Refining the Analysis of Divide and Conquer: How and When

Jérémie Barbay\textsuperscript{1}, Carlos Ochoa\textsuperscript{1}* , and Pablo Pérez-Lantero\textsuperscript{2}

\textsuperscript{1} Departamento de Ciencias de la Computación, Universidad de Chile, Chile  
Jeremy@barbay.cl, cochoa@dcc.uchile.cl.

\textsuperscript{2} Escuela de Ingeniería Civil Informática, Universidad de Valparaíso, Chile.  
pablo.perez@uv.cl.

Abstract. Divide-and-Conquer is a central paradigm for the design of algorithms, through which fundamental computational problems like sorting arrays and computing convex hulls are solved in optimal time within $O(n \log n)$ in the worst case over instances of size $n$. A finer analysis of those problems yields complexities within $O(n(1+H(n_1,\ldots,n_k))) \subseteq O(n(1+ \log k)) \subseteq O(n \log n)$ in the worst case over all instances of size $n$ composed of $k$ “easy” fragments of respective sizes $n_1,\ldots,n_k$ summing to $n$, where the entropy function $H(n_1,\ldots,n_k) = \sum_{i=1}^k \frac{n_i}{n} \log \frac{n}{n_i}$ measures the difficulty of the instance. We consider whether such refined analysis can be applied to other solutions based on Divide-and-Conquer. We describe two optimal examples of such refinements, one for the computation of planar convex hulls adaptive to the decomposition of the input into simple chains, and one for the computation of Delaunay triangulations (and, as a corollary, of Voronoi diagrams and planar convex hulls) adaptive to the decomposition of the input into monotone histograms; and four examples where such refinement yields partial results, namely for the computation of Delaunay triangulations according to various measures of difficulty.

Keywords: Divide and Conquer, Adaptive Analysis, Convex Hull, Delaunay Triangulation, Voronoi Diagrams.

1 Introduction

The Divide-and-Conquer paradigm is used to solve central computational problems such as Sorting arrays, computing Convex Hulls, Delaunay Triangulations and Voronoi Diagrams, and yields an optimal running time within $O(n \log n)$ for instances of $n$ elements. An adaptive analysis of slight variants of these algorithms has yield improved running times on large classes of instances for several of these problems, listed below, but not for all.

* Corresponding author
Concerning the problem of SORTING arrays, Munro and Spira (1976) showed that the algorithm MergeSort can be adapted to sort a multiset \( S \) of \( n \) elements in time within \( O(n(1 + \mathcal{H}(m_1, \ldots, m_\sigma))) \leq O(n(1 + \log \sigma)) \leq O(n \log n) \), where \( \sigma \) is the number of distinct elements in \( S \) and where \( m_1, \ldots, m_\sigma \) (such that \( \sum_{i=1}^{\sigma} m_i = n \)) are the multiplicities of the distinct elements in \( S \).

Taking advantage of the input order, Knuth (1973) considered sequences formed by runs (contiguous ascending subsequences) and described an algorithm sorting such sequences in time within \( O(n(1 + \log \rho)) \leq O(n \log n) \), where \( \rho \) is the number of runs in the sequence. Barbay and Navarro (2013) improved the analysis of this algorithm in time within \( O(n(1 + \mathcal{H}(r_1, \ldots, r_\rho))) \leq O(n(1 + \log \rho)) \leq O(n \log n) \), where \( r_1, \ldots, r_\rho \) (such that \( \sum_{i=1}^{\rho} r_i = n \)) are the sizes of these runs.

Concerning the computation of CONVEX HULLS in the plane, Kirkpatrick and Seidel (1986) described a Divide-and-Conquer algorithm to compute the convex hull of a set of \( n \) planar points in time within \( O(n(1 + \log h)) \leq O(n \log n) \), where \( h \) is the number of vertices in the convex hull, a complexity which was later improved by Afshani et al. (2009) within \( O(n(1 + \mathcal{H}(n_1, \ldots, n_h))) \leq O(n(1 + \log h)) \leq O(n \log n) \), where \( n_1, \ldots, n_h \) (such that \( \sum_{i=1}^{h} n_i = n \)) are the sizes of a partition of the input, such that every element of the partition is a singleton or can be enclosed by a triangle whose interior is completely below the upper hull of the set, with the minimum possible value for \( \mathcal{H}(n_1, \ldots, n_h) \). The function \( \mathcal{H}(n_1, \ldots, n_h) \) is the minimum entropy of the distribution of the points into a certificate of the instance.

Taking advantage of the fact that the convex hull of a simple chain can be computed in linear time, and that simplicity can also be tested in linear time, Levcopoulos et al. (2002) described a Divide-and-Conquer algorithm for computing the convex hull of a polygonal chain. They measured the complexity of this algorithm in terms of the minimum number of simple subchains \( \kappa \) into which the chain can be cut. They showed that its time complexity is within \( O(n(1 + \log \kappa)) \leq O(n \log n) \).

**Hypothesis.** Can refinements, similar to those described above, be applied to the algorithm from Levcopoulos et al. (2002)? Are there other problems on which such refinement can be applied?

**Our Results.** In Section 2, we classify a selection of refined analysis between those which are Structure Based and those which are Input-Order Based, showing in particular that Levcopoulos et al. (2002)’s solutions for computing the Convex Hull or the Constrained Delaunay Triangulation of a polygonal chain is in fact input-order based. In Section 3 we describe two distinct refined analysis for problems in computational geometry, which yield various optimal adaptive results on the computation of Convex Hulls, Delaunay Triangulations and Voronoi Diagrams. In Section 3.1 we refine the analysis of Levcopoulos et al. (2002)’s algorithm for computing the Convex Hull of polygonal chains in time within \( O(n(1 + \mathcal{H}(n_1, \ldots, n_\kappa))) \leq O(n(1 + \log \kappa)) \leq O(n \log n) \), where \( n_1, \ldots, n_\kappa \) are the lengths of the subchains of a partition of a polygonal chain of \( n \) points into the minimum number \( \kappa \) of simple subchains. In Section 3.2 we describe a refined analysis of the computation of Voronoi Diagrams and
Delaunay Triangulations for sequences $S$ formed by $n$ points, which yields a time complexity within $O(n(1 + H(v_1, \ldots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$, where $v_1, \ldots, v_\mu$ are the sizes of the minimum number $\mu$ of monotone histograms in which $S$ can be cut, with respect to two fixed orthogonal directions. In Section 4, we describe some more difficult applications of such refined analysis, which yield potentially non-optimal results for the computation of Delaunay Triangulations and Voronoi Diagrams in the plane. In Section 5, we discuss various open problems that arise as a consequence of these limitations.

2 Background

We review here some results on the refined analysis of algorithms for Sorting arrays (Section 2.1), for computing planar Convex Hulls (Section 2.2), and for computing Delaunay Triangulations and Voronoi Diagrams (Section 2.3). We classify the various refined analysis between those focusing on the structure of the instance versus those focusing on the order in which the input is given.

2.1 Sorting

We classify the adaptive results for Sorting arrays in two categories: those taking advantage of the frequencies of the values (i.e. the structure of the instance) and those taking advantage of the order of the values in the input (i.e. the input-order):

MergeSort is a divide-and-conquer sorting algorithm in the comparison model. This algorithm relies on a linear time merge process, that combines two ordered sequences into a single ordered sequence.

Structure Based Results. Munro and Spira (1976) considered the task of sorting a multiset $S = \{x_1, \ldots, x_n\}$ of $n$ real numbers with $\sigma$ distinct values, of multiplicities $m_1, \ldots, m_\sigma$ such as $\sum_{i=1}^\sigma m_i = n$. They showed that adding counters to various classical algorithms (among which the algorithm MergeSort) yields adaptive variants sorting $S$ in time within $O(n(1 + H(m_1, \ldots, m_\sigma))) \subseteq O(n(1 + \log \sigma)) \subseteq O(n \log n)$, where $H(m_1, \ldots, m_\sigma) = \sum_{i=1}^\sigma \frac{m_i}{n} \log \frac{n}{m_i}$ measures the entropy of the distribution of the multiplicities $\langle m_1, \ldots, m_\sigma \rangle$.

Input-Order Based Results. Knuth (1973) described an adaptive sorting algorithm that takes advantage of permutations formed by sorted blocks called runs, that is, subsequences of consecutive positions in the input with a positive gap between successive values, from beginning to end. He showed that the time complexity of this algorithm is within $O(n(1 + \log \rho)) \subseteq O(n \log n)$, where $\rho$ is the number of runs in the permutation (e.g. $(1,2,6,7,8,9,3,4,5)$ is composed of 2 such sorted blocks $(1,2,6,7,8,9)$ and $(3,4,5)$). Barbay and Navarro (2013) showed that, if the permutation $\pi$ is formed by $\rho$ runs of sizes given by the
vector \( \langle r_1, \ldots, r_\rho \rangle \), then \( \pi \) can be sorted in time within \( O(n(1+\mathcal{H}(r_1, \ldots, r_\rho))) \subseteq O(n(1+\log \rho)) \subseteq O(n \log n) \). The main idea is to detect the runs first and then merge them pairwise, using a mergesort-like step. The detection of ascending runs can be done in linear time by a scanning process identifying the positions \( i \) in \( \pi \) such that \( \pi(i) > \pi(i+1) \). Merging the two shortest runs at each step further reduces the number of comparisons, making the running time of the merging process adaptive to the entropy of the sequence of the lengths of the runs. The merging process is then represented by a tree with the shape of a Huffman tree, built from the distribution of \( R \). They extend this result to mix ascending and descending runs.

2.2 Convex Hull

Given a set \( P \) of \( n \) points, the Convex Hull of \( P \) is the smallest convex set containing \( P \) (see Figure 1). We classify the adaptive results for computing Convex Hulls in dimensions two and three in two categories: those taking advantage of the positions of the points (i.e. the structure of the instance), and those taking advantage of the order in which those points are given (i.e. the input-order).

![Fig. 1. A point set \( P \). a) The convex hull of \( P \), b) the Delaunay triangulation of \( P \), and c) the Delaunay triangulation and the Voronoi diagram of \( P \).](image)

Structure Based Results. [Kirkpatrick and Seidel 1986] described an algorithm to compute the convex hull of a set of \( n \) planar points in time within \( O(n(1 + \log h)) \subseteq O(n \log n) \), where \( h \) is the number of vertices in the convex hull. The algorithm relies on a variation of the Divide-and-Conquer paradigm, which they call the “Marriage-Before-Conquest” principle. For computing the upper hull, the algorithm finds a vertical line that divide the input point set into
two approximately equal-size parts in linear time. Next, it determines the edge of the upper hull that intersects this line in linear time. It then eliminates the points that lie underneath this edge and finally applies the same procedure to the two sets of the remaining points on the left and right side of the vertical line. A similar algorithm computes the lower hull.

Afshani et al. (2009) refined the complexity analysis of this algorithm to within \( O(n(1 + H(n_1, \ldots, n_h))) \subseteq O(n(1 + \log h)) \subseteq O(n \log n) \), where \( n_1, \ldots, n_h \) are the sizes of a partition of the input, such as every element of the partition is a singleton or can be enclosed by a triangle whose interior is completely below the upper hull of the set, and \( H(n_1, \ldots, n_h) \) has the minimum possible value (minimum entropy of the distribution of the points into a certificate of the instance).

**Input-Order Based Results.** A polygonal chain is a curve specified by a sequence of points \( p_1, p_2, \ldots, p_n \). The curve itself consists of the line segments connecting the pairs of consecutive points. A polygonal chain \( C \) is simple if any two edges of \( C \) that are not adjacent are disjoint, or if the intersection point is a vertex of \( C \); and any two adjacent edges share only their common vertex. Melkman (1987) described an algorithm that computes the convex hull of a simple polygonal chain in linear time, and Chazelle (1991) described an algorithm for testing the simplicity of polygonal chains in linear time.

Levcopoulos et al. (2002) combined these results to yield an adaptive Divide-and-Conquer algorithm for computing the convex hull of polygonal chains. They measured the complexity of their algorithm in terms of the minimum number of simple subchains \( \kappa \) into which the chain \( C \) can be cut. The algorithm tests if the chain \( C \) is simple, using Chazelle (1991)'s algorithm: if the chain \( C \) is simple, the algorithm computes the convex hull of \( C \) in linear time, using Melkman (1987)'s algorithm. If the chain \( C \) is not simple, the algorithm cuts \( C \) at the middle point into the subsequences \( C' \) and \( C'' \), and recurses on them to merge the resulting convex hulls using Preparata and Shamos (1985)'s algorithm. They showed that the time complexity of this algorithm is within \( O(n(1 + \log \kappa)) \subseteq O(n \log n) \).

We show how to refine the analysis of this algorithm in Section 3.1.

### 2.3 Delaunay Triangulations and Voronoi Diagrams

Given a set \( P \) of \( n \) planar points, a triangulation of \( P \) is a subdivision of the convex hull of \( P \) into triangles with vertex set the set \( P \). A Delaunay Triangulation \( DT(P) \) of \( P \) is a triangulation where for every edge \( e \) there exists a disk \( C \) with the following properties: (i) the endpoints of edge \( e \) are on the boundary of \( C \), and (ii) no other point of \( P \) is in the interior of \( C \): it is named after Boris Delaunay for his work on this topic from 1934. An equivalent definition is such that no point in \( P \) is inside the circumcircle of any triangle of \( DT(P) \). Computing the DELAUNAY TRIANGULATION is equivalent to computing its dual, called the VORONOI DIAGRAM: each one can be constructed from the other in linear time (Preparata and Shamos (1985)) (see Figure 2). Computing the DELAUNAY
Triangulation of a set of points in two dimensions reduces to computing the Convex Hull in three dimensions of the projections of those points on an hyperbolic plane. The projection of \( P \) onto the unit elliptic paraboloid \( z = x^2 + y^2 \) yields a point set \( P' \). The Convex Hull \( CH(P') \) of \( P' \) contains every point of the set. The downward-facing facet of \( CH(P') \) are those whose normal vectors have a negative \( z \)-value. Projecting the edges of downward-facing facet in \( CH(P') \) onto the plane yields the Delaunay triangulation of \( P \).

By showing tight bounds for input-order oblivious (i.e. structure-based) algorithms for computing Convex Hulls in three dimensions, Afshani et al. (2009) indirectly proved that no planar Delaunay Triangulation algorithm can take advantage of the position of the points.

**Theorem 1 (Afshani et al. (2009)).** Consider a set of \( n \) points in the plane. For any algorithm \( A \) computing the Delaunay triangulation in the algebraic decision tree model, \( A \) performs in time within \( \Omega(n \log n) \) on average on a random order of the points. This implies that there is an order of those points for which \( A \) performs in time within \( \Omega(n \log n) \).

We describe in Sections 3 and 4 some algorithms taking advantage of the order of the input to compute Delaunay Triangulations and Voronoi Diagrams, among other desirable objects in computational geometry.

### 3 Refined Analysis: Two Examples

We show in this section how to refine the analysis of the algorithm from Levcopoulos et al. (2002) for the decomposition of a polygonal chain into simple sequences and how to extend the analysis to the computation of Voronoi diagrams and Delaunay triangulations for another measure of difficulty based on monotone histograms.

#### 3.1 Computing Convex Hulls: Adaptivity to Simple Subchains

The idea behind the algorithm from Levcopoulos et al. (2002) is to partition the input chain into simple subchains. If it was possible to partition the input chain into the minimum number \( \kappa \) of simple subchains in linear time, then the same approach described by Barbay and Navarro (2013) could be applied to obtain a refined analysis in function of \( O(n(1 + H(n_1, \ldots, n_\kappa))) \) \( \subseteq O(n(1 + \log \kappa)) \) \( \subseteq O(n \log n) \). But, as far as we know, there does not exist any linear time algorithm to accomplish this task. We refine the analysis of the algorithm from Levcopoulos et al. (2002) to obtain this result.

In the recursion tree of the execution of the algorithm described by Levcopoulos et al. (2002) on input \( C \) of \( n \) points, every node represents a subchain of \( C \). The cost of every node is linear in the size of the subchain that it represents. The simplicity test and the merge process are both linear in the number of points in the subchain. When this subchain is simple the node that represents this subchain is a leaf. Every time the algorithm discovers that the polygonal
chain is simple, it executes a number of operations linear in the size of the chain and the corresponding node in the recursion tree becomes a leaf.

**Width Analysis: A Warm-up.** Consider a polygonal chain $C$ of $n = 2^m$ planar points such that $C$ can be partitioned into the minimum number $\kappa = m$ of simple subchains of lengths $2^1, 2^1, 2^2, 2^3, \ldots, 2^{m-1}$. Every time the algorithm cuts in half the chain $C$ into $C'$ and $C''$, the right subchain $C''$ is simple and the recursive call is made only in the left subchain $C'$. Hence, the recursion tree of the algorithm from [Levcopoulos et al. (2002)] on input $C$ has only two nodes per level, one of which is a leaf. (see Figure 2). The running time of the algorithm

![Fig. 2. The recursion tree of A on C. Each node represents a recursive call. Noted in each node is the asymptotic complexities of the simplicity test and the merging process on the subchain that it represent.](image)

on $C$ is

$$2 \sum_{i=1}^{\lfloor \log n \rfloor} \frac{cn}{2^i} \leq 2cn \sum_{i=1}^{\lfloor \log n \rfloor} \frac{1}{2^i} \leq O(n),$$

for some constant $c$ independent of $n$.

**Definition 1 (Width).** The width $\omega$ of the recursion tree in the execution of the algorithm from [Levcopoulos et al. (2002)] on input $C$ is the maximum number of nodes at any level.

[Levcopoulos et al. (2002)] analyzed the complexity of their algorithm in the worst case over instances of fixed size $n$ and $\kappa$. The followig lemma gives an alternate analysis in the worst case over instances of fixed size $n$ and width $\omega$.

**Lemma 1.** Let $\omega$ be the width of the recursion tree in the execution of the algorithm from [Levcopoulos et al. (2002)] on input $C$ of $n$ planar points. The complexity of this algorithm on input $C$ is within $\omega \sum_{i=1}^{\lfloor \log n \rfloor} \frac{cn}{2^i} \leq \omega cn \sum_{i=1}^{\lfloor \log n \rfloor} \frac{1}{2^i} \subseteq O(\omega n)$, for some constant $c$ independent of $n$. 
Refined Analysis. Let $\langle \ell_1, \ldots, \ell_m \rangle$ be the vector formed by the sizes of the subchains represented by the $m$ leaves of the recursion tree of the algorithm from Levcopoulos et al. (2002) on input $C$ (such that $\sum_{i=1}^{m} \ell_i = n$). The number of operations “saved” by the algorithm every time it discovers a leaf of size $\ell_i$ is within $\Omega(\ell_i \log \ell_i)$ (the cost of the subtree of the perfect binary tree rooted in a node of size $\ell_i$) minus $O(\ell_i)$ (the operations in the leaf are not saved). The time complexity $T(C)$ of this algorithm on input $C$ is within $O(n \log n)$ (the cost of the perfect binary tree) minus $\Omega(\sum_{i=1}^{m} \ell_i \log \ell_i - \ell_i)$ (the number of operations saved by the algorithm). So, $T(C) \subseteq O(n \log n - \sum_{i=1}^{m} (\ell_i \log \ell_i - \ell_i)) = O(n (1 + H(\ell_1, \ldots, \ell_m))) \subseteq O(\omega n) \cap O(n \log m) \subseteq O(n \log n)$.

The following lemma summarizes this finer analysis of $A$.

Lemma 2. Let $\langle \ell_1, \ldots, \ell_m \rangle$ be the vector formed by the sizes of the subchains represented by the $m$ leaves of the recursion tree in the execution of the algorithm $A$ on input $C$ of $n = \sum_{i=1}^{m} \ell_i$ planar points. Let $\omega$ be the maximum width of the recursion tree. The time complexity of $A$ on $C$ is within $O(n (1 + H(\ell_1, \ldots, \ell_m))) \subseteq O(\omega n) \cap O(n \log m) \subseteq O(n \log n)$.

Is there a relationship between the vector $\langle \ell_1, \ldots, \ell_m \rangle$ formed by the sizes of the subchains represented by the leaves of the recursion tree in the execution of the algorithm from Levcopoulos et al. (2002) on input $C$ and the vector $\langle \kappa_1, \ldots, \kappa_\kappa \rangle$ formed by the sizes of a partition of $C$ into $\kappa$ simple subchains? The following theorem describes an optimal refinement of the algorithm from Levcopoulos et al. (2002) adaptive to the decomposition of the input into simple subchains.

Theorem 2. Let $\langle \kappa_1, \ldots, \kappa_\kappa \rangle$ be the vector formed by the sizes of the subchains of a partition $\Pi$ of the chain $C$ into the minimum number $\kappa$ of simple subchains. The time complexity of the algorithm from Levcopoulos et al. (2002) on input $C$ is within $O(n (1 + H(\kappa_1, \ldots, \kappa_\kappa))) \subseteq O(n (1 + \log \kappa)) \subseteq O(n \log n)$, which is worst-case optimal in the comparison model.

Proof. Fix the subchain $c_i$ of size $n_i$ in $\Pi$. The algorithm in the worst case considers the $n_i$ 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 $\lceil \log \frac{n}{n_i} \rceil + 1$. In the next level, one of the leaves $\ell$ of the recursion tree fits completely inside $c_i$ and at least $\frac{n}{4}$ points from $c_i$ are dismissed for the following levels. The remaining points of $c_i$, considered by the algorithm, are in the left or the right ends of subchains represented by nodes in the same level of $\ell$ in the recursion tree. In all of the following levels, the number of operations of the algorithm involving points from $c_i$ is divided in half. As a result, the number of operations of the algorithm involving points from $c_i$ is within $O(n_i \log \frac{n}{n_i} + n_i)$. In total, the time complexity of the algorithm is within $O(n + \sum_{i=1}^{\kappa} n_i \log \frac{n}{n_i}) = O(n (1 + H(\kappa_1, \ldots, \kappa_\kappa))) \subseteq O(n (1 + \log \kappa)) \subseteq O(n \log n)$.

We prove the optimality of this complexity by giving a tight lower bound. Barbay and Navarro (2013) showed a lower bound of $\Omega(n (1 + H(r_1, \ldots, r_p)))$ for sorting a sequence of $n$ numbers, in the worst case over instances covered...
by \( \rho \) runs (ascending or descending) of lengths \( r_1, \ldots, r_\rho \), summing to \( n \) in the comparison model. The \textsc{Sorting} problem can be reduced in linear time to the problem of computing the \textsc{Convex Hulls} of a chain of \( n \) planar points that can be cut into \( \rho \) simple subchains of lengths \( r_1, \ldots, r_\rho \). For each real number \( r \), this is done by producing a point with \((x, y)\)-coordinates \((r, r^2)\). The \( \rho \) runs (alternating ascending and descending) are transformed into \( \rho \) simple subchains of the same lengths. The sorted sequence of the numbers can be obtained from the convex hull of the points in linear time.

\[\blacksquare\]

3.2 Computing Delaunay Triangulations: Adaptivity to Monotone Histograms

Djidjev and Lingas \cite{Djidjev1995} defined a \textit{monotone histogram} as a sequence of points sorted with respect to two orthogonal directions. They described an algorithm which, given a monotone histogram, computes the Voronoi diagram (and hence the Delaunay triangulation) of the input sequence in linear time.

A monotone histogram is also a simple polygonal chain. So, an algorithm for computing convex hulls adaptive to the decomposition of the input into monotone histograms is obtained as a corollary of Theorem 2. We extend the refined analysis to the computation of Delaunay triangulations and Voronoi diagrams adaptive to the decomposition of the input into monotone histograms.

Let \( d_1 \) and \( d_2 \) be two orthogonal directions. The algorithm described by Djidjev and Lingas \cite{Djidjev1995} suggests a way to partition the input into subsequences such that the Delaunay triangulation of each subsequence can be computed in linear time in its length. The partition algorithm cuts the sequence into the minimum number of monotone histograms with respect to \( d_1 \) and \( d_2 \). The first two points of the subsequence determine the ordering defined by the combination of ascending and descending with respect to \( d_1 \) and \( d_2 \).

Given a sequence of points and two orthogonal directions, it is possible to test whether the sequence is a monotone histograms with respect to these two directions in linear time.

**Binary Merge of Voronoi Diagrams.** Kirkpatrick \cite{Kirkpatrick1979} described a linear time algorithm for the merging of two arbitrary Voronoi diagrams. Given the Voronoi diagrams of two disjoint point sets \( P \) and \( Q \), the algorithm finds the Voronoi diagram of \( P \cup Q \) in time within \( O(|P| + |Q|) \). The plane is partitioned into points closer to \( P \), points closer to \( Q \), and points equidistant from \( P \) and \( Q \). The points equidistant from \( P \) and \( Q \) are defined as the \textit{contour} separating \( P \) and \( Q \). The \textit{contour} is composed of straight line segments: it is formed from the edges of the Voronoi diagram of \( P \cup Q \) that separates the points in \( P \) from the points in \( Q \). Inside the region of points closer to \( P \) (resp. \( Q \)) the Voronoi diagram of \( P \cup Q \) and the Voronoi diagram of \( P \) (resp. \( Q \)) are identical. Thus, the merging of two Voronoi diagrams can be seen as the process of cutting the Voronoi diagrams of \( P \) and \( Q \) along the contour.

This leads to a divide-and-conquer algorithm for constructing the Voronoi diagram of a set of \( n \) points and hence for computing the Delaunay triangulation.
in time within $O(n \log n)$. This time complexity of $O(n \log n)$ is asymptotically optimal in the comparison model in the worst case over instances composed of $n$ points. Shamos and Hoey (1975) showed that the construction of any triangulation over $n$ points requires $\Omega(n \log n)$, as sorting can be reduced to computing the triangulation of $n + 1$ points, which yields an asymptotic computational lower bound of $\Omega(n \log n)$ in the worst case over sets of $n$ planar points, in the comparison model.

Multinary Merge. Given $\mu$ Delaunay triangulations of sizes $v_1, \ldots, v_\mu$ to be merged, we make a sequence of binary merges, reducing at each step the number of Delaunay triangulations to be merged by 1. The merging process can be represented by a binary tree where the internal nodes are the merged Delaunay triangulations, and the leaves are the original $\mu$ Delaunay triangulations. Merging the two Delaunay triangulations of minimum sizes at each step further improves the merging process, which takes advantage of the potential disequilibrium in the distribution of the points between the $\mu$ Delaunay triangulations. We can apply the Huffman (1952) algorithm to the vector $(v_1, \ldots, v_\mu)$, thus obtaining a Huffman-shaped tree representing the merging process.

Lemma 3 (Multinary Merge). Given $\mu$ Delaunay triangulations of respective sizes $(v_1, \ldots, v_\mu)$ summing to $n = \sum_{i=1}^{\mu} v_i$, there is an algorithm computing the Delaunay triangulation of the $n$ points in time within $O(n(1 + H(v_1, \ldots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$.

Proof. The algorithm follows the same steps as the algorithm suggested by Huffman (1952) on a set of $\rho$ messages of probabilities $\{r_i/n\}_{i \in [1..\rho]}$:  
1. Initialize a heap $H$ with the $\rho$ Voronoi diagrams, indexed by their size;  
2. while $H$ contains more than one Voronoi diagram  
   - extract the two smallest Voronoi diagrams from $H$, of respective sizes $n_1$ and $n_2$,  
   - merge them into a Voronoi diagram $T$ of size $n_1 + n_2$, and  
   - insert $T$ in $H$.

This algorithm executes in time within $O(n(1 + H(v_1, \ldots, v_\mu)))$. The merging process is then represented by a tree with the shape of a Huffman (1952) tree. Consider the $i$-th Voronoi diagram of the input $\forall i \in [1..\mu]$; let $c_i$ be the binary string describing the path leading from the root to the corresponding leaf, and $l_i$ the length of this path. The sum of the computational costs of the binary merges is the sum of the sizes of the Voronoi diagram computed. The $i$-th Voronoi diagram contributes a cost within $O(v_i)$ to $l_i$ levels, which has a sum within $O(\sum_{i=1}^{\mu} l_i v_i)$. Consider the binary tree where the $\mu$ initial Voronoi diagrams are leaves and the $\mu - 1$ computed Voronoi diagrams are internal nodes:

- The set of binary strings $\{c_1, \ldots, c_\mu\}$ is a prefix free code, i.e. no code is prefix of another root-to-leaf path, simply because they are paths in a tree.
– The lengths of those codes minimize $\sum_{i=1}^{\mu} l_i r_i$ as a property of Huffman (1952) codes.

By the optimality of Huffman codes, this complexity is within a linear term of the entropy of the distribution $(v_1, \ldots, v_\mu)$, i.e. $\sum_{i=1}^{\mu} l_i r_i \in O(n(1 + H(v_1, \ldots, v_\mu)))$. This yields the final time complexity, within $O(n(1 + H(v_1, \ldots, v_\mu)))$. □

The combination of the partitioning algorithm and the merging process yields an optimal algorithm computing these structures adaptive to the decomposition of the input into monotone histograms.

**Theorem 3.** Let $d_1$ and $d_2$ be two perpendicular directions. Let $S$ be a sequence of $n$ planar points. Let $\mu$ and $(v_1, \ldots, v_\mu)$ be the minimum number of monotone histograms with respect to $d_1$ and $d_2$ and the sizes of these monotone histograms, respectively, in which $S$ can be cut. Then, the Delaunay triangulation and the Voronoi diagram of $S$ can be computed in time within $O(n(1 + H(v_1, \ldots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$, which is worst-case optimal in the comparison model.

**Proof.** The combination of the partitioning algorithm and the merging process yields an algorithm computing these structures within this time. In order to provide a lower bound, we use again the result of $\Omega(n(1 + H(r_1, \ldots, r_\rho)))$ for sorting a sequence of $n$ numbers, in the worst case over instances covered by $\rho$ runs of lengths $r_1, \ldots, r_\rho$ summing $n$ in the comparison model, showed by Barbay and Navarro (2013). This problem can be reduced in linear time to the problem of computing the Delaunay Triangulation of a sequence of $n$ planar points covered by $\rho$ monotone histograms of lengths $r_1, \ldots, r_\rho$ with respect to the coordinates axes. The $\rho$ runs are transformed into $\rho$ monotone histograms of the same lengths, using points on the parabola, in linear time. The sorted sequence of the numbers can be obtained from the Delaunay triangulation of the points in linear time. □

Next, we show the limitations of this refinement technique, by describing some other refined analysis, for which we were not able to obtain optimality.

### 4 Non-Optimal Refinements

Each different partitioning algorithm, in combination with the merging process described in the previous section, yields a different algorithm adaptive to the input order. We describe one such combination in Sections 4.1 to 4.3 and discuss alternate combinations in Section 4.4.

#### 4.1 Incremental Construction

Many Computational Geometry algorithms use tests known as the orientation and incircle tests (Shewchuk (1997)). The orientation test determines whether a point lies to the left of, to the right of, or on a line or plane defined by other
points. The *incircle test* determines whether a point lies inside, outside, or on a circle defined by other points.

Green and Sibson [1978] proposed the first incremental algorithm for computing the Delaunay triangulation of a point set which finds the triangle containing each new point, and updates the diagram by correcting the edges violating the circumcircle condition—an operation named *flipping*.

The algorithm adds points to the structure one by one. For each point, it performs two basic steps:

1. The algorithm finds the triangle containing the new point using the structure as a guide to the relative position of the points. A greedy approach for locating the point is to start at an edge in the structure and to walk across adjacent edges in the direction of the new point until the correct triangle is found. Orientation tests (Shewchuk 1997) are performed on each edge of such a path to see whether the new point lies on the correct side of that edge. We call the operations involved in this walk *navigation operations*.

2. It then updates the structure adding three new edges from the point inserted to the vertices of the triangle containing the point and flips all invalid edges resulting from the insertion. Note that flipping an edge can make another edge invalid but that each edge is flipped at most once, so each insertion can trigger at most a linear number of flips.

The time complexity of this incremental algorithm for computing the Delaunay triangulation is within $O(dn) \subseteq O(n^2)$, where $d \in [1..n]$ is an upper bound on the amount of operations required to insert each point and to correct the Delaunay triangulation for the instance being considered.

### 4.2 Adaptivity to D-linear Runs

Consider an incremental algorithm $G$ computing the Delaunay triangulation of $n$ planar points in time within $O(n)$ in the best case; and an algorithm $M$ merging two Delaunay triangulations of sizes summing to $n$ in time within $O(n)$ in the worst case. Given a sequence $S$ of $n$ distinct planar points and a constant $k$, the following naive algorithm computes its Delaunay triangulation in time within $O(n)$ in the best case, in time within $O(n \log n)$ in the worst case, and in time within $O(n(1 + H(r_1, \ldots, r_\rho)) \subseteq O(n \log \rho) \subseteq O(n \log n)$ in the general case, where $\rho \in [1..n]$ and $r_1, \ldots, r_\rho$ (such that $n = \sum_{i=1}^\rho r_i$) measure the difficulty of partitioning the instance into “easy” subinstances:

1. Run the incremental algorithm $G$ on $S$ until, for the $i$-th point $p$, it performs either more than $k$ operations to locate the point on the triangulation, or more than $k$ flip operations on it.

2. Store the Delaunay triangulation of the $i - 1$ first points, and restart the greedy incremental algorithm $G$ on the sequence formed by $p$ and its $n - i$ successors in the input sequence $S$, until no points are left in $S$. 
3. Let $\rho \in [1..n]$ and $r_1, \ldots, r_\rho$ (such that $n = \sum_{i=1}^\rho r_i$) be the number of Delaunay triangulations computed in this way and the sizes of these Delaunay triangulations, respectively. Merge the $\rho$ Delaunay triangulations in overall time within $O(n(1 + H(r_1, \ldots, r_\rho))) \subseteq O(n(1 + \log \rho)) \subseteq O(n \log n)$.

**Greedy Partitioning.** The algorithm described above suggests a way to partition the input into subsequences such that the Delaunay triangulation of each subsequence can be computed in linear time in its length.

We use the algorithm described above to give a definition of “easy” sequences of points for the computation of the Delaunay triangulation:

**Definition 2 (D-linear).** Consider a sequence $S$ of $n$ distinct planar points. Given an integer value $k > 0$, $S$ is $k$-D-linear (for “Delaunay linear”) if $G$ performs at most $k$ location and flip operations for each point while computing the Delaunay triangulation of $S$. If $S$ is $k$-D-linear and $k \in O(1)$ is a constant independent of $n$, we say that $S$ is D-linear.

Such a simple definition yields a simple partitioning of the input sequence into subsequences of consecutive positions such that the Delaunay triangulation of each subsequence can be computed in linear time.

**Definition 3 (D-linear Run).** A D-linear run in a sequence $S$ of points is a D-linear subsequence formed by consecutive points in $S$.

This partition algorithm greedily adds points to the D-linear run while it has not executed a number of operations exceeding the threshold $k$.

### 4.3 Greedy partition vs Optimal partition

Given a sequence $S$, an optimal partition of $S$ into D-linear runs is a partition with the minimum number of D-linear runs. We show in this section that the greedy partitioning algorithm into D-linear runs does not always yield a partition into the minimum possible number of D-linear runs.

We describe a family of sequences $S$, such that for all positive integer $n$, there is a sequence $S_n$ of $n$ points in $S$ where the greedy partitioning algorithm yields three D-linear runs when $S_n$ can be optimally partitioned into just two D-linear runs.

The Delaunay triangulation of $m \geq 2$ points on the parabola $y = x^2$ is such that the leftmost point is adjacent to all the other $m - 1$ points, and the left-to-right order is a D-linear run. The proof of the following lemma is based on this fact.

**Lemma 4.** For all positive integers $n$, there exists a sequence of $n$ planar points where the greedy partitioning algorithm yields three D-linear runs, whereas the optimal partition of this sequence has only two D-linear runs.
Fig. 3. The sequences $A$, $B$ and $A'$. 

Proof. Let $k$ be the threshold used in the definition of D-linear sequences. Let $A$ be a set of $m > k$ points in the parabola $y = x^2$, denoted $p_1, \ldots, p_m$ from left to right. Let $C$ be the circumcircle of the triangle $p_1, p_2, p_3$. Let $u, v, w$ be three points such that the triangle $\Delta uvw$ with vertex set $\{u, v, w\}$ is small enough with respect to the convex hull of $A$, and located inside $C$. Let $A' = \{p'_1, \ldots, p'_m\}$ be a scaled copy of $A$, located inside $\Delta uvw$. Let $B$ be the $k + 3$ vertex set of $k + 1$ decreasing area and disjoint triangles such as every two consecutive triangles share an edge (see Fig. 3). Suppose that $B$ is ordered such that the sequence of points is a D-linear run. The points $u, v$ and $w$ are the last three points of $B$. The triangle formed by the first three points of $B$ contains the point $p_1$ of $A$. Let $P = A \cup A' \cup B$ be ordered as $(B, p_1, \ldots, p_m, p'_1, \ldots, p'_m)$. Applying the greedy partitioning algorithm to $P$ gives the three D-linear runs $(B)$, $(p_1, \ldots, p_m)$, and $(p'_1, \ldots, p'_m)$ since: (1) adding $p_1$ to the Delaunay triangulation of $B$ requires traversing $k$ triangles to locate $p_1$; and (2) adding $p'_1$ to the Delaunay triangulation of $A$ requires creating $m$ triangles (i.e. $p'_1$ is adjacent to every point in $A$). However, $(B \setminus \{u, v, w\})$ is a D-linear run and $(u, v, w, p_1, \ldots, p_m, p'_1, \ldots, p'_m)$ is another one. Note that the triangle $\Delta uvw$ blocks the points in $A'$ of being adjacent to any point in $A$. 

This family of sequences $S$ can be generalized to show a gap of $\Theta(n)$ between the number of D-linear runs yielded by the greedy partitioning algorithm in $S_n$ and the minimum number of D-linear runs in which $S_n$ can be partitioned.

Lemma 5. For all positive integer $n$, there exists a sequence of $n$ planar points where the greedy partitioning algorithm yields $\rho \in \Theta(n)$ D-linear runs, and the optimal partition of this sequence has only 2 D-linear runs.

Proof. Let $A$ be a set of $m > k$ consecutive points in the parabola $y = x^2$ denoted $p_1, \ldots, p_m$ from left to right. Let $C$ be the circumcircle of the triangle $p_1, p_2, p_3$. Let $u, v, w$ be three points such that the triangle $\Delta uvw$ with vertex set $\{u, v, w\}$ is small enough with respect to the convex hull of $A$, and located inside $C$. Let $A'$ be a D-linear run sequence of $l$ points, of increasing $y$-coordinate and decreasing $x$-coordinate, inside $C$ denoted $p'_1, \ldots, p'_l$. During the incremental
construction of the Delaunay triangulation of the sequence \( AA' \) the first point of \( A' \) forces a flipping of \( m \) edges in \( A \) (i.e. \( m \) new edges are created connecting \( p'_1 \) with all points in \( A \)). Every next point of \( A' \) forces a flipping of the \( m \) edges connecting points in \( A \) with points in \( A' \). Let \( q'_1 \) and \( q'_2 \) be two consecutive points from the sequence \( A' \) and let \( q_1, \ldots, q_{k+1} \) be \( k+1 \) consecutive points from \( A \). The sequence \( Q = \langle q'_1, q_1, \ldots, q_{k+1}, q'_2 \rangle \) is not a \( \mathcal{D} \)-linear run because \( q'_2 \) forces a flipping of the \( k+1 \) edges connecting \( q'_1 \) with all the points in \( q_1, q_2, \ldots, q_{k+1} \). Let \( B \) be the \((k+3)\) vertex sequence of \( k+1 \) decreasing area and disjoint triangles so that every two consecutive triangles share an edge. The points \( u, v \) and \( w \) are the last three points of \( B \). The triangle formed by the first three points of \( B \) contains inside the point \( p_1 \) of \( A \), \( B \) is ordered such that the sequence of vertices is a \( \mathcal{D} \)-linear run. Let \( P = A \cup A' \cup B \) be ordered as \( \langle B, p_1, \ldots, p_{k+1}, p'_1, p_{k+2}, \ldots, p_{2k+3}, p'_2, \ldots, p_m \rangle \), that is, first the points of \( B \), then alternate \( k+1 \) points of \( A \) with 1 point of \( A' \), resulting subsequences similar to \( Q \). The greedy algorithm divides \( P \) into \( \min(l, m, \frac{m}{k+1})+1 \) \( \mathcal{D} \)-linear runs: \( \langle B \rangle, \langle p_1, \ldots, p_{k+1} \rangle \) and every subsequence starting with 1 point from \( A' \) followed by \( k+1 \) points from \( A \). Since: (i) adding \( p_1 \) to the Delaunay triangulation of \( B \) requires traversing \( k \) triangles to locate \( p_1 \); (ii) adding \( p'_1 \) to the Delaunay triangulation of \( p_1 \ldots p_{k+1} \) requires creating \( k \) triangles (i.e. \( p'_1 \) is adjacent to every point in \( p_1 \ldots p_{k+1} \)); and (iii) adding \( p'_{i+1} \) to \( \langle p'_1, p_{i+k+j}, \ldots, p_{(i+1)k+j+1} \rangle \) produces a sequence similar to \( Q \), and hence is not a \( \mathcal{D} \)-linear run. However, \( \langle B \setminus \{u, v, w\} \rangle \) is a \( \mathcal{D} \)-linear run and \( \langle u, v, w, p_1, \ldots, p_k, p'_1, p_{k+1}, p_{k+2}, \ldots, p_{2k}, p'_2, \ldots, p_m \rangle \) is another one. Note that the triangle \( \triangle uvw \) blocks the points in \( A' \), any vertex in \( A \). It is possible to build a sequence \( l = \frac{m}{k+1} = \frac{n-k+3}{k+2} \) (1 point in \( A' \) for each sequence of \( k+1 \) points in \( A \)) such that the number of \( \mathcal{D} \)-linear runs yielded by the greedy partitioning algorithm is \( \frac{n-k+3}{k+2}+1 \) and the optimal partition has only 2 \( \mathcal{D} \)-linear runs.

Given a sequence \( S \) of \( n \) planar points, one can partition \( S \) into an optimal number of \( \mathcal{D} \)-linear runs in time within \( O(n^2) \): suffices to compute for each point \( p \) in \( S \) the largest \( \mathcal{D} \)-linear run starting at \( p \), respective sizes \( n_1, \ldots, n_p \), summing to \( n \) such that \( H(n_1, \ldots, n_p) \) is within a constant additive term of the minimal entropy among partitions in runs.

The running time of the partitioning algorithm must be within \( O(n \log n) \) but adaptive to the same parameter as the merging in order to obtain an adaptive algorithm computing the Delaunay triangulation in time within \( o(n \log n) \) on some classes of instances.

### 4.4 Other Partitioning Schemes

Since the greedy partitioning algorithm does not yield an optimal partition, we explore more sophisticated partition techniques and show that they suffer similar problems.

**Test and Divide.** We adapt the Levcopoulos et al. (2002) partition technique to cut a sequence of planar points into \( \mathcal{D} \)-linear runs. The test and divide par-
Partitioning is another partition technique based on the incremental algorithm $G$ for the computation of the Delaunay triangulation (seen in Sect. 4.1). Given a sequence $S = \langle p_1, \ldots, p_n \rangle$ of $n$ planar points we first test whether the sequence $S$ is D-linear. In such a case, we identify the sequence $S$ as a D-linear run. If not, we partition the sequence $S$ into $S' = \langle p_1, \ldots, p_{\lfloor n/2 \rfloor} \rangle$ and $S'' = \langle p_{\lfloor n/2 \rfloor+1}, \ldots, p_n \rangle$ and recursively the same procedure is applied to $S'$ and $S''$. While the greedy partitioning has a linear time complexity, this partitioning has a time complexity within $O(n \log \rho) \subseteq O(n \log n)$ where $\rho$ is the number of D-linear runs identified by the partitioning process.

There exists a sequence $S$ of $n$ points where the test and divide partitioning yields a number of D-linear runs linear on $n$, and $S$ can be partitioned optimally in just 2 D-linear runs.

**Lemma 6.** For all positive integer $n$, there exists a sequence of $n$ planar points where the test and divide partitioning yields $\rho \in \Theta(n)$ D-linear runs and the optimal partition of this sequence has only 2 D-linear runs.

**Proof.** Consider the sequence $S$ used in the proof of Lemma 5 where the number of points in $B$ is changed to $\lfloor n/2 \rfloor$ instead of $k + 3$ then the test and divide partitioning will cut $S$ in the last point of $B$. $B$ is a D-linear run, but the bisection partitioning will cut the rest of $S$ in $\frac{n}{2(k+2)}$ D-linear runs. \qed

**Amortized Greedy Partitioning.** The greedy partitioning algorithm adds the point $p_i$ to the current run whether the number of location and flip operations of the incremental algorithm $G$ in $p_i$ is less than a threshold $k$. But it is possible that in some points the number $r$ of operations would be much lower than $k$. We define a partition algorithm that makes use of the $k-r$ remaining operations in the subsequent points, similar to the accounting method for amortized analysis. The number $k-r$ of remaining operations are credited to be used later, so that the point $p_i$ will be added to the current run if the number of navigation and flip operations of the incremental algorithm $G$ is less than $k$ plus this credit.

We use this algorithm to give another definition of “easy” sequences of points for the computation of the Delaunay triangulation:

**Definition 4 (Amortized D-linear).** Consider a sequence $S = \langle p_1, \ldots, p_n \rangle$ of $n$ distinct planar points. Given an integer value $k > 0$, $S$ is Amortized $k$-D-linear if for each point $p_i \in S$, the incremental algorithm $G$ performs at most $k+i$ navigation and flip operations while computing the Delaunay triangulation of the sequence $p_1, \ldots, p_i$. If $S$ is Amortized $k$-D-linear and $k \in O(1)$ is a constant independent of $n$, we say that $S$ is Amortized D-linear.

Such definition yields a partitioning of the input sequence into subsequence of consecutive positions:

**Definition 5 (Amortized D-linear Run).** An Amortized D-linear run in a sequence $S$ of points is an Amortized D-linear subsequence formed by consecutive points in $S$. 
We define the optimal partition into Amortized D-Linear runs as the partition with the minimum number of Amortized D-linear runs.

**Lemma 7.** For all positive integer $n$, there exists a sequence of $n$ planar points where the amortized greedy partitioning yields $\rho \in \Theta(n)$ Amortized D-linear runs and the optimal partition has only 2 Amortized D-linear runs.

**Proof.** The construction of the sequence follows the same scheme of previous constructions (see the proof of Lemma 5). The sequence $B$ is such that the incremental algorithm $G$ performs $k$ navigation and flip operations in almost every point in $B$, so that $B$ is still a run but the credit is almost zero at the end of $B$. Again, the last 3 points of $B$ form a triangle that blocks the points in $A'$ of being adjacent to any vertex in $A$. The sequence $S$ alternates points in $A$ with points in $A'$ such that each new point in $A'$ is adjacent to every point in $A$, thus when the numbers of points in $A$ is large enough to exceed the credit, the Amortized D-linear run is cut. This partition algorithm yields close to $n/k^2$ Amortized D-linear runs. This sequence can be partitioned optimally in just 2 Amortized D-linear runs. Note that the triangle $\Deltauvw$ blocks the points in $A'$ of being adjacent to any point in $A$. $\Box$

**Amortized Test and Divide Partitioning.** The amortized test and divide partitioning is a combination of the test and divide partitioning and the amortized greedy partitioning. Given a sequence $S = \langle p_1, \ldots, p_n \rangle$ of $n$ planar points we first test whether the sequence $S$ is Amortized D-linear. In such a case, we identify the sequence $S$ as an Amortized D-linear run. If not, we partition the sequence $S$ into $S' = \langle p_1, \ldots, p_{\lfloor n/2 \rfloor} \rangle$ and $S'' = \langle p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n \rangle$ and recursively the same procedure is applied to $S'$ and $S''$.

**Lemma 8.** For all positive integer $n$, there exists a sequence of $n$ planar points where the amortized test and divide partitioning yields $\rho \in \Theta(n)$ Amortized D-linear runs and the optimal partition has only 2 Amortized D-linear runs.

**Proof.** The construction of the sequence follows the same scheme of previous constructions (see the proof of Lemma 5). The sequence $B$ is such that the incremental algorithm $G$ performs $k$ navigation and flip operations in almost every point in $B$, so that $B$ is still a run but the credit is almost zero at the end of $B$. Again, the last 3 points of $B$ form a triangle that blocks the points in $A'$ of being adjacent to any vertex in $A$. The sequence $S$ alternates points in $A$ with points in $A'$ such that each new point in $A'$ is adjacent to every point in $A$, thus when the numbers of points in $A$ is large enough to exceed the credit, the Amortized D-linear run is cut. This partition algorithm yields close to $n/k^2$ Amortized D-linear runs. This sequence can be partitioned optimally in just 2 Amortized D-linear runs. Note that the triangle $\Deltauvw$ blocks the points in $A'$ of being adjacent to any point in $A$. $\Box$
5 Discussion

We described two techniques to refine the analysis of Divide-and-Conquer input order adaptive algorithms for computing the Convex Hull, Voronoi Diagram and Delaunay Triangulation of $n$ planar points within $O(n(1 + \mathcal{H}(n_1, \ldots, n_k))) \subseteq O(n(1 + \log k)) \subseteq O(n \log n)$ for some parameters $n_1, \ldots, n_k$ summing $n$, that measure the difficulty of the instance. We also showed the optimality of these results by providing lower bounds that match the refined analyses.

One of these techniques is based on the partitioning of a sequence of $n$ distinct planar points into $\rho$ subsequences, which corresponding $\rho$ Voronoi diagrams and Delaunay triangulations can be computed quickly, and a technique to efficiently merge those $\rho$ Voronoi diagrams and Delaunay triangulations into a single one, inspired by Huffman (1952) codes. Each combination of partition and merging technique yields adaptive algorithms to the input order for computing Voronoi Diagrams and Delaunay Triangulations.

Those results show that the Delaunay Triangulation can be computed in time within $o(n \log n)$ on some classes of instances, yet we described examples for various partitioning algorithm where the complexity achieved is sub-optimal. It is possible that the Delaunay Triangulation algorithms based on these partitioning methods are not optimally adaptive, i.e. that they are not taking full advantage of the input order.

This leaves open problems such as whether to find better partitioning algorithms (e.g. approximating by a polynomial term an optimal partition of size $\rho$ in time within $O(n \log \rho)$); or to prove that, for every partitioning algorithm running in time within $O(n \log \rho)$, there exists a family of sequences of planar points such that the gap between the size of the partition yielded by the algorithm on one hand, and the partition with the minimum size $\rho$ on the other hand, is within $\Theta(n)$. 
Afshani, P., Barbay, J., and Chan, T. (2009). Instance-optimal geometric algorithms. In *Proceedings 50th IEEE Symposium on Foundations of Computer Science (FOCS)*.

Barbay, J. and Navarro, G. (2013). On compressing permutations and adaptive sorting. *Theoretical Computer Science (TCS)*, 513:109–123.

Chazelle, B. (1991). Triangulating a simple polygon in linear time. *Discrete & Computational Geometry (DCG)*, 6(5):485–524.

Djidjev, H. and Lingas, A. (1995). On computing voronoi diagrams for sorted point sets. *International Journal of Computational Geometry & Applications (IJCGA)*, 5(3):327–337.

Green, P. J. and Sibson, R. (1978). Computing dirichlet tessellations in the plane. *The Computer Journal (TCJ)*, 21(2):168–173.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers (IRE)*, 40(9):1098–1101.

Kirkpatrick, D. G. (1979). Efficient computation of continuous skeletons. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, Washington, DC, USA. IEEE Computer Society.

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

Knuth, D. E. (1973). *The Art of Computer Programming, Vol 3*, chapter Sorting and Searching, Section 5.3. Addison-Wesley.

Levcopoulos, C., Lingas, A., and Mitchell, J. S. B. (2002). Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 80–89, London, UK. Springer-Verlag.

Melkman, A. A. (1987). On-line construction of the convex hull of a simple polyline. *Information Processing Letters (IPL)*, 25(1):11–12.

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

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag.

Shamos, M. and Hoey, D. (1975). Closest point problems. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162.

Shewchuk, J. R. (1997). Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363.