CMSC 754: Lecture 3
Convex Hulls: Lower Bounds and Output Sensitivity

Reading: Chan's output sensitive algorithm can be found in T. Chan, "Optimal output-sensitive convex hull algorithms in two and three dimensions", *Discrete and Computational Geometry*, 16, 1996, 361–368.

Lower Bound and Output Sensitivity: Last time we presented two planar convex hull algorithms, Graham's scan and the divide-and-conquer algorithm, both of which run in $O(n \log n)$ time. A natural question to consider is whether we can do better. Today, we will consider this question.

Recall that the output of the convex hull problem a convex polygon, that is, a cyclic enumeration of the vertices along its boundary. Thus, it would seem that in order to compute the convex hull, we would "need" to sort the vertices of the hull. It is well known that it is not generally possible to sort a set of n numbers faster than $\Omega(n \log n)$ time, assuming a model of computation based on binary comparisons. (There are faster algorithms for sorting small integers, but these are not generally applicable for geometric inputs.)

Can we turn this intuition into a formal lower bound? We will show that in $O(n)$ time it is possible to reduce the sorting problem to the convex hull problem. This implies that any $O(f(n))$-time algorithm for the convex hull problem implies an $O(n + f(n))$-time algorithm for sorting. Clearly, $f(n)$ cannot be smaller than $\Omega(n \log n)$ for otherwise we would obtain an immediate contradiction to the lower bound on sorting.

The reduction works by projecting the points onto a convex curve. In particular, let $X = \{x_1, \ldots, x_n\}$ be the n values that we wish to sort. Suppose we "lift" each of these points onto a parabola $y = x^2$, by mapping x_i to the point $p_i = (x_i, x_i^2)$. Let P denote the resulting set of points (see Fig. 1). Note that all the points of P lie on its convex hull, and the sorted order of points along the lower hull is the same as the sorted order X. Since it is trivial to obtain the lower hull vertices in $O(n)$ time, we can obtain the sorted order from the hull. This implies the following theorem.

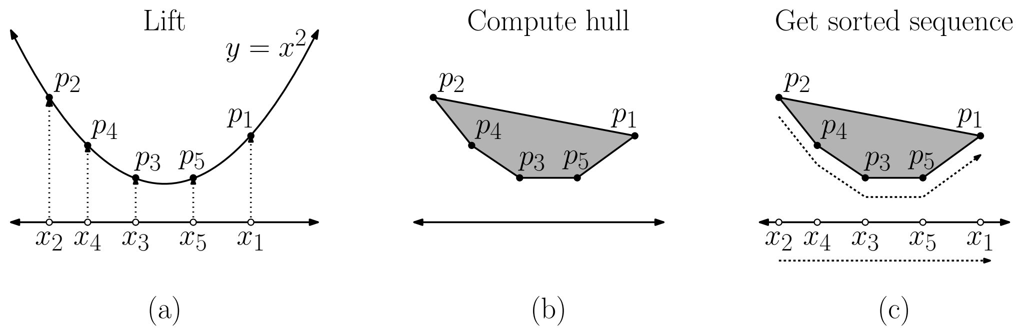

Fig. 1: Reduction from sorting to convex hull.

Theorem: Assuming computations based on comparisons (e.g., orientation tests) any algorithm for the convex hull problem requires $\Omega(n \log n)$ time in the worst case.

Is this the end of the story? Well, maybe not ...

- What if we don't require that the points be enumerated in cyclic order? For example, suppose we just want to count number of points on the convex hull. Can we do better?
- Suppose that we are not interested in worst-case behavior. For example, in many instances of convex hull, relatively few points lie on the boundary of the hull.

We will present three other results in this lecture:

- We will present a convex hull algorithm that runs $O(nh)$ time, where h is the number of vertices on the hull. (This is beats the worst-case bound is h is asymptotically smaller than $O(\log n)$.)
- We will present Chan's algorithm, which computes convex hulls in $O(n \log h)$ time.
- We will present a lower bound argument that shows that, assuming a comparison-based algorithm, even answering the question "does the convex hull have h distinct vertices?" requires $\Omega(n \log h)$ time.

The last result implies that Chan's algorithm is essentially the best possible as a function of h and n.

Gift-Wrapping and Jarvis's March: Our next convex hull algorithm, called *Jarvis's march*, computes the convex hull in $O(nh)$ time by a process called "gift-wrapping." In the worst case, $h = n$, so this is inferior to Graham's algorithm for large h, it is superior if h is asymptotically smaller than $\log n$, that is, $h = o(\log n)$. An algorithm whose running time depends on the output size is called *output sensitive.*

The algorithm begins by identifying any one point of P that is guaranteed to be on the hull, say, the point with the smallest y-coordinate. Call this p_1. It then repeatedly finds the next vertex on the hull in counterclockwise order (see Fig. 2(a)). Suppose that p_{i-1} and p_i are the last two vertices of the hull. The next vertex is the point $p_k \in P \setminus \{p_{i-1}, p_i\}$ that minimizes the angle between the *source ray* $\overrightarrow{p_{i-1}p_i}$ and the *target ray* $\overrightarrow{p_i p_k}$ (see Fig. 2(b)). As usual, we assume general position, so this point is unique. But if not, we take the one that is farthest from p_i. Note that we do not need to compute actual angles. This can all be done with orientation tests. (Try this yourself.) The algorithm stops on returning to p_1.

Clearly, each iteration can be performed in $O(n)$ time, and after h iterations, we return to the starting vertex. Thus, the total time is $O(nh)$. As a technical note, the algorithm can be simplified by adding a *sentinel point* p_0 at the (conceptual) coordinates $(-\infty, 0)$. The algorithm starts with the horizontal ray $\overrightarrow{p_0 p_1}$ (see Fig. 2(c)).

Chan's Algorithm: Depending on the value of h, Graham's scan may be faster or slower than Jarvis' march. This raises the intriguing question of whether there is an algorithm that *always* does as well or better than these algorithms. Next, we present a planar convex hull algorithm by Timothy Chan whose running time is $O(n \log h)$.

While this algorithm is too small an improvement over Graham's algorithm to be of significant practical value, it is quite interesting nonetheless from the perspective of the techniques that it uses:

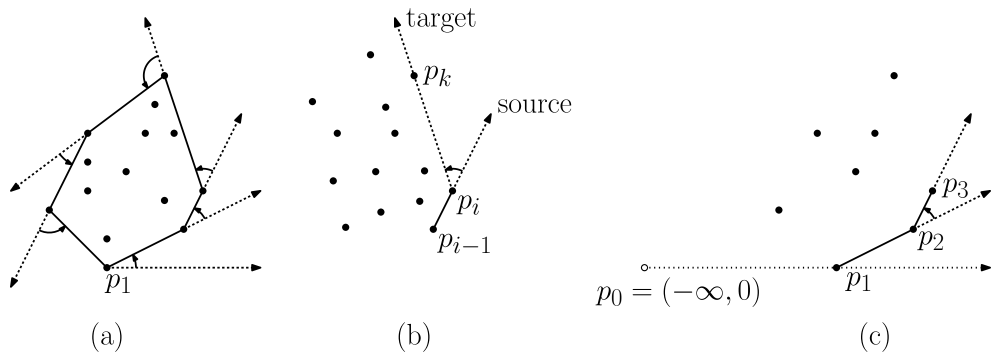

Fig. 2: Jarvis's march.

- It combines two slower algorithms, Graham's and Jarvis's, to form a faster algorithm.
- It employs an interesting guessing strategy to determine the value of a key unknown parameter, the number h of vertices on the hull.

Chan's algorithm: The principal shortcoming of Graham's scan is that it sorts all the points, and hence is doomed to having an $\Omega(n \log n)$ running time, irrespective of the size of the hull. While Jarvis's algorithm is not limited in this way, it is way too slow if there are many points on the hull.

The first observation needed for a better approach is that, if we hope to achieve a running time of $O(n \log h)$, we can only afford a log factor depending on h. So, if we run Graham's algorithm, we are limited to sorting sets of size at most h.

Actually, any polynomial in h will work as well. For example, we could sort a set of size h^2, provided that h^2 is $O(n)$. This is because $h^2 \log(h^2) = 2h^2 \log h = O(n \log h)$. This observation will come in handy later on. So, henceforth, let us imagine that a "little magical bird" tells us a number h^* such that the actual number of vertices on the convex hull satisfies $h \leq h^* \leq \min(h^2, n)$. (We will address this issue of the little magical bird later on.)

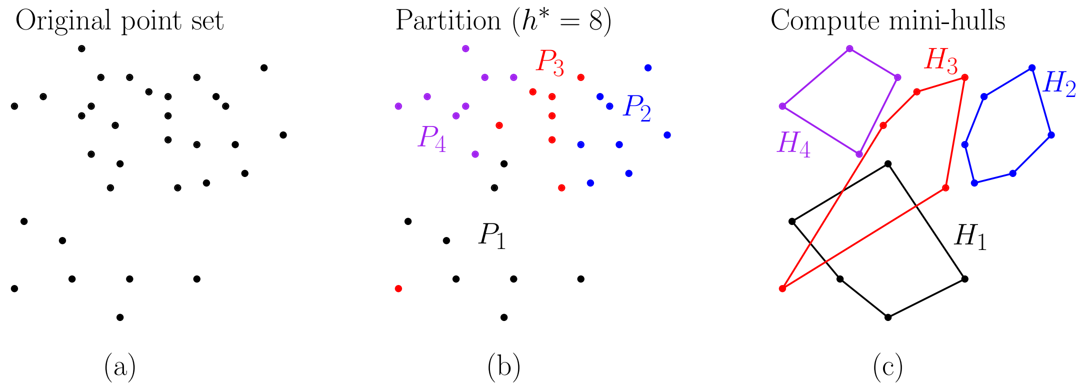

Fig. 3: Partition and mini-hulls.

Step 1: Mini-hulls We start by partitioning the point set P into groups of size h^* (the last group may have fewer than h^* elements). Call these $P_1, \ldots, P_m$ where $r = \lceil n/h^* \rceil$

(see Fig. 3(b)). This can be done arbitrarily, without any regard for their geometric structure. By Graham's algorithm, we can compute the convex hull of each subset in time $O(h^* \log h^*)$. Let $H_1, \ldots, H_m$ denote the resulting *mini-hulls*. The total time to compute all the mini-hulls is

$$O(r(h^* \log h^*)) \;=\; O((n/h^*)h^* \log h^*) \;=\; O(n \log h^*) \;=\; O(n \log h).$$

Good so far. We are within our overall time budget

Step 2: Merging the minis: The high-level idea is to run Jarvis's algorithm, but we treat each mini-hull as if it is a "fat point" (see Fig. 4(a)). Recall that in Jarvis's algorithm, we computed the angle between a source ray and a target ray, where the source ray $\overrightarrow{p_{i-1}p_i}$ was the previous edge of the hull and the target ray $\overrightarrow{p_i p_k}$ went to the next vertex of the hull. We modify this so that the target ray will now be a "tangent ray" or more properly a line of support for a mini-hull H_k that passes through p_i and has H_k lying to the left of the ray, from the perspective of someone facing the direction of the ray (see Fig. 4(b)). Among all the mini-hulls, H_k is the one that minimizes the angles in these rays (see Fig. 4(c)).

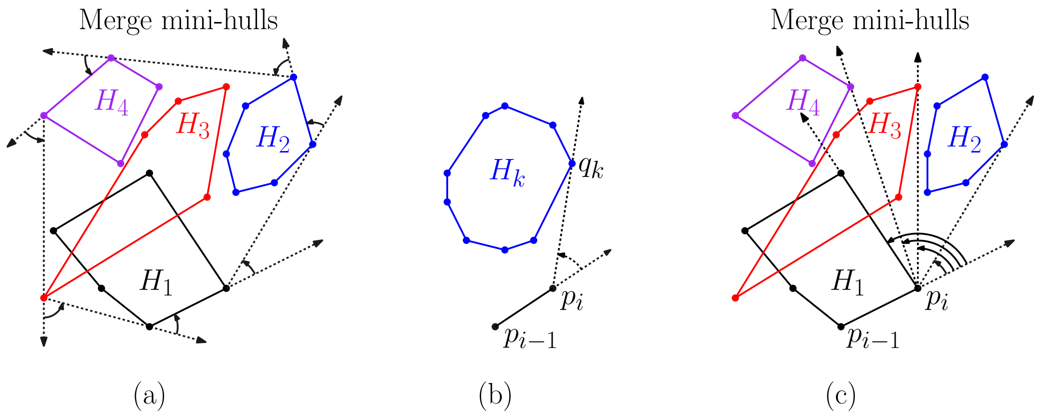

Fig. 4: Using Jarvis's algorithm to merge the mini-hulls.

Note that the current edge $\overrightarrow{p_{i-1}p_i}$ is on the global convex hull, so it cannot lie in the interior of any of the mini-hulls. Among all these tangents, we take the one that yields the smallest external angle (see Fig. 4(c)). Since each of the mini-hulls is represented as a convex polygon having at most h^* vertices, we claim that we can compute this tangent in $O(\log h^*) = O(\log h)$ time through a variant of binary search. This is formalized in the following lemma, whose proof we will leave as an exercise.

Lemma: Consider a convex polygon K in the plane stored as an array of vertices in cyclic order, and let p be any point external to K. The two supporting lines of K passing through p can each be computed in time $O(\log m)$, where m is the number of vertices of K.

Each step of Jarvis's algorithm on the mini-hulls takes $O(r \log h^*) = O(r \log h)$ time to compute the support lines and select the one forming the smallest angle.

The Conditional Algorithm: We can now present a conditional algorithm for computing the convex hull. The algorithm is given a point set P and an estimate h^* of the number of vertices on P's convex hull. Letting h denote the actual number of vertices, if $h \leq h^*$, then this algorithm computes the final hull. Otherwise, the algorithm "fails", reporting that $h > h^*$, and terminates. This is presented in the code block below.

Chan's Algorithm for the Conditional Hull Problem

```
ConditionalHull($P, h^*$) :
    (1) Let $r \leftarrow \lceil n/h^* \rceil$
    (2) Partition $P$ into disjoint subsets $P_1, \ldots, P_r$, each of size at most $h^*$
    (3) For $i \leftarrow 1$ to $r$:
            i Compute $H_i = \text{conv}(P_i)$ using Graham's scan and store the vertices in an ordered array
    (4) Let $p_0 \leftarrow (-\infty, 0)$ and let $p_1$ be the bottommost point of $P$
    (5) For $i \leftarrow 1$ to $h^*$:
        (a) For $j \leftarrow 1$ to $r$:
                i Compute the support line of $H_j$ that passes through $p_i$, and let $q_j$ be the associated vertex
                  of $H_j$
        (b) Set $p_{i+1}$ be the point of $\{q_1, \ldots, q_r\}$ that minimizes the angle between the rays $\overrightarrow{p_{i-1}p_i}$ and
            $\overrightarrow{p_iq_j}$
        (c) If $p_{i+1} = p_1$ then return success ($\langle p_1, \ldots, p_k \rangle$ is the final hull)
    (6) Return failure (conv($P$) has more than $h^*$ vertices)
```

Observe the following: (1) the Jarvis phase never performs for more than h^* stages, and (2) if $h \leq h^*$, the algorithm succeeds in computing the entire hull. To analyze its running time, recall that the computation of the mini-hulls takes $O(n \log h)$ time (under the assumption that $h^* \leq h^2$). Each iteration of the Jarvis phase takes $O(r \log h)$ time, where $r \approx n/h^*$. Since there cannot be more than h^* iterations, this takes total time $O(h^* r \log h) = O(h^*(n/h^*) \log h) = O(n \log h)$ time. So, we are within our overall time budget.

Determining the Hull's Size: The only question remaining is how do we know what value to give to h^*? Remember that, if $h^* \geq h$, the algorithm will succeed in computing the hull, and if $h^* \leq h^2$, the running time of the restricted algorithm is $O(n \log h)$. Clearly we do not want to try a value of h^* that is way too high, or we are doomed to having an excessively high running time. So, we should start our guess small, and work up to larger values until we achieve success. Each time we try a test value $h^* < h$, the restricted hull procedure may tell us we have failed, and so we need to increase the value if h^*.

As a start, we could try $h^* = 1, 2, 3, \ldots, i$, until we luck out as soon as $h^* = h$. Unfortunately, this would take way too long. (Convince yourself that this would result in a total time of $O(nh \log h)$, which is even worse than Jarvis's march.)

The next idea would be to perform a *doubling search.* That is, let's try $h^* = 1, 2, 4, 8, \ldots, 2^i$. When we first succeed, we might have overshot the value of h, but not by more than a factor of 2, that is $h \leq h^* \leq 2h$. The convex hull will have at least three points, and clearly for $h \geq 3$, we have $2h \leq h^2$. Thus, this value of h^* will satisfy our requirements. Unfortunately, it turns out that this is still too slow. (You should do the analysis yourself and convince yourself that it will result in a running time of $O(n \log^2 h)$. Better but still not the best.)

So if doubling is not fast enough, what is next? Recall that we are allowed to overshoot the actual value of h by as much as h^2. Therefore, let's try *repeatedly squaring* the previous guess. In other words, let's try $h^* = 2, 4, 16, \ldots, 2^{2^i}$. Clearly, as soon as we reach a value for which the restricted algorithm succeeds, we have $h \leq h^* \leq h^2$. Therefore, the running time for this stage will be $O(n \log h)$. But what about the total time for all the previous stages?

To analyze the total time, consider the ith guess, $h_i^* = 2^{2^i}$. The ith trial takes time $O(n \log h_i^*) = O\big(n \log 2^{2^i}\big) = O(n 2^i)$. We know that we will succeed as soon as $h_i^* \geq h$, that is if $i = \lceil \lg \lg h \rceil$. (Throughout the semester, we will use "lg" to denote logarithm base 2 and "log" when the base does not matter.[1]) Thus, the algorithm's total running time (up to constant factors) is

$$T(n,h) \;=\; \sum_{i=1}^{\lg \lg h} n 2^i \;=\; n \sum_{i=1}^{\lg \lg h} 2^i.$$

The summation is a geometric series. It is well known that a geometric series is asymptotically dominated by its largest term. Thus, we obtain a total running time of

$$T(n,h) \;<\; n \cdot 2^{\lceil \lg \lg h \rceil} \;<\; n \cdot 2^{1 + \lg \lg h} \;=\; n \cdot 2 \cdot 2^{\lg \lg h} \;=\; 2n \lg h \;=\; O(n \log h),$$

which is just what we want. In other words, by the "miracle" of the geometric series, the total time to try all the previous failed guesses is asymptotically the same as the time for the final successful guess. The final algorithm is presented in the code block below.

Chan's Complete Convex Hull Algorithm

```
Hull(P) :
    (1) $h^* \leftarrow 2$; status $\leftarrow$ fail
    (2) while status $\neq$ fail:
        (a) Let $h^* \leftarrow \min((h^*)^2, n)$
        (b) status $\leftarrow$ ConditionalHull$(P, h^*)$
    (3) Return $L$.
```

Lower Bound (Optional): We show that Chan's result is asymptotically optimal in the sense that any algorithm for computing the convex hull of n points with h points on the hull requires $\Omega(n \log h)$ time. The proof is a generalization of the proof that sorting a set of n numbers requires $\Omega(n \log n)$ comparisons.

If you recall the proof that sorting takes at least $\Omega(n \log n)$ comparisons, it is based on the idea that any sorting algorithm can be described in terms of a *decision tree*. Each comparison has at most three outcomes ($<$, $=$, or $>$). Each such comparison corresponds to an internal node in the tree. The execution of an algorithm can be viewed as a traversal along a path in the resulting ternary (3-way splitting) tree. The height of the tree is a lower bound on the worst-case running time of the algorithm. There are at least $n!$ different possible inputs, each of which must be reordered differently, and so you have a ternary tree with at least $n!$

[1]When $\log n$ appears as a factor within asymptotic big-O notation, the base of the logarithm does not matter provided it is a constant. This is because $\log_a n = \log_b n / \log_b a$. Thus, changing the base only alters the constant factor.

leaves. Any such tree must have $\Omega(\log_3(n!))$ height. Using Stirling's approximation for $n!$, this solves to $\Omega(n \log n)$ height. (For further details, see the algorithms book by Cormen, Leiserson, Rivest, and Stein.)

We will give an $\Omega(n \log h)$ lower bound for the convex hull problem. In fact, we will give an $\Omega(n \log h)$ lower bound on the following simpler decision problem, whose output is either yes or no.

Convex Hull Size Verification Problem (CHSV): Given a point set P and integer h, does the convex hull of P have h distinct vertices?

Clearly if this takes $\Omega(n \log h)$ time, then computing the hull must take at least as long. As with sorting, we will assume that the computation is described in the form of a decision tree. The sorts of decisions that a typical convex hull algorithm will make will likely involve orientation primitives. Let's be even more general, by assuming that the algorithm is allowed to compute *any* algebraic function of the input coordinates. (This will certainly be powerful enough to include all the convex hull algorithms we have discussed.) The result is called an *algebraic decision tree.*

The input to the CHSV problem is a sequence of $2n = N$ real numbers. We can think of these numbers as forming a vector in real N-dimensional space, that is, $(z_1, z_2, \ldots, z_N) = \vec{z} \in \mathbb{R}^N$, which we will call a *configuration*. Each node branches based on the sign of some function of the input coordinates. For example, we could implement the conditional $z_i < z_j$ by checking whether the function $(z_j - z_i)$ is positive. More relevant to convex hull computations, we can express an orientation test as the sign of the determinant of a matrix whose entries are the six coordinates of the three points involved. The determinant of a matrix can be expressed as a polynomial function of the matrices entries. Such a function is called *algebraic*. We assume that each node of the decision tree branch three ways, depending on the sign of a given multivariate algebraic formula of degree at most d, where d is any fixed constant. For example, we could express the orientation test involving points $p_1 = (z_1, z_2)$, $p_2 = (z_3, z_4)$, and $p_3 = (z_5, z_6)$ as an algebraic function of degree two as follows:

$$\det \begin{pmatrix} 1 & z_1 & z_2 \\ 1 & z_3 & z_4 \\ 1 & z_5 & z_6 \end{pmatrix} = (z_3 z_6 - z_5 z_4) - (z_1 z_6 - z_5 z_2) + (z_1 z_4 - z_3 z_2).$$

For each input vector $\vec{z}$ to the CHSV problem, the answer is either "yes" or "no". The set of all "yes" points is just a subset of points $Y \subset \mathbb{R}^N$, that is a region in this space. Given an arbitrary input $\vec{z}$ the purpose of the decision tree is to tell us whether this point is in Y or not. This is done by walking down the tree, evaluating the functions on $\vec{z}$ and following the appropriate branches until arriving at a leaf, which is either labeled "yes" (meaning $\vec{z} \in Y$) or "no". An abstract example (not for the convex hull problem) of a region of configuration space and a possible algebraic decision tree (of degree 1) is shown in the following figure. (We have simplified it by making it a binary tree.) In this case the input is just a pair of real numbers.

We say that two points $\vec{u}, \vec{v} \in Y$ are in the same *connected component* of Y if there is a path in $\mathbb{R}^N$ from $\vec{u}$ to $\vec{v}$ such that all the points along the path are in the set Y. (There

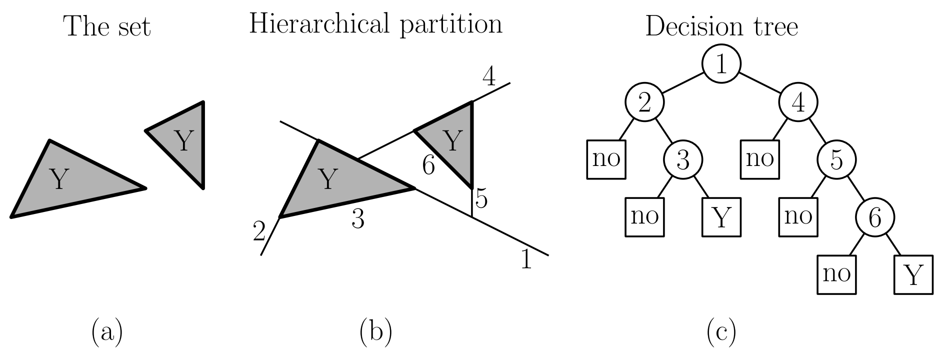

Fig. 5: The geometric interpretation of an algebraic decision tree.

are two connected components in the figure.) We will make use of the following fundamental result on algebraic decision trees, due to Ben-Or. Intuitively, it states that if your set has M connected components, then there must be at least M leaves in any decision tree for the set, and the tree must have height at least the logarithm of the number of leaves.

Theorem: Let $Y \in \mathbb{R}^N$ be any set and let T be any d-th order algebraic decision tree that determines membership in W. If W has M disjoint connected components, then T must have height at least $\Omega((\log M) - N)$.

We will begin our proof with a simpler problem.

Multiset Size Verification Problem (MSV): Given a multiset of n real numbers and an integer k, confirm that the multiset has exactly k distinct elements.

Lemma: The MSV problem requires $\Omega(n \log k)$ steps in the worst case in the d-th order algebraic decision tree

Proof: In terms of points in $\mathbb{R}^n$, the set of points for which the answer is "yes" is

$$Y = \{(z_1, z_2, \ldots, z_n) \in \mathbb{R}^n \ : \ |\{z_1, z_2, \ldots, z_n\}| = k\}.$$

It suffices to show that there are at least $k!k^{n-k}$ different connected components in this set, because by Ben-Or's result it would follow that the time to test membership in Y would be

$$\Omega(\log(k!k^{n-k}) - n) \ = \ \Omega(k \log k + (n-k)\log k - n) \ = \ \Omega(n \log k).$$

Consider the all the tuples $(z_1, \ldots, z_n)$ with $z_1, \ldots z_k$ set to the distinct integers from 1 to k, and $z_{k+1} \ldots z_n$ each set to an arbitrary integer in the same range. Clearly there are $k!$ ways to select the first k elements and k^{n-k} ways to select the remaining elements. Each such tuple has exactly k distinct items, but it is not hard to see that if we attempt to continuously modify one of these tuples to equal another one, we must change the number of distinct elements, implying that each of these tuples is in a different connected component of Y.

To finish the lower bound proof, we argue that any instance of MSV can be reduced to the convex hull size verification problem (CHSV). Thus any lower bound for MSV problem applies to CHSV as well.

Theorem: The CHSV problem requires $\Omega(n \log h)$ time to solve.

Proof: Let $Z = (z_1, \ldots, z_n)$ and k be an instance of the MSV problem. We create a point set $\{p_1, \ldots, p_n\}$ in the plane where $p_i = (z_i, z_i^2)$, and set $h = k$. (Observe that the points lie on a parabola, so that all the points are on the convex hull.) Now, if the multiset Z has exactly k distinct elements, then there are exactly $h = k$ points in the point set (since the others are all duplicates of these) and so there are exactly h points on the hull. Conversely, if there are h points on the convex hull, then there were exactly $h = k$ distinct numbers in the multiset to begin with in Z.

Thus, we cannot solve CHSV any faster than $\Omega(n \log h)$ time, for otherwise we could solve MSV in the same time.

The proof is rather unsatisfying, because it relies on the fact that there are many duplicate points. You might wonder, does the lower bound still hold if there are no duplicates? Kirkpatric and Seidel actually prove a stronger (but harder) result that the $\Omega(n \log h)$ lower bound holds even you assume that the points are distinct.