates suggest "difficult" instances). We define a "language" of basic "arguments" for such certificates: *eliminator* (which discards points from the input) and *convex* (which justifies the presence of points in the output) arguments, and their key positions in the instance. A certificate will be verified by checking each of its arguments: each argument can be checked in time proportional to the number of blocks in it.

Given an upper hull sequence $\mathcal{U}_i$, let $\mathcal{U}_i[a]$ and $\mathcal{U}_i[b..c]$ denote the $a$-th point and the block of consecutive points corresponding to the positions from $b$ to $c$ in $\mathcal{U}_i$, respectively. Given two points $p$ and $q$, let $m(p, q)$ denote the slope of the straight line that passes through $p$ and $q$.

▶ **Definition 2.** Given the points $\mathcal{U}_i[a]$ and $\mathcal{U}_j[b]$, let $\ell$ be the straight line that passes through $\mathcal{U}_i[a]$ and $\mathcal{U}_j[b]$, and let $m_\ell$ be the slope of $\ell$. $\langle \mathcal{U}_i[a], \mathcal{U}_j[b] \supset \mathcal{U}_k[c..d..e] \rangle$ is an *Elementary Eliminator Argument* if the points of the block $\mathcal{U}_k[c..e]$ are between the vertical lines through $\mathcal{U}_i[a]$ and $\mathcal{U}_j[b]$, $m(\mathcal{U}_k[d-1], \mathcal{U}_k[d]) \geq m_\ell \geq m(\mathcal{U}_k[d], \mathcal{U}_k[d+1])$, and the point $\mathcal{U}_k[d]$ lies below $\ell$.

If $\langle \mathcal{U}_i[a], \mathcal{U}_j[b] \supset \mathcal{U}_k[c..d..e] \rangle$ is an elementary eliminator argument, then the points of the block $\mathcal{U}_k[c..e]$ can not contribute to the UPPER HULL of the union (see Figure 4 for an example). Several blocks that are "eliminated" by the same pair of points can be combined into a single argument, a notion captured by the *block eliminator argument*.

▶ **Definition 3.** $\langle \mathcal{U}_i[a], \mathcal{U}_j[b] \supset \mathcal{U}_{k_1}[c_1..d_1..e_1], \ldots, \mathcal{U}_{k_t}[c_t..d_t..e_t]\rangle$ is a *Block Eliminator Argument* if $\langle \mathcal{U}_i[a], \mathcal{U}_j[b] \supset \mathcal{U}_{k_1}[c_1..d_1..e_1]\rangle, \ldots, \langle \mathcal{U}_i[a], \mathcal{U}_j[b] \supset \mathcal{U}_{k_t}[c_t..d_t..e_t]\rangle$ are elementary eliminator arguments.

It is not enough to discard all the points that can not contribute to the output. Certifying still requires additional work: a correct algorithm must justify the optimality of its output. To this end we define elementary convex arguments and block convex arguments.

▶ **Definition 4.** $\langle \mathcal{U}_i[a] \dashv \mathcal{U}_{j_1}[b_1], \ldots, \mathcal{U}_{j_t}[b_t]\rangle$ is an *Elementary Convex Argument* if there is a straight line $\ell$ that passes through $\mathcal{U}_i[a]$ of slope $m_\ell$ such that $m(\mathcal{U}_{j_1}[b_1-1], \mathcal{U}_{j_1}[b_1]) \geq m_\ell \geq m(\mathcal{U}_{j_1}[b_1], \mathcal{U}_{j_1}[b_1+1]), \ldots, m(\mathcal{U}_{j_t}[b_t-1], \mathcal{U}_{j_t}[b_t]) \geq m_\ell \geq m(\mathcal{U}_{j_t}[b_t], \mathcal{U}_{j_t}[b_t+1])$; $m(\mathcal{U}_i[a-1], \mathcal{U}_i[a]) \geq m_\ell \geq m(\mathcal{U}_i[a], \mathcal{U}_i[a+1])$; and the points $\mathcal{U}_{j_1}[b_1], \ldots, \mathcal{U}_{j_t}[b_t]$ lie below $\ell$.

If $\langle \mathcal{U}_i[a] \dashv \mathcal{U}_{j_1}[b_1], \ldots, \mathcal{U}_{j_t}[b_t]\rangle$ is an elementary convex argument, then the point $\mathcal{U}_i[a]$ is in the Upper Hull of the union of the upper hull sequences $\mathcal{U}_i, \mathcal{U}_{j_1}, \ldots, \mathcal{U}_{j_t}$. Some blocks can be "easily" certified as part of the output using similar arguments:

▶ **Definition 5.** Given the points $\mathcal{U}_i[a]$ and $\mathcal{U}_i[b]$, let $\ell$ be the straight line that passes through $\mathcal{U}_i[a]$ and $\mathcal{U}_i[b]$ and let $m_\ell$ be the slope of $\ell$. $\langle \mathcal{U}_i[a..b] \dashv \mathcal{U}_{j_1}[c_1..d_1..e_1], \ldots, \mathcal{U}_{j_t}[c_t..d_t..e_t]\rangle$ is a *Block Convex Argument* if $\langle \mathcal{U}_i[a] \dashv \mathcal{U}_{j_1}[c_1], \ldots, \mathcal{U}_{j_t}[c_t]\rangle$ and $\langle \mathcal{U}_i[b] \dashv \mathcal{U}_{j_1}[e_1], \ldots, \mathcal{U}_{j_t}[e_t]\rangle$ are elementary convex arguments; $m(\mathcal{U}_{j_1}[d_1-1], \mathcal{U}_{j_1}[d_1]) \geq m_\ell \geq m(\mathcal{U}_{j_1}[d_1], \mathcal{U}_{j_1}[d_1+1]), \ldots, m(\mathcal{U}_{j_t}[d_t-1], \mathcal{U}_{j_t}[d_t]) \geq m_\ell \geq m(\mathcal{U}_{j_t}[d_t], \mathcal{U}_{j_t}[d_t+1])$, and the points $\mathcal{U}_{j_1}[d_1], \ldots, \mathcal{U}_{j_t}[d_t]$ lie below $\ell$.

If $\langle \mathcal{U}_i[a..b] \dashv \mathcal{U}_{j_1}[c_1..d_1..e_1], \ldots, \mathcal{U}_{j_t}[c_t..d_t..e_t]\rangle$ is a block convex argument, then the points in the block $\mathcal{U}_i[a..b]$ are in the Upper Hull of the union of the upper hulls $\mathcal{U}_i, \mathcal{U}_{j_1}, \ldots, \mathcal{U}_{j_t}$. Those atomic arguments combine into a general definition of a certificate that any correct algorithm for Merging Upper Hulls in the algebraic decision tree computational model can be modified to output. Those arguments are a generalization to the two-dimensional space of the arguments from Demaine et al. [7] for Merging Multiset, and are inspired by the ones introduced by Barbay and Chen [3] for binary Merging Upper Hulls.

▶ **Definition 6.** Consider $\rho$ upper hull sequences $\mathcal{U}_1, \ldots, \mathcal{U}_\rho$ and their Upper Hull $\mathcal{U}$ of the union expressed as several blocks on the upper hull sequences: $\langle \mathcal{U} = \mathcal{U}_{k_1}[c_1..e_1], \ldots, \mathcal{U}_{k_t}[c_t..e_t]\rangle$ such that $k_i \in [1..\rho]$ for $i \in [1..t]$. A *Certificate* of $\mathcal{U}$ is a set of arguments such that the Upper Hull $\mathcal{U}'$ of the union of any instance formed by $\rho$ upper hull sequences $\mathcal{U}'_1, \ldots, \mathcal{U}'_\rho$ satisfying those arguments is given by the same description of $\mathcal{U}$: $\langle \mathcal{U}' = \mathcal{U}'_{k_1}[c_1..e_1], \ldots, \mathcal{U}'_{k_t}[c_t..e_t]\rangle$. The *Length* of a certificate is the number of arguments in it.

The algorithm `Quick Union Hull` partitions the upper hull sequences into blocks of consecutive points, where each block is either discarded or output. A block is discarded if it is underneath the Upper Hull of the union, or is output if it forms part of the Upper Hull of the union. Each block forms part of an argument of the certificate of the union computed by the algorithm. The key of the analysis is to separate the doubling search steps from the other steps of the algorithm.

▶ **Lemma 7.** *Given an upper hull sequence $\mathcal{U}_i$, the cumulated time complexity of the doubling searches (i.e., steps $2, 3, 4, 5$ and $6$) of the algorithm `Quick Union Hull` considering only points of $\mathcal{U}_i$ is within $O(\sum_{j=1}^{\beta} \log s_j)$; where $s_1, \ldots, s_\beta$ are the sizes of the $\beta$ blocks into which the algorithm partitions $\mathcal{U}_i$.*

**Proof.** Every time the algorithm finds the insertion rank of the $x$-coordinate of each "pivot" point $p$ in $\mathcal{U}_i$, it also finds a position $d$ inside a block whose points will be discarded. The discarded points steps, that search for the tangents between $p$ and $\mathcal{U}_i$, start the searches in the position $d$. The time complexity of finding the two tangents between $p$ and $\mathcal{U}_i$ are bounded by $O(\log s_b)$, where $s_b$ is the size of the discarded block $b$. In case that $p \in \mathcal{U}_i$, the time complexity of the step that computes the block $\mathcal{B}$ of the output is bounded by $O(\log s_{\mathcal{B}})$, where $s_{\mathcal{B}}$ is the size of the block $\mathcal{B}$. These steps partition $\mathcal{U}_i$ in positions separating the blocks to the left of $b$, the block $b$ itself and the blocks to the right of $b$. The doubling search steps can be represented as a tree. Each internal node has two children, which correspond to the two subproblems into which the recursive steps partition $\mathcal{U}_i$. The cost is bounded by $O(\log s_b + \log s)$, where $s$ is the minimum between the sum of the sizes of the blocks to the left of $b$ and to the right of $b$, because for each tangent, the algorithm executes two doubling searches in parallel starting at both ends of $\mathcal{U}_i$. The size of each node is the size of the block discarded or output in the step represented by this node.

We prove that at each step, the total cost is bounded by eight times the sum of the logarithms of the sizes of the nodes in the tree. This is done by induction over the number of steps. If the number of steps is zero then there is no cost. For the inductive step, if the number of steps increases by one, then a new step is done and a leaf subproblem is partitioned into two new subproblems. At this step, a leaf of the tree is transformed into an internal node and two new leaves are created. Let $w$ and $z$ such that $w \leq z$ be the sizes of the new leaves created. Note that $w$ and $z$ are the sum of the sizes of the blocks to the left and to the right of the discarded or output block $b$ in this step, respectively. The cost of this step is less than $4\log w + 4\log s_b$. The cost of all the steps then increases by $4\log w + 4\log s_b$, and hence eight times the sum of the logarithms of the nodes in the tree increases by $8(\log w + \log z + \log s_b - \log(w + z + s_b))$. But if $w \geq 3$, $w \geq s_b$ and $w \leq z$ then the result follows. ◀

We bound next the time complexity of the steps that compute the median of the slopes of the middle edges of the upper hull sequences.

▶ **Lemma 8.** *Given $\rho$ upper hull sequences, the cumulated time complexity of the steps that compute the medians of the slopes of the middle edges (i.e., step 1) of the algorithm* `Quick Union Hull` *is within $O(\sum_{i=1}^{\delta} \log \binom{\rho}{m_i})$; where $\delta$ is the length of the certificate $\mathcal{C}$ computed by the algorithm; and $m_1, \ldots, m_\delta$ is a sequence where $m_i$ is the number of blocks that form the $i$-th argument of $\mathcal{C}$.*

**Proof.** We prove this lemma by induction over $\delta$ and $\rho$. The time complexity of one of these steps is linear in the number of upper hull sequences in the sub-instance (i.e., ignoring all the empty upper hull sequences of this sub-instance). Let $\mathcal{T}(\delta, \rho)$ be the cumulated time complexity of these steps. We prove that $\mathcal{T}(\delta,\rho) \leq \sum_{i=1}^{\delta} m_i \log \frac{\rho}{m_i} - \rho$, where $m_i$ is the number of blocks that form the $i$-th argument of $\mathcal{C}$. Let $p$ be the first "pivot" point computed by the algorithm. Once computed the tangents between $p$ and the upper hull sequences, let $c$ and $d$ be the number of upper hull sequences that have non-discarded point only to the left of $p$ and to the right of $p$, respectively. Let $b$ be the number of upper hull sequences that have non-discarded points to the left and to the right of $p$. Let $\delta_c$ and $\delta_d$ be the number of arguments computed by the algorithm to the left and to the right of $p$, respectively. Then, $\mathcal{T}(\delta, \rho) = \mathcal{T}(\delta_c, c+b) + \mathcal{T}(\delta_d, d+b) + \rho$ because of the two recursive calls and the step 1. By Induction Hypothesis, $\mathcal{T}(\delta_c, c+b) \leq \sum_{i=1}^{\delta_c} m_i \log \frac{c+b}{m_i} - c - b$ and $\mathcal{T}(\delta_d, d+b) \leq \sum_{i=1}^{\delta_d} m_i \log \frac{d+b}{m_i} - d - b$. We need to prove that $c + d \leq \sum_{i=1}^{\delta_c} m_i \log\left(1 + \frac{d}{c+b}\right) + \sum_{i=1}^{\delta_d} m_i \left(1 + \frac{c}{d+b}\right)$, but this is a consequence of $\sum_{i=1}^{\delta_c} m_i \geq c + b, \sum_{i=1}^{\delta_d} m_i \geq d + b$ (the number of blocks is greater than or equal to the number of upper hull sequences); in the worst case, $c \leq d + b, d \leq c + b$ (at least $\frac{\rho}{2}$ sequences are left to the left and to the right of $p$); and $\log\left(1 + \frac{y}{x}\right)^x \geq y$ for $y \leq x$. ◀

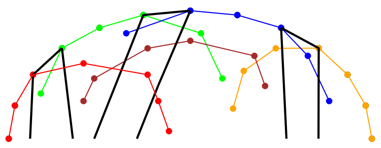

**Figure 5** A partition certificate of size 7 of an instance of the MERGING UPPER HULL problem. The black lines mark the division between the 7 convex regions.

Combining Lemma 7 and Lemma 8 yields the following theorem:

▶ **Theorem 9.** *Given $\rho$ upper hull sequences, there is an algorithm that computes the* UPPER HULL *of the union of these upper hull sequences in time within $O(\sum_{j=1}^{\beta}\log s_j+\sum_{i=1}^{\delta}\log\binom{\rho}{m_i})$; where $\beta$ is the number of blocks in the certificate $\mathcal{C}$ of the union computed by the algorithm; $s_1,\dots,s_\beta$ are the sizes of these blocks; $\delta$ is the length of $\mathcal{C}$; and $m_1,\dots,m_\delta$ is a sequence where $m_i$ is the number of blocks that form the $i$-th argument of $\mathcal{C}$.*

In the next section, in the context of computing the CONVEX HULL, we prove that a synergistic algorithm based on this result outperforms the algorithm originally proposed by Kirkpatrick and Seidel [10], even when taking into account the refined analysis described by Afshani et al. [1], and the algorithm originally proposed by Levcopoulos et al.'s algorithm [11], even when taking into account the refined analysis described in Section 2. We depict a family of instances where the synergistic algorithm is faster than the previous algorithms by a factor logarithmic in the size of the input.

We define the concept of partition certificate of an instance as a partition of the plane into regions such that, in each region, the points of the upper hull sequence are "easy" to certify, whether they form part of the output or not (see Figure 5 for an example of a partition certificate). We describe an analysis of the algorithm `Quick Union Hull` in function of the smallest possible size $\delta$ of such a partition certificate for a particular instance, and show the optimality of the algorithm in the worst case over all instances of signature $\rho, r_1, \dots, r_\rho$, and at least one partition certificate of size $\delta$, within the algebraic decision tree model.

▶ **Definition 10.** Given an instance of the MERGING UPPER HULL problem, a *Partition Certificate* of this instance is a partition of the plane into convex regions, such that, in each region, which points belong to the output can be decided using a constant number of arguments. The *Size* of a partition certificate is its number of convex regions.

The next theorem provides a tight bound on the computational complexity of merging:

▶ **Theorem 11.** *Given $\rho$ upper hull sequences of sizes $r_1,\dots,r_\rho$ such that their union admits a partition certificate of size $\delta$, there is an algorithm that computes the union of the $\rho$ upper hull sequences in time within $O(\delta\sum_{i=1}^{\rho}\log\frac{r_i}{\delta})$. This time complexity is optimal in the worst case over all instances of $\rho$ upper hull sequences of sizes $r_1,\dots,r_\rho$ admitting a partition certificate of size $\delta$.*

**Proof.** The number of arguments of the certificate of the union computed by the algorithm `Quick Union Hull` is a constant factor of the number of blocks in a description of $\mathcal{U}$ of minimal size (see Definition 6 for an example of the description of $\mathcal{U}$), such that each block of the description can be certified using a single block convex argument. Fix any partition certificate $\mathcal{P}$ of minimal size $\delta$. Let $\mathcal{R}$ be a convex region of $\mathcal{P}$. The key argument is to prove that if $\mathcal{R}$ contains a block $\mathcal{B}$ that forms part of the UPPER HULL $\mathcal{U}$ of the union, then the algorithm `Quick Union Hull` can certify, using a constant number of argument, that $\mathcal{B}$

forms part of $\mathcal{U}$. But, this is a consequence of the optimality of the step that computes the block $\mathcal{B}$ of the output (i.e., step 5). This step computes the block of maximal size of the output (i.e., the block that contains the "pivot" point $p$) that can be certified using a single block convex argument. Using the results from Lemma 7 and Lemma 8 with the concavity of the logarithm function (i.e., $\sum_{j=1}^{\beta} \log s_j \leq \beta \log \frac{\sum_{j=1}^{\beta} s_j}{\beta}$), we obtain that the time complexity of the algorithm is within $O(\sum_{i=1}^{\delta} \sum_{j=1}^{\rho} s_{ij} + \delta\rho) \subseteq O(\delta \sum_{i=1}^{\rho} \log \frac{r_i}{\delta})$, where $s_{ij}$ is the size of the $i$-th block of the $j$-th upper hull sequence. We defer the proof of the corresponding tight lower bound to the Appendix B. ◀

In the following section, we combine the results from Sections 2 and 3 into a synergistic algorithm that computes the Convex Hull of a set of planar points.

## 4 Synergistic Computation of Convex Hulls

The synergistic algorithm for computing the Convex Hull decomposes first the input sequence of points into simple subchains, computes their convex hulls, and then merges their convex hulls. There are two noteworthy *advantages* to this approach: (1) the synergistic algorithm decomposes the points into "easy" instances (these "easy" instances are determined by the order in which the points are given), and computes the convex hull sequences of these instances, both steps in time linear in the number of points; and (2) the merging convex hulls algorithm takes advantage of the number of convex hull sequences, of the fact that the points in the convex hull sequences are given in sorted order, and of the relative position of those.

The time complexity of the partitioning and merging steps of the Levcopoulos et al.'s algorithm [11] (seen in Section 2) are within $O(n(1+\mathcal{H}(n_1,\dots,n_\kappa))) \subseteq O(n(1+\log\kappa))$. We describe a partitioning algorithm running in time linear in the size of the input, which is key to the synergistic result of Theorem 13. From left to right, the `Doubling Search Partition` algorithm searches for the largest integer $t$ such that the subchain of size $2^t$ is simple. It identifies this subchain as simple and restarts the computation in the rest of the sequence. The following theorem describes the properties of the partition obtained by this algorithm.

▶ **Theorem 12.** *Given a sequence $S$ of $n$ planar points. The algorithm* `Doubling Search Partition` *computes in linear time a partition of $S$ into $k$ simple subchains of consecutive points of sizes $r_1,\dots,r_k$, such that $n(1+\mathcal{H}(r_1,\dots,r_k)) \in O(n(1+\alpha))$, where $\alpha$ is the minimal entropy $\mathcal{H}(r_1,\dots,r_\kappa)$ of any partition of $S$ into $\kappa$ simple subchains of consecutive positions, of respective sizes $r_1,\dots,r_\kappa$, and $\mathcal{H}(r_1,\dots,r_\kappa) = \sum_{i=1}^{\kappa} \frac{r_i}{n} \log \frac{n}{r_i}$.*

**Proof.** Similar to the proof of Theorem 1, the number of operations for each simple subchain of size $r_i$ of any partition into simple subchains can be bounded separately to $O(r_i \log \frac{n}{r_i} + r_i)$. ◀

The algorithm `(Simple,Structure) Synergistic Hull` proceeds in two phases. It first decomposes the input into simple subchains of consecutive positions using the algorithm `Doubling Search Partition`, that searches for simple chains from left to right. It computes their upper hull sequences, and then merges those using the algorithm `Quick Union Hull`.

▶ **Theorem 13.** *Consider a sequence $\mathcal{S}$ of $n$ planar points that can be partitioned into $\kappa$ simple subchains of consecutive points of sizes $r_1,\dots,r_\kappa$ (such that $\sum_{i=1}^{\kappa} r_i = n$); and also can be partitioned into $h$ sets of sizes $n_1,\dots,n_h$ (such that $\sum_{i=1}^{h} n_i = n$), where each set can be enclosed by a triangle completely below the* Upper Hull *of $\mathcal{S}$. There is an algorithm that computes the* Upper Hull *of $S$ in time within $O(n + \sum_{i=1}^{\delta} \log \binom{\kappa}{m_i}) \subseteq O(n(1+\min(\mathcal{H}(r_1,\dots,r_\kappa),\mathcal{H}(n_1,\dots,n_h)))) \subseteq O(n(1+\min(\log\kappa,\log h))) \subseteq O(n\log n)$,*

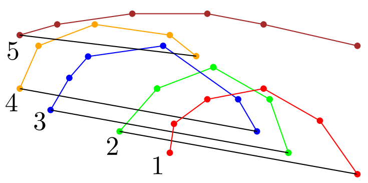

**Figure 6** A sequence of points and its decomposition into $\kappa = 5$ simple subchains. The numbers indicate the order in which the sequence of points are given: each from left to right internally, and mark the simple subchains. The line segment that joins the leftmost and rightmost points of the simple subchain 5 is an elementary eliminator argument for all other simple subchains. The simple subchains $1, 2, 3$ and 4 are more intricately woven together.

*where $\delta$ is the length of the certificate $\mathcal{C}$ of the union computed by the merging algorithm once the simple subchains are identified ($\delta \leq h$); and $m_1, \ldots, m_\delta$ is a sequence where $m_i$ is the number of blocks of the upper hull sequences of the simple subchains that form the $i$-th argument of $\mathcal{C}$ ($m_i \leq \kappa$ for $i \in [1..\delta]$).*

**Proof.** This result is a consequence of Theorem 9 and Theorem 12. ◀

The `(Smooth,Structure) Synergistic Hull` algorithm outperforms both the algorithm described by Levcopoulos et al. [11] and the one described by Kirkpatrick and Seidel [10], as well as any dovetailing combination of them. Consider for example the instance depicted in Figure 6: in this instance, the time complexity of the algorithm described by Kirkpatrick and Seidel [10], as refined by Afshani et al. [1], is within $O(h \log n)$ (all the points in the sequences $1, 2, 3$ and 4 can be enclosed by a triangle completely below the Upper Hull, hence $n_1 = \cdots = n_{h-1} = 1$ and $n_h = n - h + 1$ in the formula $O(n(1 + \mathcal{H}(n_1, \ldots, n_h)))$, where $h - 1$ is the number of points in the sequence 5). The time complexity of the algorithm described by Levcopoulos et al. [11], as refined in Section 2, is within $O(\kappa \log n)$ (there are $\kappa$ simple subchains, suppose that the sizes of the subchains labeled 1 to $\kappa - 1$ are a constant $c$ and that the size of the subchain labeled $\kappa$ is $n - (\kappa - 1)c$, then $r_1 = \cdots = r_{\kappa-1} = c$ and $r_\kappa = n - (\kappa - 1)c$ in the formula $O(n(1 + \mathcal{H}(r_1, \ldots, r_\kappa))))$. On the other hand, the time complexity of the algorithm `(Smooth,Structure) Synergistic Hull` is within $O(n)$: once it computes the first supporting line and the first point in the output, it discards all the points except the points in the upper hull sequence labeled 5. If $h \in \Theta(n)$ and $\kappa \in \Theta(n)$, then the `(Smooth,Structure) Synergistic Hull` algorithm is faster than the previous algorithms by a factor logarithmic in the size of the input. This concludes the description of our synergistic results. In the next section, we discuss the issues left open for improvement.

## 5 Discussion

The results based on the partition of a sequence of points into simple subchains from Levcopoulos et al. [11] can be generalized to take advantage of the input order when computing Maxima Sets. Similarly, the results about Merging Upper Hulls can be generalized to Merging Maxima Set in the plane. The combination of those results yields a synergistic algorithm for computing Maxima Set adaptive to the input order and structure, at the same time. The analysis of the synergistic algorithm for computing Maxima Set is similar to the analysis of the synergistic algorithm for computing Convex Hulls.

Considering the computation of the Maxima Set and of the Convex Hull, we have built upon previous results taking advantage either of the input order or of the input structure, to describe solutions which take advantage of both in a synergistic way. There are various other ways in which those results can be further improved: we list only a selection here. In the same line of thought, Ahn and Okamoto [2] described a notion of input order other than the one we considered here, which can potentially yield another synergistic solution in combination with the notion of input structure defined by Afshani et al. [1]. This is true for any of the many notions of input order which could be adapted from Sorting [12]. Whereas being adaptive to as many measures of difficulty as possible at once is a worthy goal in theory, it usually comes at a price of an increase in the constant factor of the running time of the algorithm: it will become important to measure, for the various practical applications of each problem, which measures of difficulty take low value in practice. It is necessary to do more theoretical work to identify what to look for in the practical applications, and equally important to measure the practical difficulties of the instances.

## References

1 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *Journal of the ACM (JACM)*, 64(1):3:1–3:38, 2017.

2 Hee-Kap Ahn and Yoshio Okamoto. Adaptive algorithms for planar convex hull problems. *IEICE Transactions*, 94-D(2):182–189, 2011.

3 Jérémy Barbay and Eric Y. Chen. Convex hull of the union of convex objects in the plane: an adaptive analysis. In *Proceedings of the Annual Canadian Conference on Computational Geometry (CCCG)*, 2008.

4 Jérémy Barbay, Carlos Ochoa, and Srinivasa Rao Satti. Synergistic solutions on multisets. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2017.

5 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters (IPL)*, 5(3):82–87, 1976.

6 Timothy M. Chan, Jack Snoeyink, and Chee-Keng Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional voronoi diagrams. *Discrete & Computational Geometry (DCG)*, 18(4):433–454, 1997.

7 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the* $11^{th}$ *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

8 Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys (ACMCS)*, 24(4):441–476, 1992.

9 David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of the Annual Symposium on Computational Geometry (SoCG)*, pages 89–96, New York, NY, USA, 1985. ACM.

10 David G Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing (SICOMP)*, 15(1):287–299, 1986.

11 Christos Levcopoulos, Andrzej Lingas, and Joseph S. B. Mitchell. Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In *Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 80–89, London, UK, 2002.

12 Alistair Moffat and Ola Petersson. An overview of adaptive sorting. *Australian Computer Journal (ACJ)*, 24(2):70–77, 1992.

13 J. Ian Munro and Philip M. Spira. Sorting and searching in multisets. *SIAM Journal on Computing (SICOMP)*, 5(1):1–8, 1976.

14 M. Sion. On general minimax theorems. *Pacific Journal of Mathematics (PJM)*, 1:171–176, 1958.

15 Tadao Takaoka and Yuji Nakagawa. Entropy as computational complexity. *Journal of Information Processing (JIP)*, 18:227–241, 2010.

## Appendix

## A Proof of the Tight Lower Bound of Theorem 1

**Proof.** We prove the optimality of this complexity in the worst-case over instances of $n$ points that can be partitioned into $\kappa$ simple subchains of sizes $r_1, \dots, r_\kappa$ by giving a tight lower bound. Takaoka and Nakagawa [15] showed a lower bound of $\Omega(n(1+\mathcal{H}(r_1, \dots, r_\kappa)))$ in the comparison model for Sorting a sequence of $n$ numbers, in the worst case over instances covered by $\kappa$ runs (increasing or decreasing) of sizes $r_1, \dots, r_\kappa$ (such that $\sum_{i=1}^{\kappa} r_i = n$), respectively. The Sorting problem can be reduced in linear time to the problem of computing the Convex Hulls of a chain of $n$ planar points that can be partitioned into $\kappa$ simple subchains of sizes $r_1, \dots, r_\kappa$, respectively. For each real number $r$, this is done by producing a point with $(x, y)$-coordinates $(r, r^2)$. The $\kappa$ runs (alternating increasing and decreasing) are transformed into $\kappa$ simple subchains of the same sizes. The sorted sequence of the numbers can be obtained from the Convex Hull of the points in linear time. ◀

## B Proof of the Tight Lower Bound of Theorem 11

We describe the intuition for the lower bound below: it is a simple adversary argument, based on the definition of a family of "hard" instances for each possible value taken by the parameters of the analysis, building over each other.

First, we verify the lower bound for "easy" instances, of finite difficulty: general instances of $\rho$ upper hull sequences of sizes $r_1, \dots, r_\rho$ that admit a partition certificate of size $\delta = 1$ (i.e., the upper hull of the union is one of the set). Such an instance requires $\Omega(\sum_{i=1}^{\rho} \log r_i)$ operation (no correct algorithm can afford to ignore a single upper hull sequence, which could be the upper hull of the union, and for each discarded upper hull sequence it needs to certify that all points lie underneath the union). This lower bound yields a distribution of instances of $\rho$ upper hull sequences of sizes $r_1, \dots, r_\rho$, such that the time complexity of any deterministic algorithm is within $\Omega(\sum_{i=1}^{\rho} \log r_i)$, on average on a uniform distribution of those instances.

Such distributions of "elementary" instances can be duplicated so that to produce various distributions of elementary instances; and combined so that to define a distribution of harder instances:

▶ **Lemma 14.** *Given the positive integers $\rho, r_1, \dots, r_\rho, \delta$, there is a family of instances of $\rho$ upper hull sequences of sizes $r_1, \dots, r_\rho$ of smallest partition certificates of size $\delta$, such that on average on a uniform distribution of these instances, the time complexity of any algorithm computing the* Upper Hull *of the union of the $\rho$ upper hull sequences is within $\Omega(\delta \sum_{i=1}^{\rho} \log \frac{r_i}{\delta})$, in the algebraic decision tree model.*

Finally, any such distribution with a computational lower bound on average yields a computational lower bound for the worst case instance complexity of any randomized algorithm, on average on its randomness; and as a particular case a lower bound on the worst case complexity of any deterministic algorithm:

▶ **Corollary 15.** *Given the positive integers $\rho, r_1, \dots, r_\rho, \delta$, and an algorithm $A$ computing the union of $\rho$ upper hull sequences in the algebraic decision tree model (whether deterministic or randomized), there is an instance $I$ such that the time complexity of $A$ is within $\Omega(\delta \sum_{i=1}^{\rho} \log \frac{r_i}{\delta})$ when it computes the union of the $\rho$ upper hull sequences in $I$.*

**Proof.** A direct application of Yao's minimax principle [14]. ◀