Using the Crowd for Top-k and Group-by Queries

Susan B. Davidson
University of Pennsylvania
susan@cis.upenn.edu

Sanjeev Khanna
University of Pennsylvania
sanjeev@cis.upenn.edu

Tova Milo
Tel Aviv University
milo@cs.tau.ac.il

Sudeepa Roy*
University of Washington
sudeepa@cs.washington.edu

ABSTRACT

Group-by and top-k are fundamental constructs in database queries. However, the criteria used for grouping and ordering certain types of data – such as unlabeled photos clustered by the same person ordered by age – are difficult to evaluate by machines. In contrast, these tasks are easy for humans to evaluate and are therefore natural candidates for being crowd-sourced.

We study the problem of evaluating top-k and group-by queries using the crowd to answer either *type* or *value* questions. Given two data elements, the answer to a type question is "yes" if the elements have the same type and therefore belong to the same group or cluster; the answer to a value question orders the two data elements. The assumption here is that there is an underlying ground truth, but that the answers returned by the crowd may sometimes be erroneous. We formalize the problems of top-k and group-by in the crowd-sourced setting, and give efficient algorithms that are guaranteed to achieve good results with high probability. We analyze the crowd-sourced cost of these algorithms in terms of the total number of type and value questions, and show that they are essentially the best possible. We also show that fewer questions are needed when values and types are correlated, or when the error model is one in which the error decreases as the distance between the two elements in the sorted order increases.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms And Problem Complexity**]: General; H.2.0 [**Database Management**]: General

General Terms

Algorithms, Theory

Keywords

crowd sourcing, top-k, group by, clustering, lower bounds

*This work was done while the author was at the University of Pennsylvania.

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy

1. INTRODUCTION

Consider the following problem: You have a massive database of photos. Currently, the photos are tagged and organized along a dimension of geo-location. However, many of these photos contain people and for those that are focused on a single person – Alice standing in front of Niagara Falls or Bob in front of the Louvre – you wish to be able to group the photos by person. Furthermore, you wish to find the most recent photo of each person, or alternatively the 5 most recent photos of each person. Thus one of the queries you wish to run over the photo database is:

```
SELECT most-recent(photo)
FROM photoDB
WHERE singlePerson(photo)
GROUP BY Person(photo)
```

This query contains several (non-traditional) functions, *most-recent*, *singlePerson*, and the grouping function *Person*. While some of the functions in this query could be done by your photo processing software – for example, *singlePerson*, or allowing you to annotate faces in photos with names that could be used for grouping – it may not have been done for all the photos in your database. The annotation results are also not impressive for a time span of 20 years when the person has aged from babyhood to adult. Furthermore, the date field is missing in many of the photos or is not completely trustworthy as some people forget to set the date and time when they buy a new camera. You therefore decide to ask the database to crowd-source some of the functions in this query, in particular the group-by function (photos of the same person) and the "max" function *most-recent* (returning the most recent photo of each person).

In this paper, we formally study the problem of evaluating such max/top-k and group-by queries using the crowd to answer either *type* or *value* questions. Given two data elements, the answer to a type question is "yes" if the elements have the same type and therefore belong to the same group or cluster; the answer to a value question orders the two data elements. The assumption here is that there is an underlying ground truth that the crowd is expected to know. However, due to differing skill levels or the amount of time and effort spent, the answers returned by the crowd may sometimes be erroneous. That is, the crowd may not correctly identify pictures as being of the same person (type) or of one picture of a person being more recent than another picture of that same person (value).

There are several ways of modeling this human error. In the first and the more standard one, we assume that each type or value question is answered correctly by the crowd with a constant probability $> \frac{1}{2}$ (*constant error model*). We then propose a more interesting

error model, called the *variable error model*, in this paper. Here the error is related to how close the elements are in the ordering of interest. For example, if a value question involves two pictures of the same person, one of which is them as a baby and the other of which is them at their high school graduation it is easy to decide which is the most recent. However, if the value question involves two pictures of the same person separated by a week, it is much harder to decide which is the most recent.

Using these error models, we formalize the max/top-k problems and clustering problems for group-by queries in the crowd-sourced setting, and give efficient algorithms that are guaranteed to achieve the desired results with high probability. For max/top-k, we ask value questions to the crowd, whereas for clustering, we ask type questions. Asking questions to the crowd using platforms like Amazon Mechanical Turk [6] involves monetary cost. Therefore, we analyze the crowd-sourced cost of these algorithms in terms of the total number of type and value questions asked to the crowd.

There are scenarios where type and value questions are needed to be used together, *e.g.* in the database query given above, where we want to find max/top-k elements from each cluster. However it turns out that, even for the simple clustering problem when the types and values of the elements are correlated, both type and value questions can be used to reduce the total number of questions. Note that in our example photo database, there is no correlation between the type and value questions; not all "older" people are the same person.[1] However, there are other examples in which there is such a correlation. For example, suppose that you have a database of hotels in a city. If the hotels were to be clustered by districts, then there would be a high correlation between the district and the quality (luxury level or star ratings) of a hotel. In this case, a value question – the luxury level of a hotel – would give some information about its type, *i.e.* the district of the hotel. Similar correlation is expected to exist when the value of a hotel in a given district is its (average/maximum) price, and the type is its quality. We show that this type of correlation can be used to advantage.

Contributions. We study max/top-k and clustering problems in the crowd-sourced setting. Here is a summary of our contributions.

- We propose a *variable error model* for erroneous answers from the crowd. Suppose the two elements being compared by a value question is Δ apart in the sorted order on values. Then the probability of error is $\leq \frac{1}{f(\Delta)}$ for a monotone non-negative error function f, *i.e.* the error in the answer decreases when the distance Δ between the elements increases. We study the max/top-k problems under this model. For max and top-k with small values of k, we show that only $n + o(n)$ value questions are sufficient to find the answers with high probability given any strictly monotone error function f. Here $o(n)$ denotes a function of n that is strictly asymptotically smaller than the linear function.

- For the general clustering problem, we give a lower bound of $\Omega(nJ)$ where J is the number of clusters, even for randomized algorithms, even when the answers to the type questions are always correct, and the value of J is known by the algorithm. We show that this bound is essentially the best possible by providing a simple algorithm that compares $O(nJ)$ pairs of elements by type questions (if the answers to the type questions are erroneous, the number of questions increases by a factor of $O(\log n)$ to output the clusters with high probability). We also note that our algorithm does not require an a priori knowledge of J.

- We formalize the scenario when the types and values are correlated, and show that $O(n \log J)$ type and value questions in total (with additional logarithmic factors for erroneous answers) are sufficient when elements from same types form contiguous blocks in the sorted order (called the *full correlation case*). In the *partial correlation case*, there are at most α changes in types between any two element of the same type in the sorted order by value. We show that in this case $O(\alpha J + n \log(\alpha J))$ questions (with logarithmic factors) suffice. We also discuss the problem of finding max/top-k elements from each cluster.

Outline of paper. We review related work in Section 2. Section 3 presents the model and definitions that will be used throughout the paper. Results on max and top-k (*value* comparisons) for the variable error model are given in Section 4, while results on clustering (*type* comparisons) are given in Section 5. In Section 6, we consider the case when values and types are correlated, and give an improved clustering algorithm using both type and value questions. We conclude in Section 7.

2. RELATED WORK

Crowd-sourcing is a topic of recent interest to the database community, and has been suggested as a method for data cleansing, data integration, entity resolution, schema expansion, and data analytics see *e.g.* [10, 17, 19, 1, 11]. Different crowd-sourced databases like CrowdDB [8], Deco [15], Qurk [12], AskIt![4] have been built. These systems help decide which questions should be asked of the crowd, taking into account various requirements such as latency, monetary cost, quality, etc.

The problem of finding max in the crowd-sourced setting has been considered in [10]. However, instead of finding the maximum element exactly, [10] focuses on the *judgment problem:* given a set of comparison results, which element has the maximum likelihood of being the maximum element, and the *next vote problem:* given a set of results, which future comparisons will be most effective. Finding the exact solution to these problems is shown to be hard, and efficient heuristics are proposed that work well in practice. Finding max in crowd-sourced settings has also been considered in [18]. It provides efficient heuristics and evaluate them empirically which can be tuned using parameters like execution time, cost, and quality of the result. The more general sorting problem is considered in [13] which aims to minimize the cost of asking questions using the Qurk system by optimization techniques like batching and replacing pairwise comparisons by numerical ratings. The focus of [13] is mainly on handling different implementation aspects in contrast to obtaining rigorous theoretical results which is the goal of this paper.

On the other hand, the problem of finding max/top-k elements in the presence of noisy comparison operators has been extensively studied in the the theory community (see the references in [18, 10]). Our work is closest to [7] which considers the more standard constant error model; we compare our work with [7] in detail in Section 4, and show that we obtain better bounds for max and for small values of k in top-k when the error function in the variable error model is strictly monotone.

[1]The most recent photo of a person is the one in which they are the oldest.

The clustering problem in the crowd-sourced setting has been considered in [9]. However, the goal of [9] is to *define* cluster types instead of only *classifying* the elements. It also assumes that workers may have different clustering criteria and each worker only has a partial view of the data. This paper proposes a Bayesian model of how the workers can approach clustering. This is in contrast to our model where we assume that there is a fixed (but unknown) set of clusters partitioning the elements. Previous work has also considered filtering data items based on a set of properties that can be manually verified [16], and also entity resolution using crowd-sourcing [19]. However, to the best of our knowledge, our work is the first one that formally considers clustering in the crowd-sourced setting assuming a ground truth on the clusters, and studies the correlation between types and values.

3. PRELIMINARIES

In this section, we present some preliminary notions and formally define the problems that we study in this paper.

There are n elements $x_1, \cdots, x_n$. Each element x_i is associated with a *type* (denoted by $\texttt{type}(x_i)$) and a *value* (denoted by $\texttt{val}(x_i)$). We denote by J, the number of distinct types. Types induce a partition of the elements into J clusters where each cluster contains elements of the same type. The clusters are *balanced* when the ratio of maximum and minimum cluster size is bounded by some given constant c, *i.e.* when each cluster has about n/J elements. The values of the elements are distinct and there is a total order on the values, *i.e.* for any two elements x_i and x_j, either $\texttt{val}(x_i) > \texttt{val}(x_j)$ or $\texttt{val}(x_j) > \texttt{val}(x_i)$. For simplicity, we will use $x_i > x_j$ instead of $\texttt{val}(x_i) > \texttt{val}(x_j)$ when it is clear from the context.

In the `photoDB` example mentioned in the introduction, each photo is an element. The type of a photo is the person appearing in the photo, while the value of a photo is the age of the person in that photo (or, the date when the photo was taken). The number of clusters J is the number of distinct people in the database who appear in a `singlePerson` photo. Since some people may appear in many more `singlePerson` photos than others, the clusters are not necessarily balanced. Note that the name or age of the individuals may not be explicitly recorded in the photos, but that there is an underlying ground truth.

Group-by and top-k database queries. Consider the elements as tuples in a relational setting. The types and values of the elements can be assumed to be two different attributes of the tuples which are not explicitly mentioned in the database. The goal of a *group-by query* is to group together elements having same types, *i.e.* to find the clusters mentioned above. An example group-by query that clusters the photos based on the individuals appearing in them is the following (for simplicity assume that group photos have already been filtered out and *ind-photoDB* contains photos focusing on individual people):

```
SELECT Person(photo)
FROM ind-photoDB
GROUP BY Person(photo)
```

On the other hand, in a *top-k query*, predicates can be added in the select clause to find the maximum or k elements having the highest values for each cluster in the query (the smallest elements can be found similarly). As an example, consider the scenario where a person accesses only her photos in the database and wants to find the most recent photo, or where someone wants to find the most recent photo of a place of interest given a number of such photos. Both can be answered by the following query (where *my-photoDB* represents the set of photos of interest) :

```
SELECT most-recent(photo)
FROM my-photoDB
```

Here value of a photo is the date when it was taken (or the age of the person was at the time the photo was taken). In particular, group-by and top-k queries can be combined in the natural way when we want to find top-k/maximum element from each cluster, as in the example database query given in the introduction.

Questions for the crowd. As mentioned in the introduction, the wisdom of the crowd can be used to compute the functions "`GROUP BY` Person(photo)" or "*most-recent(photo)*" in the above queries. This is done by posing *questions* to the crowd that ask them to compare types or values of two elements, followed by (possibly in rounds) some computation performed within the system. In our model, the crowd can be asked either a *type question*: given two elements x_i and x_j, whether $\texttt{type}(x_i) = \texttt{type}(x_j)$, or a *value question*: given two elements x_i and x_j, whether $\texttt{val}(x_i) > \texttt{val}(x_j)$. Note that the answers to these questions are always "yes" or "no", and we cannot ask what the type or value of a given element is. This is motivated by the examples given, since the crowd may not know the exact date when the photo was taken, who the person is in the photos they are shown, or where the places of interest are. From now on, we use "queries" to denote database group-by or top-k queries, and "questions" or "comparisons" to denote the type or value questions asked to the crowd.

Problem definitions. The problems studied in this paper are:

(i) **Max and top-k**: Here our goal is find the maximum, and in general the top-k elements having the highest values in order to compute top-k/max functions in queries. That is, assuming without loss of generality that $x_1 > x_2 > \cdots, x_n$, we want to find x_1 (for max) or $x_1, \cdots, x_k$ (top-k) using value questions.

(ii) **Clustering**: Here we want to find the J clusters using type questions grouping together elements having the same type. This allows us to find the groups in group-by queries.

(iii) **Clustering with correlated types and values:** The problem stated above uses only type questions to cluster the elements. However, sometimes the types and values of elements are highly correlated. For instance, consider average/maximum price of hotels (or rooms in a hotel) of a district vs. their quality. Here the elements are hotels or rooms in the same hotel; quality of a room or a hotel (luxury level or star ratings) captures its type, and the value is the (average) price of the room or the hotel. When we sort rooms in the same hotel according to their prices, it is likely that rooms of similar quality will form a contiguous block in the sorted order. This we call the *full correlation case*. On the other hand, when we sort the hotels according to their average prices, the hotels of similar quality are expected to form "almost" contiguous blocks in the sorted order (due to other factors like location). This we call the *partial correlation case*. In this problem, once again we want to find the J clusters for group-by queries. However, we use both type and value questions in order to exploit any correlation as above between types and values.

The above intuition is formalized as follows. Suppose $x_1 > x_2 > \cdots > x_n$. There are at most α changes in types of elements in the sorted order between any two elements of the same type for some value of α. Formally, consider any two elements $x_i > x_j$ such that $\texttt{type}(x_i) = \texttt{type}(x_j)$. There exists a value of α, $\alpha \in [1, n-1]$[2], such that between x_i and x_j there are at most $\alpha - 1$ elements x_ℓ, $i \leq \ell < j$, where $\texttt{type}(x_\ell) \neq \texttt{type}(x_{\ell+1})$. For instance, when $\texttt{type}(x_1) = \texttt{type}(x_n)$, and all other elements have distinct types, $\alpha = n - 1$. On the other hand, when $\alpha = 1$, we get the full correlation case. In general, when the value of α is small, the clusters are nearly sorted according to the values of the elements in them, and we get the partial correlation case. We will show that fewer number of type and value questions in total are needed compared to the number of type questions in (ii) when the value of α is small. We also discuss how we can find top-k elements from each cluster using both type and value questions (for queries using both top-k/max function and `GROUP BY` clause).

A dominant cost factor in crowd sourcing applications is the number of questions being asked.[3] This is because answering may incur monetary cost, involves human efforts, and may be slow. Therefore, we will provide upper and lower bounds on the cost to solve the above problems by counting the total number of type and value questions.

Error model. Although each element has a fixed type and a fixed value, due to differing skill levels, or the amount of time and effort spent, the answers returned by the crowd may be erroneous. We first model the potential noisy answers by a simple and standard *error model for the questions*: both type and value questions are answered correctly with probability $\geq \frac{1}{2} + \epsilon$, where $0 < \epsilon \leq \frac{1}{2}$ is a fixed constant[4]. This ensures that the answer returned by the crowd is always correct with higher probability than a random "yes" or "no" answer returned with probability $\frac{1}{2}$. We call this the *constant error model*.

For the max/top-k problem, we will see the effect of a more refined *variable error model* for value questions, where the probability of error decreases when two elements that are far apart in the total order on values are compared. For instance given two photos of an individual, it is easier for the crowd to decide which one has been taken earlier if the time difference between the photos is 10 years than when the photos are taken a week apart. We formalize this concept as follows: A function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ [5] is *monotone* (respectively, *strictly monotone*) if for al $n_1 \geq n_2$, $f(n_1) \geq f(n_2)$ (respectively, $f(n_1) > f(n_2)$). Given two distinct elements x_i, x_j such that $x_i > x_j$

$$\Pr[x_j \text{ is returned as the larger element}] \leq \frac{1}{f(j-i)} - \epsilon \qquad (1)$$

where f is a monotone function, $f(1) \geq 2$, and $\epsilon > 0$ is a constant. The conditions $f(1) \geq 2$ and $\epsilon > 0$ ensure that even if x_i and x_j are consecutive elements in the total order, *i.e.* $j = i + 1$, then the probability of making the right decision is also strictly greater than $\frac{1}{2}$. Note that when $f(\Delta) = 2$ for all inputs Δ, the variable error model is the same as the constant error model for value questions[6].

The function f is called the *error function*, and can have different rate of growth depending on the dataset and skill level of the crowd. For instance, when the set of n photos spans a time line of few weeks, the probability of error in ordering them according to age of people is likely to be high even when the oldest and most recent photos are compared. However, if the n photos span a time line of over 20 years, the probability of error is much smaller when the first and last photos are compared. In practice, the nature of error can be estimated by asking a test set of questions to the crowd on a small set of elements for which the correct answer is already known, prior to asking the actual questions for finding max/top-k. In Section 4, we will study the effect of different error functions on the number of questions asked to the crowd.

Error in the answers to value and type questions results in errors in the answer to a database query. Therefore, we seek to find solutions to the queries that are correct with high probability: given any constant $\delta > 0$, our goal is to find the exact max/top-k elements or the exact clusters with probability $1 - \delta$.

We will frequently use the standard notation $O()$ and $\Omega()$ for asymptotic upper and lower bounds, and $o()$ and $\omega()$ for strict asymptotic upper and lower bounds respectively [5]. We will also use the notation $\tilde{O}()$ or $\tilde{\Theta}()$ which hides the associated logarithmic terms in an expression. All logs used in the paper refer to logarithms with base 2. Next we discuss the three problems mentioned above in detail in Sections 4, 5 and 6 respectively.

4. MAX AND TOP-K

The problem of finding the maximum and top-k elements with faulty comparisons has been thoroughly studied in [7] under the constant error model. When value comparisons are performed by the crowd, the probability of error is likely to be less when two elements far apart in the sorted order are compared, which can reduce the number of questions asked to the crowd. This motivates the study of max and top-k problems under the variable error model which is formalized in Equation (1). In this section, we show that when the error function f in Equation (1) is strictly monotone, only $n + o(n)$ value questions are sufficient to find the max or the top-k elements for a small value of k (which is typically the case in practice) with high probability. We start with the algorithm for finding max in Section 4.1, and then use that algorithm as building block to find the top-k elements in Section 4.2.

4.1 Finding Max

Suppose $x_1 > x_2 > \cdots > x_n$. Given $\delta > 0$, we want to find x_1 with probability $\geq 1 - \delta$ using a small number of value questions. When the answers to value questions are correct, $n - 1$ questions are necessary and sufficient to find x_1. On the other hand in the constant error model, where each value question is answered correctly with probability $\geq \frac{1}{2} + \epsilon$ for a constant ϵ, [7] gives a simple algorithm to find the maximum with probability $\geq 1 - \delta$ using $O(n \log \frac{1}{\delta})$ questions (we sketch the algorithm later). They also show that this bound is tight as stated in the following theorem:

THEOREM 1. *[7] For all $\delta \in (0, \frac{1}{2})$, $\Theta(n \log(\frac{1}{\delta}))$ value questions (comparisons) are both sufficient and necessary to find the*

[2]For positive integers a, b, where $a \leq b$, $[a, b]$ denotes the interval $a, a+1, \cdots, b$.

[3]There are other natural cost function like the *latency* or *number of rounds of questions* asked of the crowd, which we leave as future work.

[4]The results obtained are similar if we assume two different ϵ values for type and value questions.

[5]$\mathbb{N}, \mathbb{R}$, and $\mathbb{R}^{\geq 0}$ respectively denote the set of natural numbers, the set of real numbers, and the set of non-negative real numbers.

[6]Our results also hold for an alternative 'value-based' error model when adjacent values in the sorted order are well-separated (see Section 7).

Algorithm 1 Algorithm for finding the maximum element.

1: – Choose a random permutation Π of the elements $x_1, \cdots, x_n$.
2: **for** levels $L = 1$ to $\log n$ in the comparison tree **do** { *leaves are in level 0, the root is in level* $\log n$* }
3: – If $L \leq \log X$ (lower $\log X$ levels), do one comparison at each internal node. Propagate the winners to the level above.
4: – If $L > \log X$ (upper $\log \frac{n}{X}$ levels), do N_L comparison at each internal node. Take majority vote and propagate the winners to the level above.
5: **end for**
6: **return** The element at the root node of the comparison tree.

maximum with probability $\geq 1 - \delta$ *in the constant error model.*

Moreover, the proof of the lower bound shows a stronger result when $\delta + \epsilon \leq \frac{1}{2}$, *i.e.* when a high probability of success is needed in spite of a high probability of error.

OBSERVATION 4.1. *[7] There exists a constant* $c > 0$ *such that at least* $(1+c)n$ *comparisons are needed to compute the maximum with probability* $1 - \delta$ *for any* δ, ϵ *satisfying* $\delta + \epsilon \leq \frac{1}{2}$.

In this section we show that a much better upper bound can be obtained in the variable error model that almost matches both the upper bound of $n - 1$ comparisons when the value questions are correctly answered, and the lower bound of $(1 + c)n$ stated in Observation 4.1. Recall the probability of error for value questions in the variable error model in terms of the error function f given in (1) in the previous section. The following theorem shows that for *all* strictly monotone functions f, $n + o(n)$ questions suffice (for constant ϵ, δ) to find the maximum with high probability. Further, the number of questions improves to $n + O(\log \log n)$ when f is at least linear (*i.e.* $f(\Delta) = \Omega(\Delta)$) and to $n + O(1)$ when f is exponential ($f(\Delta) = 2^{\Delta}$).

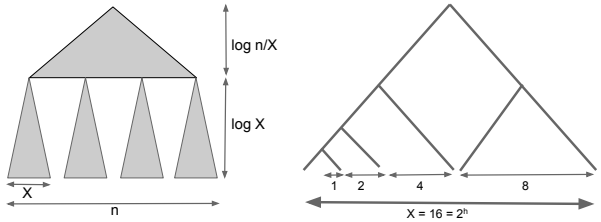
THEOREM 2. *For all strictly growing functions* f *and constant* $\epsilon, \delta > 0$, $n + o(\frac{n}{\delta} \log \frac{1}{\delta})$ *value questions are sufficient to output the maximum element* x_1 *with probability* $\geq 1 - \delta$ *in the variable error model.*

Further, if $f(\Delta) = \Omega(\Delta)$, *then* $n + O(\frac{\log \log n}{\delta^2} \log \frac{1}{\delta})$ *questions are sufficient. If* $f(\Delta) = 2^{\Delta}$, *then* $n + O(\log^2 \frac{1}{\delta})$ *questions are sufficient.*

Next we present our algorithm and prove the bounds given in the theorem.

Our Algorithm. Our algorithm uses the tournament approach using a *comparison tree*. The key idea is to choose a random permutation of the elements $x_1, \cdots, x_n$ which appear as leaves in the tree. These leaves are grouped into $\frac{n}{2}$ pairs, the two elements in each pair are compared using value questions, and the winners (larger elements) propagate to the level above. This is continued until only one element remains as the root of the tree which is declared as the maximum.

We divide the $\log n$ levels of the comparison tree into upper $\log \frac{n}{X}$ levels and lower $\log X$ levels (see Figure 1a). In the lower levels, only one value comparison is performed at each internal node. In the upper levels $L = \log X + 1$ to $\log n$, N_L comparisons are performed at each internal node, and a majority vote is taken to decide the larger element. Algorithm 1 presents our method; the parameters X and N_L will be calculated later depending on the nature of error function f.



(a) (b)

Figure 1: (a) General framework of Algorithm 1 with a comparison tree, (b) Amplified single X-tree with X nodes and (lower) $\log X$ levels.

Analysis. The total number of comparisons performed by the algorithm is

$$n - \frac{n}{X} + \sum_{L=\log X+1}^{\log n} N_L \leq n + \sum_{L=\log X+1}^{\log n} N_L$$

We will show that the maximum element x_1 is returned with probability $\geq 1 - 6\delta$; to obtain the desired $1 - \delta$ probability as stated in Theorem 2, the algorithm is run with $\delta' = \delta/6$.

We analyze the upper $\log \frac{n}{X}$ levels and the lower $\log X$ levels separately: (i) in the upper levels, we use the algorithm from [7] which returns the maximum element with probability $\geq 1 - \delta$; (ii) in the lower levels, we show that x_1 does not lose in any comparison with probability $\geq 1 - 5\delta$, even when only one comparison is performed at each internal node in the lower levels. Therefore, by union bound, the maximum element x_1 will be returned with probability $\geq 1 - 6\delta$. Moreover, the chosen value of X ensures that the number of comparisons in the upper levels in $o(n)$ (for constant ϵ, δ) for any strictly monotone error function f.

Analysis of the Upper Levels. For the upper $\log \frac{n}{X}$ levels, we simply use the fact that each value question is answered correctly with probability $\geq \frac{1}{2} + \epsilon$ irrespective of the function f. Then we can use the algorithm and bounds given in [7] for constant error model (see Theorem 1). We briefly sketch the algorithm for the sake of completeness.

Consider the sub-tree with the upper $\log \frac{n}{X}$ levels, the number of nodes in the subtree is $\frac{n}{X}$. Each internal node in the levels $\ell = 1$ to $\log \frac{n}{X}$ uses $S_\ell = (2\ell - 1) * O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ comparisons, and $N_L = S_{L - \log X}$. It is easy to verify that $\sum_{L=\log X+1}^{\log n} N_L = O(\frac{n}{X} \log \frac{1}{\delta})$ for constant $\epsilon > 0$ (see the appendix). Therefore, given constant $\epsilon, \delta > 0$, to find the maximum element in the upper $\log \frac{n}{X}$ levels with probability $\geq 1 - \delta$,

$$\text{it suffices to ask } O(\frac{n}{X} \log \frac{1}{\delta}) \text{ value questions} \qquad (2)$$

Analysis of the Lower Levels. The expression in (2) bounds the number of comparisons in the upper $\log \frac{n}{X}$ levels; the number of comparisons in the lower $\log X$ levels is bounded by n. Next we show that there exists a value of X such that $\frac{n}{X} = o(\frac{n}{\delta})$ for any strictly monotone function f, and the maximum element does not lose in any comparison with probability $\geq 1 - 5\delta$ in the lower levels.

Algorithm 1 starts with a random permutation Π of the elements

$x_1, \cdots, x_n$. Let us partition Π into block of size (at most) X of consecutive elements. Let us call the sub-trees of the comparison tree in the bottom $\log X$ levels on each block of X elements an *X-tree* (the sub-trees in Figure 1a); the number of X-trees is $\frac{n}{X}$. The algorithm performs only one comparison at each non-leaf node of each X-tree.

Consider the X-tree that contains the maximum element in Figure 1b. Without loss of generality, assume that the leftmost leaf is the maximum element x_1. Consider the left-most path of length h to the root of the X-tree. We will compute the probability that x_1 is never eliminated along this path.

The height of the X-tree is $h = \log X$. Let $r_\ell, \ell \in [1, h]$, be a non-leaf node on the left-most path of the X-tree. The right subtree of r_ℓ will have $2^{\ell-1}$ leaf nodes (see Figure 1b). Note that, if x_1 survives all the comparisons in levels 1 to $\ell - 1$, in the internal node r_ℓ, x_1 can only be compared with the nodes in the right subtree of r_ℓ. For parameters Δ_ℓ, $\ell \in [1, h]$, to be decided later, we will bound the following probabilities:

1. $\delta_\ell = \Pr[$ At least one leaf in the right subtree of r_ℓ corresponds to an element from the set $\{x_j : 2 < j \leq \Delta_\ell + 1\}]$.
2. p_ℓ = $\Pr[x_1$ loses the comparison at node r_ℓ, when none of the leaves in the right subtree of r_ℓ correspond to an element from the set $\{x_j : 2 < j \leq \Delta_\ell + 1\}]$.

In particular, we prove the following proposition:

PROPOSITION 4.1. *There exist values of $h = \log X$ and Δ_ℓ, $\ell \in [1, h]$ such that $\sum_{\ell=1}^{h} \delta_\ell + \sum_{\ell=1}^{h} p_\ell \leq 5\delta$ and $\frac{n}{X} = o(\frac{n}{\delta})$ for any monotone error function f.*

It follows from Proposition 4.1 (using union bound) that the maximum element x_1 cannot lose any comparison in the lower levels. Next, we choose values of h and Δ_ℓ, $\ell \in [1, h]$, and prove Proposition 4.1 (see Lemmas 4.1 and 4.3 below).

Since Π is a random permutation, given a fixed position of x_1, any of the $n - 1$ elements other than x_1 can appear in another given position in Π with probability $\frac{1}{n-1}$. Therefore by union bound,

$$\delta_\ell \leq \Delta_\ell 2^{\ell-1}/(n-1) \tag{3}$$

On the other hand, if the right subtree of r_ℓ does not contain any element from $\{x_j : 2 < j \leq \Delta_\ell + 1\}$, then the minimum distance between the ranks of x_1 and the elements in the right subtree of r_ℓ is $(\Delta_\ell + 2) - 1 = \Delta_\ell + 1$. In this case

$$p_\ell \leq \frac{1}{f(\Delta_\ell + 1)} - \epsilon \leq \frac{1}{f(\Delta_\ell)} \tag{4}$$

Inequality (4) follows from Equation (1), since f is an increasing function and $\epsilon > 0$. Given any $\delta > 0$, for all $\ell \in [1, h]$, we set

$$\frac{\Delta_\ell 2^{\ell-1}}{n-1} = \frac{(h - \ell + 1)\delta}{2^{h-\ell}} \tag{5}$$

The following lemma (proved in the appendix) gives a bound on $\sum_{\ell=1}^{h} \delta_\ell$:

LEMMA 4.1. *If $\frac{\Delta_\ell 2^{\ell-1}}{n-1} = \frac{(h-\ell+1)\delta}{2^{h-\ell}}$, then $\sum_{\ell=1}^{h} \delta_\ell \leq 4\delta$*

The bound on $\sum_{\ell=1}^{h} p_\ell$ is obtained in two steps. First, in Lemma 4.2, we give an upper bound on $\sum_{\ell=1}^{h} p_\ell$ for any monotone function f in terms of $h = \log X$. Then in Lemma 4.3, we show that there exists a value of X such that $\sum_{\ell=1}^{h} p_\ell \leq \delta$ and $\frac{n}{X} = o(\frac{n}{\delta})$ which will complete the proof of Proposition 4.1.

LEMMA 4.2. $\sum_{\ell=1}^{h} p_\ell \leq \frac{h}{f\left(\frac{\delta n}{2^h}\right)}$.

PROOF. By Inequality (4) and Equation (5), $\Delta_\ell = \frac{(h-\ell+1)\delta(n-1)}{2^{h-1}}$, and $p_\ell \leq \frac{1}{f(\Delta_\ell)}$. Then $\sum_{\ell=1}^{h} p_\ell \leq \sum_{\ell=1}^{h} \frac{1}{f(\Delta_\ell)}$
$= \sum_{\ell=1}^{h} \frac{1}{f\left(\frac{(h-\ell+1)\delta(n-1)}{2^{h-1}}\right)} = \sum_{\ell=1}^{h} \frac{1}{f\left(\frac{2(h-\ell+1)\delta(n-1)}{2^h}\right)}$
$\leq \sum_{\ell=1}^{h} \frac{1}{f\left(\frac{(h-\ell+1)\delta n}{2^h}\right)}$ (for $n \geq 2$, $2(n-1) \geq n$). Therefore,

$$\sum_{\ell=1}^{h} p_\ell \leq \sum_{q=1}^{h} \frac{1}{f\left(\frac{q\delta n}{2^h}\right)} \tag{6}$$

$$\leq \frac{h}{f\left(\frac{\delta n}{2^h}\right)} \quad \text{(since } f \text{ is monotone)} \tag{7}$$

□

LEMMA 4.3. *Given any strictly monotone function f, and a constant $\delta > 0$, there exists a h and $n_0 \in \mathbb{N}$, such that for all $n \geq n_0$, $\sum_{\ell=1}^{h} p_\ell \leq \delta$, and $\frac{n}{X} = \frac{n}{2^h} = o(\frac{n}{\delta})$.*

PROOF. Since f is strictly monotone, $f(\Delta) = \omega(1)$ (super-constant growth rate). We choose h as follows:

- $h = \log n - \log\log n - \log\frac{1}{\delta}$, if $f(\Delta) = \omega(\Delta)$.
  Then $n/2^h = \frac{\log n}{\delta} = o(\frac{n}{\delta})$.
- $h = \log f(n^{1/4}) - \log\frac{1}{\delta}$, if $f(\Delta) = O(\Delta)$. Then $n/2^h = \frac{n}{\delta f(n^{1/4})} = \frac{n}{\delta\omega(1)} = o(\frac{n}{\delta})$.

If $f(\Delta) = \omega(\Delta)$, $\sum_{\ell=1}^{h} p_\ell \leq \frac{h}{f\left(\frac{\delta n}{2^h}\right)}$ (from Lemma 4.2)
$= \frac{\log n - \log\log n - \log\frac{1}{\delta}}{f\left(\frac{\delta n}{\frac{\delta n}{\log n}}\right)} \leq \frac{\log n}{f(\log n)} = \frac{\log n}{\omega(\log n)} = o(1) \leq \delta$

when $\delta > 0$ is a constant, there exists n_0 such that for all $n \geq n_0$ the last step holds.

If $f(\Delta) = O(\Delta)$, $\sum_{\ell=1}^{h} p_\ell \leq \frac{h}{f\left(\frac{\delta n}{2^h}\right)}$ (from Lemma 4.2)
$= \frac{\log f(n^{1/4}) - \log\frac{1}{\delta}}{f\left(\frac{\delta n}{\delta f(n^{1/4})}\right)} \leq \frac{\log f(n^{1/4})}{f\left(\frac{n}{O(n^{1/4})}\right)} = \frac{\log f(n^{1/4})}{f(\Omega(n^{3/4}))} \leq \frac{\log f(n^{1/4})}{f(n^{1/4})}$
(for large enough n, since f is strictly monotone) $\leq \delta$.

For all constant $\delta > 0$, there exists n_0 such that for all $n \geq n_0$ the last step holds. □

Since $\frac{n}{X} = \frac{n}{2^h} = o(\frac{n}{\delta})$, by the expression in (2), the upper level uses $o(\frac{n}{\delta}\log\frac{1}{\delta})$ comparisons in total. Combining with the total number of comparisons in the lower levels, which is $\leq n$, and summing up the bad probabilities in the upper and lower levels

by union bound (from the expression in (2) and Proposition 4.1), the maximum element is found with probability $\geq 1 - 6\delta$ with $n + o(\frac{n}{\delta} \log \frac{1}{\delta})$ value questions.

The proof of Lemma 4.3 also shows that, when $f(\Delta) = \omega(\Delta)$, $n + O(\frac{\log n}{\delta} \log \frac{1}{\delta})$ value questions suffice. However, the following Lemma 4.4 shows that better bound can be obtained when $f(\Delta) = \Omega(\Delta)$ or $f(\Delta) = 2^{\Delta}$, by a tighter analysis using Inequality (6). This lemma also gives a tighter bound for the slowly growing logarithmic error function. This lemma is analogous to Lemma 4.3; the proof of the lemma is given in the appendix.

LEMMA 4.4. *Given any $\delta > 0$,*

1. *for **exponential error function**, there exists a value of X such that $\frac{n}{X} = O(\log^2 \frac{1}{\delta})$, and $\sum_{\ell=1}^{h} p_\ell \leq \delta$,*
2. *for **linear error function**, there exists a value of X such that $\frac{n}{X} = O(\frac{\log \log n}{\delta^2})$ and and $\sum_{\ell=1}^{h} p_\ell \leq \delta$.*
3. *for **logarithmic error function**, there exists a value of X such that $\frac{n}{X} = O\left(\frac{n^{\frac{1}{1+\delta}}}{\delta^{\frac{\delta}{\delta+1}}}\right)$, and $\sum_{\ell=1}^{h} p_\ell \leq \delta$.*

Substituting the value of $\frac{n}{X}$ in the expression in (2), the better upper bounds for functions f such that $f(\Delta) = \Omega(\Delta)$ or $f(\Delta) = 2^{\Delta}$ in Theorem 2 can be obtained. This completes the proof of Theorem 2.

4.2 Finding Top-k

Suppose $x_1 > x_2 > \cdots > x_n$. Given an integer k, Feige et. al. [7] has given an algorithm to find the top-k elements $x_1, \cdots, x_k$ in the constant error model. This algorithm uses $O(n \log \frac{\min(k, n-k)}{\delta})$ comparisons to find the k-th largest element with probability $\geq 1 - \delta$. For simplicity, assume that $k \leq \frac{n}{2}$, *i.e.* $\min(k, n-k) = k$.

In practice, for database top-k queries, the value of k is likely to be much smaller than the total number of elements n. When the value of k is small, a better bound on the number of value comparisons can be obtained using Theorem 2 in Section 4.1 and the algorithm given in [7] with strictly monotone error functions. In particular, we show the following corollary to Theorem 2 that solves the top-k problem with high probability.

COROLLARY 1. *For all strictly growing functions f and constant $\epsilon, \delta > 0$, $n + o(\frac{nk}{\delta} \log \frac{k}{\delta}) + O(\frac{k^2}{\delta} \log \frac{k}{\delta})$ value questions are sufficient to output all the top-k elements $x_1, \cdots, x_k$ with probability $\geq 1 - \delta$.*

Further, if $f(\Delta) = \Omega(\Delta)$, then $n + O(\frac{k \log \log n}{\delta^3} \log \frac{k}{\delta}) + O(\frac{k^2}{\delta} \log \frac{k}{\delta})$ value questions are sufficient. If $f(\Delta) = 2^{\Delta}$, then $n + O(\frac{k^2}{\delta} \log \frac{k}{\delta})$ value questions are sufficient.

When $k = O(1)$ and $\delta > 0$ is constant, we once again get a bound of $n + o(n)$ even to find all the top-k elements with high probability, which exceeds n only by lower order additive terms. Note that, even when the comparisons are exact, the linear time recursive selection algorithm [3] requires cn comparisons for a constant $c > 1$ to find the k-th element (although it works for all values of k). The same guarantee of $n + o(n)$ can be obtained for any $k = o(\sqrt{n})$, when the error function $f(\Delta) = \Omega(\Delta)$ (the growth-rate is at least linear). The algorithm in [7] gives a better bound for other error functions and values of k.

As an aside, we note that for any fixed $\delta > 0$, when the answers to value questions have no errors, and when $k = o(\sqrt{n})$, our techniques give a bound of $n + g(k, \delta)$ on the value comparisons for finding the top-k elements with probability $\geq 1 - \delta$. Here $g(k, \delta)$ is a polynomial function of k and $\frac{1}{\delta}$ independent of n (see the appendix).

To conclude this section, we sketch how we can obtain Corollary 1 using Theorem 2; the details are given in the appendix. Once again, we use a comparison tree and start with a random permutation of the elements in the leaves of this tree. We also divide the $\log n$ levels of the comparison tree into lower $\log X$ levels and upper $\log \frac{n}{X}$ levels. In the lower levels, we have $\frac{n}{X}$ X-trees. We show that there is a value of X such that with high probability each of $x_1, \cdots, x_k$ appear in different X-trees so that they are the maximum elements in their respective X-trees. In all the X-trees we use Algorithm 1, and argue that all of $x_1, \cdots, x_k$ are the winners in their respective X-trees with high probability. Therefore, in the upper $\log \frac{n}{X}$ levels, $x_1, \cdots, x_k$ remain to be the top-k elements. In the upper levels, which has $\frac{n}{X}$ elements, we use the top-k algorithm from [7]. We show that the total number of questions asked to the crowd is given by the expressions in Corollary 1. We also argue that the total probability of error (probability that exactly $x_1, \cdots, x_k$ are not returned) is bounded by δ, so with probability $\geq 1 - \delta$ we find the top-k elements.

5. CLUSTERING

We study the clustering problem motivated by group-by queries in this section. Recall that there are J distinct types, and the goal of the clustering problem is to find the J clusters, *i.e.* the groups of elements having the same type. For the max and top-k problems discussed in the previous chapter we used value questions, where the crowd is asked to order two elements according to their values. We will use type questions in this section for clustering purposes. Here the crowd is asked to decide whether the elements have the same type, for instance, whether two photos capture the same person or place.

We prove the following theorem which gives a bound on the number of type questions that are necessary and sufficient to exactly find the J clusters. In our algorithms, we *do not* assume that J is known a priori – when a set of questions regarding a photo database is asked, the crowd may not know the number of people participating in the database. However, our (tight) lower bounds hold even when the value of J is known. Recall that we assume the constant error model for type questions, *i.e.* each type question is answered correctly with probability $\geq \frac{1}{2} + \epsilon$, for a constant $\epsilon > 0$.

THEOREM 3. *For all $\delta > 0$, to group n elements into J clusters with probability $\geq 1 - \delta$, $O(nJ \log \frac{n}{\delta})$ type questions in expectation are sufficient in the constant error model.*

On the other hand, $\Omega(nJ)$ type questions are necessary (i) even if the algorithm is randomized, (ii) even when answers to all type questions are exact, and (iii) even when the value of J is known.

Proof of Upper Bound. Algorithm 2 finds the J clusters with high probability.

Algorithm 2 Algorithm for clustering with only type questions (given n elements, and the values of $\epsilon, \delta > 0$))

1: – List the elements in arbitrary order L.
2: – Initialize a set for clusters $P = \emptyset$.
3: **while** L is not empty **do**
4: Let y be the first element in L.
5: Find elements with the same type as y among the remaining elements in L as follows: For each remaining element x in L, ask the type question $\texttt{type}(x) = \texttt{type}(y)$ $O(\frac{1}{\epsilon^2}(\log \frac{n}{\delta}))$ times. If the majority of the answers are "yes", x, y are decided to have the same type; otherwise they are decided to have different types.
6: Collect all elements of the same type, make a cluster C, add to P, and delete these elements from L.
7: **end while**
8: **return** the clusters in P.

Analysis. With appropriate choices of constants, by Chernoff bound, whether the elements x, y compared in Step 5 have the same type is decided incorrectly with probability $\leq \frac{\delta}{n^3}$. Since $J \leq n$, $nJ \leq n^2$. By union bound, with probability $\geq 1 - \frac{\delta}{n}$, for all pairs of elements considered by the algorithm whether they have the same type is decided correctly. When the type comparisons are correct, it is easy to check that the correct clusters are returned in J iterations. This happens with probability $\geq 1 - \frac{\delta}{n}$.

Note that in each iteration, at least the first remaining element from the list L is deleted, therefore the loop is run at most n times. However, as argued above, with probability $\geq 1 - \frac{\delta}{n}$, the number of iterations of the while loop is J (when the clusters are correctly returned), and with probability $\leq \frac{\delta}{n}$, the number of iterations is $\leq n$. Hence the expected number of iterations is $O(J)$. In each iteration, at most $O(\frac{n}{\epsilon^2}\log\frac{n}{\delta})$ type questions are asked. Therefore, in expectation and for constant ϵ, the bound given in Theorem 3 follows.

Proof of Lower Bound. First we give the proof of lower bound for deterministic algorithm, when there is no error in the answers to the type questions (exact comparisons), and when the value of J is known. Then we prove the lower bound for randomized algorithms.

Lower bound for deterministic algorithms. Let $s_{\max}$ be the size of the maximum cluster in an instance, and $s_{\min}$ be the size of the minimum cluster. Recall that the instance is called balanced if $s_{\max}/s_{\min} = O(1)$. We will prove the lower bound of $\Omega(nJ)$ for deterministic algorithms even when the clusters are balanced.

Consider any deterministic algorithm A that solves the clustering problem. Let us number the clusters arbitrarily as $C_1, \cdots, C_J$. The adversary starts by assigning $2n/3J$ elements to each of the clusters $C_1, C_2, ..., C_J$ and reveals these elements for free to A (therefore, algorithm A knows the value of J). At this point, number of unassigned elements is $n/3$. Let this be the set U. The adversary now plays an evasive game on this set U. An element $x \in U$ is active iff it has been compared with $(J-1)/2$ elements. For an active element, whenever the algorithm A asks a question involving it, the answer is always "no". Once an element ceases to be active, it has at least $J - (J-1)/2 = (J+1)/2$ valid clusters among $C_1, \cdots, C_J$ to which it can still be assigned. We always assign it to a cluster with smallest number of elements, breaking ties arbitrarily. This ensures that no cluster C_i ever gets assigned more than $\frac{n/3}{(J+1)/2} < 2n/3J$ elements from U. So the minimum cluster size is $2n/3J$ (recall that all clusters initially had $2n/3J$ elements), and the maximum cluster size is $4n/3J$. The ratio is bounded by 2. The total work done is clearly $\Omega(nJ)$.

Lower bound for randomized algorithms. We next show that an $\Omega(nJ)$ lower bound holds for randomized algorithms as well even when all type comparisons are exact. By Yao's min-max principle [14], it suffices to exhibit a distribution on input instances such that any deterministic algorithm needs $\Omega(nJ)$ comparisons in expectation with respect to that distribution.

Suppose the clusters are $C_1, \cdots, C_J$. For each element, we randomly choose $j \in [1, J]$ and assign it to cluster C_j. Let us call an element x to be *settled*, if either the algorithm performs $J-1$ comparisons involving x, or if the algorithm performs a type comparison between x and some element y whose result is a "yes". Note that to cluster all n elements, each element must be settled. This is because if $\leq J-2$ comparisons are performed involving x, and all of the comparisons return "no", there are still at least two clusters where x can go to. Next we compute the expected number of comparisons needed to make an element settled by a "yes" answer.

Suppose ℓ comparisons have been performed involving x all of which answered "no". Since type of each element is chosen uniformly at random, under the above assumptions, for any element x that has participated so far in ℓ type comparisons each of which resulted in a "no", the probability that the next type comparison returns a "yes" is bounded by $\frac{1}{J-\ell}$. The probability that each of the first ℓ type comparisons of x returns a "no" is at least $\frac{J-\ell}{J}$. Thus the expected number of type comparisons before an element gets a "yes" answer is at least $\sum_{\ell=1}^{J} \ell \times \frac{1}{J-\ell} \times \frac{J-\ell}{J} = \frac{J+1}{2}$. Therefore the expected number of comparisons for an element to get settled is $\Omega(J)$. Since every type comparison involves exactly two elements, it follows by linearity of expectation that the total number of type comparisons is $\Omega(nJ)$[7].

6. CLUSTERING WITH CORRELATED TYPES AND VALUES

In the previous section, we used only type questions for clustering that compares whether two elements have the same type. We showed that, to cluster n elements into J clusters $\tilde{\Theta}(nJ)$ questions are necessary and sufficient. However, as mentioned in Section 3, types and values can be correlated in some scenarios and elements of the same type can form (almost) contiguous blocks in the sorted order according to the values (*e.g.* quality of hotels as types vs. their prices as values). Recall that we formalized this idea assuming that there are at most α changes in types between any two elements of the same type. In this section we will see that this bound improves to $\tilde{O}(n \log J)$ when α is small and both type and value questions are asked. Note that both value and type questions are answered correctly with probability $\geq \frac{1}{2} + \epsilon$, given a constant $\epsilon > 0$.

THEOREM 4. *Given any $\delta > 0$, it is sufficient to ask $O\left((n\log(\alpha J) + \alpha J)\log\frac{n}{\delta}\right)$ type and value questions in expectation to cluster n elements into J clusters with probability $\geq 1 - \delta$.*

[7]We leave the exact bound for randomized algorithms for the balanced case as an open problem.

Similar to the previous section, we do not assume that the value of J is known a priori. But we assume that the value of α (or an upper bound on α) is known. In this section we will also explain why the bound given in the above theorem is tight in a certain sense, and briefly discuss how top-k/maximum elements from each of the J clusters can be found with high probability using both type and value questions.

Recall that when $\alpha = 1$, we have the full correlation case, where elements from the same type exactly form contiguous blocks in the sorted order on values. When the value of α is small, we have the partial correlation case. Here we present our key ideas assuming the full correlation case in Algorithm 3 and assuming that the answers to the type and value questions are exact. We analyze this algorithm and discuss how erroneous answers to the type and value questions can be handled. Then we discuss how the ideas can be extended to work for general α.

6.1 Clustering for Full Correlation

Since the clusters do not have any name to identify them, Algorithm 3 forms the clusters as a forest. For each cluster C, except one element in C, each element y stores a pointer $\texttt{link}(y)$ which points to another element z in the same cluster (*i.e.* $\texttt{type}(y) = \texttt{type}(z)$). The element with no link (null) is the root of this tree. Clearly, from this structure all the clusters can be output in $O(n)$ time.

Analysis. We first analyze Algorithm 3 assuming that the answers to all type and value questions are correct. To argue the correctness, we show that exactly one element from each type has null link which forms the tree of the corresponding cluster, and the rest of the elements link to another element from the same cluster. Note that we assign $\texttt{link}(z) = y$ if and only if $\texttt{type}(y) = \texttt{type}(z)$, therefore we never set $\texttt{link}(z)$ incorrectly for any element z. Also whenever an element z is deleted (Step 19), another element y such that $\texttt{type}(y) = \texttt{type}(z)$ is retained, *i.e.* we never delete all elements from a type. Therefore, we argue that the algorithm returns exactly one element y from each type such that $\texttt{link}(y) = null$, which proves its correctness.

For the sake of analysis, consider the elements $x_1 > \cdots > x_n$ in sorted order. In the full correlation case, elements from the same type form contiguous blocks in the sorted order. We argue that the two while loops in the algorithm keep exactly one element from each such block.

The algorithm tries to identify these blocks by dividing the list of elements (in arbitrary order L) into intervals. As long as there is one interval with more than two types, the interval is partitioned into two halves by the median, which also ensures that all elements before (resp. after) the median are greater (resp. smaller) than the median. Therefore, repeatedly finding the medians divide the list of elements into intervals such that all elements of any earlier interval is larger than all elements in any interval after. When the variable `repeat_loop` is set to FALSE, only one element from each block (*i.e.* from each cluster) is retained. These elements are returned by the algorithm as elements with null link.

Number of questions asked. The following lemma bounds on the total number of type and value questions.

LEMMA 6.1. *The total number of type and value questions used by the algorithm is $O(n \log J)$ assuming the answers to these questions are correct.*

Algorithm 3 Algorithm for clustering in the full correlation case (given $\epsilon, \delta > 0$)

1: – List all elements in L in an arbitrary order.
2: – Initialize $\texttt{link}(y) = null$ for each element y.
3: – Set `repeat_loop` = true.
4: **while** `repeat_loop` is true **do**
5: – Let $s = |L|$.
6: – Initially, the entire L forms a single interval.
7: **while** $|L| > s/2$ **do** {*/*The total number of elements in L is not halved*/*}
8: **if** each interval has exactly one element **then**
9: – `repeat_loop` = false
10: **else**
11: /* Divide each interval in half to form two smaller intervals*/
12: **for** each interval B with two or more elements **do**
13: – Find the median of the elements in B.
14: – Partition the elements in B in two halves comparing with the median by value questions.
15: – Each of these two halves forms a new interval, say B_1 and B_2.
16: **for** both B_i, $i \in \{1, 2\}$ **do**
17: – Check if B_i has at least two types: The first element y in B_i is compared with each of the other elements z in B_i to check if there is a z such that $\texttt{type}(y) \neq \texttt{type}(z)$.
18: – If B_i has at least two types, B_i is called an *active interval*. Do nothing.
19: – If B_i is not active (all elements have the same type), choose an arbitrary element y from B_i. For the other elements z in the interval, set $\texttt{link}(z) = y$. Delete all elements in B_i from L except y.
20: **end for**
21: **end for**
22: **end if**
23: **end while**
24: **end while**
25: **return** all elements y with their link $\texttt{link}(y)$.

PROOF. First we compute the total number of questions asked in each iteration of the inner while loop (Step 7). Let us count unit cost for the repeated value and type questions in Steps 13, 14 and 17. Consider the first `for` loop with original intervals in Step 12. Let the number of intervals be b, and let the number of elements in these intervals be $n_1, \cdots, n_b$. Since the intervals are disjoint, $n_1 + \cdots + n_b \leq s$. In the j-th interval, the linear-time selection algorithm [3] can find the median using $O(n_j)$ value questions (Step 13) and the partition can also be done using $O(n_j)$ value questions (Step 14). This `for` loop further partitions the intervals into two disjoint intervals B_1, B_2. In the inner `for` loop (Step 16), only one element from each interval is compared with the other elements using type questions, hence the total number of type questions in B_1, B_2 is $O(n_j)$. Therefore the total number of value and type questions in the outer `for` loop (Step 12) is $\sum_{i=1}^{b} O(n_j) = O(s)$.

Next we compute the number of iterations in the inner while loop. Consider the contiguous blocks of elements of the same type in

the sorted order. Since there are J clusters, the number of blocks is J. Suppose the algorithm reduces the number of elements in P iterations of the inner while loop. Then the number of element at the start of the while loop s is divided into 2^P intervals. When $2^P = 4J$, at most J intervals may be active (two or more types). The active intervals have $\leq s/4$ elements in total. Each interval is of size $s/4J$ and one element is retained from each inactive interval. Hence the number of remaining elements will be $s/4 + s/4J \leq s/2$. Therefore, the while loop will terminate in $P = O(\log J)$ iterations.

Now we compute the number of questions in the outer `while` loop in Step 4. The inner while loop ensures that the problem size s is halved in each of its iterations. Hence the number of questions $Q(n)$ with input size n is captured by the following recurrence relation (counting unit cost in Steps 14 and 17):

$$Q(n) = Q(\frac{n}{2}) + O(n \log J) \tag{8}$$

The solution of this recurrence relation is $O(n \log J)$. □

Handling erroneous answers to type and value questions. Here we discuss how erroneous answers to type and value questions can be handled using standard techniques. When type and value comparisons are correct, $cn \log J$ questions suffice, for some constant c. Now consider the case when the comparisons are erroneous, but correct answers are returned with probability $\geq \frac{1}{2} + \epsilon$, for constant $\epsilon > 0$. In this case we repeat each type or value comparison performed by Algorithm 3 between two elements $O(\frac{1}{\epsilon^2} \log \frac{n}{\delta})$ times and take the majority vote (omitted in the algorithm for simplicity) to decide whether they have the same type, or to order them according to their values. This adds the multiplicative $\log \frac{n}{\delta}$ factor in the total number of questions asked. Moreover, we abort the algorithm after comparing cn^2 pairs of elements (note that we do not assume that the value of J is known).

With appropriate choices of the constants, by Chernoff bound, the answer to each type and value comparison between any two elements is correct with probability $\geq 1 - \frac{\delta}{cn^2}$. By union bound, the total bad probability in the $cn \log J \leq cn^2$ comparisons is bounded by δ. Hence with probability $\geq 1 - \delta$, all the comparisons are correct, and the above analysis holds. The expected number of questions asked by the algorithm is $O(n \log J) \times O(\log \frac{n}{\delta})$: $cn \log J$ comparisons are performed with probability $\geq 1 - \delta$ and cn^2 comparisons are performed only with probability $\leq \delta$. This proves Theorem 4 for the full correlation case.

6.2 Extension to general α

For arbitrary α, there are at most α changes in types between any two elements of the same type. Again partition the elements in the sorted order $x_1 > \cdots > x_n$ into consecutive blocks of the same type. Algorithm 3 as it is run for the case of general α. As argued above, this will give one representative element from each block. However, when now there may be more than one representative element from the same cluster, which we need to group together. Further note that, these representative elements will be sorted according to their values due to repeated partitioning using medians of active intervals.

To group elements of the same types, consider the list of remaining elements L returned by Algorithm 3. While L is not empty, select the first element y in L. For the next α elements z in L, check if y and z have the same type. For all elements z with $\texttt{type}(y) = \texttt{type}(z)$, set $\texttt{link}(z) = y$. Delete these elements from S. Then repeat the procedure with the remaining elements in L (in order).

We already argued in the full correlation case that Algorithm 3 leaves one element from each consecutive block of the same type. In the additional step to group elements of the same types, the consecutive α elements in L are examined by type questions. At most α changes in types are present between the first and the last element of any cluster in the sorted order. This ensures that all elements of the same type as the first element y in S will be grouped together. This process is repeated until the list L is empty, which returns all clusters.

To count the number of type and value questions, note that the number of consecutive blocks for general α is $\leq \alpha J$. Therefore, Algorithm 3 asks $O(n \log(\alpha J))$ questions (similar to Lemma 6.1). The additional step to group elements of the same type needs J iterations. So $O(n \log(\alpha J) + \alpha J)$ comparisons suffice when the answers to type and value questions are exact. Errors in the answers can be handled by repeating each comparison $O(\log \frac{n}{\delta})$ times and taking the majority vote, as described for the full correlation case.

6.3 Lower Bounds

Let us briefly discuss why the bounds given in Theorem 4 is tight upto logarithmic factors in a certain sense even when there is no error in the comparisons. Recall that we proved $\Omega(nJ)$ lower bound for clustering in Section 5. Since α in the worst case is $\Omega(n)$ (no correlation between types and values), we cannot hope to get a better bound than $O(\alpha J)$ for all values of α. Further, there is also a lower bound of $\Omega(n \log n)$ which explains that we cannot get a better bound than $O(n \log(\alpha J))$ for all values of α and J. This lower bound follows from the *element distinctness problem*, *i.e.* given n elements check if any two elements have the same value, which is known to have a lower bound of $\Omega(n \log n)$ [2]. In the reduction, two elements belong to the same cluster if and only if they have the same value, and a cluster has ≥ 2 elements if and only if the elements are not distinct.

6.4 Max/Top-k from Each Cluster using Constant and Variable Error Model

When both type and value questions are asked, a natural question to ask is to find top-k or the maximum element from each cluster (see the query example given in the introduction). This can be achieved by combining our results on clustering and top-k: first find the clusters using Algorithms 2 or 3, and then find the top-k or max from each cluster using the algorithms in Section 4. Clearly, to guarantee that top-k elements are found from all clusters with probability $\geq 1 - \delta$ given $\delta > 0$, the value of δ in the max or top-k algorithm has to be replaced by $\delta' = \frac{\delta}{J}$ (when the clusters are constructed, the value of J is known). The maximum elements from each cluster can be found by a small modification of Algorithm 2: as each cluster is computed, in addition to type comparisons between two elements, also compare their values; retain the element with larger value. We can also use the algorithms from Section 4 for a monotone error function f to obtain better bounds on the number of questions asked[8].

[8]Let Δ and Δ' be the distance of two elements having same type in the entire sorted order, and in the sorted order restricted to the cluster containing them respectively. Since $\Delta \geq \Delta'$, for monotone error function f, the probability of error in value comparisons $\frac{1}{f(\Delta)} \leq \frac{1}{f(\Delta')}$. Therefore, the same error function can be assumed even when the elements in respective clusters are compared.

7. CONCLUSIONS

In this paper, we studied max/top-k and clustering problems in the crowd sourcing setting. These problems are motivated by top-k and group-by database queries, where the criteria used for grouping and ordering are difficult to evaluate by machines but much easier to evaluate by the crowd (*e.g.* grouping photos by the individuals occurring in them or finding their most recent photo). We gave efficient algorithms that ask a small number of type and value questions to the crowd. For max/top-k queries, we proposed a variable error model for erroneous answers and showed that fewer queries are needed compared to the constant error model. On the other hand, for the clustering problem or group-by queries, fewer questions are needed when there is a correlation between the types and values of the elements.

There are many interesting future directions. We focused on the objective of minimizing the number of comparisons performed by the crowd to find the exact top-k elements or the exact clusters. It will be interesting to consider alternative cost-based objectives that are amenable to better bounds on the number of comparisons. Further, there is a natural 'value-based' alternative to the 'ranking-based' variable error model considered in the paper, where the error probability is a monotone function of the difference in values of the two elements being compared instead of the difference in their ranks in the sorted order. The upper bounds given in this paper for the ranking-based error model also hold for the value-based error model when the difference in the values of two consecutive elements in the sorted order is at least one (then the value-based error is bounded above by the ranking-based error); it will be interesting to further explore the value-based error model. Other reasonable objectives include minimizing latency or the number of rounds of interactions with the crowd. We assumed that the cost is directly proportional to the number of questions. One can look at cost functions that are concave in the number of questions, so that when asking more questions one needs to pay less per question (as repetitive tasks are considered easier). It will also be interesting is to minimize the probability of errors for top-k or clustering questions, when a budget on the number of comparisons is provided.

Acknowledgements. We thank the anonymous reviewers for their insightful comments. This work was supported in part by the NSF Awards IIS-0803524, CCF-1116961, and IIS-0904314, the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 291071-MoDaS, the Israel Ministry of Science, the Binational (US-Israel) Science Foundation, and a Google Ph.D. Fellowship.

8. REFERENCES

[1] Eyal Baharad, Jacob Goldberger, Moshe Koppel, and Shmuel Nitzan. Distilling the wisdom of crowds: weighted aggregation of decisions on multiple issues. *Autonomous Agents and Multi-Agent Systems*, 22(1):31–42, January 2011.

[2] Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 80–86, New York, NY, USA, 1983. ACM.

[3] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, August 1973.

[4] Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang-Chiew Tan. Asking the right questions in crowd data sourcing. *ICDE*, 0:1261–1264, 2012.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[6] *https://www.mturk.com/.*

[7] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994.

[8] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, New York, NY, USA, 2011. ACM.

[9] Ryan Gomes, Peter Welinder, Andreas Krause, and Pietro Perona. Crowdclustering. In *NIPS*, pages 558–566, 2011.

[10] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, New York, NY, USA, 2012. ACM.

[11] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: a crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040–1051, June 2012.

[12] Adam Marcus, Michael S. Bernstein, Osama Badar, David R. Karger, Samuel Madden, and Robert C. Miller. Twitinfo: aggregating and visualizing microblogs for event exploration. In *CHI*, pages 227–236, 2011.

[13] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[14] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[15] Aditya Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative crowdsourcing. Technical report, Stanford University.

[16] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, New York, NY, USA, 2012. ACM.

[17] Joachim Selke, Christoph Lofi, and Wolf-Tilo Balke. Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. *Proc. VLDB Endow.*, 5(6):538–549, February 2012.

[18] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, New York, NY, USA, 2012. ACM.

[19] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

APPENDIX

A. OMITTED PROOFS

In this section we give the proofs that are omitted in the previous sections.

A.1 The number of questions in the upper log (n/X) levels for Algorithm 1

Recall that, each internal node in the levels $\ell = 1$ to $\log \frac{n}{X}$ uses $S_\ell = (2\ell - 1) * O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ comparisons. By a simple application of Chernoff bound [14], it can be shown that the max element in

this sub-tree can be found with probability $\geq 1-\delta$. Level ℓ in the sub-tree has $\frac{n}{X2^\ell}$ nodes, $\ell = 1$ to $\log \frac{n}{X}$. Clearly, $N_L = S_{L-\log X}$ for $L = \log X + 1$ to $\log n$. The total number of comparisons $\sum_{L=\log X+1}^{\log n} N_L$ is $\sum_{\ell=1}^{\log \frac{n}{X}} (2\ell-1) \times \frac{n}{X2^\ell} \times O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ = $O(\frac{n}{\epsilon^2 X})\log(\frac{1}{\delta})$. For constant $\epsilon > 0$, the expression in (2) follows.

A.2 Proof of Lemma 4.1

PROOF. Let $S = \sum_{\ell=1}^{h} \delta_\ell \leq \sum_{\ell=1}^{h} \frac{(h-\ell+1)\delta}{2^{h-\ell}}$ (from (3)). Then

$$\begin{aligned} S &= \delta + 2\delta/2^1 + 3\delta/2^2 + \cdots + h\delta/2^{h-1} \\ S/2 &= \delta/2^1 + 2\delta/2^2 + \cdots + (h-1)\delta/2^{h-1} + h\delta/2^h \\ \Rightarrow S/2 &= \delta + \delta/2^1 + \delta/2^2 + \cdots + \delta/2^{h-1} - h\delta/2^h \leq 2\delta \\ \Rightarrow S &\leq 4\delta \end{aligned}$$

□

A.3 Proof of Lemma 4.4

PROOF. The particular choices of the exponential, linear or logarithmic functions in the proof ensure that $f(1) \geq 2$, the results also hold for any other choices of these functions (and functions having steeper growth rates than these functions).

Exponential error function. Suppose $f(\Delta) = 2^\Delta$. From (4) and (5), $p_\ell \leq \frac{1}{2^{\Delta_\ell}}$ and $\Delta_\ell = \frac{(h-\ell+1)\delta(n-1)}{2^{h-1}}$. We set $X = 2^h = \frac{2\delta(n-1)}{\log(2/\delta)}$. With these choice of X, $\sum_{\ell=1}^{h} p_\ell \leq \sum_{\ell=1}^{h} \frac{1}{2^{\Delta_\ell}}$ $\leq \sum_{q=1}^{h} \frac{1}{\frac{q\delta n}{2^h}}$ (from (6)) $\leq \sum_{q=1}^{\infty} \frac{1}{\frac{q\delta n}{2^h}} \leq \sum_{q=1}^{\infty} \frac{1}{\left(\frac{\delta n}{2^h}\right)^q} \leq$ $\frac{2}{\frac{\delta(n-1)}{2^{h-1}}} = \frac{2}{\frac{\delta(n-1)}{\log(2/\delta)}} = \frac{2}{2/\delta} = \delta$.

Therefore, $\frac{n}{X} = \frac{n\log(2/\delta)}{2\delta(n-1)} \leq \frac{n\log(2/\delta)}{\delta n} = \log(2/\delta)$.

Linear error function. Suppose $f(\Delta) = \Delta + 1$. We set $X = 2^h = \frac{\delta^2 n}{\log\log n}$. With this choice of X, $\sum_{\ell=1}^{h} p_\ell \leq \sum_{\ell=1}^{h} \frac{1}{\Delta_\ell}$ $\leq \sum_{q=1}^{h} \frac{1}{\frac{q\delta n}{2^h}}$ (from (6)) $= \frac{2^h}{\delta n}\sum_{q=1}^{h} \frac{1}{q} \leq \frac{2^h \log h}{\delta n} \leq \frac{\delta^2 n}{\log\log n} \times$ $\frac{\log(\log \frac{\delta^2 n}{\log\log n})}{\delta n} \leq \frac{\delta^2 n}{\log\log n} \times \frac{\log\log n}{\delta n} = \delta$.

Therefore, $\frac{n}{X} = n \times \frac{\log\log n}{\delta^2 n} = \frac{\log\log n}{\delta^2}$.

Logarithmic error function. Suppose $f(\Delta) = \log\Delta + 2$. We set $X = (2\delta n)^{\frac{\delta}{\delta+1}}$. With this choice of X, $\sum_{\ell=1}^{h} p_\ell \leq \sum_{\ell=1}^{h} \frac{1}{\log\Delta_\ell}$ $\leq \frac{h}{\log\left(\frac{\delta n}{2^h}\right)}$ (from (7)) $= \frac{\log(\delta n)^{\frac{\delta}{\delta+1}}}{\log\left(\frac{\delta n}{(\delta n)^{\frac{\delta}{\delta+1}}}\right)} \leq \frac{\frac{\delta}{\delta+1}\log(\delta n)}{\log\left(\frac{\delta n}{(\delta n)^{\frac{\delta}{\delta+1}}}\right)}$ $\leq \frac{\frac{\delta}{\delta+1}\log(\delta n)}{\frac{1}{\log(\delta n)^{\frac{\delta}{\delta+1}}}} = \delta$.

With this choice of X, $\frac{n}{X} = \frac{n}{(\delta n)^{\frac{\delta}{\delta+1}}} = \frac{n^{\frac{1}{1+\delta}}}{\delta^{\frac{\delta}{\delta+1}}}$. □

A.4 Proof of Corollary 1

Here we show that we obtain the exact top-k elements with probability $\geq 1-8\delta$. To obtain the bound of $1-\delta$, we need to run our algorithm with $\delta' = \delta/8$. We assume that $k = o(\sqrt{n})$, as discussed in Section 4.2, otherwise the algorithm in [7] gives a better bound on the number of value comparisons.

Once again, we start with a random permutation Π and the comparison tree is again divided into upper and lower levels. In the lower levels, n elements in Π are partitioned into $\frac{n}{X}$ number of X-trees. Instead of focusing on the single X-tree that contains the largest element x_1, we consider all the X-trees that contain the top-k elements $x_1, \cdots, x_k$. We show that with probability $1-6\delta$, (A) each of $x_1, \cdots, x_k$ appear in different X-trees so that they are the maximum elements in their respective X-trees, and (B) none of these elements lose any comparison in their X-trees.

First we consider (A) and argue that no two elements in $x_1, \cdots, x_k$ appear in the same X-tree with probability $\geq 1-\delta$ if $X \leq \frac{\delta n}{k^2}$. Since Π is a random permutation, any of the other $n-1$ elements have equal probability of belonging to any fixed leaf of the X-tree containing x_i, for any $i \in [1,k]$. Therefore, by union bound, the probability that this X-tree contains another element from $x_1, \cdots, x_k$ is $\leq \frac{k(X-1)}{n-1} \leq \frac{kX}{n}$ which is $\leq \frac{\delta}{k}$ when $X \leq \frac{\delta n}{k^2}$. We choose $X = \frac{\delta n}{k^2}$. Applying union bound for all x_i, $i \in [1,k]$, with probability $\geq 1-\delta$ no two top-k elements appear in the same X-tree. From now on, we will consider that the elements $x_1, \cdots, x_k$ belong to separate X-trees.

Now consider (B). Given (A), each x_i, $i \in [1,k]$ is the maximum element in their respective X-trees. We apply Algorithm 1 on all $\frac{n}{X}$ X-trees with $\delta' = \delta/k$. Hence total number of comparisons = $(\frac{n}{X}) \times X + o(X) \times O(\frac{1}{\delta'}\log\frac{1}{\delta'})$ = $n + o(n) \times O(\frac{k}{\delta}\log\frac{k}{\delta})$. By Theorem 2, all of $x_1, \cdots, x_k$ are decided as the maximum elements in their respective X-trees with probability $\geq 1-6\delta/k$. By union bound, all of them go to the upper levels with probability $\geq 1-6\delta$.

In the upper levels, we employ the algorithm given in [7] to find the top-k elements which are still $x_1, \cdots, x_k$. Since these upper levels have $\frac{n}{X}$ elements, $O(\frac{n}{X}\log\frac{k}{\delta}) = O(\frac{k^2}{\delta}\log\frac{k}{\delta})$ comparisons suffice to find $x_1, \cdots, x_k$ with probability $1-\delta$.

Combining with the number of comparisons in the lower levels, and the bad probabilities from (A) and (B), with probability $\geq 1-8\delta$ the top-k elements are found with the stated number of comparisons. For error functions $f(\Delta) = \Omega(\Delta)$ or $f(\Delta) = 2^\Delta$, better bound can be obtained by using Lemma 4.4 in the lower levels.

An upper bound of $n + O(\frac{k^2}{\delta})$ for exact comparison .

Suppose the answers to the value comparisons are exact. Here we sketch how our techniques presented above can find all top-k elements with probability $\geq 1-\delta$ given $\delta > 0$ using only $n + O(\frac{k^2}{\delta})$ comparisons when $k = o(\sqrt{n})$. As discussed above, we choose $X = \frac{\delta n}{k^2}$. This ensures that with probability $\geq 1-\delta$, all top-k elements appear in different X-trees and therefore survive in the upper levels. In each X-tree, we perform $X-1$ comparisons to perform the maximum element in it. Clearly, all $x_1 \cdots, x_k$ are chosen as maximum elements in their respective X-trees. The upper level has $\frac{k^2}{\delta}$ elements, and we run the linear time selection algorithm [3] to find x_k. A linear pass on the upper level finds $x_1, \cdots, x_k$ which are larger than x_1. The total number of value comparisons performed is $\leq n + O(\frac{k^2}{\delta})$.