Lower Bounds

Michiel Smid*

December 7, 1999

1 Lower bounds for comparison based algorithms

We start by considering algorithms whose execution only depends on the outcomes of comparisons between input variables. Such an algorithm gets as input a sequence $s_1, s_2, \ldots, s_n$ of elements from some ordered universe U, and can only compare pairs of input elements. The outcome of a comparison between s_i and s_j is either "$s_i \leq s_j$" or "$s_i > s_j$". At any moment during the execution of the algorithm, the two elements that are compared only depend on the outcomes of previous comparisons. The *complexity* of such an algorithm is defined as the number of comparisons made in the worst case.

The basic idea for proving a lower bound for comparison based algorithms is as follows. Let $\mathcal{P}$ be a problem that has N possible solutions, and let $\mathcal{A}$ be any algorithm that solves $\mathcal{P}$. Assume that in one comparison, $\mathcal{A}$ can at most halve the number of possible solutions. Then, in the worst case, this algorithm has to make at least $\log N$ comparisons. Put differently, $\mathcal{A}$ has to obtain $\log N$ bits of information in order to solve $\mathcal{P}$. In one comparison, it obtains at most one bit of information. Therefore, at least $\log N$ comparisons have to be made. This lower bound is often called the *information theoretic bound*.

1.1 A lower bound for the sorting problem

Let us apply this idea to the *sorting problem*. First note that many sorting algorithms, such as insertion-sort, merge-sort, heapsort, and quicksort, are comparison based.

*Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg. E-mail: michiel@latrappe.cs.uni-magdeburg.de.

If a sorting algorithm gets an input sequence of length n, then it basically has to determine the permutation of $1, 2, \ldots, n$ that puts this sequence into sorted order. Since there are $n!$ permutations, we expect a lower bound of $\log n!$ on the worst-case number of comparisons that have to be made by *any* comparison based sorting algorithm. We know from Stirling's approximation that $\log n! = n \log n - O(n)$. Hence, we will get a lower bound of $n \log n - O(n)$ on the complexity of the sorting problem. In the rest of this section, we will give a formal proof of this lower bound.

As an introduction to this formal proof, consider the following algorithm that sorts any sequence s_1, s_2, s_3 of length three. The first comparison is between s_1 and s_2. If $s_1 \leq s_2$, then s_2 is compared with s_3. If $s_2 \leq s_3$, then the input sequence is in sorted order already. Otherwise, if $s_2 > s_3$, one more comparison—between s_1 and s_3—is needed to determine the sorted order. In case the first comparison has $s_1 > s_2$ as outcome, we proceed similarly. We can represent this algorithm by a binary tree, called a *decision tree*. The internal nodes of this tree are labeled with comparisons. A node labeled $s_i : s_j$ denotes a comparison between s_i and s_j. The edge leading to the left child corresponds to the outcome $s_i \leq s_j$, whereas the edge leading to the right child corresponds to the outcome $s_i > s_j$. The leaves of the tree are labeled with permutations of $1, 2, 3$. Applying such a permutation to the indices of the input sequence gives this sequence in sorted order. See Figure 1.

Definition 1 A *decision tree* is a finite binary tree whose internal nodes have labels of the form $s_i : s_j$. Each internal node has two outgoing edges; the left one is labeled with $\leq$, whereas the right one is labeled with $>$.

Let T be a decision tree. This tree corresponds to a comparison based algorithm $\mathcal{A}_T$: Let $s_1, s_2, \ldots, s_n$ be an input sequence for this algorithm. Then $\mathcal{A}_T$ starts in the root of T, and follows a path down the tree. Assume the algorithm has reached node u, and let this node have label $s_i : s_j$. Then $\mathcal{A}_T$ compares s_i with s_j. If $s_i \leq s_j$, then $\mathcal{A}_T$ proceeds to the left child of u. Otherwise, if $s_i > s_j$, it proceeds to the right child of u. The algorithm terminates when it reaches a leaf of T.

Definition 2 Let T be a decision tree. Assume that the leaves of T are labeled with permutations of $1, 2, \ldots, n$. We say that T *solves the sorting problem* for n elements, if the following holds.

1. For any sequence $s_1, s_2, \ldots, s_n \in U$, if the leaf reached by algorithm $\mathcal{A}_T$, when given this sequence as input, has the permutation π as its label, then
$$s_{\pi(1)} \leq s_{\pi(2)} \leq s_{\pi(3)} \leq \ldots \leq s_{\pi(n)}.$$

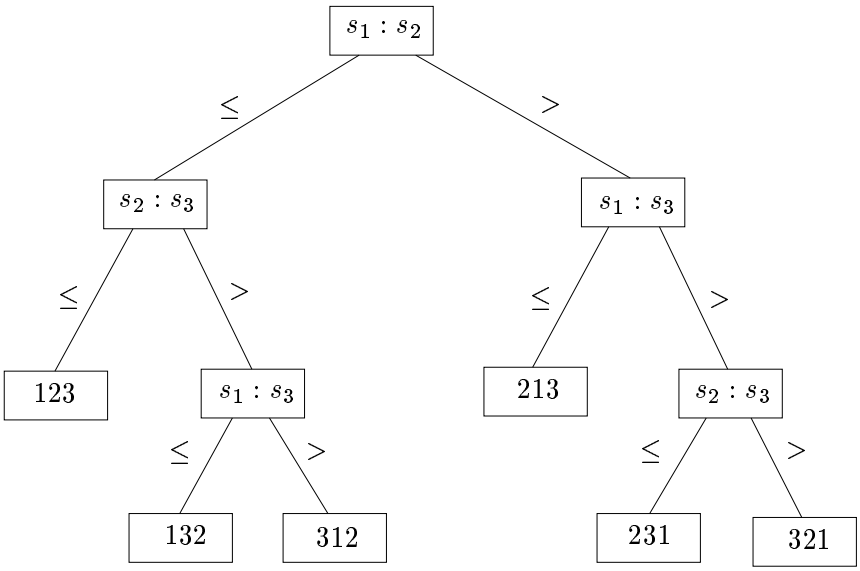

Figure 1: *A decision tree that sorts input sequences of length three.*

---

2. For every leaf w of T, there is an input sequence on which algorithm $\mathcal{A}_T$ terminates in w.

Exercise 1 Convince yourself that this definition expresses what you intuitively expect.

Given a decision tree T, we have defined a corresponding comparison based algorithm $\mathcal{A}_T$, and we have defined what it means that T solves the sorting problem. Conversely, given any comparison based sorting algorithm $\mathcal{B}$, we can define a decision tree T, such that $\mathcal{A}_T = \mathcal{B}$. Hence, there is a one-to-one correspondence between comparison based sorting algorithms and decision trees that solve the sorting problem. Therefore, we speak about *decision tree algorithms*.

Remark 1 A decision tree is only used to *analyze* the complexity of an algorithm; it is *not* constructed. Such a tree represents all possible behaviors of the algorithm.

Let T be a decision tree that solves the sorting problem for n elements, and let $s_1, s_2, \ldots, s_n$ be an input sequence. Then the length of the path in T that algorithm $\mathcal{A}_T$ traverses on this input is exactly the number of comparisons made. Therefore, since for each leaf w of T there is an input sequence

on which algorithm $\mathcal{A}_T$ terminates in w, the worst-case complexity of a comparison based sorting algorithm is equal to the height of the corresponding decision tree.

Definition 3 The *complexity of the sorting problem* for n elements in the decision tree model is defined as the minimal height of any decision tree solving this problem.

We now prove our lower bound on the complexity of the sorting problem.

Theorem 1 *The complexity of the sorting problem in the decision tree model is at least equal to* $\lceil \log n! \rceil$. *That is, any decision tree algorithm for sorting* n *elements from an ordered universe makes at least* $\lceil \log n! \rceil$ *comparisons in the worst case.*

Proof. Let T be an arbitrary decision tree solving the sorting problem for n elements, and let h be its height. Since T is a binary tree, it has at most 2^h leaves. On the other hand, since there are $n!$ permutations of $1, 2, \ldots, n$, it follows from Definition 2 that T must have at least $n!$ leaves. It follows that $n! \leq 2^h$, which shows that T has height at least $\lceil \log n! \rceil$. ∎

Exercise 2 We have mentioned already that, by Stirling's approximation, Theorem 1 gives a lower bound of $n \log n - O(n)$ for decision tree sorting algorithms. Let $C(n)$ denote the number of comparisons made by the merge-sort algorithm on an input sequence of length n. Then $C(1) = 0$, and $C(n) = n - 1 + C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil)$ for $n > 1$. Prove that

$$C(n) = n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1.$$

This implies that, in the decision tree model, the complexity of the sorting problem is at most equal to $n \log n + 1$.

1.2 The element uniqueness problem

In this section, we prove a lower bound for the *element uniqueness problem*, which is defined as follows. Given a sequence $s_1, s_2, \ldots, s_n$ of elements from an ordered universe U, decide whether these elements are pairwise distinct, i.e., $s_i \neq s_j$ for all $i \neq j$.

By considering all pairs s_i, s_j, where $i \neq j$, this problem can clearly be solved by making $\binom{n}{2} = O(n^2)$ comparisons. There is a much faster algorithm: First sort the sequence using the merge-sort algorithm. Now observe that the elements of the input sequence are pairwise distinct if and only if any

two elements that are neighbors in the sorted sequence are distinct. Hence, after having sorted the sequence, we can solve the problem by making $n-1$ comparisons. This algorithm makes overall at most $n \log n + n$ comparisons. In this section, we will show that this is optimal in the decision tree model.

A decision tree T solving the element uniqueness problem for n elements is again a binary tree. As before, internal nodes of T are labeled with comparisons $s_i : s_j$, and have two outgoing edges which are labeled with $\leq$ and $>$. Each leaf of T has a label YES or NO. On input $s_1, s_2, \ldots, s_n$, the algorithm $\mathcal{A}_T$ corresponding to T terminates in a leaf labeled YES if and only if the input elements are pairwise distinct.

We will prove that any decision tree solving the element uniqueness problem for n elements has at least $n!$ leaves. This will imply that the height of such a tree is at least $\lceil \log n! \rceil$. As a result, any comparison based algorithm that solves the element uniqueness problem makes at least $\lceil \log n! \rceil = n \log n - O(n)$ comparisons in the worst case.

So let T be an arbitrary decision tree solving the element uniqueness problem for n elements. If π is a permutation of $1, 2, \ldots, n$, then we denote by $w(\pi)$ the leaf of T in which algorithm $\mathcal{A}_T$ terminates when given as input the sequence

$$\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n).$$

Note that such a leaf $w(\pi)$ has label YES.

Theorem 2 *If π and ρ are two different permutations of $1, 2, \ldots, n$, then $w(\pi) \neq w(\rho)$.*

This theorem implies that different permutations lead to different leaves of the decision tree T. Hence, this tree has at least $n!$ leaves. In the rest of this section, we will prove Theorem 2. When reading the proof for the first time, it may help to assume that π is the identity permutation.

The proof of Theorem 2 is by contradiction. So assume that $w(\pi) = w(\rho)$. Let $\alpha_k := \pi(k)$, for $1 \leq k \leq n$. Note that $\pi^{-1}(\alpha_k) = k$.

Lemma 1 *There is an index i, $1 \leq i < n$, such that $\rho^{-1}(\alpha_i) > \rho^{-1}(\alpha_{i+1})$.*

Proof. If such an index does not exist, then

$$\rho^{-1}(\alpha_1) < \rho^{-1}(\alpha_2) < \ldots < \rho^{-1}(\alpha_n).$$

Therefore, since ρ^{-1} is a permutation of $1, 2, \ldots, n$, we must have $\rho^{-1}(\alpha_k) = k$ for all k, $1 \leq k \leq n$. It follows that $\pi^{-1} = \rho^{-1}$, which implies that $\pi = \rho$. This is a contradiction. ∎

Lemma 2 *Let i be an index for which $\rho^{-1}(\alpha_i) > \rho^{-1}(\alpha_{i+1})$, and let u be an internal node on the path in T from the root to leaf $w(\pi)$. This node u is not labeled with the comparison $s_{\alpha_i} : s_{\alpha_{i+1}}$ or $s_{\alpha_{i+1}} : s_{\alpha_i}$.*

Proof. Assume that u is labeled with the comparison $s_{\alpha_i} : s_{\alpha_{i+1}}$. On input $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$, algorithm $\mathcal{A}_T$ passes through node u. In this node, it compares the α_i-th element with the α_{i+1}-st element. Since $\pi^{-1}(\alpha_i) = i < i + 1 = \pi^{-1}(\alpha_{i+1})$, the algorithm proceeds to the left child of u.

Similarly, on input $\rho^{-1}(1), \rho^{-1}(2), \ldots, \rho^{-1}(n)$, the algorithm visits node u, and compares $\rho^{-1}(\alpha_i)$ with $\rho^{-1}(\alpha_{i+1})$. It follows from Lemma 1, that the algorithm proceeds to the right child to u.

This is a contradiction, because the fact that $w(\pi) = w(\rho)$ implies that the computations for both input sequences terminate in the same leaf of T.

The case when node u is labeled with the comparison $s_{\alpha_{i+1}} : s_{\alpha_i}$ can be handled in a symmetric way. ∎

Again, let i be an index for which $\rho^{-1}(\alpha_i) > \rho^{-1}(\alpha_{i+1})$. Consider the input sequence $r_1, r_2, \ldots, r_n$, where

$$r_k := \begin{cases} \pi^{-1}(k) & \text{if } 1 \le k \le n \text{ and } k \ne \alpha_{i+1}, \\ i & \text{if } k = \alpha_{i+1}. \end{cases}$$

The sequences $r_1, r_2, \ldots, r_n$ and $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$ are the same, except at position α_{i+1}. The r-sequence has the value i at this position, whereas the π^{-1}-sequence has the value $i + 1$ there. Also, since $r_{\alpha_i} = r_{\alpha_{i+1}}$, the r-sequence is a NO-input for the element uniqueness problem.

Lemma 3 *On input $r_1, r_2, \ldots, r_n$, algorithm $\mathcal{A}_T$ terminates in leaf $w(\pi)$.*

Proof. We first show that

$$r_k \le r_\ell \text{ if and only if } \pi^{-1}(k) \le \pi^{-1}(\ell), \tag{1}$$

for all k and ℓ, such that $1 \le k \le n$, $1 \le \ell \le n$, and $\{k, \ell\} \ne \{\alpha_i, \alpha_{i+1}\}$.

Clearly, (1) holds if $k = \ell$. So assume that $k \ne \ell$. If $k \ne \alpha_{i+1}$ and $\ell \ne \alpha_{i+1}$, then $r_k = \pi^{-1}(k)$ and $r_\ell = \pi^{-1}(\ell)$ and, hence, (1) holds. Next, consider the case when $k = \alpha_{i+1}$ and $\ell \ne \alpha_i$. Since $r_k = i$, $r_\ell = \pi^{-1}(\ell)$, $\pi^{-1}(\ell) \ne i$, and $\pi^{-1}(k) = i + 1$, we have

$$\begin{aligned} r_k \le r_\ell \quad & \text{iff} \quad i \le \pi^{-1}(\ell) \\ & \text{iff} \quad i + 1 \le \pi^{-1}(\ell) \\ & \text{iff} \quad \pi^{-1}(k) \le \pi^{-1}(\ell), \end{aligned}$$

i.e., (1) holds. It remains to consider the case when $\ell = \alpha_{i+1}$ and $k \neq \alpha_i$. Since $\pi^{-1}(k) \neq i+1$, we have

$$\begin{aligned} r_k \leq r_\ell \quad &\text{iff} \quad \pi^{-1}(k) \leq i \\ &\text{iff} \quad \pi^{-1}(k) \leq i+1 \\ &\text{iff} \quad \pi^{-1}(k) \leq \pi^{-1}(\alpha_{i+1}) = \pi^{-1}(\ell), \end{aligned}$$

which shows that also in this case (1) holds.

The path in T that is followed by algorithm $\mathcal{A}_T$ on a particular input sequence only depends on the outcomes of comparisons. If $\mathcal{A}_T$ compares the k-th and ℓ-th input elements, where $\{k, \ell\} \neq \{\alpha_i, \alpha_{i+1}\}$, then (1) implies that the outcomes are the same for the input sequences $r_1, r_2, \ldots, r_n$ and $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$. We know from Lemma 2 that on the path to $w(\pi)$ none of the two comparisons $s_{\alpha_i} : s_{\alpha_{i+1}}$ and $s_{\alpha_{i+1}} : s_{\alpha_i}$ is made. It follows that the computations on the r-sequence and the π^{-1}-sequence follow the same path and, therefore, terminate in the same leaf, namely $w(\pi)$. ■

We saw already that the leaf $w(\pi)$ is labeled YES. By Lemma 3, however, this leaf should have label NO, because the r-sequence contains two equal elements. It follows that our decision tree T does not correctly solve the element uniqueness problem. This is a contradiction and, therefore, our assumption that $w(\pi) = w(\rho)$ was wrong. Hence, we have proved Theorem 2. This, in turn, implies the following result.

Theorem 3 *Any decision tree algorithm solving the element uniqueness problem for n elements from an ordered universe makes at least $\lceil \log n! \rceil = n \log n - O(n)$ comparisons in the worst case.*

2 Lower bounds for algebraic decision tree algorithms

In this section, we strengthen the lower bounds obtained so far. Until now, we assumed that the elements belong to an arbitrary ordered universe. The only operation allowed was the comparison of two input elements, and the complexity of an algorithm was equal to the worst-case number of comparisons made. Assume now that the elements are real numbers. Such elements can be added, multiplied, etc. Hence, the question arises whether we can use arithmetic to solve the sorting problem or the element uniqueness problem in $o(n \log n)$ time. For example, could it be that comparisons such as $s_1 s_5 + s_2^2 : s_4 - s_3$ lead to faster algorithms?

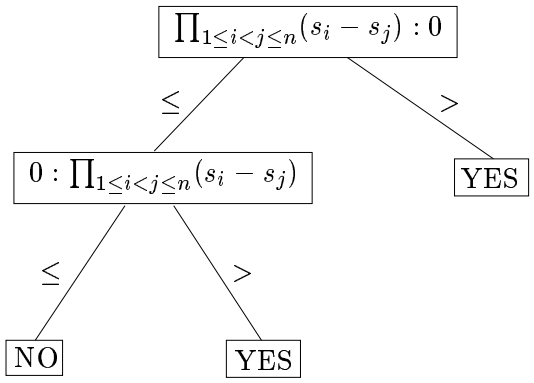

Figure 2: *Solving the element uniqueness problem by making at most two comparisons.*

If we allow arithmetic operations, then counting only the number of comparisons does not give a realistic measure for the running time of an algorithm. Consider for example the decision tree in Figure 2. The corresponding algorithm solves the element uniqueness problem on any input sequence $s_1, s_2, \ldots, s_n$, by making only two comparisons in the worst case. Clearly, since the algorithm has to compute the product $\prod_{1\leq i<j\leq n}(s_i - s_j)$, it is not realistic to say that its running time is bounded by a constant. That is, we have to count both the number of comparisons and arithmetic operations.

Hence, we have to generalize decision trees such that they also model the cost of arithmetic operations. We will do this for algorithms that use—besides comparisons—the addition ($+$), subtraction ($-$), multiplication ($*$), division ($/$), and square root operations. We assume that all these operations are performed exactly.

Definition 4 An *algebraic decision tree* for real inputs $s_1, s_2, \ldots, s_n$ is a finite tree T in which each node has at most two children, and that satisfies the following three conditions.

1. Each leaf is labeled with either YES or NO.
2. Each node u having one child is labeled with a variable $Z(u)$ and an assignment of the form
   (a) $Z(u) := A_1 \& A_2$, where $\& \in \{+, -, *, /\}$, or
   (b) $Z(u) := \sqrt{A_1}$,

where, for $i = 1, 2$, (i) $A_i = Z(u')$ for some proper ancestor u' of u in T, (ii) $A_i \in \{s_1, s_2, \ldots, s_n\}$, or (iii) A_i is a real number. In the latter case, this real number is a "constant" that is independent of the input sequence, although it may depend on n.

3. Each node u having two children is labeled with a comparison of the form $A : 0$, where (i) $A = Z(u')$ for some proper ancestor u' of u in T, or (ii) $A \in \{s_1, s_2, \ldots, s_n\}$. The two outgoing edges leading to the left and right child of u are labeled with $\leq$ and $>$, respectively.

An algebraic decision tree T corresponds to an algorithm $\mathcal{A}_T$ that computes a function $f : \mathbb{R}^n \to \{\text{YES,NO}\}$: Given an input sequence $s_1, s_2, \ldots, s_n$, this algorithm traverses a path down the tree T, starting in the root. If the current node u has one child, then $\mathcal{A}_T$ performs the corresponding arithmetic operation, assigns the result to the variable $Z(u)$, and proceeds to the child of u. If the current node u has two children, then the corresponding comparison is made and, depending on the outcome, $\mathcal{A}_T$ proceeds to the left or right child of u. If $\mathcal{A}_T$ reaches a leaf, say w, then it outputs the label (YES or NO) stored at w, and terminates. This label is the value $f(s_1, s_2, \ldots, s_n)$.

We require that no computation leads to a division by zero or taking the square root of a negative number. Also, for each leaf w of T, there is an input on which algorithm $\mathcal{A}_T$ terminates in w.

An algorithm $\mathcal{A}$ that only makes comparisons and the arithmetic operations $+, -, *, /$, and $\sqrt{\ }$, will be called an *algebraic decision tree algorithm*, if there is an algebraic decision tree that solves the same problem as $\mathcal{A}$. In Exercise 5, we will see that *not* every such algorithm is an *algebraic decision tree algorithm*.

The *complexity* of an algebraic decision tree algorithm is defined as the height of the corresponding tree. Hence, each comparison and each of the elementary operations $+, -, *, /$, and $\sqrt{\ }$ takes unit time.

Let $\mathcal{P}$ be a decision problem whose value, YES or NO, depends on n real numbers $s_1, s_2, \ldots, s_n$. We associate with $\mathcal{P}$ the subset $V_\mathcal{P}$ of $\mathbb{R}^n$ consisting of those points $(s_1, s_2, \ldots, s_n)$ for which $\mathcal{P}(s_1, s_2, \ldots, s_n)$ has value YES. Conversely, for any subset V of $\mathbb{R}^n$, there is a decision problem $\mathcal{P}$ for which $V_\mathcal{P} = V$. Hence, we can identify decision problems with subsets of $\mathbb{R}^n$. For example, if $\mathcal{P}$ is the element uniqueness problem for n elements, then

$$V_\mathcal{P} = \left\{ (s_1, s_2, \ldots, s_n) \in \mathbb{R}^n \mid \prod_{1 \leq i < j \leq n} (s_i - s_j) \neq 0 \right\}.$$

We say that a decision problem $\mathcal{P}$ is *decidable* in the algebraic decision tree model, if there is an algebraic decision tree that, when given an arbitrary

input $s_1, s_2, \ldots, s_n$, outputs YES if $\mathcal{P}(s_1, s_2, \ldots, s_n) = \text{YES}$, and outputs NO otherwise.

Definition 5 Let V be a subset of $\mathbb{R}^n$, and let $\mathcal{P}$ be the corresponding decision problem. If $\mathcal{P}$ is decidable in the algebraic decision tree model, then we define the *complexity* of V as the minimum height of any algebraic decision tree that outputs YES if its input is contained in V, and outputs NO otherwise.

In the rest of this section, we will show that the topological structure of a set $V \subseteq \mathbb{R}^n$ yields a lower bound on the complexity of V. Before we prove this lower bound for general algebraic decision trees, we consider a more restricted class of algorithms.

2.1 Linear decision trees

We consider algorithms that can only perform the following operations, each in unit time. First, input elements, values that have previously been computed, and constants can be added and subtracted. Second, any input element and previously computed value can be multiplied by a constant. As before, these constants do not depend on the input sequence, although they may depend on n. Finally, any input element and previously computed value can be compared with zero; the outcome is either "$\leq$" or "$>$".

Hence, each variable that occurs during the execution is a linear function of the input variables. An example is the function $4+3s_1+7s_2-s_5+2s_4-8s_3$. Therefore, we call these algorithms *linear decision tree algorithms*.

Consider a decision problem $\mathcal{P}$ that is decidable in this linear model, and let $V = V_{\mathcal{P}} \subseteq \mathbb{R}^n$ be the corresponding set of YES-inputs. Let T be any linear decision tree that decides $\mathcal{P}$. Then the algorithm $\mathcal{A}_T$ corresponding to T outputs YES if and only if it gets an element of V as input.

Let w be any leaf of T, and consider the path $u_1, u_2, \ldots, u_\ell$ from the root u_1 to $u_\ell = w$. Let $v_1, v_2, \ldots, v_k$ be the nodes on this path that have two children. In each node v_i, algorithm $\mathcal{A}_T$ makes a comparison, which can be written as

$$f_i(s_1, s_2, \ldots, s_n) : 0,$$

where f_i is a linear function of the n input variables $s_1, s_2, \ldots, s_n$. Let $R(w)$ be the set consisting of all points $(s_1, s_2, \ldots, s_n) \in \mathbb{R}^n$, such that for all i, $1 \leq i \leq k$,

1. $f_i(s_1, s_2, \ldots, s_n) \leq 0$ if the path in T to w proceeds from v_i to its left child,

2. $f_i(s_1, s_2, \ldots, s_n) > 0$ if the path in T to w proceeds from v_i to its right child.

Hence, $R(w)$ contains exactly those inputs on which algorithm $\mathcal{A}_T$ terminates in leaf w.

Lemma 4 *The set $R(w)$ is connected, i.e., for any two points P and Q in $R(w)$, there is a (continuous) curve in $\mathbb{R}^n$ between P and Q that is completely contained in $R(w)$.*

Proof. Each inequality $f_i(s_1, s_2, \ldots, s_n) : 0$ defines a closed or open halfspace in $\mathbb{R}^n$. The set $R(w)$ is the intersection of these halfspaces. Hence, since each halfspace is convex, $R(w)$ is also convex and, therefore, connected. ∎

We claim that we can obtain a lower bound on the complexity of problem $\mathcal{P}$ from the topological structure of the set V. This set V consists of one or more *connected components*. Let A and B be two different connected components of V, and let $P = (s_1, s_2, \ldots, s_n)$ and $Q = (t_1, t_2, \ldots, t_n)$ be points in A and B, respectively. Let w_P be the leaf of T in which algorithm $\mathcal{A}_T$ terminates on input P. Define w_Q similarly w.r.t. input Q.

Lemma 5 *The leaves w_P and w_Q are distinct.*

Proof. Assume that $w_P = w_Q$, and denote this leaf by w. Note that P and Q are both contained in the set $R(w)$. We saw in the proof of Lemma 4 that $R(w)$ is convex. Therefore, the line segment PQ is completely contained in $R(w)$. Hence, for each point $X = (x_1, x_2, \ldots, x_n)$ that is on PQ, algorithm $\mathcal{A}_T$, when given X as input, terminates in w. Since $P \in V$, leaf w is labeled with YES. This proves that each point X on PQ belongs to the set V. As a result, the points P and Q are connected by a curve—the line segment PQ—that is completely inside V. This is a contradiction, because these points are in different connected components of V. ∎

This lemma immediately implies the following lower bound, which was first proved by Dobkin and Lipton [3]. For any subset V of $\mathbb{R}^n$, we denote the number of its connected components by $CC(V)$.

Theorem 4 (Dobkin, Lipton) *Let $\mathcal{P}$ be a decision problem that is decidable in the linear decision tree model, and let $V_{\mathcal{P}} \subseteq \mathbb{R}^n$ be the corresponding set of YES-inputs. The complexity of any linear decision tree algorithm that decides the set $V_{\mathcal{P}}$ is at least equal to* $\log CC(V_{\mathcal{P}})$.

Proof. Let T be any linear decision tree that decides $V_{\mathcal{P}}$. Then, by Lemma 5, T has at least $CC(V_{\mathcal{P}})$ leaves. Hence, this tree has height at least $\log CC(V_{\mathcal{P}})$. ∎

Let us apply this result to the element uniqueness problem, in which case we have

$$V_{\mathcal{P}} = \left\{ (s_1, s_2, \ldots, s_n) \in \mathbb{R}^n \mid \prod_{1 \le i < j \le n} (s_i - s_j) \neq 0 \right\}.$$

In order to get a lower bound on this problem for linear decision tree algorithms, we need to estimate the number of connected components of $V_{\mathcal{P}}$.

Let π and ρ be two different permutations of $1, 2, \ldots, n$, and consider the points $P := (\pi(1), \pi(2), \ldots, \pi(n))$ and $R := (\rho(1), \rho(2), \ldots, \rho(n))$ in $\mathbb{R}^n$. Clearly, both P and R belong to the set $V_{\mathcal{P}}$. We will show that these two points belong to different connected components of $V_{\mathcal{P}}$. This will prove that $CC(V_{\mathcal{P}}) \ge n!$.

Because π and ρ are distinct, there are indices i and j such that $\pi(i) < \pi(j)$ and $\rho(i) > \rho(j)$. (The proof is similar to that of Lemma 1.) Hence the points P and R are on different sides of the hyperplane $x_i = x_j$. Any curve in $\mathbb{R}^n$ between P and R must pass through this hyperplane. That is, this curve contains a point $Q := (q_1, q_2, \ldots, q_n)$ for which $q_i = q_j$. As a result, this curve is not completely contained in the set $V_{\mathcal{P}}$. Therefore, P and R belong to different connected components of $V_{\mathcal{P}}$. Theorem 4 immediately implies the following lower bound.

Theorem 5 *The complexity of any linear decision tree algorithm solving the element uniqueness problem for n real numbers is at least equal to $\lceil \log n! \rceil = n \log n - O(n)$.*

This lower bound is stronger than that of Theorem 3. It says that the arithmetic operations of addition and subtraction, and multiplication by constants do not help to obtain an $o(n \log n)$-time algorithm for the element uniqueness problem. It does not rule out, however, $o(n \log n)$-time algorithms that can multiply, divide and take square roots. In the next section, we will show that this still does not help.

Here is another reason why linear decision tree algorithms are not general enough. Geometric algorithms often make comparisons whose outcomes are decided by multiplying input variables. An example is the test if a planar point c is to the left, right, or on the directed line through the points a and b.

This can be decided by computing the sign of the determinant

$$\begin{vmatrix} a_1 & a_2 & 1 \\ b_1 & b_2 & 1 \\ c_1 & c_2 & 1 \end{vmatrix},$$

which is a polynomial in the input variables, having degree two. Hence, in order to obtain meaningful lower bounds for geometric problems, we have to consider the more general algebraic decision tree model.

2.2 The general lower bound

The proof of Theorem 4 heavily depends on the fact that the set $R(w)$ associated with a leaf w is connected. For an algebraic decision tree, this set is in general not connected. For example, if $n = 2$, then $R(w)$ may be defined by the two inequalities $s_1^2 + s_2^2 - 1 \leq 0$ and $-8s_1^2 + s_2 + 2 \leq 0$. That is, $R(w)$ is the set of those points that are inside the unit circle, and below the parabola $s_2 = 8s_1^2 - 2$. This set clearly consists of two connected components.

In this section, we will prove that the arguments of Section 2.1 can nevertheless be generalized. As we will see, the number of connected components of the set $V_{\mathcal{P}}$ of YES-inputs still gives a lower bound on the complexity of the (decidable) decision problem $\mathcal{P}$. The proof will be based on the following result from algebraic topology, which was proved independently by Milnor [5] and Thom [8].

Theorem 6 (Milnor, Thom) *Let k be a positive integer, and let $f_1, f_2, \ldots, f_k$ be polynomials in N variables, each having degree at most g. Let*

$$W := \{(x_1, x_2, \ldots, x_N) \in \mathbb{R}^N \,|\, f_i(x_1, x_2, \ldots, x_N) = 0 \text{ for all } i,\ 1 \leq i \leq k\}.$$

The set W has at most $g(2g-1)^{N-1}$ connected components.

The proof of Theorem 6 is beyond the scope of this course.

Exercise 3 Prove Theorem 6 for $N = 1$.

Consider an algebraic decision tree T, and let w be a leaf of T. Later in this section, we will show that the set $R(w)$ of all inputs on which algorithm $\mathcal{A}_T$ terminates in w can be described by a system of polynomial equations and inequalities, each having degree at most two. Our goal is to derive an upper bound on the number of connected components of $R(w)$. This will be done by transforming the system of equations and inequalities into a system containing polynomial equations only, and then applying Theorem 6. The details of this transformation are given in the following theorem, which is due to Ben-Or [1].

Theorem 7 *Let a, b and c be non-negative integers, and let $f_1, \ldots, f_a$, $p_1, \ldots, p_b$, $q_1, \ldots, q_c$ be polynomials in N variables, each having degree at most two. Let W be the set of all points $(x_1, \ldots, x_N)$ in $\mathbb{R}^N$ such that*

1. *$f_i(x_1, \ldots, x_N) = 0$, for all i, $1 \leq i \leq a$,*
2. *$p_i(x_1, \ldots, x_N) \leq 0$, for all i, $1 \leq i \leq b$, and*
3. *$q_i(x_1, \ldots, x_N) > 0$, for all i, $1 \leq i \leq c$.*

The set W has at most 3^{N+b+c} connected components.

Proof. We first remark that the number of connected components of W is finite. (See Milnor [6].) Let $d := CC(W)$. For each j, $1 \leq j \leq d$, let $P_j \in \mathbb{R}^N$ be an arbitrary point in the j-th connected component of W. Define

$$\epsilon := \min\{q_i(P_j) | 1 \leq i \leq c, 1 \leq j \leq d\}.$$

Then $\epsilon > 0$. Let W_ϵ be the set of all points $(x_1, \ldots, x_N) \in \mathbb{R}^N$ such that

- $f_i(x_1, \ldots, x_N) = 0$, for all i, $1 \leq i \leq a$,
- $p_i(x_1, \ldots, x_N) \leq 0$, for all i, $1 \leq i \leq b$, and
- $q_i(x_1, \ldots, x_N) \geq \epsilon$, for all i, $1 \leq i \leq c$.

Then $W_\epsilon \subseteq W$, and W_ϵ contains the points $P_1, P_2, \ldots, P_d$.

We transform the equations and inequalities that define W_ϵ into a system of polynomial equations by introducing $b+c$ new variables $x_{N+1}, \ldots, x_{N+b+c}$: Let W' be the set of all points $(x_1, \ldots, x_{N+b+c})$ in $\mathbb{R}^{N+b+c}$ such that

- $f_i(x_1, \ldots, x_N) = 0$, for all i, $1 \leq i \leq a$,
- $p_i(x_1, \ldots, x_N) + x_{N+i}^2 = 0$, for all i, $1 \leq i \leq b$, and
- $q_i(x_1, \ldots, x_N) - x_{N+b+i}^2 - \epsilon = 0$, for all i, $1 \leq i \leq c$.

The projection of W' onto the first N coordinates is exactly the set W_ϵ, i.e.,

$$W_\epsilon = \{(x_1, \ldots, x_N) | \exists x_{N+1}, \ldots, x_{N+b+c} \in \mathbb{R} : (x_1, \ldots, x_{N+b+c}) \in W'\}.$$

For each j, $1 \leq j \leq d$, let P'_j be a point in W' such that its projection onto the first N coordinates is the point P_j. Since the points $P_1, P_2, \ldots, P_d$ are in pairwise distinct connected components of W, and since $W_\epsilon \subseteq W$, it follows that the points $P'_1, P'_2, \ldots, P'_d$ are in pairwise distinct connected components of W'. Hence, $CC(W') \geq d$.

The set W' is defined by polynomial equations in $N+b+c$ variables, each having degree at most two. Therefore, by Theorem 6, we have

$$CC(W') \leq 2 \cdot 3^{N+b+c-1} \leq 3^{N+b+c}.$$

This completes the proof. ∎

Now we are ready to prove the lower bound for algebraic decision tree algorithms.

Theorem 8 (Ben-Or [1]) *Let $\mathcal{P}$ be a decision problem that is decidable in the algebraic decision tree model, and let $V_{\mathcal{P}} \subseteq \mathbb{R}^n$ be the corresponding set of YES-inputs. The complexity of any algebraic decision tree algorithm that decides the set $V_{\mathcal{P}}$ is at least equal to*

$$\frac{\log CC(V_{\mathcal{P}}) - n \log 3}{1 + 2 \log 3}.$$

Proof. Let T be an arbitrary algebraic decision tree that decides $V_{\mathcal{P}}$, and let w be a leaf of T. Let $R(w)$ be the set of all points $(s_1, s_2, \ldots, s_n) \in \mathbb{R}^n$ such that the computation in T on input $s_1, s_2, \ldots, s_n$ terminates in w. We will derive an upper bound on the number of connected components of $R(w)$.

Consider the path $u_1, u_2, \ldots, u_{k+1}$ in T from the root u_1 to $u_{k+1} = w$. Let r be the number of nodes on this path that have exactly one child, and let s be the number of such nodes that are labeled with a $\sqrt{\ }$-assignment. We will define a system of $k + s$ polynomial equations and inequalities in the variables $x_1, \ldots, x_{n+k}$. The variables $x_1, \ldots, x_n$ represent the input variables $s_1, \ldots, s_n$, whereas the variables $x_{n+1}, \ldots, x_{n+k}$ represent the program variables $Z(u_1), \ldots, Z(u_k)$.

Let $1 \leq i \leq k$, and consider node u_i.

Case 1: u_i has one child, i.e., u_i is a computation node.

Node u_i is labeled with an assignment of the form $Z(u_i) := A_1 \& A_2$ or $Z(u_i) := \sqrt{A_1}$, where $\& \in \{+, -, *, /\}$, and, for $m = 1, 2$, (i) $A_m = Z(u_j)$ for some index j, $1 \leq j < i$, (ii) $A_m \in \{s_1, s_2, \ldots, s_n\}$, or (iii) $A_m = c$ for some real constant c. (See Definition 4.)

Depending on the form of this assignment, we add either one equation, or one equation and one $\leq$-inequality to our system. In Table 1, all possibilities are listed. For example, if the assignment is $Z(u_i) := s_a / Z(u_\ell)$, then we add the equation $x_{n+i} x_{n+\ell} - x_a = 0$. Here, x_{n+i} represents the program variable $Z(u_i)$; $x_{n+\ell}$ represents $Z(u_\ell)$; and x_a represents the input variable s_a. If the assignment is $Z(u_i) := \sqrt{s_a}$, then we add the equation $x_{n+i}^2 - x_a = 0$ and the inequality $-x_{n+i} \leq 0$. (Note that $x_{n+i} = \sqrt{x_a}$ if and only if $x_{n+i}^2 = x_a$ and $x_{n+i} \geq 0$.)

assignment	equation
$Z(u_i) := Z(u_j) + Z(u_\ell)$	$x_{n+i} - x_{n+j} - x_{n+\ell} = 0$
$Z(u_i) := Z(u_j) - Z(u_\ell)$	$x_{n+i} - x_{n+j} + x_{n+\ell} = 0$
$Z(u_i) := Z(u_j) * Z(u_\ell)$	$x_{n+i} - x_{n+j}x_{n+\ell} = 0$
$Z(u_i) := Z(u_j)/Z(u_\ell)$	$x_{n+i}x_{n+\ell} - x_{n+j} = 0$
$Z(u_i) := \sqrt{Z(u_j)}$	$x_{n+i}^2 - x_{n+j} = 0$ and $-x_{n+i} \leq 0$
$Z(u_i) := s_a + Z(u_\ell)$	$x_{n+i} - x_a - x_{n+\ell} = 0$
$Z(u_i) := s_a - Z(u_\ell)$	$x_{n+i} - x_a + x_{n+\ell} = 0$
$Z(u_i) := Z(u_\ell) - s_a$	$x_{n+i} - x_{n+\ell} + x_a = 0$
$Z(u_i) := s_a * Z(u_\ell)$	$x_{n+i} - x_a x_{n+\ell} = 0$
$Z(u_i) := s_a/Z(u_\ell)$	$x_{n+i}x_{n+\ell} - x_a = 0$
$Z(u_i) := Z(u_\ell)/s_a$	$x_{n+i}x_a - x_{n+\ell} = 0$
$Z(u_i) := s_a + s_b$	$x_{n+i} - x_a - x_b = 0$
$Z(u_i) := s_a - s_b$	$x_{n+i} - x_a + x_b = 0$
$Z(u_i) := s_a * s_b$	$x_{n+i} - x_a x_b = 0$
$Z(u_i) := s_a/s_b$	$x_{n+i}x_b - x_a = 0$
$Z(u_i) := \sqrt{s_a}$	$x_{n+i}^2 - x_a = 0$ and $-x_{n+i} \leq 0$
$Z(u_i) := c + Z(u_\ell)$	$x_{n+i} - c - x_{n+\ell} = 0$
$Z(u_i) := c - Z(u_\ell)$	$x_{n+i} - c + x_{n+\ell} = 0$
$Z(u_i) := Z(u_\ell) - c$	$x_{n+i} - x_{n+\ell} + c = 0$
$Z(u_i) := c * Z(u_\ell)$	$x_{n+i} - cx_{n+\ell} = 0$
$Z(u_i) := c/Z(u_\ell)$	$x_{n+i}x_{n+\ell} - c = 0$
$Z(u_i) := Z(u_\ell)/c$	$cx_{n+i} - x_{n+\ell} = 0$
$Z(u_i) := c + s_b$	$x_{n+i} - c - x_b = 0$
$Z(u_i) := c - s_b$	$x_{n+i} - c + x_b = 0$
$Z(u_i) := s_b - c$	$x_{n+i} - x_b + c = 0$
$Z(u_i) := c * s_b$	$x_{n+i} - cx_b = 0$
$Z(u_i) := c/s_b$	$x_{n+i}x_b - c = 0$
$Z(u_i) := s_b/c$	$cx_{n+i} - x_b = 0$
$Z(u_i) := c + d$	$x_{n+i} - c + d = 0$
$Z(u_i) := c - d$	$x_{n+i} - c + d = 0$
$Z(u_i) := c * d$	$x_{n+i} - cd = 0$
$Z(u_i) := c/d$	$dx_{n+i} - c = 0$
$Z(u_i) := \sqrt{c}$	$x_{n+i}^2 - c = 0$ and $-x_{n+i} \leq 0$

Table 1: *The equations corresponding to all possible assignments of computation node u_i. The indices j and ℓ satisfy $1 \leq j < i$ and $1 \leq \ell < i$; the indices a and b satisfy $1 \leq a \leq n$ and $1 \leq b \leq n$; c and d are real constants.*

Case 2: u_i has two children, i.e., u_i is a comparison node.

Node u_i is labeled with a comparison of the form $A : 0$, where (i) $A = Z(u_j)$ for some index j, $1 \leq j < i$, or (ii) $A \in \{s_1, s_2, \ldots, s_n\}$.

In case the path in T to w proceeds from u_i to its left child, we do the following. If the comparison in u_i has the form $Z(u_j) : 0$, then we add the inequality $x_{n+j} \leq 0$ to our system. Otherwise, the comparison in u_i has the form $s_a : 0$, in which case we add the inequality $x_a \leq 0$.

In case the path to w proceeds from u_i to its right child, we do the following. If the comparison in u_i has the form $Z(u_j) : 0$, then we add the inequality $x_{n+j} > 0$. Otherwise, the comparison in u_i has the form $s_a : 0$, in which case we add the inequality $x_a > 0$.

Recall that r denotes the number of computation nodes on the path to w, and s denotes the number of these nodes that are labeled with a $\sqrt{\ }$-assignment. Let t be the number of times this path proceeds from a comparison node to its left child. Then we have obtained a system of r polynomial equations, $s+t$ polynomial $\leq$-inequalities, and $k-r-t$ polynomial $>$-inequalities, in the variables $x_1, \ldots, x_{n+k}$. Each of these polynomials has degree at most two. Let $W \subseteq \mathbb{R}^{n+k}$ be the set of all points that satisfy these equations and inequalities. Then, by Theorem 7, W has at most $3^{n+2k+s-r}$ connected components.

The projection of W onto the first n coordinates is the set $R(w)$. This implies that $CC(R(w)) \leq CC(W)$ and, hence, $CC(R(w)) \leq 3^{n+2k+s-r}$. Let h be the height of our algebraic decision tree T. Then, since $k \leq h$ and $s \leq r$, we have proved that $CC(R(w)) \leq 3^{n+2h}$.

Now we can complete the proof of the theorem. Recall that $V_{\mathcal{P}} \subseteq \mathbb{R}^n$ is the set of YES-inputs for the decision problem $\mathcal{P}$. Since

$$V_{\mathcal{P}} = \bigcup_{w:\text{YES-leaf of } T} R(w),$$

we have

$$CC(V_{\mathcal{P}}) \leq \sum_{w:\text{YES-leaf of } T} CC(R(w)).$$

Hence, the number of connected components of $V_{\mathcal{P}}$ is less than or equal to 3^{n+2h} times the number of YES-leaves of T. Since T has height h, it has at most 2^h leaves. Therefore,

$$CC(V_{\mathcal{P}}) \leq 3^{n+2h} \cdot 2^h.$$

Taking logarithms, and rewriting the inequality, we obtain

$$h \geq \frac{\log CC(V_{\mathcal{P}}) - n \log 3}{1 + 2 \log 3},$$

which is exactly what we wanted to show. ∎

Remark 2 Let $\mathcal{P}$ be a decision problem that is decidable in the algebraic decision tree model, and let $V \subseteq \mathbb{R}^n$ be the corresponding set of YES-inputs. It follows from the proof of Theorem 8 that the number of connected components of V is finite.

2.3 Applications of Ben-Or's theorem

First, let us again consider the element uniqueness problem. As we have seen already in Section 2.1, the corresponding subset $V_{\mathcal{P}}$ of $\mathbb{R}^n$ has at least $n!$ connected components. Hence, Theorem 8 gives a lower bound of

$$\frac{\log n! - n \log 3}{1 + 2 \log 3} = \Omega(n \log n)$$

on the complexity of this problem.

Theorem 9 *Any algebraic decision tree algorithm that solves the element uniqueness problem for n real numbers has complexity $\Omega(n \log n)$.*

Using simple reductions, this theorem implies several other lower bounds.

Corollary 1 *Each of the following problems has complexity $\Omega(n \log n)$ in the algebraic decision tree model.*

1. *The sorting problem for n real numbers.*
2. *The closest pair problem for n points in $\mathbb{R}^d$, for any $d \geq 1$.*
3. *Constructing the Voronoi diagram of n points in the plane.*

Proof. Let $\mathcal{A}$ be any algebraic decision tree algorithm solving the sorting problem, and let $T(n)$ be its complexity. The following algorithm $\mathcal{B}$ solves the element uniqueness problem: Given real numbers $s_1, s_2, \ldots, s_n$, first use algorithm $\mathcal{A}$ to sort them. Then compare any two elements that are neighbors in the sorted sequence. Output YES, if no two equal elements are encountered; otherwise, output NO. Algorithm $\mathcal{B}$ has complexity $T(n) + O(n)$, which, by Theorem 9, must be $\Omega(n \log n)$. It follows that $T(n) = \Omega(n \log n)$.

The lower bound for the closest pair problem follows immediately from Theorem 9, because the input sequence contains two equal elements if and only if the distance of the closest pair is zero.

Finally, let $\mathcal{C}$ be an arbitrary algebraic decision tree algorithm that constructs the Voronoi diagram of any set of n points in the plane. Given such

a set S, we can first use $\mathcal{C}$ to construct the Voronoi diagram of S, and then use this diagram to find a closest pair in S in $O(n)$ time. Hence, algorithm $\mathcal{C}$ has complexity $\Omega(n \log n)$. ∎

Exercise 4 Prove that, in the algebraic decision tree model, the following problems have complexity $\Omega(n \log n)$:

1. Constructing the convex hull of a set of n points in the plane. The convex hull vertices should be reported in clockwise (or counter clockwise) order.

2. Constructing an arbitrary triangulation of a set of n points in the plane.

This exercise shows that the complexity of computing the *ordered* sequence of convex hull vertices is $\Omega(n \log n)$. What is the complexity if we are satisfied with just the vertices of the convex hull, in no particular order? We will use Theorem 8 to prove that the complexity is still $\Omega(n \log n)$. Note that in order to use this theorem, we need a decision problem. Therefore, we first prove the following result. Recall that a point that is in the interior of a convex hull edge is not a vertex of the convex hull.

Theorem 10 *Any algebraic decision tree algorithm that, when given a set of n points in the plane, decides whether they are all vertices of the convex hull, has complexity $\Omega(n \log n)$.*

Proof. Let V be the set of all points $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n) \in \mathbb{R}^{2n}$ such that the convex hull of the n planar points (x_i, y_i), $1 \leq i \leq n$, has n vertices. We claim that V has at least $(n-1)!$ connected components. Hence, by Theorem 8, the complexity of any algebraic decision tree algorithm that decides V is at least equal to

$$\frac{\log(n-1)! - 2n \log 3}{1 + 2 \log 3} = \Omega(n \log n),$$

which is exactly the statement of the theorem.

To prove the claim, let $P = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n) \in \mathbb{R}^{2n}$ be a point of V, and let z be a point (in $\mathbb{R}^2$) in the interior of the convex hull of the points (x_i, y_i), $1 \leq i \leq n$. The circular order of the indices of the points (x_i, y_i), sorted in counter clockwise order around z, is called the *order type* of P. Since the first element of an order type is arbitrary, there are $(n-1)!$ different order types.

Any (continuous) curve in $\mathbb{R}^{2n}$ that connects two points P and Q of V that have different order types must pass through a point in $\mathbb{R}^{2n}$ that does

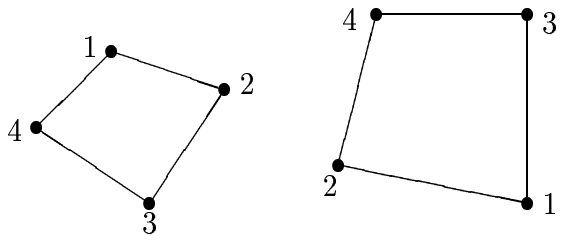

Figure 3: *Illustrating the proof of Theorem 10 for $n = 4$.*

---

not belong to V. Hence, P and Q are in different connected components of V. This proves that $CC(V) \geq (n-1)!$.

As an example, let $n = 4$, and consider the two "elements" of V in Figure 3. In this figure, the point (x_i, y_i) has label i, $1 \leq i \leq 4$. The order type of the left element is $(1, 4, 3, 2)$; for the right element, it is $(1, 3, 4, 2)$.

Consider a continuous movement of the left "point" (considered as an element of $\mathbb{R}^8$) to the right "point". In Figure 3, this means that for each i, $1 \leq i \leq 4$, point i on the left moves continuously to point i on the right. During this movement, the points 3 and 4 must change their relative order. This is only possible by moving through a configuration in which at least three of the four points are colinear. In this configuration, the convex hull of these points has size less than four. Hence, this configuration corresponds to a point in $\mathbb{R}^8$ that is not contained in the set V. ■

Corollary 2 *Any algebraic decision tree algorithm that, when given a set of n points in the plane, computes the vertices of their convex hull (in no particular order) has complexity $\Omega(n \log n)$.*

2.4 Some problems that are not decidable in the algebraic decision tree model

Let $\mathcal{P}$ be a decision problem, and let $V \subseteq \mathbb{R}^n$ be the corresponding set of YES-inputs. If V has an infinite number of connected components, then it follows from Remark 2 that $\mathcal{P}$ is not decidable in the algebraic decision tree model. Hence, by taking $V := \mathbb{N}$, we obtain the following result:

Theorem 11 *There is no algebraic decision tree algorithm that, when given an arbitrary real number x as input, decides if $x \in \mathbb{N}$.*

Exercise 5 In Figure 4, an algorithm is given that gets an arbitrary real number x as input, and outputs YES if and only if $x \in \mathbb{N}$. Does this contradict Theorem 11?

```
Algorithm(x):
(* x is a real number *)
if x < 0
then output NO
else k := 0;
     while k ≤ x
     do k := k + 1
     endwhile;
     ℓ := k − 1;
     if x > ℓ
     then output NO
     else output YES
     endif
endif
```

Figure 4: *An algorithm that decides the set* $\mathbb{N}$.

Exercise 6 Prove that there is no algebraic decision tree algorithm that, on an arbitrary input $x \in \mathbb{R}$, computes the value $\lfloor x \rfloor$. (Hint: Use Theorem 11.)

If a set $V \subseteq \mathbb{R}^n$ has an infinite number of connected components, then V is not decidable in the algebraic decision tree model. Is every set V with a finite number of connected components decidable in this model? The answer is "no": Let $n = 2$, and

$$V := \{(x, y) \in \mathbb{R}^2 \,|\, y \leq \sin x\}.$$

Then V consists of one connected component. Since

$$\{x \in \mathbb{R} \,|\, \sin \pi x = 0 \text{ and } x \geq 0\} = \mathbb{N},$$

Theorem 11 implies the following result:

Theorem 12 *There is no algebraic decision tree algorithm that, when given two arbitrary real numbers x and y as input, decides if $y \leq \sin x$.*

2.5 Another example: computing an approximate minimum weight matching

Let S be a set of $2n$ points in $\mathbb{R}^d$, where $d \geq 1$ is a (small) constant. We consider sets of edges having the points of S as vertices. Such a set M is

called a *perfect matching* of S, if each point of S is a vertex of exactly one edge in M. In other words, a perfect matching is a partition of S into n subsets of size two. The *weight* $wt(M)$ of a perfect matching M is defined as the sum of the Euclidean lengths of all edges in M. The *minimum weight matching* $MWM(S)$ of S is the perfect matching of S that has minimum weight.

The best known algorithm that computes a minimum weight matching is due to Vaidya [9]; its running time is bounded by $O(n^{5/2}(\log n)^4)$ if $d = 2$, and $O(n^{3-1/c^d})$ if $d > 2$, for some constant $c > 1$.

Exercise 7 Prove that the minimum weight matching of a set of $2n$ real numbers—i.e., one-dimensional points—can be computed in $O(n \log n)$ time.

Let us consider the (hopefully) easier problem of approximating the minimum weight matching. Let $r > 1$ be a real number. A perfect matching M of S is called an *r-approximate MWM*, if $wt(M) \leq r \cdot wt(MWM(S))$.

Rao and Smith [7] have shown that an r-approximate MWM, for

$$r = c \cdot \exp(8 \cdot 2^{1-1/(d-1)}\sqrt{d})$$

where c is a constant, can be computed in $O(n \log n)$ time.

We will show that Rao and Smith's algorithm is optimal in the algebraic decision tree model: Any algorithm in this model that, when given an arbitrary set S of $2n$ points in $\mathbb{R}^d$ and an arbitrary real number $r > 1$, computes an r-approximate MWM for S, has running time $\Omega(n \log n)$. In fact, this lower bound even holds for dimension $d = 1$. Moreover, it holds for *any* approximation factor r, even one that depends on n. For example, computing a 2^{2^n}-approximate MWM has complexity $\Omega(n \log n)$.

In the rest of this section, we will prove the lower bound for the one-dimensional case. Clearly, this implies the same lower bound for any dimension $d \geq 1$.

Let $\mathcal{A}$ be an arbitrary algebraic decision tree algorithm that, when given as input a sequence of $2n$ real numbers $x_1, x_2, \ldots, x_{2n}$ and a real number $r > 1$, computes an r-approximate MWM for the x_i's. We will use Theorem 8 to prove that $\mathcal{A}$ has complexity $\Omega(n \log n)$.

Note that algorithm $\mathcal{A}$ solves a computation problem. In order to apply Theorem 8, we need a decision problem, i.e., a problem having values YES and NO. Below, we will define such a decision problem; in fact, we will define the corresponding subset $V \subseteq \mathbb{R}^{2n}$ of YES-inputs.

Fix the integer n and the real number $r > 1$. We define an algorithm $\mathcal{B}$ that takes as input any sequence of $2n$ real numbers. On input sequence $x_1, x_2, \ldots, x_{2n}$, algorithm $\mathcal{B}$ does the following.

Step 1. Check if $x_i = i$, for all i, $1 \leq i \leq n$. If not, output NO, and terminate. Otherwise, go to Step 2.

Step 2. Let $\epsilon := 1/(2rn)$. Run algorithm $\mathcal{A}$ on the input $x_1, x_2, \ldots, x_{2n}, r$. Let M be the r-approximate *MWM* that is computed by $\mathcal{A}$. Check if all edges of M have length ϵ. If so, output YES. Otherwise, output NO.

Let $T_{\mathcal{A}}(n)$ and $T_{\mathcal{B}}(n)$ denote the complexities of algorithms $\mathcal{A}$ and $\mathcal{B}$, respectively. Then, it is clear that

$$T_{\mathcal{B}}(n) \leq T_{\mathcal{A}}(n) + cn,$$

for some constant c. Therefore, if we can show that $\mathcal{B}$ has complexity $\Omega(n \log n)$, then it follows immediately that $\mathcal{A}$ has complexity $\Omega(n \log n)$ as well.

Let V be the set of all points $(x_1, x_2, \ldots, x_{2n})$ in $\mathbb{R}^{2n}$ that are accepted by algorithm $\mathcal{B}$. We will show that V has at least $n!$ connected components. As a result, Theorem 8 implies the $\Omega(n \log n)$ lower bound on the complexity of $\mathcal{B}$ and, hence, on the complexity of $\mathcal{A}$.

Lemma 6 *Let π be any permutation of $1, 2, \ldots, n$, and let $\epsilon = 1/(2rn)$. Then the point*

$$P := (1, 2, \ldots, n, \pi(1) + \epsilon, \pi(2) + \epsilon, \ldots, \pi(n) + \epsilon)$$

is contained in the set V.

Proof. Let M^* be the *MWM* of the elements $1, 2, \ldots, n, \pi(1) + \epsilon, \pi(2) + \epsilon, \ldots, \pi(n) + \epsilon$. Since $0 < \epsilon < 1/2$, it is easy to see that M^* consists of the edges $(i, i + \epsilon)$, $1 \leq i \leq n$.

Consider what happens when algorithm $\mathcal{B}$ is run on input P. Clearly, this input "survives" Step 1. Let M be the r-approximate *MWM* that is computed in Step 2. We will show below that $M = M^*$. Having proved this, it follows that algorithm $\mathcal{B}$ accepts the input P, i.e., $P \in V$.

Suppose that $M \neq M^*$. Then M contains an edge of the form (i, j), $(i, j + \epsilon)$, or $(i + \epsilon, j + \epsilon)$, for some integers i and j, $i \neq j$. (We consider edges to be undirected.) Since $0 < \epsilon < 1/2$, it follows that this edge and, hence, also the matching M, has weight more than $1/2$. Clearly, the optimal matching M^* has weight $n\epsilon = 1/(2r)$. Therefore, $wt(M) > 1/2 = r \cdot wt(M^*)$. This is a contradiction, because M is an r-approximate *MWM*. ∎

Lemma 7 *The set V has at least $n!$ connected components.*

Proof. Let π and ρ be two different permutations of $1, 2, \ldots, n$. Consider the points

$$P := (1, 2, \ldots, n, \pi(1) + \epsilon, \pi(2) + \epsilon, \ldots, \pi(n) + \epsilon)$$

and

$$R := (1, 2, \ldots, n, \rho(1) + \epsilon, \rho(2) + \epsilon, \ldots, \rho(n) + \epsilon).$$

in $\mathbb{R}^{2n}$. By Lemma 6, both these points are contained in the set V. We will show that they are in different connected components of V.

Let C be an arbitrary curve in $\mathbb{R}^{2n}$ that connects P and R. Since π and ρ are distinct permutations, there are indices i and j such that $\pi(i) < \pi(j)$ and $\rho(i) > \rho(j)$. Hence, the curve C contains a point Q,

$$Q = (p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_n),$$

such that $q_i = q_j$. We claim that Q is not contained in V. This will prove that P and R are in different connected components of V.

To prove the claim, first assume that there is an index k, $1 \leq k \leq n$, such that $p_k \neq k$. Then point Q is rejected by algorithm $\mathcal{B}$ and, therefore, $Q \notin V$. Hence, we may assume that

$$Q = (1, 2, \ldots, n, q_1, q_2, \ldots, q_n).$$

Let us see what happens if we run algorithm $\mathcal{B}$ on input Q. This input "survives" Step 1. Let M be the r-approximate MWM that is constructed in Step 2.

If M contains an edge of the form $(p_k, p_\ell) = (k, \ell)$, then algorithm $\mathcal{B}$ rejects point Q, because such an edge has length more than ϵ. Hence, we may assume that each edge of M has the form $(p_k, q_\ell) = (k, q_\ell)$. Let a and b be the integers, $1 \leq a, b \leq n$, such that (a, q_i) and (b, q_j) are edges of M. Since (i) a and b are distinct integers, (ii) $q_i = q_j$, and (iii) $0 < \epsilon < 1/2$, one of these two edges must have length more than ϵ. Hence, algorithm $\mathcal{B}$ rejects point Q. This completes the proof. ∎

We summarize the result of this section.

Theorem 13 *Let $d \geq 1$ be an integer. Any algebraic decision tree algorithm that, when given a set of $2n$ points in $\mathbb{R}^d$ and a real number $r > 1$, computes an r-approximate MWM, has complexity $\Omega(n \log n)$.*

3 Final remarks

In the algebraic decision tree model, square roots can be computed (exactly) in unit time. Theorem 8 also holds, if k-th roots can be computed in unit time, where k is an element of a finite set of positive integers. (The size of this set may not depend on n.)

In Section 2.1, we have considered the linear decision tree model. I do not think that in this model, the convex hull of a set of planar points can be computed, although I do not know how to prove this. How do we answer a question like "given three points in $\mathbb{R}^2$, are they colinear?", in this linear model?

A wealth of information about algebraic algorithms can be found in the book by Bürgisser, Clausen, and Shokrollahi [2].

The lower bound proofs of this chapter are *not* valid for algorithms that use indirect addressing. See Section II.3 in the book by Mehlhorn [4].

We have only considered lower bounds for deterministic algorithms. Note that quicksort is a *randomized* sorting algorithm. See Section II.1.6 in [4] for lower bounds on the expected running time of randomized algorithms.

References

[1] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[2] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory.* Springer-Verlag, Berlin, 1996.

[3] D. P. Dobkin and R. J. Lipton. On the complexity of computations under varying sets of primitives. *J. Comput. Syst. Sci.*, 18:86–91, 1979.

[4] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Springer-Verlag, Heidelberg, Germany, 1984.

[5] J. W. Milnor. On the Betti numbers of real algebraic varieties. *Proc. Amer. Math. Soc.*, 15:275–280, 1964.

[6] J. W. Milnor. *Singular Points of Complex Hypersurfaces.* Princeton University Press, Princeton, NJ, 1968.

[7] S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 540–550, 1998.

[8] R. Thom. Sur l'homologie des variétés algébriques reélles. In S. S. Cairns, editor, *Differential and Combinatorial Topology*, pages 255–265. Princeton University Press, Princeton, NJ, 1965.

[9] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18:1201–1225, 1989.