Tight Bounds for Distributed Selection

Fabian Kuhn
Institute of Theoretical Computer Science
ETH Zurich, Switzerland
kuhn@inf.ethz.ch

Thomas Locher
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
lochert@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

ABSTRACT

We revisit the problem of distributed k-selection where, given a general connected graph of diameter D consisting of n nodes in which each node holds a numeric element, the goal is to determine the k^{th} smallest of these elements. In our model, there is no imposed relation between the magnitude of the stored elements and the number of nodes in the graph. We propose a randomized algorithm whose time complexity is $O(D \log_D n)$ with high probability. Additionally, a deterministic algorithm with a worst-case time complexity of $O(D \log_D^2 n)$ is presented which considerably improves the best known bound for deterministic algorithms. Moreover, we prove a lower bound of $\Omega(D \log_D n)$ for any randomized or deterministic algorithm, implying that the randomized algorithm is asymptotically optimal.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*;
G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*

General Terms

Algorithms, Theory

Keywords

Data Aggregation, Distributed Algorithms, In-Network Aggregation, Median, Time Complexity, Sensor Networks

SPAA'07, June 9–11, 2007, San Diego, California, USA.

1. INTRODUCTION

There is a recent growing interest in distributed aggregation, thanks to emerging application areas such as, e.g., data mining or sensor networks [5, 17, 18]. The goal of distributed aggregation is to compute an aggregation function on a set of distributed values, each value stored at a node in a network. Typical aggregation functions are *max, sum, count, average, median, variance, k^{th} smallest or largest value*, or combinations thereof such as, e.g., "What is the average of the 10% largest values?"

The database community classifies aggregation functions into three categories: distributive (*max, min, sum, count*), algebraic (*plus, minus, average, variance*), and holistic (*median, k^{th} smallest or largest value*). Combinations of these functions are believed to support a wide range of reasonable aggregation queries.[1]

It is well-known that distributive and algebraic functions can easily be computed using the so-called *convergecast* operation, a straightforward flooding-echo procedure executed on a spanning tree. Convergecast is fast, as it terminates after at most $2D$ time, where D denotes the diameter of the network. On the other hand, it is believed that holistic functions cannot be supported by convergecast. After all, the very name "holistic" indicates that one "cannot look into" the set of values, more precisely, that all the values need to be centralized at one node in order to compute the holistic function. Bluntly, in-network aggregation is considered to be practically impossible for holistic functions.

For arbitrary k, a selection algorithm answers questions about the k^{th} smallest value in a set or network. The well-known *median problem* is a special case of the k-selection problem. Generally speaking, selection solves aggregation queries about order statistics and percentiles. Surprisingly, little is known about distributed (network) selection, despite being a significant missing piece of understanding data aggregation.

In this paper, we shed some new light on the problem of distributed selection for general networks with n nodes and diameter D. In particular, we prove that distributed selection is strictly harder than convergecast by giving a lower bound of $\Omega(D \log_D n)$ on the time complexity in Section 5. In other words, to the best of our knowledge, we are the first to formally confirm the preconception about holistic functions being strictly more difficult than distributive or

[1]We encourage the reader to think of a natural aggregation (single value result) query that cannot be formulated by a combination of distributive, algebraic, and holistic functions.

algebraic functions. In addition, in Section 4.1, we present a novel *Las Vegas algorithm* which matches this lower bound with high probability, improving the best randomized algorithm. As for many networks this running time is strictly below collecting all values at one node, our new upper bound proves that (contrary to common belief) in-network aggregation is possible also for holistic functions; in fact, in network topologies where the diameter is large, e.g. in grids, selection can be performed within the same asymptotic time bounds as convergecast. As a third result, in Section 4.2, we derandomize our algorithm, and arrive at a deterministic distributed selection algorithm with a time complexity of $O(D \log_D^2 n)$ which constitutes a substantial improvement over prior art.

2. RELATED WORK

Finding the k^{th} smallest value among a set of n elements is a classic problem which has been extensively studied in the past approximately 30 years, both in a distributed and a non-distributed setting. The problem of finding the median, i.e., the element for which half of all elements are smaller and the other half is larger, is a special case of the k-selection problem which has also received a lot of attention. Blum et al. [1] proposed the first deterministic sequential algorithm that, given an array of size n, computes the k^{th} smallest element in $O(n)$ time. Subsequently, Schönhage et al. [15] developed an algorithm requiring fewer comparisons in the worst-case.

As far as distributed k-selection is concerned, a rich collection of algorithms has been amassed for various models over the years. A lot of work focused on special graphs such as stars and complete graphs [6, 10, 13, 14]. The small graph consisting of two connected nodes where each node knows half of all n elements has also been studied and algorithms with a time complexity of $O(\log n)$ have been presented [2, 9]. For deterministic algorithms in a restricted model, this result has been shown to be tight [9]. Frederickson [3] proposed algorithms for rings, meshes, and also complete binary trees whose time complexities are $O(n)$, $O(\sqrt{n})$, and $O(\log^3 n)$, respectively.

Several algorithms, both of deterministic [7, 8, 16] and probabilistic nature [11, 12, 16], have also been devised for arbitrary connected graphs. Some of these deterministic algorithms restrict the elements the nodes can hold in that the maximum numeric item x_{max} has to be bounded by $O(n^{O(1)})$. Given this constraint, applying binary search results in a time complexity of $O(D \log x_{max}) = O(D \log n)$ [8]. Alternatively, by exponentially increasing the initial guess of $x_k = 1$, the solution can be found in $O(D \log x_k)$ [7]. To the best of our knowledge, the only non-restrictive deterministic k-selection algorithm for general graphs with a sublinear time complexity in the number of nodes is due to Shrira et al. [16]. Their adaptation of the classic sequential algorithm by Blum et al. for a distributed setting has a worst-case running time of $O(Dn^{0.9114})$. In the same work, a randomized algorithm for general graphs is presented. The algorithm simply inquires a random node for its element and uses this guess to narrow down the number of potential elements. The expected time complexity is shown to be $O(D \log n)$. Kempe et al. [4] proposed a gossip-based algorithm that, with probability at least $1-\epsilon$, computes the k^{th} smallest element within $O((\log n + \log \frac{1}{\epsilon})(\log n + \log\log \frac{1}{\epsilon}))$ rounds of communication on a complete graph.

If the number of elements N is much larger than the number of nodes, in $O(D \log\log \min\{k, N-k+1\})$ expected time, the problem can be reduced to the problem where each node has exactly one element using the algorithm proposed by Santoro et al. [11, 12]. However, their algorithm depends on a particular distribution of the elements on the nodes. Patt-Shamir [8] showed that the median can be approximated very efficiently, again subject to the constraint that the maximum element must be bounded by a polynomial in n.

3. MODEL AND DEFINITIONS

In our system model, we are given a connected graph $G = (V, E)$ of diameter D with node set V and edge set E. The cardinality of the node set is $|V| = n$ and the nodes are denoted $v_1, \ldots, v_n$. The *diameter* of a graph is the length of the longest shortest path between any two nodes. Each node v_i holds a single element x_i.[2] Without loss of generality, we can assume that all elements x_i are unique. If two elements x_i and x_j were equal, node IDs, e.g. i and j, could be used as tiebreakers. The goal is to efficiently compute the k^{th} smallest element among all elements $x_1, \ldots, x_n$, and the nodes can achieve this goal by exchanging messages. Nodes v_i and v_j can directly communicate if $(v_i, v_j) \in E$.

The standard asynchronous model of communication is used. Throughout this paper, the communication is considered to be reliable, there is no node failure, and all nodes obediently follow the mandated protocol. We do not impose any constraint on the magnitude of the stored elements. However, we restrict the size of any single message such that it can contain solely a constant number of both node IDs and elements, and also at most $O(\log n)$ arbitrary additional bits. Such a restriction on the message size is needed, otherwise a single convergecast would suffice to accumulate all elements at a single node, which could subsequently solve the problem locally.

As both proposed algorithms are *iterative* in that they continuously reduce the set of possible solutions, we need to distinguish between nodes holding elements that are still of interest from the other nodes. Henceforth, the first set of nodes is referred to as *candidate nodes* or *candidates*. We call the reduction of the search space by a certain factor a *phase* of the algorithm. The number of candidate nodes in phase i is denoted $n^{(i)}$.

We assume that all nodes know the diameter D of the graph. Furthermore, it is assumed that a *breadth first search* spanning tree rooted at the node initiating the algorithm has been computed beforehand. These assumptions are not critical as both the diameter and the spanning tree can be computed in $2D$ time.

The main complexity measure used is the time complexity which is, for deterministic algorithms, the time required from the start of an execution to its completion in the worst case for every legal input and every execution scenario. The time complexity is normalized in that the slowest message is assumed to reach its target after one time unit. As far as our randomized algorithm is concerned, we determine the time after which the execution of the algorithm has completed

[2]Our results can easily be generalized to the case where more than one element is stored at each node. The time complexities are then stated in terms of the number of elements $N > n$ instead of the number of nodes.

with high probability, i.e., with probability at least $1 - \frac{1}{n^c}$ for a constant $c \geq 1$. Thus, in both cases, we do not assign any cost to local computation.

4. ALGORITHMS

As mentioned before, the algorithms presented in this paper operate in sequential phases in which the space of candidates is steadily reduced. This pattern is quite natural for k-selection and used in all other proposed algorithms including the non-distributed case. The best known deterministic distributed algorithm for general graphs uses the well-known *median-of-median* technique, resulting in a time complexity of $O(Dn^{0.9114})$ for a constant group size. A straightforward modification of this algorithm in which the group size in each phase i is set to $O(\sqrt{n^{(i)}})$ results in a much better time complexity. It can be shown that the time complexity of this variant of the algorithm is bounded by $O(D(\log n)^{\log\log n + O(1)})$. However, since our proposed algorithm is substantially better, we will dispense with the analysis of this median-of-median-based algorithm. Due to the more complex nature of the deterministic algorithm, we will treat the proposed randomized algorithm first.

4.1 Randomized Algorithm

While the derivation of an expedient deterministic algorithm is somewhat intricate, it is remarkably simple to come up with a fast randomized algorithm. An apparent solution, proposed by Shrira et al. [16], is to choose a node randomly and take its element as an initial guess. After computing the number of nodes with smaller and larger elements, it is likely that a considerable fraction of all nodes no longer need be considered. By iterating this procedure on the remaining candidate nodes, the k^{th} smallest element can be found quickly for all k.

A node can be chosen randomly using the following scheme: A message indicating that a random element is to be selected is sent along a random path in the spanning tree starting at the root. If the root has l children $v_1, \ldots, v_l$ where child v_i is the root of a subtree with n_i candidate nodes including itself, the root chooses its own element with probability $1/(1+\sum_{j=1}^{l} n_j)$. Otherwise, it sends a message to one of its children. The message is forwarded to node v_i with probability $n_i/(1+\sum_{j=1}^{l} n_j)$ for all $i \in \{1, \ldots, l\}$, and the recipient of the message proceeds in the same manner. It is easy to see that this scheme selects a node uniformly at random, and that it requires at most $2D$ time, because the times to reach any node and to report back are both bounded by D. Note that after each phase the probabilities change as they depend on the altered number of candidate nodes remaining in each subtree. However, having determined the new interval in which the solution must lie, the number of nodes satisfying the new predicate in all subtrees can again be computed in $2D$ time.

This straightforward procedure yields an algorithm that finds the k^{th} smallest element in $O(D \log n)$ expected time, as $O(\log n)$ phases suffice in expectation to narrow down the number of candidates to a small constant. It can even be shown that the time required is $O(D \log n)$ with high probability. The key observation to improve this algorithm is that picking a node randomly always takes $O(D)$ time, therefore several random elements ought to be chosen in a single phase in order to further reduce the number of candidate nodes. The method to select a single random element can easily be modified to allow for the selection of several random elements by including the number of needed random elements in the request message. A node receiving such a message locally determines whether its own element is chosen, and also how many random elements each of its children's subtrees has to provide. Subsequently, it forwards the requests to all of its children whose subtrees must produce at least one random element. Note that all random elements can be found in D time independent of the number of random elements, but due to the restriction that only a constant number of elements can be packed into a single message, it is likely that not all elements can propagate back to the root in D time. However, all elements still arrive at the root in $O(D)$ time if the number of random elements is bounded by $O(D)$.

Algorithm 1 $\mathcal{A}^{rand}(t,k)$

```
1: x_{j-1} := -∞; x_j := ∞
2: repeat
3:    x_0 := x_{j-1}; x_{t+1} := x_j
4:    {x_1, ..., x_t} := getRndElementsInRange(t, (x_{j-1}, x_j))
5:    for i = 1, ..., t in parallel do
6:       r_i := countElementsInRange((x_{i-1}, x_i])
7:    od
8:    if x_0 ≠ -∞ then r_1 := r_1 + 1 fi
9:    j := min_{l∈{1,...,t+1}} Σ_{i=1}^{l} r_i > k
10:   k := k - Σ_{i=1}^{j-1} r_i
11:   if k ≠ 0 and j ≠ 1 then k := k + 1 fi
12: until r_j ≤ t or k = 0
13: if k = 0 then
14:    return x_j
15: else
16:    {x_1, ..., x_s} := getElementsInRange([x_{j-1}, x_j])
17:    return x_k
18: fi
```

Pseudo-code for the algorithm $\mathcal{A}^{rand}$ which determines the k^{th} smallest element by making use of a larger number of random elements is depicted in Algorithm 1.

The algorithm $\mathcal{A}^{rand}$ takes two parameters, the number of random elements t considered in each phase, and the position of interest k. The function *getRndElementsInRange* collects t random elements in the range (x_{j-1}, x_j). As mentioned before, this operation includes in a first step the counting of nodes whose elements lie in the specified interval. Once the number of candidate nodes in each subtree has been determined, the t random elements are selected and reported back to the root. Hence, this function call overall takes $O(D + t)$ time. After acquiring the random elements, which are ordered such that $x_1 < \ldots < x_t$, the number of elements r_i in the intervals $(x_{i-1}, x_i]$ are counted using the function *countElementsInRange*. All these counting requests can be sent one after the other, thus there is no need to wait for one single counting operation to complete. By counting the nodes in each fraction in parallel, the time complexity of this operation is again $O(D+t)$. Afterwards, the interval $(x_{j-1}, x_j]$ in which the wanted element is to be found is determined, and k is updated accordingly. These steps are repeated until the solution is found, i.e., $k = 0$, or the fraction is small enough such that all elements can be collected in $O(D+t)$ time and the solution can be computed locally.

It is evident that the number of iterations determines the overall time complexity, as each operation can be performed in $O(D)$ time provided that $t \in O(D)$. The following lemma states how many phases are required in order to find the solution with high probability.

LEMMA 4.1. *In a connected graph of diameter $D \geq 2$ consisting of n nodes, algorithm $\mathcal{A}^{rand}(8\lambda D, k)$, where $\lambda \geq 1$, determines the k^{th} smallest element in less than $3\log_D n$ phases w.h.p.*

PROOF. First, we compute an upper bound on the probability that after any phase i the wanted element is in a fraction of size at least $c\frac{\log D}{D}$ times the size of the fraction after phase $i-1$ for a suitable constant c, i.e., $n^{(i)} \geq n^{(i-1)}\frac{c\log D}{D}$. Let $\hat{x}_1 < \ldots < \hat{x}_n$ denote the total order of all elements. The probability that none of the elements $\hat{x}_{k+1}, \ldots, \hat{x}_{k+\frac{c\log D}{2D}}$ are among the $t = 8\lambda D$ random elements is bounded by $\left(1 - \frac{c\log D}{2D}\right)^{8\lambda D}$. The same argument holds for the elements $\hat{x}_{k-1}, \ldots, \hat{x}_{k-\frac{c\log D}{2D}}$ and thus, by virtue of a union-bound argument, we have that

$$\begin{aligned}\mathbb{P}\left[n^{(i)} \geq n^{(i-1)}\frac{c\log D}{D}\right] &\leq 2\left(1 - \frac{4\lambda c\log D}{8\lambda D}\right)^{8\lambda D} \\ &\leq 2e^{-4\lambda c\log D}.\end{aligned}$$

We call phase i *successful* if $n^{(i)} < n^{(i-1)}\frac{c\log D}{D}$. By setting $c := \frac{16}{17}$, less than $2\log_D n$ successful phases are required to find $\hat{x}_k$ as it holds for all $D \geq 1$ that $\left(\frac{c\log D}{D}\right)^2 < \frac{1}{D}$ for this choice of c.

Let the random variable $\mathcal{U}(\tau)$ denote the number of *unsuccessful* phases out of τ phases in total. The probability that $\tau := 3\log_D n$ phases suffice is therefore

$$\begin{aligned}&\mathbb{P}[\mathcal{U}(3\log_D n) \geq \log_D n] \\ &\leq \sum_{i=\log_D n}^{3\log_D n}\binom{3\log_D n}{i}\left(2e^{-4\lambda c\log D}\right)^i\left(1 - 2e^{-4\lambda c\log D}\right)^{3\log_D(n)-i} \\ &\leq \binom{3\log_D n}{\log_D n}\left(2e^{-4\lambda c\log D}\right)^{\log_D n} \\ &\leq \left(\frac{6ee^{-4\lambda c\log D}\log_D n}{\log_D n}\right)^{\log_D n} \\ &\leq \left(e^{-(4\lambda c\log D - \ln 6e)}\right)^{\log_D n} \\ &\leq \left(\frac{1}{D^\lambda}\right)^{\log_D n} = \frac{1}{n^\lambda}.\end{aligned}$$

This holds because $4\lambda c\frac{\ln D}{\ln 2} - \ln 6e > \lambda \ln D$ for $c = \frac{16}{17}$ and $D \geq 2$. Hence, with high probability, the algorithm terminates after less than $3\log_D n$ phases. □

As the number of phases is $O(\log_D n)$ with high probability, we immediately get the following result.

THEOREM 4.2. *In a connected graph of diameter $D \geq 2$ consisting of n nodes, the time complexity of algorithm $\mathcal{A}^{rand}(8\lambda D, k)$, where $\lambda \geq 1$, is $O(D\log_D n)$ w.h.p.*

This algorithm is considerably faster than the algorithm selecting only a single random element in each phase. In Section 5, we prove that no deterministic or probabilistic algorithm can be better asymptotically, i.e., algorithm $\mathcal{A}^{rand}$ is *asymptotically optimal*.

4.2 Deterministic Algorithm

The difficulty of deterministic iterative algorithms for k-selection lies in the selection of elements that provably allow for a reduction of the search space in each phase. Once these elements have been found, the reduced set of candidate nodes can be determined in the same way as in the randomized algorithm. Therefore, the only difference between the two algorithms is the way these elements are chosen. While the function *getRndElementsInRange* performs this task in the randomized algorithm $\mathcal{A}^{rand}$, a suitable counterpart for the deterministic algorithm, referred to as $\mathcal{A}^{det}$, has to be derived.

A simple idea to go about this problem is to start sending up elements from the leaves of the spanning tree, accumulating the elements from all children at the inner nodes, and then recursively forwarding a selection of t elements to the parent. The problem with this approach is the reduction of all elements received from the children to the desired t elements. If a node v_i receives t elements from each of its c_i children in the spanning tree, the t elements that partition all $c_i t$ nodes into segments of approximately equal size ought to be found. However, in order to find these elements, the number of elements in each segment has to be counted starting at the leaves. Since this counting has to be repeated in each step along the path to the root, the time required to find a useful partitioning into k segments requires $O(D(D + Ct))$ time, where $C := \max_{i\in\{1,\ldots,n\}} c_i$. This approach suffers from several drawbacks: It takes at least $O(D^2)$ time just to find a partitioning, and the time complexity depends on the structure of the spanning tree.

Our proposed algorithm $\mathcal{A}^{det}$ solves these issues in the following manner. In any phase i, the algorithm splits the entire graph into $O(\sqrt{D})$ groups, each of size $O(n^{(i)}/\sqrt{D})$. Recursively, in each of those groups a particular node initiates the same partitioning into $O(\sqrt{D})$ groups as long as the group size is larger than $O(\sqrt{D})$. Groups of size at most $O(\sqrt{D})$ simply report all their elements to the node that initiated the grouping at this recursion level. Once such an initiating node v has received all $O(\sqrt{D})$ elements from each of the $O(\sqrt{D})$ groups it created, it sorts those $O(D)$ elements, and subsequently issues a request to count the nodes in each of the $O(D)$ intervals induced by the received elements. Assume that all the groups created by node v together contain $n_v^{(i)}$ nodes in phase i. The intervals can locally be merged into $O(\sqrt{D})$ intervals such that each interval contains at most $O(n_v^{(i)}/\sqrt{D})$ nodes. These $O(\sqrt{D})$ elements are recursively sent back to the node that created the group to which node v belongs. Upon receiving the $O(D)$ elements from its $O(\sqrt{D})$ groups and counting the number of nodes in each interval, the root can initiate phase $i+1$ for which it holds that $n^{(i+1)} < n^{(i)}/O(\sqrt{D})$. The procedure *getPartitionInRange* that computes this partitioning is depicted in Algorithm 2.

We will now study each part of the function *getPartitionInRange* in greater detail. In a first step, groups are created using the function *createGroups*. This operation includes the counting of the number of remaining candidate nodes in phase i by accumulating the updated counters from each subtree. Simultaneously, the groups are built using the following procedure. The leaf nodes return 0 if their elements do not fulfill the predicate of the current phase, and 1 otherwise. Any inner node v with children $c_1, \ldots, c_p$ whose

Algorithm 2 getPartitionInRange$(g, (x_{j-1}, x_j))$

```
1: n' := countElementsInRange((x_{j-1}, x_j))
2: if n' > g then
3:    {v_1, ..., v_l} := createGroups(g, (x_{j-1}, x_j))
4:    for i = 1, ..., l in parallel do
5:       {x_{i1}, ..., x_{im}} := getPartitionInRange(g, (x_{j-1}, x_j))
         from v_i
6:    od
7:    X := ⋃_{i=1,...,l} {x_{i1}, ..., x_{im}}
8:    {x_1, ..., x_s} := sort(X)
9:    for i = 1, ..., s − 1 in parallel do
10:      r_i := countElementsInRange((x_i, x_{i+1}])
11:   od
12:   {x'_1, ..., x'_m} := reduce({x_1, ..., x_s}, {r_1, ..., r_{s−1}})
13: else
14:   {x'_1, ..., x'_m} := getElementsInRange((x_{j-1}, x_j))
15: fi
16: return {x'_1, ..., x'_m}
```

subtrees contain $n_1^{(i)}, \ldots, n_p^{(i)}$ candidates in phase i creates groups $g_1, \ldots, g_t$, where $\forall h \in \{1, \ldots, t\} : g_h \subseteq \{1, \ldots, p\}$, $\bigcup_{h \in \{1,\ldots,t\}} g_h = \{1, \ldots, p\}$, and $\forall h, l \in \{1, \ldots, t\} : g_h \cap g_l = \emptyset$. The size of a group g_h is defined as $s(g_h) := \sum_{j \in g_h} n_j^{(i)}$. The groups are created such that $s(g_h) \leq \lceil n_v^{(i)}/\sqrt{D} \rceil$ for all h, and t is minimal. Unless all groups have exactly size $\lceil n_v^{(i)}/\sqrt{D} \rceil$, node v adds itself to any suitable group if its element is still of interest. For each group g_h not including node v, v selects a *leader* which is any node c_l, where $l \in g_h$. This particular node is informed about its duty to find a partitioning within its group in the next step. Subsequently, all nodes are informed about their group membership. Let s denote the size of the group to which v belongs, including v. If $s = \lceil n_v^{(i)}/\sqrt{D} \rceil$ or v is the root node, v becomes the leader of this group as well. Otherwise, s is propagated to the parent node in order to further enlarge the group and to find a suitable leader. Note that, while inner nodes can act as relay nodes for group communication, any edge at a specific recursion level is used strictly by at most one group. All the selected leaders report to the root which concludes the operation *createGroups*. As the size of each group is larger than $\lceil n_v^{(i)}/\sqrt{D} \rceil/2$, otherwise at least two groups could be merged, the total number of groups is bounded by $l \leq g := 2\sqrt{D}$.

Once these groups are set up, *getPartitionInRange* is called recursively at each leader in order to further partition the groups. The return value of this function call at a particular node v is a subset of the elements stored at candidate nodes belonging to the subtree rooted at v. If its group consists of less than $g = 2\sqrt{D}$ candidates, all elements are returned to the node that issued this request for a further partitioning. In case the group is larger, the resulting sets of the recursive calls are accumulated and sorted using the function *sort*. Afterwards, the number of elements belonging to candidate nodes that are part of this group are counted for each induced interval. The function *reduce* merges adjacent intervals as long as each interval contains at most $n_v^{(i)}/\sqrt{D}$ elements, and the reduced set of elements $\{x'_1, \ldots, x'_m\} \subset \{x_1, \ldots, x_{s-1}\}$ inducing these new intervals is returned. The following lemma bounds the number of elements returned and the number of elements in each interval.

LEMMA 4.3. *At any node v in phase i,* getPartitionInRange *returns a set of at most $2\sqrt{D}+1$ elements which induce intervals containing at most $n_v^{(i)}/\sqrt{D}$ elements each.*

PROOF. We will prove the lemma by induction. Both claims are obviously true if the group is of size smaller than $g = 2\sqrt{D}$. Assume now that the subtree rooted at node v contains more than g candidates, and that, by the induction hypothesis, both claims hold for all l returned sets of elements. Let $n_{v_j}^{(i)}$ denote the number of elements in the group containing node v_j in phase i. As any interval $\{x_j, x_{j+1}\}$ for $j \in \{1, \ldots, s-2\}$ contains elements from at most one interval from each of the l groups, we have that the total number of elements in this interval is bounded by $\sum_{j=1}^{l} n_{v_j}^{(i)}/\sqrt{D} \leq n_v^{(i)}/\sqrt{D}$. If two adjacent intervals together contain less than $n_v^{(i)}/\sqrt{D}$ elements, they are combined into one interval. Once no more intervals can be merged, any two consecutive segments contain more than $n_v^{(i)}/\sqrt{D}$ elements and it therefore holds that $2n_v^{(i)} = 2\sum_{j=1}^{l} n_{v_j}^{(i)} > (l-1)n_v^{(i)}/\sqrt{D}$. Hence it follows that l is upper bounded by $2\sqrt{D}+1$. □

The root uses the elements returned by *getPartitionInRange* to narrow down the range of potential candidates as described in Section 4.1. Similarly to $\mathcal{A}^{rand}$, the algorithm $\mathcal{A}^{det}$ takes two parameters which are the group size g and the value k. We are now in the position to prove the following theorem.

THEOREM 4.4. *In a connected graph of diameter $D \geq 2$ consisting of n nodes, the time complexity of algorithm $\mathcal{A}^{det}(2\sqrt{D}, k)$ is $O(D \log_D^2 n)$.*

PROOF. From Lemma 4.3 it follows that $n^{(i+1)} \leq n^{(i)}/\sqrt{D}$, thus the number of phases is bounded by $O(\log_D n)$. The groups can be created in $O(D)$ time. The only other non-local operations are the collection of the sets from all subgroups and the counting of all elements in each interval within the entire group. As l is bounded by $2\sqrt{D}$ and each group returns at most $2\sqrt{D}+1$ elements according to Lemma 4.3, at most $4D + 2\sqrt{D}$ elements have to be collected at the initiating node which can be done in $O(D)$ time. Consequently, the number of elements in each interval can also be counted in $O(D)$ time. Let $\mathcal{T} : \mathbb{N} \to \mathbb{N}$ where $\mathcal{T}(n)$ denotes the time complexity of *getPartitionInRange* if there are n candidate nodes. We have that $\mathcal{T}(n) \leq \mathcal{T}(n/\sqrt{D}) + cD$ for a suitable constant c, implying that $\mathcal{T}(n) \in O(D \log_D n)$. The time complexity of $\mathcal{A}^{det}$ is therefore bounded by $O(D \log_D^2 n)$. □

5. LOWER BOUND

In this section, we prove a time lower bound for *generic* distributed selection algorithms which shows that the time complexity of the simple randomized algorithm of Section 4.1 for finding an element of rank k is asymptotically optimal for most values of k. Informally, we call a selection algorithm generic if it does not exploit the structure of the element space except for using the fact that there is a global order on all the elements. Formally, this means that the only access to the structure of the element space is by means of the comparison function. Equivalently, we can assume that all elements assigned to the nodes are fixed but that the ordering of elements belonging to different nodes is determined by an adversary and initially not known to the nodes. For

the lower bound, we use a simpler synchronous communication model where time is divided into rounds and in every round each node can send a message to each of its neighbors. Note that since the synchronous model is strictly more restrictive than the asynchronous model, a lower bound for the synchronous model directly carries over to the asynchronous model. We show that if in any round only $B \geq 1$ elements can be transmitted over any edge, such an algorithm needs at least $\Omega(D \log_D n)$ rounds to find the median with reasonable probability. A lower bound for finding an element with an arbitrary rank k can then be derived by using a simple reduction.

We prove the lower bound in two steps. First we prove a $\Omega(\log_{2B} n)$ time lower bound for protocols between two nodes if each of the nodes starts with half of the elements. In a second step, we construct a graph $T(D)$ for every diameter $D \geq 3$ such that every median algorithm on $T(D)$ can be simulated by two nodes to compute the median in a two-party protocol. For simplicity, we prove the lower bound for deterministic algorithms and use Yao's principle to obtain a lower bound for randomized algorithms.

Let us therefore first consider protocols between two nodes u and v such that u and v each have $N \geq 1$ elements $u_0 \prec u_1 \prec \cdots \prec u_{N-1}$ and $v_0 \prec v_1 \prec \cdots \prec v_{N-1}$, respectively, where $\prec$ is the global order according to which we want to find the median. We denote the sets of elements of u and v by S_u and S_v, respectively. Without loss of generality, assume again that no element occurs twice, i.e., there is a total of $2N$ distinct elements. Each message $M = (S, X)$ between the two nodes is further assumed to contain a set S of at most B elements and some arbitrary additional information X. Assume M is a message from u to v. In this case, X can be everything which can be computed from the results of the comparisons between all the elements u has seen so far, as well as all the additional information u has received so far. The only restriction on X is that it cannot be used to transmit information about additional elements. We call a protocol between u and v which only sends messages of the form $M = (S, X)$ as described above, a generic two-party protocol.

The time needed to compute the median in the above model of course depends on the way in which the $2N$ elements are distributed among u and v. Therefore, we first need to determine this distribution. Equivalently, we need to determine the outcomes of comparisons between two elements $u_i \in S_u$ and $v_j \in S_v$. Let us first describe the general idea. We construct N different distributions of elements (i.e., N different orders between the elements in S_u and S_v) in such a way that the N distributions result in N different elements for the median. We choose one of the N distributions uniformly at random and show that in each communication round, the probability for reducing the number of possible distributions by more than a factor of λB is exponentially small in λ.

For simplicity, we assume that $N = 2^\ell$ is a power of 2. Let $X_0, \ldots, X_{\ell-1} \sim \text{Bernoulli}(1/2)$ be ℓ independent Bernoulli variables, i.e., all X_i take values 0 or 1 with equal probability. The distribution of the $2N$ elements among u and v is determined by the values of $X_0, \ldots, X_{\ell-1}$. If $X_{\ell-1} = 0$, the $N/2$ smallest of the $2N$ elements are assigned to u and the $N/2$ largest elements are assigned to v. If $X_{\ell-1} = 1$, it is the other way round. In the same way, the value of $X_{\ell-2}$ determines the assignment of the smallest and largest $N/4$ of the remaining elements: If $X_{\ell-2} = 0$, u gets the elements with ranks $N/2+1, \ldots, 3N/4$ and v gets the elements with ranks $5N/4+1, \ldots, 3N/2$ among all $2N$ elements. Again, the remaining elements are recursively assigned in the same way depending on the values of $X_{\ell-3}, \ldots, X_0$ until only the two elements with ranks N and $N+1$ (i.e., the two median elements) remain. The element with rank N is assigned to u and the element with rank $N+1$ is assigned to v. Formally, the resulting global order can be defined as follows. Let u_α and v_β be two elements of u and v, respectively. Recall that u_α is the $(\alpha+1)$-smallest element of u and that v_β is the $(\beta+1)$-smallest element of v. Let $\alpha_{\ell-1} \ldots \alpha_0$ and $\beta_{\ell-1} \ldots \beta_0$ be the base 2 representations of α and β, i.e., $\alpha = \sum_{i=0}^{\ell-1} \alpha_i 2^i$ and $\beta = \sum_{i=0}^{\ell-1} \beta_i 2^i$. Assume that there is an index i for which $\alpha_i = X_i$ or $\beta_i \neq X_i$. Let i^* be the largest such index. If $X_{i^*} = 0$, we have $u_\alpha \prec v_\beta$, whereas if $X_{i^*} = 1$, we obtain $v_\beta \prec u_\alpha$. If there is no index i for which $\alpha_i = X_i$ or $\beta_i \neq X_i$, $u_\alpha \prec v_\beta$. In this case, u_α and v_β are the elements with ranks N and $N+1$ among all $2N$ elements, i.e., u_α and v_β are the median elements. Since the median elements u_α and v_β are those elements for which $\alpha_i \neq \beta_i = X_i$ for all $i \in \{0, \ldots, \ell-1\}$, it immediately follows that finding the median is equivalent to determining the values of X_i for all i.

Let $\mathcal{A}$ be a deterministic, generic two-party algorithm between u and v which computes the median. Consider the state of u and v after the first t rounds of an execution of $\mathcal{A}$. Let $S_{uv}(t) \subseteq S_u$ and $S_{vu}(t) \subseteq S_v$ be the sets of elements that u and v have sent to each other in the first t rounds. After t rounds, everything u and v can locally compute has to be a function of the results of comparisons between elements in $S_u \cup S_{vu}(t)$ and of comparisons between elements in $S_v \cup S_{uv}(t)$, respectively. Note that except for the elements themselves, everything u and v can send to each other can be computed from comparisons between elements within these two sets. Let us therefore define the combined state $\mathbf{state}_{u,v}(t)$ of u and v at time t as the partial order which is induced by comparing all pairs of elements in $S_u \cup S_{vu}(t)$ and by comparing all pairs of elements in $S_v \cup S_{uv}(t)$. Because knowledge of the median implies the knowledge of the values of X_i for all i, it follows that if after t rounds, u and v know the median, the values of all X_i can be computed as a function of $\mathbf{state}_{u,v}(t)$. For an element $u_\alpha \in S_u$ let $I(u_\alpha) := \max\big\{i \in \{0, \ldots, \ell-1\} \big| X_i = \alpha_i\big\}$ where α_i is defined as above. If there is no i for which $X_i = \alpha_i$, we define $I(u_\alpha) := -1$. Similarly, for an element $v_\beta \in S_v$ let $J(v_\beta) := \max\big\{j \in \{0, \ldots, \ell-1\} \big| X_j \neq \beta_j\big\}$. Again, $J(v_\beta) := -1$ if $X_j = \beta_j$ for all j. The following lemma quantifies how much we can deduce about the values of the random variables X_i from a given combined state $\mathbf{state}_{u,v}(t)$.

LEMMA 5.1. *Let $S_{uv}(t)$ and $S_{vu}(t)$ be defined as above. Further, we define $I^*(t) = \min_{u_\alpha \in S_{uv}(t)} I(u_\alpha)$, $J^*(t) = \min_{v_\beta \in S_{vu}(t)} J(v_\beta)$, and $H^*(t) = \min\{I^*(t), J^*(t)\}$. The combined state $\mathbf{state}_{u,v}(t)$ of u and v at time t is statistically independent of X_i for $i < H^*(t)$.*

PROOF. We prove that $\mathbf{state}_{u,v}(t)$ can be computed if we know X_i for all $H^*(t) \leq i \leq \ell - 1$. The lemma then follows because we have chosen the random variables X_i to be independent of each other.

In order to prove that we can compute $\mathbf{state}_{u,v}(t)$ from the values of X_i for $i \geq H^*(t)$, we have to show that the re-

sults of all comparisons between two elements in $S_u \cup S_{vu}(t)$ and between two elements in $S_v \cup S_{uv}(t)$ can be deduced from the knowledge of these X_i. Let us therefore consider an element $u_\alpha \in S_u$ and an element $v_\beta \in S_{vu}(t)$. Assume that there is an index i for which $\alpha_i = X_i$ or $\beta_i \neq X_i$ and let i^* be the largest index for which this holds. We have seen that in this case $u_\alpha \prec v_\beta$ if $X_{i^*} = 0$ and $v_\beta \prec u_\alpha$ if $X_{i^*} = 1$, i.e., the outcome of the comparison between u_α and v_β is determined by the value of X_{i^*}. However, by the definition of $J(v_\beta)$, we have $i^* \geq J(v_\beta) \geq J^*(t) \geq H^*(t)$. If there is no index i for which $\alpha_i = X_i$ or $\beta_i \neq X_i$, $H^*(t) = -1$ and the lemma trivially holds. Symmetrically, we can show that every comparison between two elements in $S_v \cup S_{uv}(t)$ is determined by the value of a variable $X_{i'}$ with $i' \geq H^*(t)$ if $H^*(t) \geq 0$. This concludes the proof. □

Based on Lemma 5.1, we are able to prove a time lower bound for finding the element with rank k by a two-party protocol when each node starts with N elements.

THEOREM 5.2. *Let $h = \min\{k, 2N - k\}$. Every, possibly randomized, generic two-party protocol needs at least $\Omega(\log_{2B} h)$ rounds to find the element with rank k in expectation and with probability at least $1/h^\delta$ for any constant $\delta < 1/2$.*

PROOF. For simplicity, we assume that B is a power of 2. We first prove the lower bound for $k = N$, i.e., for finding the median. For the state after t rounds, we define $I^*(t) = \min_{u_\alpha \in S_{uv}(t)} I(u_\alpha)$, $J^*(t) = \min_{v_\beta \in S_{vu}(t)} J(v_\beta)$, and $H^*(t) = \min\{I^*(t), J^*(t)\}$ as above and let $H^*(0) = \ell$. Assume that a given protocol $\mathcal{A}$ needs T rounds to find the median. We then have $H^*(0) = \ell$ and $H^*(T) = 0$. We define the progress of round t as $progress(t) := H^*(t) - H^*(t-1)$. In the following, we show that the progress of every round is at best geometrically distributed:

$$\forall t: \ \mathbb{P}\big[progress(t) \geq \xi\big] \leq \frac{2B}{2^\xi}. \tag{1}$$

Consider $\mathbf{state}_{u,v}(t-1)$ and $\mathbf{state}_{u,v}(t)$. By the definition of $H^*(t)$, in round t, either one of the B elements u_α which u sends to v satisfies $I(u_\alpha) = H^*(t)$ or one of the B elements v_β which v sends to u satisfies $J(v_\beta) = H^*(t)$. If $I(u_\alpha) = H^*(t)$, the $(\ell - H^*(t))$-highest priority bits of the base-2 representation of α equal $X_{H^*}, \ldots, X_{\ell-1}$. Similarly, if $I(v_\beta) = H^*(t)$, the $(\ell - H^*(t))$-highest priority bits of the base-2 representation of β equal the complements of $X_{H^*}, \ldots, X_{\ell-1}$. Therefore at least one of the $2B$ elements sent in round t contains all information about the values of X_i for $i \geq H^*(t)$. Because by Lemma 5.1, the combined state $\mathbf{state}_{u,v}(t-1)$ after round $t-1$ is independent of the random variables X_i for $H^*(t) \leq i < H^*(t-1)$, the probability to have $H^*(t) \leq H^*(t-1) - \xi$ is at most $2B/2^\xi$ which proves Inequality (1).

Let $P_t := \sum_{i=1}^{t} progress(t)$. If $\mathcal{A}$ finds the median in T rounds, we have $P_T = \ell$. For $t \neq t'$, the random variables $progress(t)$ and $progress(t')$ are not independent. However, by Lemma 5.1 and the above observation, it is possible to upper bound the random variables $progress(t)$ by independent random variables Z_t with $\mathbb{P}[Z_t = \log 2B + i - 1] = 1/2^i$ for $i \geq 1$. That is, we can define independent random variables Z_t such that $progress(t) \leq Z_t$, and for which

$$\forall t: \ \mathbb{P}\big[Z_t \geq \xi\big] \leq \frac{2B}{2^\xi}. \tag{2}$$

We can bound the probability that the number of rounds T to compute the median is lower bounded by some value τ by using a generalized Chernoff-type argument:

$$\begin{aligned}
\mathbb{P}[T \leq \tau] &= \mathbb{P}\left[\sum_{t=1}^{\tau} progress(t) \geq \ell\right] \\
&\leq \mathbb{P}\left[\sum_{t=1}^{\tau} Z_t \geq \ell\right] \\
&\underset{\gamma>0}{=} \mathbb{P}\left[e^{\gamma \cdot \sum_{t=1}^{\tau} Z_t} \geq e^{\gamma \cdot \ell}\right] && (3) \\
&\leq \frac{\mathrm{E}\left[e^{\gamma \cdot \sum_{t=1}^{\tau} Z_t}\right]}{e^{\gamma \cdot \ell}} = \frac{\mathrm{E}\left[\prod_{t=1}^{\tau} e^{\gamma \cdot Z_t}\right]}{e^{\gamma \cdot \ell}} && (4) \\
&= \frac{\prod_{t=1}^{\tau} \mathrm{E}\left[e^{\gamma \cdot Z_t}\right]}{e^{\gamma \cdot \ell}} && (5) \\
&= \frac{1}{e^{\gamma \cdot \ell}} \cdot \prod_{t=1}^{\tau} \sum_{\xi=0}^{\infty} \mathbb{P}[Z_t = \xi] \cdot e^{\gamma \cdot \xi} \\
&= \frac{1}{e^{\gamma \cdot \ell}} \cdot \prod_{t=1}^{\tau} \Bigg(\mathbb{P}[Z_t \geq 0] + \\
&\qquad \sum_{\xi=1}^{\infty} \mathbb{P}[Z_t \geq \xi] \cdot \left(e^{\gamma \cdot \xi} - e^{\gamma \cdot (\xi-1)}\right) \Bigg) \\
&\leq \frac{1}{e^{\gamma \cdot \ell}} \cdot \Bigg(1 + \sum_{\xi=1}^{\lfloor \log(2B) \rfloor} \left(e^{\gamma \cdot \xi} - e^{\gamma \cdot (\xi-1)}\right) + \\
&\qquad \sum_{\xi=\lfloor \log(2B) \rfloor + 1}^{\infty} \frac{2B}{2^\xi} \cdot \left(e^{\gamma \cdot \xi} - e^{\gamma \cdot (\xi-1)}\right) \Bigg)^{\tau} && (6) \\
&= \frac{1}{e^{\gamma \cdot \ell}} \cdot \Bigg(e^{\gamma \cdot \lfloor \log(2B) \rfloor} + \\
&\qquad \sum_{\xi=\lfloor \log(2B) \rfloor + 1}^{\infty} \frac{2B}{2^\xi} \cdot e^{\gamma \cdot \xi} \cdot \left(1 - \frac{1}{e^\gamma}\right) \Bigg)^{\tau}.
\end{aligned}$$

Equation (3) holds for all $\gamma > 0$, the inequality in (4) is obtained by applying the Markov inequality, Equation (5) follows from the independence of different Z_t, and Inequality (6) is a consequence of Inequality (2). By setting $\gamma = \ln(\sqrt{2})$, we obtain

$$\begin{aligned}
\mathbb{P}[T \leq \tau] &\leq \frac{1}{2^{\ell/2}} \cdot \left(\sqrt{2B} + \frac{2\sqrt{2B}}{2 - \sqrt{2}} \cdot \left(1 - \frac{1}{\sqrt{2}}\right) \right)^{\tau} \\
&= \frac{(8B)^{\tau/2}}{2^{\ell/2}}.
\end{aligned}$$

If we substitute $\tau = \log_{8B}(N)/c$ for a constant $c > 1$, we then get

$$\begin{aligned}
\mathbb{P}\left[T \leq \frac{1}{3c} \cdot \log_{2B}(N)\right] &\underset{B \geq 1}{\leq} \mathbb{P}\left[T \leq \frac{\log_{8B}(N)}{c}\right] \\
&\leq \frac{(8B)^{1/2 \cdot \log_{8B}(N)/c}}{2^{\log(N)/2}} \\
&= \frac{1}{N^{\frac{1-1/c}{2}}},
\end{aligned}$$

which proves the claimed lower bound for finding the median by a deterministic algorithm. The lower bound for randomized algorithms follows by applying Yao's principle. Note that the lower bound for randomized algorithms could also

be proven directly in exactly the same way as the deterministic lower bound. This would however require to include randomness in all the definitions.

To obtain the lower bound for selecting an element with arbitrary rank $k < N$, we show that finding the element with rank $k < N$ is at least as hard as finding the median if both nodes start with k instead of N values. To do so, we assign elements $u_1, \ldots, u_k$ and $v_1, \ldots, v_k$ to u and v as described above such that finding the median of the $2k$ elements is difficult. The remaining elements are assigned such that $u_i \prec u_j$, $u_i \prec v_j$, $v_i \prec u_j$, and $v_i \prec v_j$ for all $i \leq k$ and $j > k$. If $k > N$, we get the lower bound by lower bounding the time to find the k^{th} smallest element with respect to the complementary ordering relation. □

Based on the above lower bound for generic two-party protocols, we can now prove a lower bound for generic selection algorithms on general graphs. In the following, we assume that every node of a graph with n nodes starts with one element and that we have to find the k^{th} smallest of all n elements. In every round, every node can send B elements to each of its neighbors. The next theorem shows how to reduce the problem on general graphs to the two-party problem.

THEOREM 5.3. *For every $n \geq D \geq 3$, there is a graph $G(D)$ with n nodes and diameter D such that $\Omega(D \log_D \min\{k, n-k\})$ rounds are needed to find the k^{th} smallest element in expectation and with probability at least $1/(\min\{k, n-k\})^{\delta}$ for every constant $\delta < 1/2$. In particular, finding the median requires time at least $\Omega(D \log_D n)$.*

PROOF. We can certainly assume that $n = \omega(D)$ since even finding the minimum of two elements or the median of three values requires $\Omega(D)$ rounds. Without loss of generality, we can also assume that $k \leq n/2$. For $k > n/2$, the theorem then follows by symmetry. As in the two-party case, we first consider only deterministic algorithms and apply Yao's principle to obtain the lower bound for randomized algorithms.

For simplicity, assume that $n - D$ is an odd number. Let $N = (n - D + 1)/2$. We consider the graph $G(D)$ defined as follows: The graph $G(D)$ consists of two nodes u and v that are connected by a path of length $D - 2$ (i.e., it contains $D - 1$ nodes). In addition, there are nodes $u_1, \ldots, u_N$ and $v_1, \ldots, v_N$ such that u_i is connected to u and v_i is connected to v for all $i \in \{1, \ldots, N\}$.

We can assume that only the leaf nodes u_i and v_i for $i \in \{1, \ldots, N\}$ hold an element and that we need to find the k^{th} smallest of these $2N$ elements. We can simply assign dummy elements which are larger than these $2N$ elements to all other nodes. Since only the leaves start with an element, we can assume that in the first round, all leaves u_i send their element to u and all leaves v_i send their element to v, as this is the only possible useful communication in the first round. By this, the problem reduces to finding the k^{th} smallest element of $2N$ elements on a path of length $D-2$ if initially each of the two end nodes u and v of the path holds N elements. Note that the leaf nodes u_i and v_i of $G(D)$ do not need to further participate in a distributed selection protocol since u and v know everything their respective leaves know and can locally simulate all actions of their leaf nodes.

Let w be any node on the path and let $\mathcal{M}_w(t)$ be a message that w sends in round t. Because we consider deterministic algorithms and because only u and v initially hold elements, $\mathcal{M}_w(t)$ can be computed when knowing the elements u and v send to their neighbors in rounds prior to t. In fact, since information needs time $d(w, w')$ to be transmitted from a node w to a node w', $\mathcal{M}_w(t)$ can be computed when knowing all elements u sends in rounds prior to $t - d(u, w) + 1$ and all elements v sends in rounds prior to $t - d(v, w) + 1$, where $d(w, w')$ denotes the distance between two nodes w and w' of the path connecting u and v. In particular, u and v can compute their own messages of round t when knowing the elements sent by v and u in rounds prior to $t - (D - 2) + 1$, respectively.

Let us consider the following alternative model. A round lasts $D - 2$ time units. In every round, u and v can send a message containing $D - 2$ elements to all other nodes of the path (they need to send the same message to all nodes). By the above observation, u and v can send all messages of rounds r for $(i-1) \cdot (D-2) < r \leq i \cdot (D-2)$ in round i of the alternative model. Additionally, after round i in the alternative model, u and v can locally compute all communication on the path in rounds $r \leq i \cdot (D-2)$. The time needed to find the k^{th} smallest element in the alternative model is therefore upper bounded by the time needed in the original model. Hence, a lower bound for the alternative model implies a lower bound for the original model. However, the time needed in the alternative model is exactly the time needed by a two-party protocol if every round lasts $D-2$ time units and if in every round $D-2$ elements can be transmitted. Applying Theorem 5.2, $n = \omega(D)$, and therefore $2N = n(1 - o(1))$, this completes the proof. □

Remark: We assumed that every node starts the algorithm with exactly one element. Note that results can easily be generalized to more general settings. If the total number of elements in the system is $N \geq n$, we then obtain an $\Omega(D \log_D \min\{k, N-k\})$ time lower bound for finding the k^{th} smallest value.

6. REFERENCES

[1] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time Bounds for Selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[2] F. Y. L. Chin and H. F. Ting. An Improved Algorithm for Finding the Median Distributively. *Algorithmica*, 2:77–86, 1987.

[3] G. N. Frederickson. Tradeoffs for Selection in Distributed Networks. In *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–160, 1983.

[4] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, pages 131–146, 2002.

[6] J. M. Marberg and E. Gafni. An Optimal Shout-Echo Algorithm for Selection in Distributed Sets. In *Proc. 23rd Allerton Conf. Comm., Control, and Computing*, 1985.

[7] A. Negro, N. Santoro, and J. Urrutia. Efficient Distributed Selection with Bounded Messages. *IEEE Trans. Parallel and Distributed Systems*, 8(4):397–401, 1997.

[8] B. Patt-Shamir. A Note on Efficient Aggregate Queries in Sensor Networks. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 283–289, 2004.

[9] M. Rodeh. Finding the Median Distributively. *J. Computer and System Science*, 24(2):162–166, 1982.

[10] D. Rodem, N. Santoro, and J.B. Sidney. Shout-Echo Selection in Distributed Files. *Networks*, 16:235–249, 1986.

[11] N. Santoro, M. Scheutzow, and J. B. Sidney. On the Expected Complexity of Distributed Selection. *J. Parallel and Distributed Computing*, 5(2):194–203, 1988.

[12] N. Santoro, J. B. Sidney, and S. J. Sidney. A Distributed Selection Algorithm and Its Expected Communication Complexity. *Theoretical Computer Science*, 100(1):185–204, 1992.

[13] N. Santoro and J.B. Sidney. Order Statistics on Distributed Sets. In *Proc. 20th Allerton Conf. Comm., Control, and Computing*, pages 251–256, 1982.

[14] N. Santoro and E. Suen. Reduction Techniques for Selection in Distributed Files. *IEEE Trans. Computers*, 38(6):891–896, 1989.

[15] A. Schönhage, M. S. Paterson, and N. Pippenger. Finding the Median. *Journal of Computer and System Sciences*, 13:184–199, 1976.

[16] L. Shrira, N. Francez, and M. Rodeh. Distributed k-Selection: From a Sequential to a Distributed Algorithm. In *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–153, 1983.

[17] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.

[18] J. Zhao, R. Govindan, and D. Estrin. Computing Aggregates for Monitoring Wireless Sensor Networks. In *Proc. 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003.