Finding the Repeated Median Regression Line

Andrew Stein Michael Werman

Department of Computer Science
The Hebrew University of Jerusalem
91904 Jerusalem
Israel

Abstract

The repeated median regression line is a robust regression estimate, having a maximal 50% breakdown point. This paper presents an $O(n(\log n)^2)$ algorithm for finding the repeated median regression line through n points in the plane.

1 Introduction

Given n points, $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{R}^2$, we denote the slope of the line through (x_i, y_i) and (x_j, y_j), $1 \leq i, j \leq n$, $x_i \neq x_j$ by a_{ij}. The slope of Siegel's *repeated median* regression line [9] is defined as

$$\hat{a} = \operatorname*{med}_i \operatorname*{med}_{j \neq i} a_{ij}, \tag{1}$$

where the innermost median is over the a_{ij} which are defined. This paper presents an $O(n(\log n)^2)$ algorithm for calculating this slope. The algorithm is based on the framework of the Cole, Salowe, Steiger and Szemerédi (henceforth CSSS) algorithm [3] for finding the kth smallest element of $S = \{a_{ij} : 1 \leq i < j \leq n\}$. Like the CSSS algorithm, our algorithm exploits the geometric information embedded in the given points, in order to avoid calculating all of the $N = \binom{n}{2} = n(n-1)/2$ slopes $a_{ij} \in S$. Once the slope has been found the intercept of the repeated median regression line may be estimated hierarchically[1] as

$$\hat{b} = \operatorname*{med}_i (y_i - \hat{a} x_i) \tag{2}$$

in $O(n)$ operations, leading to an $O(n(\log n)^2)$ algorithm for the repeated median regression line. We would like to note that even though the difficulty of implementing our algorithm makes it impractical, an $O(n(\log n)^4)$ version can be implemented easily. With a little care this may be reduced to $O(n(\log n)^3)$.

The *breakdown point* may be roughly defined as the smallest percentage of gross errors which may cause an estimator to take on arbitrarily large values. Both the traditional *least squares* estimator (the value of (a, b) minimizing $\sum (y_i - a x_i - b)^2$) and the *least absolute values* estimator (the value of (a, b) minimizing $\sum |y_i - a x_i - b|$) have a breakdown point of 0%, meaning that even

[1] Alternatively this algorithm may be adapted to estimate the intercept independently of the slope, as $\hat{b} = \operatorname{med}_i \operatorname{med}_{j \neq i} b_{ij}$, also in $O(n(\log n)^2)$ operations.

a single outlier in the sample can pull the estimates to ∞. On the other hand the repeated median estimator has a maximal 50% breakdown point. The other major estimator with a 50% breakdown point is Rousseeuw's *least median of squares* estimator [7], defined as the value of (a, b) minimizing $\mathrm{med}_i(y_i - ax_i - b)^2$. Several algorithms exist for finding this estimate [11] [10], the fastest in $O(n^2)$ operations. This algorithm is very complicated and the fastest practical algorithm has an expected time complexity of $O(n^2(\log n)^2)$. In short the time complexity our algorithm compares favorably with that of its main competitor. However, it should be pointed out that in many actual computer programs, for example in the PROGRESS package [8], a probabilistic approximation of the least median of squares estimator is found in $O(n)$ or $O(n \log n)$ time.

Throughout the paper, we assume that the n points are in general position: no two points have the same x-coordinate (a_{ij} is finite for every $i \neq j$), no three points are collinear, and no two of the N lines induced by a pair of points have the same slope ($a_{ij} \neq a_{i'j'}$ whenever $i \neq i'$ or $j \neq j'$). This assumption simplifies matters without affecting the complexity of the algorithm.

The median of a multiset A having $2m$ elements may be *any* value between the mth and $(m+1)$th smallest elements of A. We will denote by $\mathrm{med}^\vee A$ and $\mathrm{med}^\wedge A$ the median operators returning the mth and $(m+1)$th smallest elements of A respectively. Whenever we write $\mathrm{med}\, A$, we mean any valid single-valued median operator. Obviously, $\mathrm{med}^\vee A = \mathrm{med}^\wedge A = \mathrm{med}\, A$ for any set A having $2m-1$ elements, and is the mth element of A. Under these definitions and the general position assumption, the algorithm calculates $\mathrm{med}_i^\square \mathrm{med}_{j \neq i}\, a_{ij}$, for some valid median operator $\mathrm{med}^\square$, whenever n is even. If n is odd the algorithm calculates $\mathrm{med}_i \mathrm{med}_{j \neq i}^\vee a_{ij}$. The algorithm may be modified to calculate $\mathrm{med}_i \mathrm{med}_{j \neq i}^\wedge a_{ij}$, if n is odd.

The paper is organized as follows. In Section 2 the framework of the algorithm is set up. This section is based in the slope selection algorithm of CSSS. The heart of the algorithm is described in Section 3. This is a sub-algorithm called by the framework of Section 2. In these two sections the correctness and time complexity of the algorithm is shown. In Section 4 our implementation of the algorithm is discussed.

2 The Framework of the Algorithm

We define a mapping T which takes points and lines in the xy-plane to lines and points respectively, in the uv-plane. T takes the point $p = (\alpha, \beta)$ to the line Tp given by $v = f(u) = \alpha u + \beta$, and the line l given by $y = ax + b$ to the point $Tl = (-a, b)$. It is well known that T preserves the point-line duality. In other words if p_1 and p_2 are two points on line l, then Tp_1 and Tp_2 intersect at Tl. Similarly, if lines l_1 and l_2 intersect at point p, Tp is the line through Tl_1 and Tl_2.

We denote the image under T of the given points by $l_1, \ldots, l_n$. Because no two points have the same x-coordinate, no two of the lines l_i and l_j, $i \neq j$, are parallel. Let (u_{ij}, v_{ij}) be the intersection point of l_i and l_j. Because of the point-line duality, $u_{ij} = -a_{ij}$. Therefore, the dual problem is to find a valid estimate $\hat{u}$ of

$$u^* = \operatorname*{med}_i \operatorname*{med}_{j \neq i} u_{ij}.$$

The slope[2] $\hat{a}$ is then equal to $-\hat{u}$.

[2]In order to find $\hat{b} = \mathrm{med}_i \mathrm{med}_{j \neq i}\, b_{ij} = \mathrm{med}_i \mathrm{med}_{j \neq i}\, v_{ij}$, we use the same algorithm on the v axis instead of the u axis.

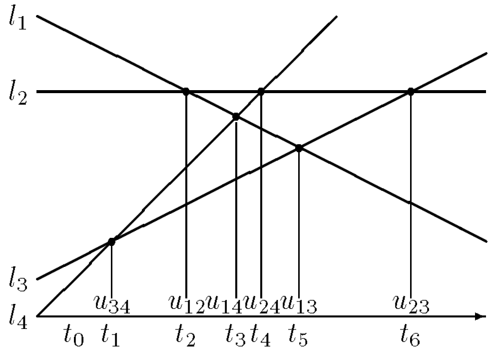

Figure 1: The dual plane.

		$\boldsymbol{\pi}$	$\boldsymbol{b}$	$\boldsymbol{c}$	M	
	t_0	$(1,2,3,4)$	$(0,0,0,0)$	$(0,0,0,0)$	0	
u_{23}	t_1	$(1,2,4,3)$	$(0,0,1,0)$	$(0,0,0,1)$	0	
u_{12}	t_2	$(2,1,4,3)$	$(1,0,1,0)$	$(0,1,0,1)$	0	
u_{14}	t_3	$(2,4,1,3)$	$(2,0,1,0)$	$(0,1,0,2)$	2	m_1, m_4
u_{24}	t_4	$(4,2,1,3)$	$(2,1,1,0)$	$(0,1,0,3)$	3	m_2
u_{13}	t_5	$(4,2,3,1)$	$(3,1,1,0)$	$(0,1,1,3)$	4	m_3
u_{23}	t_6	$(4,3,2,1)$	$(3,2,1,0)$	$(0,1,2,3)$	4	

$\boldsymbol{\rho}$ is either $(1,4,2,3)$ or $(4,1,2,3)$.
$m_{\rho_2} = u_{14} \ \leq \ u^* \ \leq \ m_{\rho_3} = u_{24}$

Table 1: Values for the example in Figure 1.

Let $t_1 < t_2 < \cdots < t_N$ be the elements of $TS = \{u_{ij} : 1 \leq i < j \leq n\}$ in sorted order. The t_i are distinct because the input points are in general position. Let t_0 be any value less than t_1. Let $\boldsymbol{\pi}(u)$ be the permutation that orders the intercepts of $l_1, \ldots, l_n$ with the vertical line at u in descending order. Thus, if $f_i(x)$ is the function whose image is l_i, $1 \leq i \leq n$, then $f_{\pi_1(u)}(u) > \cdots > f_{\pi_n(u)}(u)$. We renumber $l_1, \ldots, l_n$ so that $\boldsymbol{\pi}(t_0)$ is the identity. At the intersection points t_i, we disambiguate $\boldsymbol{\pi}$ by defining $\boldsymbol{\pi}(t_i) = \boldsymbol{\pi}(t_i + \delta)$, where $0 < \delta < \min_j(t_{j+1} - t_j)$. Under this notation, the permutation $\boldsymbol{\pi}(u)$ has no inversions if $u < t_1$, it has one inversion if $t_1 \leq u < t_2$, two inversions if $t_2 \leq u < t_3$, etc. Figure 1 shows an example of the dual plane when $n = 4$ and $N = 6$. Table 1 shows values of $\boldsymbol{\pi}(u)$ in this example.

In seeking an estimate $\hat{u}$ of u^*, we will attempt to sort $f_1(u^*), \ldots, f_n(u^*)$ in descending order. This attempt will yield a permutation $\boldsymbol{\pi}^*$, such that $f_{\pi_1^*}(u^*) > \cdots > f_{\pi_n^*}(u^*)$. Once $\boldsymbol{\pi}^*$ has been found we will choose $\hat{u}$ as

$$\hat{u} = \begin{cases} \max_{\pi_i^* > \pi_{i+1}^*} u_{\pi_{i+1}^* \pi_i^*} & \text{if } n \text{ is even} \\ \min_{\pi_i^* < \pi_{i+1}^*} u_{\pi_i^* \pi_{i+1}^*} & \text{if } n \text{ is odd.} \end{cases} \tag{3}$$

The attempt to sort $f_1(u^*), \ldots, f_n(u^*)$ will involve $O(n \log n)$ questions of the type:

Q_{ij} = "Is $f_i(u^*)$ less or greater than $f_j(u^*)$?", $1 \leq i < j \leq n$.

Some of the questions Q_{ij} will be answered in $O(n \log n)$ time by using the sub-algorithm of Section 3. This sub-algorithm returns one of three answers: LESS, GREATER or FOUND. If the answer to any question Q_{ij} is FOUND we choose

$$\hat{u} = u_{ij} \tag{4}$$

and terminate the program. It will be shown in Section 3 that if the answer is LESS or GREATER, the answer is correct, *even though we do not know* u^*. Also the choice of $\hat{u}$ by (3) or (4) is justified in Section 3.

The framework within which $\boldsymbol{\pi}^*$ is found is the same as that used by CSSS. The control structure for the sort comes from the $O(\log n)$ depth sorting network of Ajtai, Komlós and Szemerédi [1]. At each network level, $n/2$ of the questions Q_{ij} are answered. The network is just a guide for blocking these questions into groups of size $n/2$, which are then answered in series. The sort is complete once the $O(n \log n)$ answers are obtained and we have determined $\boldsymbol{\pi}^*$. If we answered all $n/2$ questions $Q_{i_1 j_1}, \ldots Q_{i_{n/2} j_{n/2}}$ on a level of the network by the sub-algorithm of Section 3, the complexity would be $O(n^2 \log n)$ for that level and $O(n^2(\log n)^2)$ overall. However, we can answer the $n/2$ questions on a level calling the sub-algorithm only $O(\log n)$ times, by applying Megiddo's technique [5] for creating sequential algorithms from parallel ones. This will lead to a $O(n(\log n)^3)$ algorithm.

For each question $Q_{i_h j_h}$, $1 \leq h \leq n/2$, $1 \leq i_h < j_h \leq n$, on a given level of the network, we set $z_h = u_{i_h j_h}$. We compute the median $z_m = \text{med}(z_1, \ldots, z_{n/2})$ in $O(n)$ time. Here we demand the median is one of the z_h even if $n/2$ is even, i.e., $\text{med}^\vee$ or $\text{med}^\wedge$. We then answer the question $Q_{i_m j_m}$ using the sub-algorithm in time $O(n \log n)$. The answer to $Q_{i_m j_m}$ resolves half the questions of the level. For example, if the answer to $Q_{i_m j_m}$ is LESS, then for each h such that $z_h \leq z_m$ the answer to $Q_{i_h j_h}$ will be LESS. The reason for this is that if $f_{i_m}(u^*) < f_{j_m}(u^*)$, then the lines l_{i_m} and l_{j_m} have crossed before u^*, i.e., $z_m = u_{i_m j_m} < u^*$. Therefore, if $u_{i_h j_h} = z_h \leq z_m < u^*$, then the lines l_{i_h} and l_{j_h} have also crossed before u^*, and so $f_{i_h}(u^*) < f_{j_h}(u^*)$. Continuing with the $n/4$ unresolved questions on this level, $Q_{i'_1 j'_1}, \ldots Q_{i'_{n/4} j'_{n/4}}$, we reset $z_h = u_{i'_h j'_h}$, $1 \leq h \leq n/4$, compute $z_m = \text{med}(z_1, \ldots, z_{n/4})$, and answer $Q_{i'_m j'_m}$ using the sub-algorithm. After $O(\log n)$ such phases, all $n/2$ questions on the level have been answered, each phase taking time $O(n \log n)$. All questions on a level are answered in time $O(n(\log n)^2)$, and the above algorithm has an overall time complexity of $O(n(\log n)^3)$.

The time complexity of the algorithm is reduced by a factor of $O(\log n)$ by using the Cole method [2] of weighing "active" questions at each phase, and asking the question corresponding to the weighted median of the active questions. A question is active if both its inputs have been determined, but the question has not yet been answered. In the beginning all the questions on the first level of the network are active. The inputs to each question on level $\lambda > 1$ depend on the answers to a pair of questions on level $\lambda - 1$. Once these two answers are known the question on level λ becomes active and remains active until it has been answered. A weight of $1/4^{\lambda-1}$ is assigned to each active question on level λ. At each phase the weighted median z_w of the intersection points corresponding to the active questions is found in linear time using Reiser's algorithm [6], and the question corresponding to z_w is answered by the sub-algorithm. As before, the answer to this question resolves all active questions $Q_{i_h j_h}$ for which $z_h \leq z_w$ or all questions for which $z_h \geq z_w$.

All answered questions will become inactive and other questions will become active. Cole shows that at each phase the total weight of all active questions is reduced by a factor of at least 1/4. In all $O(\log n)$ questions are answered directly by the sub-algorithm (instead of $O((\log n)^2)$ above), leading to an overall time complexity of $O(n(\log n)^2)$.

3 The Heart of the Algorithm: Answering Q_{ij}

Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ be a permutation of the set $\{1, \ldots, n\}$. If $p < q$ and $\pi_p > \pi_q$, the pair $(\pi_p, \pi_q) = (s, r)$ is called an *inversion* of $\boldsymbol{\pi}$. The *left inversion table* $\boldsymbol{b} = (b_1, \ldots, b_n)$ of $\boldsymbol{\pi}$ is obtained by letting b_r be the number of elements of $\boldsymbol{\pi}$ to the *left* of r, which are *greater* than r. Similarly, the *right inversion table* $\boldsymbol{c} = (c_1, \ldots, c_n)$ of $\boldsymbol{\pi}$ is obtained by letting c_s be the number of elements of $\boldsymbol{\pi}$ to the *right* of s, which are *less* than s. In other words, b_r is the number of inversions in $\boldsymbol{\pi}$ whose second component is r and c_s is the number of inversions in $\boldsymbol{\pi}$ whose first component is s. Therefore, $b_r + c_r$ is the number of inversions involving r. Knuth [4] gives an $O(n \log n)$ algorithm for calculating $\boldsymbol{b}$ from $\boldsymbol{\pi}$. A simple modification of this algorithm to calculates both $\boldsymbol{b}$ and $\boldsymbol{c}$ from $\boldsymbol{\pi}$, also in $O(n \log n)$ time.

Recalling the definition of $\boldsymbol{\pi}(u)$, we define $\boldsymbol{b}(u)$ and $\boldsymbol{c}(u)$ to be the left and right inversion tables of $\boldsymbol{\pi}(u)$ respectively. For $u < t_1$, both $\boldsymbol{b}(u)$ and $\boldsymbol{c}(u)$ are zero vectors. Each coordinate of $\boldsymbol{b}(u)$ and $\boldsymbol{c}(u)$ is a monotone step function with unit steps at some of the t_h's. If t_h is u_{rs}, $r < s$, then $b_r(u)$ and $c_s(u)$ have a step at t_h. For $u \geq t_N$, $\boldsymbol{b}(u) = (n-1, n-2, \ldots, 0)$ and $\boldsymbol{c}(u) = (0, 1, \ldots, n-1)$. Table 1 shows values of $\boldsymbol{b}(u)$ and $\boldsymbol{c}(u)$ in the example in Figure 1. As noted, for any u, $b_r(u) + c_r(u)$ is the number of inversions in $\boldsymbol{\pi}(u)$ involving r. In the dual plane this is the number of intersections of the line l_r with other lines up to the point u, that is $b_r(u) + c_r(u) = \#\{s : u_{rs} \leq u\}$. If we define

$$M(u) = \#\{r : b_r(u) + c_r(u) \geq n/2\}, \tag{5}$$

then $M(u)$ is a monotone step function going from 0 at t_0 to n at t_N, with steps of size one or two at some of the t_h's.

Let us first assume that n is even, and therefore $n-1$ is odd, so that $m_r = \text{med}_{s \neq r}\, u_{rs}$ is always defined in the same manner and is one of the points of intersection of l_r with another line. $M(u)$ is then equal to $\#\{r : m_r \leq u\}$. Let $\boldsymbol{\rho}$ be the permutation that sorts $m_1, \ldots, m_n$ in ascending order. Therefore, $m_{\rho_{n/2}} \leq u^* = \text{med}_r\, m_r \leq m_{\rho_{n/2+1}}$ for any valid definition of the med function. If $M(u) < n/2$ then less than half the m_r's are in $(-\infty, u]$, therefore, $u^* \geq m_{\rho_{n/2}} > u$. If $M(u) > n/2$ then more than half the m_r's are in $(-\infty, u]$, therefore, $u^* \leq m_{\rho_{n/2+1}} < u$. If $M(u) = n/2$ then half the m_r's are in $(-\infty, u]$ and half are in (u, ∞), therefore, $u \in [m_{\rho_{n/2}}, m_{\rho_{n/2+1}}]$. In other words if $M(u) = n/2$, u is a valid median of $m_1, \ldots, m_n$. Values of $M(u)$, the placing of the m_r and the permutation $\boldsymbol{\rho}$ for the example of Figure 1 can be seen in Table 1.

The question Q_{ij}, $1 \leq i < j \leq n$, is answered by returning LESS, FOUND or GREATER according to whether $M(u_{ij})$ is less than, equal to or greater than $n/2$ respectively. If $M(u_{ij}) < n/2$, then $u_{ij} < u^*$, and the lines l_i and l_j have crossed before u^*, therefore, $f_i(u^*) < f_j(u^*)$ and LESS is the answer to Q_{ij}. Similarly, if $M(u_{ij}) > n/2$, GREATER is the answer to Q_{ij}. If $M(u_{ij}) = n/2$ then u_{ij} is a valid value for the repeated median estimate. Therefore, returning FOUND, which will cause $\hat{u}$ to be set to u_{ij} according to (4), will lead to a correct termination of the algorithm. If the algorithm does not terminate due to FOUND being returned by this sub-algorithm, then a permutation $\boldsymbol{\pi}^*$ is found, which sorts $f_1(u^*), \ldots, f_n(u^*)$ in descending order. $\boldsymbol{\pi}^*$ is therefore $\boldsymbol{\pi}(u^*)$

because during the course of the algorithm all questions Q_{ij} needed to sort $f_1(u^*), \ldots, f_n(u^*)$ have been answered correctly. Let $1 \leq k < N$ be the index such that $t_k \leq u^* < t_{k+1}$ (then $\boldsymbol{\pi}^* = \boldsymbol{\pi}(t_k)$). However, $m_{\rho_{n/2}} \in TS$, therefore, $m_{\rho_{n/2}} \leq t_k \leq u^* \leq m_{\rho_{n/2+1}}$, so t_k is a valid median of $m_1, \ldots, m_n$. $\boldsymbol{\pi}(t_k)$ has just reversed a pair of adjacent lines in the permutation $\boldsymbol{\pi}(t_{k-1})$, therefore

$$t_k = \max_{\pi_i(t_k) > \pi_{i+1}(t_k)} u_{\pi_{i+1}(t_k)\,\pi_i(t_k)} = \max_{\pi_i^* > \pi_{i+1}^*} u_{\pi_{i+1}^* \pi_i^*},$$

and so choosing $\hat{u} = t_k$ via (3) is also a valid choice.

If n is odd, then the condition $b_r(u) + c_r(u) \geq n/2$ in the definition (5) is equivalent to $b_r(u) + c_r(u) \geq (n+1)/2$. Therefore, if we denote $m_r = \text{med}^{\wedge}_{s \neq r} u_{rs}$, and $u^{\wedge} = \text{med}_r\, m_r$, then $m_r \in TS$, $u^{\wedge} \in TS$, and $M(u)$ is once more equal to $\#\{r : m_r \leq u\}$. $M(u)$ can never be $n/2$, because $M(u)$ is an integer. If $M(u) < n/2$ (i.e., $M(u) \leq (n-1)/2$) then less than half the m_r's are in $(-\infty, u]$, therefore, $u < u^{\wedge}$. However, if $M(u) > n/2$ (i.e., $M(u) \geq (n+1)/2$) then *not less* than $(n+1)/2$ of the m_r's are in $(-\infty, u]$, therefore, $u \geq u^{\wedge}$. As before, the question Q_{ij}, $1 \leq i < j \leq n$, is answered by returning LESS or GREATER according to whether $M(u_{ij})$ is less or greater than $n/2$ respectively. If $M(u_{ij}) < n/2$ then LESS is the correct answer, on the other hand, if $M(u_{ij}) > n/2$ the answer should really be GREATER or *EQUAL*, because u_{ij} may be $u^{\wedge}$. If u_{ij} is in fact $u^{\wedge}$ then according to the disambiguation of $\boldsymbol{\pi}(u^{\wedge})$, $f_i(u^{\wedge})$ is in fact *less* than $f_j(u^{\wedge})$ and so the answer GREATER is incorrect. This will cause the algorithm to find a permutation $\boldsymbol{\pi}^* = \boldsymbol{\pi}(t_{k-1})$, when the correct permutation is $\boldsymbol{\pi}(t_k) = \boldsymbol{\pi}(u^{\wedge})$. However,

$$u^{\wedge} = t_k = \min_{\pi_i^* < \pi_{i+1}^*} u_{\pi_i^* \pi_{i+1}^*},$$

because $\boldsymbol{\pi}(t_k)$ reverses a pair of adjacent lines in $\boldsymbol{\pi}(t_{k-1}) = \boldsymbol{\pi}^*$, Therefore, for odd n, $\hat{a} = -u^{\wedge}$ will be $\text{med}_i\, \text{med}^{\vee}_{j \neq i} a_{ij}$.

If we'd prefer $\hat{a}$ to be $\text{med}_i\, \text{med}^{\wedge}_{j \neq i} a_{ij}$ for odd n, we'd have to modify the definition (5) of M to

$$M(u) = \#\{r : b_r(u) + c_r(u) \geq (n-1)/2\}.$$

In this case, we denote $m_r = \text{med}^{\vee}_{s \neq r} u_{rs}$, and $u^{\vee} = \text{med}_r\, m_r$. The question Q_{ij}, $1 \leq i < j \leq n$, is answered by returning LESS or GREATER according to whether $M(u_{ij})$ is less or greater than $n/2$ respectively. As explained above, if $M(u_{ij}) < n/2$ the answer should really be GREATER or *EQUAL*, because u_{ij} may be $u^{\vee}$. If u_{ij} is in fact $u^{\vee}$ then the answer GREATER is incorrect. However once again,

$$u^{\vee} = \min_{\pi_i^* < \pi_{i+1}^*} u_{\pi_i^* \pi_{i+1}^*}.$$

Changing the definition of M to the above will cause $\hat{a} = -u^{\vee}$ to be $\text{med}_i\, \text{med}^{\wedge}_{j \neq i} a_{ij}$.

In short, our algorithm receives a question Q_{ij}, represented by the point u_{ij}. Q_{ij} is answered using the algorithm shown in Figure 2. The sorting of $f_1(u_{ij}), \ldots, f_n(u_{ij})$ and the finding of the inversion tables $\boldsymbol{b}(u_{ij})$ and $\boldsymbol{c}(u_{ij})$, both take $O(n \log n)$ time. The other parts of the algorithm take linear time. In all $O(n \log n)$ time is needed to answer Q_{ij}.

4 Implementation

The Ajtai, Komlós and Szemerédi network [1] may have a theoretical depth of only $c \log n$, but it is a very complicated network and the constant c is very large. Therefore, in addition to the difficulty

```
algorithm answer-Q_ij(q, a);
input: q = u_ij — the query point representing Q_ij
output: a — the answer to Q_ij

for r := 1 to n do
        v_r := f_r(q);
sort(v, π);
inversion-tables(π, b, c);
M := 0;
for r := 1 to n do
        if b_r + c_r ≥ n/2 then M := M + 1;
if M < n/2 then a := LESS
else if M > n/2 then a := GREATER
else a := FOUND;
return a;


algorithm sort(v, π);
input: v — an array of values
output: π — the permutation which sorts v_1, ..., v_n in descending order


algorithm inversion-tables(π, b, c);
input: π — a permutation on {1, ..., n}
output: b and c — the left and right inversion tables of π
```

Figure 2: Algorithm for answering Q_{ij}.

in implementing the network, the implementation is only worthwhile for very large values of n. In our implementation we use the Batcher network as described by Knuth [4]. This network has a depth of exactly $\lceil \log n \rceil \left(\lceil \log n \rceil + 1 \right)/2$, and is so easy to build that it may be built during the run-time of the algorithm with almost no additional cost. In addition, we did not implement the Cole method [2] of weighing active questions, but worked one level at a time. These two simplifications increase the theoretical complexity of our algorithm to $O(n(\log n)^4)$.

The running time was improved holding two values z_L and z_R. Initially, $z_L = -\infty$ and $z_R = +\infty$. When a question Q_{ij} needs to be answered, u_{ij} is compared to z_L and z_R. If $u_{ij} \leq z_L$ then the answer is LESS and if $u_{ij} \geq z_R$ the answer is GREATER. Only if $z_L < u_{ij} < z_R$ is the sub-algorithm used to answer Q_{ij}. If the answer returned by the sub-algorithm is LESS then z_L is changed to u_{ij} and if the answer is GREATER z_R is changed to u_{ij}. Although this improvement does not change the time complexity of the algorithm, in practice the running time is decreased considerably.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[2] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, January 1987.

[3] Richard Cole, Jeffrey S. Salowe, W. L. Steiger, and Endre Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, August 1989.

[4] Donald E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[5] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, October 1983.

[6] Angelika Reiser. A linear selection algorithm for sets of elements with weights. *Information Processing Letters*, 7(3):159–162, April 1978.

[7] Peter J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, December 1984.

[8] Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection*. John Wiley, 1987.

[9] Andrew F. Siegel. Robust regression using repeated medians. *Biometrika*, 69(1):242–244, 1982.

[10] Diane L. Souvaine and J. Michael Steele. Time- and space-efficient algorithms for least median of squares regression. *Journal of the American Statistical Association*, 82(399):794–801, September 1987.

[11] J. M. Steele and W. L. Steiger. Algorithms and complexity for least median of squares regression. *Discrete Applied Mathematics*, 14:93–100, 1986.