# A note about "Faster Algorithms for Computing Hong's Bound on Absolute Positiveness" by K. Mehlhorn and S. Ray

Przemysław Koprowski[a], Kurt Mehlhorn[b], Saurabh Ray[c]

[a] *Faculty of Mathematics, University of Silesia, ul. Bankowa 14, PL-40-007 Katowice, Poland*
[b] *Saarland Informatics Campus, Building E1 4, 66123 Saarbrcken, Germany*
[c] *New York University Abu Dhabi, 129188 UAE*

**Abstract**

We show that a linear-time algorithm for computing Hong's bound for positive roots of a univariate polynomial, described by K. Mehlhorn and S. Ray in an article "Faster algorithms for computing Hong's bound on absolute positiveness", is incorrect. We present a corrected version.

## 1. Introduction

Computing an upper bound for real roots of a polynomial is an important problem in computational algebra. It has numerous applications (for instance root separation, to mention just one). Thus, in recent decades there has been intensive effort to find such bounds. Given an univariate polynomial $A = a_0 + a_1 x + \cdots + a_n x^n \in \mathbb{R}[x]$ with a positive leading coefficient $a_n$, a good bound for its positive roots, obtained by Hong [1], is $2 \cdot H(A)$ where

$$H(A) = \max_{\substack{i<n \\ a_i<0}} \min_{\substack{j>i \\ a_j>0}} \left| \frac{a_i}{a_j} \right|^{1/(j-i)}.$$

The above bound can be easily computed in $O(n^2)$ time by running over all pairs of indices $i$ and $j$. Mehlhorn and Ray [2] proposed an $O(n)$ time algorithm in the univariate case by converting the problem into a computational geometry problem that can be solved in linear time. Unfortunately, the linear time algorithm they proposed for the latter is incorrect. The aim of this note is to explain the source of the error and present a correct algorithm.

The main idea in [2] is the following one: construct a point set $P = \{p_0, \ldots, p_n\}$ with $p_i = (i, -\lg |a_i|)$ where lg denotes base 2 logarithm. The slope of a line joining points $p_i$ and $p_j$ is then

$$m_{ij} = \frac{\lg |a_i| - \lg |a_j|}{j - i} = \lg \left| \frac{a_i}{a_j} \right|^{1/(j-i)}.$$

Since $\lg(\cdot)$ is a monotonic function, we have that $H(A) = 2^{h(a)}$ where

$$h(A) = \max_{\substack{i<n \\ a_i<0}} \min_{\substack{j>i \\ a_j>0}} m_{ij}.$$

*Email addresses:* przemyslaw.koprowski@us.edu.pl (Przemysław Koprowski), mehlhorn@mpi-inf.mpg.de (Kurt Mehlhorn), saurabh.ray@nyu.edu (Saurabh Ray)

*URL:* http://z2.math.us.edu.pl/perry/ (Przemysław Koprowski), https://www.mpi-inf.mpg.de/~mehlhorn/ (Kurt Mehlhorn)

The task of computing $H(A)$ is thus reduced to the task of computing $h(A)$.

A point $p_i$ is called *positive* if $a_i > 0$ and *negative* if $a_i < 0$. Denote by $P_i^+$ the set $\{p_j : j \geq i \text{ and } a_j > 0\}$ which is the set of positive points with $x$-coordinate at least $i$. Let $\mathcal{L}_i$ denote the lower hull of $P_i^+$. For a negative point $p_i$, let $s_i$ denote the slope of a tangent from $p_i$ to $\mathcal{L}_i$. Note that $s_i = \min\{m_{ij} : j > i, a_j > 0\}$ and $h(A)$ is the maximum among the $s_i$'s computed for the negative points.

## 2. Error

The algorithm of Mehlhorn and Ray processes the points from right to left i.e., in the order $p_n$, $p_{n-1}$, ..., $p_0$ and claims to maintain the invariant that after $p_i$ has been processed, the following quantities defined for all $0 \leq i \leq n$ are available:[1]

- $\mathcal{L}_i$, the lower hull of the positive points processed so far
- $\sigma_i = \max_{j \geq i, a_j < 0} s_j$, the maximum slope of any tangent computed so far ($\sigma_n := -\infty$)
- the lower tangent $\ell_i$ to $\mathcal{L}_i$ with slope $\sigma_i$
- $t_i$, the point of tangency of $\ell_i$ on $\mathcal{L}_i$

Once all points are processed, $\sigma_0$ is the value returned for $h(A)$. In their paper, the algorithm is described twice, once as pseudocode (Algorithm 1) and once in English (Section 3.2 of their paper). The descriptions are inconsistent and both descriptions are incorrect. The error is in the maintenance of the point of the tangency $t_i$. In the `for` loop in Algorithm 1 of the discussed paper, $t_i$ is not updated when a positive point $p_i$ (i.e., when $a_i > 0$) is processed. In the description of their algorithm (see "Case 2" in Section 3.2 of their paper), they set "$\ell_i = \ell_{i+1}$" and "$t_i = t_{i+1}$". Apart from being inconsistent with Algorithm 1, this is incorrect. If $p_i$ is a positive point and lies below $\ell_{i+1}$, then $p_i$ should be the new tangent point i.e. $t_i = p_i$ and $\ell_i$ should be a line through $p_i$ with slope $\sigma_i = \sigma_{i+1}$. This error causes their algorithm to output a wrong answer. When processing a negative point $p_i$, the algorithm searches for the tangent point $t_i$ to the "right" of $t_{i+1}$. If the lower hull has changed since the last time the tangent point was updated, the algorithm may fail to find the correct tangent point. In particular, if one or more positive points were inserted into the lower hull, these points are not scanned by the algorithm. Likewise, if the previous tangent point was removed from the lower hull, the algorithm still tries to start the scan from this point.

A simple example illustrating this glitch is the following one: consider the polynomial $A = -2 + 4x + x^2$. In this case, we have three points $p_0 = (0, -1)$, $p_1 = (1, -2)$, and $p_2 = (2, 0)$. Here, $p_0$ is a negative point while $p_1$ and $p_2$ are positive points. The points are processed in the order $p_2, p_1, p_0$. After $p_2$ is processed, the lower hull $\mathcal{L}_2$ consists of just $p_2$ and $t_2 = p_2$. Then $p_1$ is processed: $\mathcal{L}_1$ is correctly updated to be the segment joining $p_1$ and $p_2$ but $t_1$ is *incorrectly* set to $t_2$. Finally, when processing the negative point $p_0$, the search for the tangent point $t_0$ starts from $t_1 = p_2$ and in fact ends there since there are no points to the "right" of $p_2$. Thus $t_0$ is wrongly set to $p_2$ instead of $p_1$. Accordingly, the algorithm erroneously outputs the slope of the line joining $p_0$ and $p_2$ i.e. $0.5$ as the computed value of $h(A)$. The correct value of $h(A)$ is the slope of the line joining $p_0$ and $p_1$ which is $-1$.

[1] Note that the indices in [2] run from 1 to $n$ while here they run from 0 to $n$.

A possible correction to the algorithm would be to always search for the tangent point $t_i$ starting from the leftmost point in $\mathcal{L}_i$. However, such an algorithm has $\Omega(n^2)$ running time in the worst case.

If the algorithm is modified as described earlier (i.e., when a positive point $p_i$ lying below $\ell_{i+1}$ is encountered, then $t_i$ should be set to $p_i$ and $\ell_i$ should be a line through $p_i$ with slope $\sigma_i = \sigma_{i+1}$), then it maintains the tangent correctly. However, there is still the question which points needs to be considered when a negative point is scanned and how the linear running time can be maintained. In the next section, we present a slightly different algorithm for which we provide a proof of correctness.

## 3. Correction

Before describing the algorithm, we prove a couple of lemmas useful for proving the correctness of our algorithm. Let $p_i = (i, b_i) = (i, -\lg|a_i|)$.

**Lemma 1.** *Let $p_i$ be a negative point. Then there is a vertex $p_j$ of $\mathcal{L}_{i+1}$ such that $s_i = {}^{(b_j - b_i)}/_{(j-i)}$.*

*Proof.* Let $s_i$ pass through $p_i$ and $p_j$ and assume that no vertex of $\mathcal{L}_{i+1}$ lies on it. In particular, $p_j$ is not a vertex. Then there are vertices $p_\ell$ and $p_k$ of $\mathcal{L}_{i+1}$ such that $p_j$ lies on or above the line segment $\overline{p_\ell p_k}$. Since $s_i$ is only defined by the non-vertex $p_j$, the slope of the ray $\overrightarrow{p_i p_j}$ is strictly smaller than the slopes of the rays $\overrightarrow{p_i p_\ell}$ and $\overrightarrow{p_i p_k}$. This is impossible. □

**Lemma 2.** *Let $p_i$ and $p_j$ be negative points with $i < j$. Let $p_h$ be a vertex of $\mathcal{L}_{j+1}$ that defines $s_j$. If $s_i > s_j$, then $s_i$ is not defined by any vertex of $\mathcal{L}_{j+1}$ whose $x$-coordinate lies strictly between the $x$-coordinates of $p_j$ and $p_h$.*

*Proof.* If $p_i$ lies on or above the line $\ell(p_j, p_h)$ through $p_j$ and $p_h$, the slope of $\overrightarrow{p_i p_h}$ is no larger than the slope of $\overrightarrow{p_j p_h}$ and hence $s_i \leq s_j$. So, $p_i$ lies below the line $\ell(p_j, p_h)$. All vertices of $\mathcal{L}_{j+1}$ whose $x$-coordinate lies between the $x$-coordinates of $p_j$ and $p_h$ lie above this line. Thus the slope of the line defined by $p_i$ and any such vertex is larger than the slope of the line defined by $p_i$ and $p_h$. □

*The Algorithm.* We process the points from right to left. As we do so, we maintain $\mathcal{L}$, the lower hull of the positive points processed so far. The lower hull is kept as a linked list with points appearing in the increasing order of $x$-coordinates and with $p_n$ as its last element. Each point in the list has a pointer to the next point in the lower hull. It also has an additional successor pointer which is used to maintain a sublist of candidates for the tangent points of tangents from negative points. We maintain pointers to the first elements of both the lists. We also maintain the line, $\ell = \ell(p_l, p_h)$, defining the current maximum slope, where $p_l$ is a negative point and $p_h$ is a positive point.

Initially, the linked list storing $\mathcal{L}$ consists only of $p_n$ and the sublist of candidates is identical to $\mathcal{L}$. We then process the points from right to left starting from $p_{n-1}$. Let $p_i$ be the point to be processed next. If $p_i$ is a positive point, we compute the tangent from $p_i$ to $\mathcal{L}_{i+1}$ by walking down the vertex list. The points that we skip over are removed from both the lists. Once we have determined the tangent, we make $p_i$ the first element of both the lists. If $p_i$ is a negative point, the lower hull does not change. If $p_i$ lies on or above $\ell$, we skip over $p_i$. Otherwise, we walk down the candidate list and determine the tangent from $p_i$ to the polygon determined by the candidate list, say the line $\ell(p_i, p_k)$. The point $p_k$ then becomes the first element of the candidate list. Our new maximum slope becomes the larger of the slopes of $\ell(p_\ell, p_h)$ and $\ell(p_i, p_k)$.

**Theorem 1.** *The algorithm is correct and works in linear time.*

*Proof.* The lower hull of the positive points is clearly maintained correctly. We remove points from the candidate list for two reasons: a) if the point ceases to be a vertex of the lower hull — this is justified by Lemma 1, or b) if the point lies between a negative point and its tangent point to the lower hull — this is justified by Lemma 2.

Linear running time follows since the time for processing any point is either constant or proportional to the number of nodes removed from (at least) one of the lists, and since any node can be removed from a list only once. □


[1] Hoon Hong. Bounds for absolute positiveness of multivariate polynomials. *J. Symbolic Comput.*, 25(5):571–585, 1998. ISSN 0747-7171. doi: 10.1006/jsco.1997.0189. URL `http://dx.doi.org/10.1006/jsco.1997.0189`.

[2] Kurt Mehlhorn and Saurabh Ray. Faster algorithms for computing Hong's bound on absolute positiveness. *J. Symbolic Comput.*, 45(6):677–683, 2010. ISSN 0747-7171. doi: 10.1016/j.jsc.2010.02.002. URL `http://dx.doi.org/10.1016/j.jsc.2010.02.002`.