# Where's the Winner? Max-Finding and Sorting with Metric Costs

Anupam Gupta[1] and Amit Kumar[2]

[1] Dept. of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213. `anupamg@cs.cmu.edu` *

[2] Dept. of Computer Science & Engineering, Indian Institute of Technology, Hauz Khas New Delhi, India - 110016. `amitk@cse.iitd.ernet.in`

**Abstract.** Traditionally, a fundamental assumption in evaluating the performance of algorithms for sorting and selection has been that comparing any two elements costs one unit (of time, work, etc.); the goal of an algorithm is to minimize the total cost incurred. However, a body of recent work has attempted to find ways to weaken this assumption—in particular, new algorithms have been given for these basic problems of searching, sorting and selection, when comparisons between different pairs of elements have different associated costs.

In this paper, we further these investigations, and address the questions of max-finding and sorting when the comparison costs form a *metric*; i.e., the comparison costs $c_{uv}$ respect the triangle inequality $c_{uv} + c_{vw} \geq c_{uw}$ for all input elements $u, v$ and $w$. We give the first results for these problems—specifically, we present

– An $O(\log n)$-competitive algorithm for max-finding on general metrics, and we improve on this result to obtain an $O(1)$-competitive algorithm for the max-finding problem in constant dimensional spaces.
– An $O(\log^2 n)$-competitive algorithm for sorting in general metric spaces.

Our main technique for max-finding is to run two copies of a simple natural online algorithm (that costs too much when run by itself) in parallel. By judiciously exchanging information between the two copies, we can bound the cost incurred by the algorithm; we believe that this technique may have other applications to online algorithms.

## 1 Introduction

The questions of optimal searching, sorting, and selection lie at the very basis of the field of algorithms, with a vast literature on algorithms for these fundamental problems [1]. Traditionally the fundamental assumption in evaluating the performance of these algorithms has been the *unit-cost comparison model*—comparing any two elements costs one unit, and one of the goals is to devise algorithms that minimize the total cost of comparisons performed. Recently, Charikar et al. [2] posed the following problem: given a set $V$ of $n$ elements, where the cost of comparing $u$ and $v$ is $c_{uv}$, how should we design algorithms for sorting and selection so as to minimize the total cost of

* Research partly supported by NSF CAREER award CCF-0448095, and by an Alfred P. Sloan Fellowship.

all the comparisons performed? Note that these costs $c_{uv}$ are known to the algorithm, which can use them to decide on the sequence of comparisons. The case where all comparison costs are identical is just the unit-cost model. To measure the performance of algorithms in this model, Charikar et al. [2] used the framework of *competitive analysis*: they compared the cost incurred by the algorithm to the cost incurred by an optimal algorithm to *prove* the output correct.

The paper of Charikar et al. [2], and subsequent work by the authors [3] and Kannan and Khanna [4] considered sorting, searching, and selection for special cost functions (which are described in the discussion on related work). However, there seems to be no work on the case where the comparison costs form a *metric space*, i.e., where costs respect the triangle inequality $c_{uv} + c_{vw} \geq c_{uw}$ for all $u, v, w \in V$. Such situations may arise if the elements reside at different places in a communication network, and the communication cost of comparing two elements is proportional to the distance between them. An equivalent, and perhaps more natural way of enforcing the metric constraint on the costs is to say that the vertices in $V$ lie in an ambitent metric space $(X, d)$ (where $V \subseteq X$), where the cost $c_{ij}$ of comparing two vertices $i$ and $j$ is the distance $d(i, j)$ between them.

**Our Results.** In this paper, we initiate the study of these problems: in particular, we consider problems of max-finding and sorting with metric comparison costs. For the max-finding problem, our results show that the lower bound of $\Omega(n)$ on the competitive ratio arises only for arguably pathological scenarios, and substantially better guarantees can be given for metric costs.

**Theorem 1.** *Max-finding with metric costs has an $O(\log n)$-competitive algorithm.*

We improve the result for the special case when the points are located in $d$-dimensional Euclidean space:

**Theorem 2.** *There is an $O(d^3)$-competitive randomized algorithm for max-finding when the nodes lie on the $d$-dimensional grid and the distance between points is given by the $\ell_\infty$ metric; this yields an $O(d^{3(1+1/p)})$-competitive algorithm for $d$-dimensional $\ell_p$ space.*

For the problem of sorting $n$ elements in a metric, we give an $O(\log n)$-competitive algorithm for the case of hierarchically well-separated trees (HSTs). We then use standard results of Bartal [5], and Fakcharoenphol et al. [6] from the theory of metric embeddings to extend our results to general metrics. Our main theorem is the following:

**Theorem 3.** *There is an $O(\log^2 n)$-competitive randomized algorithm for sorting with metric costs.*

It can be seen that any algorithm for sorting with metric costs must be $\Omega(\log n)$-competitive even when the points lie on a star or a line—indeed, one can model unit-cost sorting and searching sorted lists in these cases. The question of closing the logarithmic gap between the upper and lower bounds remains an intriguing one.

**Our Techniques.** For the max-finding algorithm for general metrics, we use a simple algorithm that uses $O(\log n)$ rounds, eliminating half the elements in each round while paying at most $\mathsf{O}PT$. Getting better results turns out to be non-trivial even for very simple metrics: an illuminating example is the case where the comparison costs for elements $V = \{v_1, v_2, \ldots, v_n\}$ are given by $c_{v_i v_j} = |i - j|$; i.e., when the metric is generated by a path. (Note that this path has no relationship to the total order on $V$: it merely specifies the costs.)

Indeed, an $O(1)$-competitive algorithm for the path requires some work: one natural idea is to divide the line into two pieces, recursively find the maximum in each of these pieces, and then compare the two maxima to compute the overall maximum. However, a closer look indicates that this algorithm also gives us a competitive ratio of $\Omega(\log n)$. To fix this problem and reduce the expected cost to $O(\mathsf{O}PT)$, we make a simple yet important change: we run *two copies* of the above algorithm in parallel, transferring a small amount of information between the two copies after every round of comparisons. Remarkably, this subtle change in the algorithm gives us the claimed $O(1)$-competitive ratio for the line. In fact, this idea extends to the $d$-dimensional grid to give us $O(d)$-competitive algorithms—while the algorithm remains virtually unchanged, the proof becomes quite non-trivial for $d$-dimensions.

For the results on sorting, we first develop an algorithm for the case when the metric is a $k$-HST (which is a tree where the edges from each vertex to its children are $k$ times shorter than the edge to its parent). For these HST's, we show how to implement a "bottom-up mergesort" to get an (existentially optimal) $O(\log n)$-competitive algorithm; this is then combined with standard techniques to get the $O(\log^2 n)$-competitiveness for general metrics.

**Previous Work.** The study of the arbitrary cost model for sorting and selection was initiated by Charikar et al. [2]. They showed an $O(n)$-competitive algorithm for finding the maximum for general metrics. Tighter upper and matching lower bounds (up to constants) for finding the maximum were shown by Hartline et al. [7] and independently by the authors [3].

In the latter paper [3], the authors considered the special case of *structured costs*, where each element $v_i$ is assumed to have an inherent *size* $s_i$, and the cost $c_{v_i v_j}$ of comparing two elements $v_i$ and $v_j$ is of the form $f(s_i, s_j)$ for some function $f$; as expected, better results could be proved if the function $f$ was "well-behaved". Indeed, for monotone functions $f$, they gave $O(\log n)$-competitive algorithms for sorting, $O(1)$-competitive algorithms for max-finding, and $O(1)$-competitive algorithms for selection for the special cases of $f$ being addition and multiplication. Subseqently, Kannan and Khanna [4] gave an $O(\log^2 n)$-competitive algorithm for selection with monotone functions $f$, and an $O(1)$-competitive algorithm when $f$ was the $\min$ function.

**Formal Problem Definition.** The input to our problems is a *complete* graph $G = (V, E)$, with $|V| = n$ vertices. These vertices represent the elements of the total order, and hence each vertex $v \in V$ has a distinct *key value* denoted by $\mathsf{key}(v)$. (We use $x \prec y$ to denote that $\mathsf{key}(x) \leq \mathsf{key}(y)$.) Each edge $e = (u, v)$ has non-negative *length* or *cost* $c_e = c_{uv}$, which is the cost of comparing the elements $u$ and $v$. We assume that these edge lengths satisfy the triangle inequality, and hence form a metric space.

In this paper, we consider the problems of finding the element in $V$ with the maximum key value, and the problem of sorting the elements in $V$ according to their key values. We work in the framework of competitive analysis, and compare the cost of the comparisons performed by our algorithm to the cost incurred by the optimal algorithm which knows the results of all pairwise comparisons and just has to *prove* that the solution produced by it is correct. We shall denote the optimal solution by $\mathsf{O}PT \subseteq E$. Given a set of edges $E' \subseteq E$, we will let $c(E') = \sum_{e \in E'} c_e$, and hence $c(\mathsf{O}PT)$ is the optimal cost. Note that a proof for max-finding is a rooted spanning tree of $G$ with the maximum element at the root, and where the key values of vertices monotonically increase when moving from any leaf to the root; for sorting, a proof is the Hamilton path where key values monotonically increase from one end to the other.

## 2 Max-Finding in Arbitrary Metrics

For arbitrary metrics, we give an algorithm for finding the maximum element $v_{\max}$ of the nodes in $V$; the algorithm incurs cost at most $O(\log n) \times c(\mathsf{O}PT)$. Our algorithm proceeds in stages. In stage $i$, we have a subgraph $G_i = (V_i, E_i)$ such that $V_i$ contains $v_{\max}$; here $V_i \subseteq V$ , and $E_i = V_i \times V_i$ with the same costs as in $G$. We start with $G_0 = G$; in stage $i$, if $G_i$ has a single node, then it must be $v_{\max}$, else we do the following steps.

1. Find a minimum cost almost-perfect matching $M_i$ in $G_i$. (I.e., at most one node remains unmatched.)
2. For every edge $e = (u, v) \in M_i$, compare the end-points of $e$. (If $u$ is greater than $v$, then $u$ "wins" and $v$ "loses".)
3. Delete nodes which lost in the comparisons above to get the new set $V_{i+1}$ from $V_i$.

It is clear that the above algorithm correctly finds $v_{\max}$; there are $O(\log n)$ rounds since $|V_i| = \lceil n/2^i \rceil$, and hence the following lemma immediately implies that the cost incurred is $O(\log n) \times c(\mathsf{O}PT)$, this proving Theorem 1.

**Lemma 1.** *The cost of edges in $M_i$ is at most $c(\mathsf{O}PT)$.*

*Proof.* Given any set of $2k$ vertices in a tree $T$, one can find a pairing of these vertices into $k$ pairs so that the paths in $T$ between these pairs are edge disjoint (see, e.g., [8, Lemma 2.4]). We use this to pair off the vertices of $V_i$ in the tree $\mathsf{O}PT$; the total cost of the paths between them is an upper bound on a min-cost almost-perfect matching of $V_i$. Finally, since the paths between the pairs are edge disjoint, their total cost is at most $c(\mathsf{O}PT)$, and hence the cost of $M_i$ is at most $c(\mathsf{O}PT)$ as well. $\square$

## 3 Finding the maximum on a line

To improve on the results of the previous section, let us consider the case of the line metric; i.e., where the vertices $V = \{1, 2, \ldots, n\}$ lie on a line, and the cost of comparing two elements in $V$ is just the distance between them in this line. Let us assume that the line is unweighted, and consecutive points in $V$ are at unit distance from each other, and hence $c_{ij} = |i - j|$; we will indicate how to remove this simplifying assumption at the end of this section. We also assume that $n$ is a power of 2. For an element $x \in V$ which is not the maximum, let $g(x)$ be a *nearest* element to $x$ which has a key greater than $\mathsf{key}(x)$, and let $d(x)$ be the distance between $x$ and $g(x)$. Observe that in $\mathsf{O}PT$, the parent of $x$ must be at distance $d(x)$ from $x$, and hence $c(\mathsf{O}PT) = \sum_{x \neq v_{\max}} d(x)$.

Let us first look at a naïve scheme: we start off with a division $D_1$ of the line into two-node *segments* $\{[1,2],[3,4],\ldots,[n-1,n]\}$. In next division $D_2$, we pair off segments of $D_1$ to get $n/4$ segments $\{[1,2,3,4],\ldots,[n-3,n-2,n-1,n]\}$; similarly, $D_i$ has $n/2^i$ segments, each with $2^i$ nodes. We maintain the invariant that we know the maximum key element in each segment of $D_i$; when merging two segments, we compute the maximum by comparing the maxima of the two segments. However, this is just the algorithm of Section 2, and if we have $1 \prec 2 \prec \cdots \prec n$, then $c(\mathsf{O}PT) = n-1$ but our scheme costs $\Omega(n \log n)$.

**An algorithm which almost works.** A natural next attempt is to introduce randomization: to form the division $D_2$ from $D_1$, we toss an unbiased coin: if the result is "heads", we merge $[1,2],[3,4]$ into one segment (which we denote using the notation [1-4]), merge $[5,6],[7,8]$ into the segment $[5-8]$, and so on. If the coin comes up "tails", we *shift* over by one: $[1,2]$ forms a segment by itself, and from then on, we merge every two consecutive segments of $D_1$. Hence, with probability $\frac{1}{2}$, the segments in $D_2$ are $\{$[1-4], [5-8], $\ldots\}$, and with probability $\frac{1}{2}$, they are $\{$[1-2], [3-6], [7-10], $\ldots\}$. To get division $D_{i+1}$ from $D_i$, we flip an unbiased coin and either merge every pair of consecutive segments of $D_i$ beginning with the *first* segment, or merge every pair of consecutive segments starting at the *second* one. It is easy to see that all segments in $D_i$, except perhaps the first and last ones, have $2^i$ nodes. Again, the natural randomized algorithm is to maintain the maximum element in each segment of $D_i$: when combining segments of $D_i$ to form segments of $D_{i+1}$, we compare the two maxima to find the maximum of the newly formed segment. (We use *stage* $i$ to refer to the comparisons performed whilst forming $D_i$; note that stages begin at 1, and there are no comparisons in the first stage.)

The correctness of the procedure is immediate; to calculate the expected cost incurred, we charge the cost of a comparison to the loser—note that each node except $v_{\max}$ pays for exactly one comparison. We would like to show that the expected cost paid by $x \in V$ in our algorithm is $O(d(x))$. Let $S_i(x)$ denote the segment of $D_i$ containing $x$; the *size* of $|S_i(x)| \leq 2^i$, and the *length* of $S_i(x)$ is $|S_i(x)| - 1$. Note that if $2^k \leq d(x) < 2^{k+1}$, then $x$ definitely wins (and hence does not pay) in stages 1 through $k$; the following lemma bounds the probability that it loses in any stage $t \geq k+1$. (Recall that depending on the coin tosses, $x$ may nor may lose to $g(x)$.)

**Lemma 2.** *Let $2^k \leq d(x) < 2^{k+1}$. Then* $\mathbf{Pr}[x$ *loses in stage* $t] \leq 2^{-(t-k-2)}$.

*Proof.* Note that the lemma is vacuously true for $t \leq k+2$. Since $d(x) < 2^{k+1}$, nodes $x$ and $g(x)$ must lie in the same or consecutive segments in stage $k+1$. Now for $x$ to lose in stage $t$, it must not have lost in stages $\{k+2, k+3, \ldots, t-1\}$, and hence the segments containing $x$ and $g(x)$ must not have merged in these stages. Since we make independent decisions at each stage, the probability of this event happening is $(1/2)^{(t-1)-(k+2)+1} = 2^{-(t-k-2)}$. □

Since $x$ may have to pay as much as $2^{t+1}$ if it loses in stage $t$, the expected cost for $x$ is $\sum_{t \geq k} \Theta(2^k)$, which may be as large as $\Theta(2^k \cdot (\log n - k))$. Before we indicate how to fix the problem, let us note that our analysis is tight: for the example with $1 \prec 2 \prec \cdots \prec n$, the randomized algorithm incurs a cost $\Omega(n \log n)$.

**Two Copies Help: The Double-Random Algorithm.** Let us modify the above algorithm to maintain two independent *copies* of the line, which we call $L$ and $L'$. The partitions in $L$ will be denoted by $D_1, D_2, \ldots$, while those in $L'$ will be called $D'_1, D'_2, \ldots$. These partitions in the two lines are chosen independent of each other. Again, we maintain the maximum element of each segment in $D_i$ and $D'_i$, but also exchange some information between the lines. Consider the step of merging segments $S_1$ and $S_2$ to get a segment $S$ in $D_{i+1}$, and let $x_i$ be the maximum element of $S_i$. Before we compare $x_1$ and $x_2$, we check if $x_1$ has lost to an element $y \in S_2$ in some previous stage in $L'$: in this case, we know that $x_1 \prec y \prec x_2$, and hence can avoid comparing $x_1$ and $x_2$. Similarly, if $x_2$ has previously lost in $L'$ to some element $z \in S_1$, we can declare $x_1$ to be the maximum element of $S$. Only if neither of these fortuitous events occur, we compare $x_1$ and $x_2$. (The process for $L'$ is similar, and uses the information from $L$'s previous rounds.) The correctness of the algorithm follows immediately.

Notice that each element $x$ now loses exactly twice, once in each line, but the second loss may be implicit (without an actual comparison being performed). As before, we say that a node $x$ *loses in stage* $i$ *of* $L$ (or $L'$) if this is the first stage in which $x$ loses in $L$ (or $L'$). The node $x$ *pays* in stage $i$ of $L$ (or $L'$) if $x$ loses in stage $i$ of $L$ (or $L'$) *and* an actual comparison was made. While $x$ loses twice, it may possibly pay only once.

**Lemma 3.** *If $x, y \in V$ are at distance $d(x,y)$, then the probability (in either line) that $x$ and $y$ lie in different segments in $D_i$ is at most $d(x,y)/2^{i-1}$.*

*Proof.* Let $2^k \leq d(x,y) < 2^{k+1}$; the statement is vacuously true for $i-1 \leq k$. In stage $i-1 > k$, the nodes $x$ and $y$ must lie in either the same or consecutive segments in $D_{i-1}$. Now, if they were in different segments in $D_{i-1}$ (which inductively happens with probability at most $d(x,y)/2^{i-2}$), the chance that these segments do not merge in stage $i$ is exactly $\frac{1}{2}$, giving us the bound of $d(x,y)/2^{i-2} \times \frac{1}{2}$. $\square$

Let the node $g(x)$ lie to the left of $x$; the other case is symmetric, and proved identically. Let the distance $d(x) = d(x, g(x))$ satisfy $2^k \leq d(x) < 2^{k+1}$. Let $h(x)$ be the nearest point to the right of $x$ such that $x \prec h(x)$, and let $2^m \leq d(x, h(x)) < 2^{m+1}$. Note that if $x$ *pays* in stage $t$ of $L$, then $t \leq m+3$. Indeed, if the segment $S_{m+3}(x)$ is the leftmost or the rightmost segment, then it either contains $g(x)$ or $h(x)$, so it must have paid by then. Else, the length of $S_{m+3}(x) = 2^{m+3} - 1$, and since $d(g(x), h(x)) < 2^{m+1} + 2^{k+1} = 2^{m+2}$, the segment $S_{m+3}(x)$ must contain one of $g(x)$ or $h(x)$, so $t \leq m+3$. Moreover, since $S_t(x)$ must contain either $g(x)$ or $h(x)$, it follows that $t > k$. The following key lemma shows us that the probability of paying in stage $t \in [k+1, m]$ is small. (An identical lemma holds for $L'$.)

**Lemma 4.** *For $t \in [k+1, m]$,* **Pr**[*$x$ pays in stage $t$ of $L$*] $\leq 2^{-2(t-k)+5}$.

*Proof (Lemma 4).* Note that if $x$ pays in stage $t \leq m$ of $L$, then $x$ must have lost to some element to its left in $L$, since $d(x, h(x)) \geq 2^m$. Depending on whether $x$ loses in $L'$ before stage $t$ or not, there are two cases.

**Case I:** *$x$ has not lost in $D'_{t-1}$.* This implies that $x$ and $g(x)$ lie in different segments in $L'$, which by Lemma 3 has probability $\leq d(x, g(x))/2^{t-2} \leq 2^{-(t-k-3)}$. Now the chance that $x$ loses in $L$ in stage $t$ is $2^{-(t-k-2)}$ (by Lemma 2). Since the partitions are independently chosen, the two events are independent, which proves the lemma.

**Case II:** *$x$ has lost in stage $l \leq t-1$ in $L'$.* Since $l \leq m$, $x$ must have lost to some element $y$ to its left in $L'$; this $y$ is either $g(x)$, or lies to the left of $g(x)$. Consider stage $t-1$ in $L$: since the distance $d(y,x) < 2^l \leq 2^{t-1}$, the three elements $x, g(x)$ and $y$ lie in the union of two adjacent segments in $D_t$. Furthermore, $x$ must lie in a different segment from $y$ and $g(x)$, otherwise $x$ would have already lost in $L$ in stage $t-1$. Recall that if $x$ loses in stage $t$ in $L$, it must lose to a node to its left—since $t \leq m$, $h(x)$ is too far to the right. But this implies that $S_{t-1}(x)$ must merge in $L$ with $S_{t-1}(y) = S_{t-1}(g(x))$; in this case, no comparisons would be performed since $x$ had already lost to $y$ in $L'$. $\square$

Note that this lemma implies that the expected payment of $x$ for stages $k+1$ through $m$ is at most $\sum_{t=k+1}^{m} 2^{-2(t-k)+5} \times 2^t = O(2^k)$. The expected payment in stages $m+1$ to $m+3$ is at most $3 \cdot O(2^m) = O(2^k)$ by Lemma 2, which proves:

**Theorem 4.** *The Double-Random algorithm is an $O(1)$-competitive algorithm for max-finding on the line.*

To end, note that the assumption of unit length edges can be removed: by scaling and translation, all distances can be made integers. We can add dummy vertices at all integers $i$ that do not correspond to a vertex in $V$, where all these vertices have key values less than those of all the non-dummy vertices. Running Double-Random on this augmented line allows us to run the algorithm without increasing the cost. (We can even space and time overhead by maintaining only segments containing at least one non-dummy node.)

## 4 Max-Finding for Euclidean metrics

In this section, we extend our algorithm for the line metric to arbirary Euclidean metrics: the basic idea of running two copies of the algorithm and judiciously exchanging information will be used again, but the proof becomes substantially more involved. We give the proof for the 2-d case; the proof for the general case is deferred to the final version of the paper.

**The General Double-Random Algorithm.** As in the case of the line, we begin with the simplifying assumption that the nodes in $V$ form a subset of the unit-weight $n \times n$ grid; we refer to this underlying grid as $\mathbb{M} = \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$. (This assumption can be easily discharged, as for the case of the line; we omit the details here.) Hence each point $v \in V$ corresponds to a point $(v_x, v_y) \in \mathbb{M}$, with $1 \leq v_x, v_y \leq n$. In fact, if $P^x$ denotes the path along the $x$-axis from 1 to $n$, and $P^y$ denotes a similar path along the $y$-axis, then we can identify the grid $\mathbb{M}$ with the cartesian product $P^x \times P^y$. To construct partitions $D_1, D_2, \ldots$ of the grid, we build stage-$i$ partitions $D_i^x$ and $D_i^y$ for the paths $P^x$ and $P^y$: the rectangles in $\mathbb{M}$'s partition correspond to the products of the segments in $D_i^x$ and $D_i^y$, and hence a square in $D_{i+1}$ is formed by merging at most four squares in $D_i$.[3] The random partitioning schemes for $P^x$ and $P^y$ evolve independently of each other.

[3] We abuse notation and say "squares" even though the pieces may be rectangles.

Again, we maintain the invariant that we know the maximum element in each square of the partition $D_i$, and as in the case of the line, we do not want to perform three comparisons when merging squares in $D_i$ to get $D_{i+1}$. Hence we maintain two independent copies $\mathbb{M}$ and $\mathbb{M}'$ of the grid, with $D_i$ and $D_i'$ being the partitions in the two grids at stage $i$. Suppose $\{x_1, x_2, x_3, x_4\}$ are the four maxima of four squares $S_i$ being merged into a new square $S$ in $\mathbb{M}$: for each $i \in [1, 4]$, we check if $x_i$ has lost to some $y \in S$ in a previous stage in $\mathbb{M}'$, and if so, we remove $x_i$ from consideration; we finally compare the $x_i$'s that remain. The correctness of the algorithm is immediate, and we just have to bound the costs incurred.

## 4.1 Cost of the Double-Random Algorithm in Two Dimensions

We charge the cost of each comparison to the node that loses in that comparison, and wish to upper bound the cost charged to any node $p \in \mathbb{M}$. Let $G(p) = \{q \mid p \prec q\}$ be the set of nodes with keys greater than $p$; fix a vertex $g(p) \in G(p)$ *closest* to $p$, and let $d(p)$ be the distance between $p$ and $g(p)$, with $2^\ell \leq d(p) < 2^{\ell+1}$. Since we focus on the node $p$ for the entire argument, we shift our coordinate system to be *centered at* $p$: we renumber the vertices on the paths $P^x$ and $P^y$ so that the vertex $p$ lies at the "origin" of the 2-d grid. Formally, we label the nodes $p_x \in P^x$ and $p_y \in P^y$ as 0; the other vertices on the paths are labeled accordingly. This naturally defines four quadrants as well. Let $D_i^x$ be the projection of the partition $D_i$ on the line $P^x$, and $D_i^y$ be its projection of $P^y$. ($D_i^{x\prime}$ and $D_i^{y\prime}$ are defined similarly for the grid $\mathbb{M}'$.)

Let us note an easy lemma, bounding the chance that $p$ and $g(p)$ are separated in the partition $D_i$ in $\mathbb{M}$. (Such a lemma holds for partition $D_i'$ of $\mathbb{M}'$, of course.)

**Lemma 5.** **Pr**[*$p$ and $g(p)$ lie in different squares of $D_i$*] $\leq 2^{-(i-\ell-3)}$.

*Proof.* Let the distance from $p$ to $g(p)$ along the two axes be $d_x = d(p_x, g(p)_x)$ and $d_y = d(p_y, g(p)_y)$ with $\max\{d_x, d_y\} = d(p)$. By Lemma 3, the projections $p_x$ and $g(p)_x$ do not lie in the same stage-$i$ interval of $P^x$ with probability at most $d_x/2^{i-1}$. Similarly, $p_y$ and $g(p)_y$ do not lie in the same stage-$i$ interval of $P^y$ with probability at most $d_y/2^{i-1}$; a trivial union bound implies that the probability that $2d(p)/2^{i-1} < 2^{-(i-\ell-3)}$. □

We now prove that the expected charge to a node $p$ is $O(2^\ell)$, where the distance between $p$ and a closest point $g(p)$ in the set $G(p) = \{q \in V \mid p \prec q\}$ lies in the interval $[2^\ell, 2^{\ell+1})$. Let $H(p) = G(p) - \{g(p)\}$. Let $S_i(p)$ and $S_i'(p)$ be the squares in $D_i$ and $D_i'$ respectively that contain $p$. We will consider two events of interest:

1. Let $\mathcal{A}_i$ be the event that $p$ pays in $\mathbb{M}$ in stage $i$, and the square $S_i(p)$ contains at least one point from $H(p)$, but does not contain $g(p)$.
2. Let $\mathcal{B}_i$ be the event that $p$ pays in $\mathbb{M}$ in stage $i$, and $S_i$ contains $g(p)$.

Note that $\mathcal{A}_i \cap \mathcal{B}_i = \emptyset$; also, if $p$ pays in stage $i$ in $\mathbb{M}$, then either $\mathcal{A}_i$ or $\mathcal{B}_i$ must occur. Also, **Pr**$[\mathcal{A}_i \cup \mathcal{B}_i] > 0$ only when $p$ and some element of $G(p)$ lie in the same square in stage $i$ in $\mathbb{M}$: since any two points in such a square are at $\ell_\infty$ distance $\leq 2^\ell - 1$ from each other, and each element of $G(p)$ has $\ell_\infty$ distance at least $2^\ell$ from $p$, it suffices to consider the case $i > \ell$. Theorems 5 and 6 will show that $\sum_i \mathbf{Pr}[\mathcal{A}_i] \times 2^i + \sum_i \mathbf{Pr}[\mathcal{B}_i] \times 2^i = O(2^\ell)$. This shows that $p$ pays only $O(2^\ell)$ in $\mathbb{M}$; a similar bound holds for $\mathbb{M}'$, which proves the claim that Double-Random is $O(1)$-competitive in the case of two-dimensional grids.

**Theorem 5.** $\sum_i \mathbf{Pr}[\mathcal{A}_i] \times 2^i = O(2^\ell)$.

*Proof.* Let us define two events $\mathcal{E}_x$ and $\mathcal{E}_y$. Let $\mathcal{E}_x$ be the event that $p_x$ and $g(p)_x$ lie in different segments in $D_i^x$, and $\mathcal{E}_y$ be the event that $p_y$ and $g(p)_y$ lie in different segments in $D_i^y$. Note that $\mathcal{A}_i \setminus (\mathcal{E}_x \cup \mathcal{E}_y) = \emptyset$, and hence

$$\mathbf{Pr}[\mathcal{A}_i] \leq \mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_x] + \mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_y] \tag{4.1}$$

Let us now estimate $\mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_x]$, the argument for the other term is similar. Assume (w.l.o.g.) that $g(p)_x$ lies to the left of $p_x$, and let the points between $g(p)_x$ and $p_x$ (including $p_x$, but not including $g(p)_x$) in $P^x$ be labeled $p_x^1, p_x^2, \ldots, p_x^k = p_x$ from left to right. Define $\mathcal{F}_j$ as the event that the segment $S_i^x(p)$ in $D_i^x$ containing $p_x$ has $p_x^j$ as its left end-point. Note that the events $\mathcal{F}_j$ are disjoint, and $\mathcal{E}_x = \cup_{j=1}^k \mathcal{F}_j$. Thus it follows that

$$\mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_x] = \sum_j \mathbf{Pr}[\mathcal{A}_i \cap \mathcal{F}_j] = \sum_j \mathbf{Pr}[\mathcal{A}_i \mid \mathcal{F}_j]\ \mathbf{Pr}[\mathcal{F}_j]. \tag{4.2}$$

If $\mathcal{F}_j$ occurs then the end-points of the edge connecting $p_x^j$ and $p_x^{j-1}$ (where $p_x^0 = g(p)_x$) lie in different segments of $D_i^x$. Lemma 3 implies that this can happen with probability at most $\frac{1}{2^{i-1}}$. Thus, we get

$$\mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_x] \leq 2^{-(i-1)} \times \sum_j \mathbf{Pr}[\mathcal{A}_i \mid \mathcal{F}_j]. \tag{4.3}$$

Define $I_i^j$ as the segment of length $2^i$ in $P^x$ containing $p_x$ and having $p_x^j$ as its left end-point. Let $q(i,j) \in H(p)$ be such that $q(i,j)_x \in I_i^j$ and $|q(i,j)_y - p_y|$ is minimum; in other words, the point closest to the $x$-axis whose projection lies in $I_i^j$. If no such point exists, then we say that $q(i,j)$ is undefined. Let $\delta(i,j) = |q(i,j)_y - p_y|$ if $q(i,j)$ is defined, and $\infty$ otherwise. Notice that for a fixed $j$, $\delta(i,j)$ is a decreasing function of $i$.

Assume $\mathcal{F}_j$ occurs for some fixed $j$: for $\mathcal{A}_i \neq \emptyset$, $S_i(p)$ must contain a point in $H(p)$, and hence $\delta(i,j) \leq 2^i$; let $i(j)$ be the smallest value of $i$ for which $\delta(i,j) \leq 2^i$. Due to $\delta(i,j)$ being a decreasing function in $i$, $\delta(i,j) > 2^i$ for all $i < i(j)$, and for all $i \geq i(j)$, $\delta(i,j) \leq 2^i$. Now suppose $i > i(j)$; note the strict inequality, which ensures that $q(i-1,j)$ exists. Again assume that $\mathcal{F}_j$ occurs: now for $\mathcal{A}_i$ to occur, the square $S_{i-1}(p)$ cannot contain any point of $H(p)$. In particular, it cannot contain $q(i-1,j)$.

**Lemma 6.** *If $\mathcal{F}_j$ occurs and $i > \ell + 1$, then $p_x$ and $q(i-1,j)_x$ lie in the same segment of $D_{i-1}^x$.*

*Proof.* It will suffice to show the claim that segment containing $p_x$ in $D_{i-1}^x$ also has $p_x^j$ as the left end-point; since $q(i-1,j)_x$ also lies in this segment, the lemma follows. To prove the claim, note that the distance $|p_x^j - p_x| \leq |g(p)_x - p_x| - 1 \leq (2^{\ell+1} - 1) - 1 = 2^{\ell+1} - 2$. Since $i > \ell$, it follows that $p_x$ and $p_x^j$ must lie in the same or in adjacent segments of $D_{i-1}^x$; we claim that the former is true. Indeed, suppose they were in different segments: since the segment of $D_{i-1}^x$ containing $p_x^j$ must have width $2^{i-1} - 1 \geq 2^{\ell+1} - 1$ which is greater than $|p_x^j - p_x|$, it must happen that $p_x^j$ lies in the interior of this segment, and hence $\mathcal{F}_j$ could not occur. $\square$

Note that since the projections of $p$ and $q(i-1,j)$ on the $x$-axis lie in the same segment implies that the projections $p_y$ and $q(i-1,j)_y$ on the $y$-axis must lie in different segments of $D^y_{i-1}$. Since this event is independent of $\mathcal{F}_j$, we can use Lemma 3 to bound the probability: indeed, we get that for $i > i(j)$,

$$\mathbf{Pr}[\mathcal{A}_i \mid \mathcal{F}_j] \leq \tfrac{\delta(i-1,j)}{2^{i-1}}. \tag{4.4}$$

We are now ready to prove the theorem.

$$\begin{aligned}
\textstyle\sum_i 2^i \cdot \mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_x] &\leq 2 \ \textstyle\sum_i \sum_j \mathbf{Pr}[\mathcal{A}_i \mid \mathcal{F}_j] && \text{(from (4.3))} \\
&= 2 \ \textstyle\sum_j \sum_{i \geq i(j)} \mathbf{Pr}[\mathcal{A}_i \mid \mathcal{F}_j] \\
&\leq 2 \ \textstyle\sum_j \left(1 + \sum_{i > i(j)} \frac{\delta(i-1,j)}{2^{i-1}}\right) && \text{(from (4.4))} \\
&\leq 2 \ \textstyle\sum_j 3 \qquad \leq 6 \cdot 2^{\ell}.
\end{aligned}$$

where penultimate inequality follows from the fact that $\delta(i,j)$ is a decreasing function of $i$, and hence $\sum_{i>i(j)} \frac{\delta(i-1,j)}{2^{i-1}}$ is a dominated by a geometric sum. A similar calculation proves that $\sum_i 2^i \cdot \mathbf{Pr}[\mathcal{A}_i \cap \mathcal{E}_y]$ is $O(2^{\ell})$, which in turn completes the proof of Theorem 5. □

Now that we have bounded the charge to $p$ due to the events $\mathcal{A}_i$ by $O(2^{\ell})$, we turn our attention to the events $\mathcal{B}_i$, and claim a similar result for this case.

**Theorem 6.** $\sum_i 2^i \cdot \mathbf{Pr}[\mathcal{B}_i] \leq O(2^{\ell})$.

*Proof (Theorem 6).* Recall that if $p$ loses in stage $i$, then $i > \ell$: hence we define a set of events $\mathcal{E}_{\ell+1}, \ldots, \mathcal{E}_{i-3}$, where $\mathcal{E}_j$ occurs if $p$ loses in stage $j$ of $\mathbb{M}'$. Also, define the event $\mathcal{E}_0$ occur if $p$ does not lose in $\mathbb{M}'$ till stage $i-3$. Note that exactly one of these events can occur, and hence

$$\mathbf{Pr}[\mathcal{B}_i] = \mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_0]\mathbf{Pr}[\mathcal{E}_0] + \textstyle\sum_{j=\ell+1}^{i-3} \mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_j]\mathbf{Pr}[\mathcal{E}_j]. \tag{4.5}$$

The next two lemmas give us bounds on the probability of each of the terms in the summation.

**Lemma 7.** *If $i > \ell + 1$, then* $\mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_0]\mathbf{Pr}[\mathcal{E}_0] \leq 2^{-(2i-2\ell-10)}$.

*Proof.* Lemma 5 implies that $\mathbf{Pr}[\mathcal{E}_0] \leq 2^{-((i-3)-\ell-3)}$. Now given $\mathcal{E}_0$, $p$ must not lose till stage $i-1$ in $\mathbb{M}$ for $\mathcal{B}_i$ to occur. But this event is independent of $\mathcal{E}_0$, and hence Lemma 5 implies that $\mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_0]$ is at most $2^{-((i-1)-\ell-3)}$. Multiplying the two completes the proof. □

**Lemma 8.** $\mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_j]\mathbf{Pr}[\mathcal{E}_j] \leq 2^{-(2i-2\ell-9)}$.

*Proof.* For the event $\mathcal{E}_j$ to occur, $p$ does not lose till stage $j-1$ in $\mathbb{M}'$; now applying Lemma 5 gives us that $\mathbf{Pr}[\mathcal{E}_j] \leq 2^{-((j-1)-\ell-3)}$. Also, note that $\ell < j \leq i-3$ for us to be in this case.

Now let us condition on $\mathcal{E}_j$ occurring: let $p$ lose to some $q$ in stage $j$ of $\mathbb{M}'$, and hence $|p_x - q_x|, |p_y - q_y| < 2^j$. Now consider stage $i-1$ of $\mathbb{M}$. We claim that $p_x, q_x$ and $g(p)_x$

do not all lie in the same segment of $D^x_{i-1}$. Indeed, since the distance $|p_y - g(p)_y| < 2^{\ell+1} \leq 2^{i-2}$, the triangle inequality ensures that $|q_y - g(p)_y| \leq |p_y - g(p)_y| + |p_y - q_y| \leq 2^{i-1}$, and hence the distance between any two points in the set $\{p_y, q_y, g(p)_y\}$ is at most $2^{i-1}$. Thus two of these points must lie in the same segment in $D^y_{i-1}$ in $\mathbb{M}$. If all three lay in the same segment of $D^x_{i-1}$, two of these points would lie in the same square in $D_{i-1}$. Now if $p$ was one of these points, then $p$ would lose before stage $i$ and $\mathcal{B}_i$ would not occur. If $g(p)$ and $q$ would lie in the same square of $D_{i-1}$, then $p$ and $q$ would be in the same square in $D_i$, and then $p$ would not pay. Therefore, all three of $p_x, q_x$ and $g(p)_x$ cannot lie in the same segment of $D^x_{i-1}$; similarly, $p_y, q_y$ and $g(p)_y$ can not lie in the same segment of $D^y_{i-1}$.

Hence one of the following two events must happen: either (1) $p_x, g(p)_x$ lie in different segments of $D^x_{i-1}$ and $p_y, q_y$ lie in different segments of $D^y_{i-1}$, or (2) $p_x, q_x$ lie in different segments of $D^x_{i-1}$ and $p_y, g(p)_y$ lie in different segments of $D^y_{i-1}$. Lemma 3 implies that the probability of either of these events is at most $2^{-(i-\ell-2)} \cdot 2^{-(i-j-2)}$, and hence $\mathbf{Pr}[\mathcal{B}_i \mid \mathcal{E}_j] \leq 2^{-(2i-\ell-j-5)}$. Finally, multiplying this with $\mathbf{Pr}[\mathcal{E}_j] \leq 2^{-((j-1)-\ell-3)}$ completes the proof.

Now combining (4.5) with Lemmas 7 and 8, we see that if $i > \ell + 1$, then $\mathbf{Pr}[\mathcal{B}_i] \leq O\big(\frac{i-l}{2^{2i-2\ell}}\big)$. Thus,

$$\textstyle\sum_{i \geq \ell} 2^i \cdot \mathbf{Pr}[\mathcal{B}_i] \leq O\big(2^\ell \cdot \sum_{i > \ell} \big(\frac{i-l}{2^{i-l}}\big)\big) = O(2^\ell). \tag{4.6}$$

This completes the proof of Theorem 6.

□

## 5 Sorting with Metric Comparison Costs

We now consider the problem of sorting the points in $V$ according to their key values. Let $\mathsf{O}PT$ be the set of $n-1$ edges going between consecutive nodes in sorted order. A rooted tree $T$ is called a 2-HST if the lengths of all edges at any level of $T$ are the same, and the lengths of consecutive edges on any root-leaf path decrease by a factor of exactly 2. We assume that each internal node of $T$ has at least 2 children. Indeed, if a node has exactly one child, we can contract this edge – this will change distances between leaves up to a constant factor only. Let us denote the set of leaves of the 2-HST tree $T$ by $V$, and let $|V| = n$. The following theorem is the main technical result of this section.

**Theorem 7.** *Given $n$ elements, and the metric generated by the leaves of a 2-HST, there is an algorithm to sort the elements with a cost of $O(\log n) \times c(\mathsf{O}PT)$.*

Using standard results on approximating arbitrary metrics by probability distributions on metrics generated by HSTs [5,6], the above theorem immediately implies Theorem 3.

*Proof (Theorem 7).* For any rooted subtree $H$ of $T$, let $\mathsf{O}PT(H)$ denote the optimal set of comparisons to sort the leaves in $H$, and let $c(\mathsf{O}PT(H))$ be their cost. Let $h$ be the root of $H$, and $h$'s children be $h_1, \dots, h_r$; let the subtree rooted at $h_i$ be $H_i$. Consider $\mathsf{O}PT(H)$, and let a *segment* of $\mathsf{O}PT(H)$ be a maximal sequence of consecutive vertices in $\mathsf{O}PT(H)$ belonging to the same sub-tree $H_i$ for some $i$. Clearly, we can divide $\mathsf{O}PT(H)$ uniquely into node-disjoint segments—let $\mathsf{segs}(H)$ denote the number of these disjoint segments. Let $d(H)$ denote the cost of an edge joining $h$ to one of its children; recall that all these edges have the same cost. We omit the proof of the following simple lemma.

**Lemma 9.** $c(\mathsf{O}PT(H)) \geq \sum_{i=1}^{r} c(\mathsf{O}PT(H_i)) + (\mathsf{segs}(H) - 1) \cdot d(H)$.

Our algorithm sorts the leaves of $T$ in a bottom-up manner, by sorting the leaves of various subtrees, and then merging the results. For subtrees which just consist of a leaf, there is nothing to do. Now let $H, h, H_i, h_i$ be as above, and assume we have sorted the leaves of $H_i$ for all $i$: we want to merge these sorted lists to get the sorted list for the leaves of $H$. The following lemma, whose proof we omit, shows that we can do this without paying too much.

**Lemma 10.** *There is an algorithm to merge the sorted lists for $H_i$ while incurring a cost of* $O(\mathsf{segs}(H) \cdot \log n \cdot d(H))$.

We now complete the proof of Theorem 7. If $\mathsf{cost}(H)$ is the cost incurred to sort the subtree $H$, we claim $\mathsf{cost}(H) \leq \alpha \cdot \log n \cdot c(\mathsf{O}PT(H))$ for some constant $\alpha$. The proof is by induction on the height of the tree: the base case is when $H$ is a leaf, and $\mathsf{cost}(H) = c(\mathsf{O}PT(H)) = 0$. If $H, H_i$ are as above, and if our claim is true for $H_i$, then Lemma 10 implies that

$$\begin{aligned}\mathsf{cost}(H) &\leq \textstyle\sum_i \mathsf{cost}(H_i) + O(\mathsf{segs}(H) \cdot \log n \cdot d(H)) \\ &\leq \textstyle\sum_i \alpha \cdot \log n \cdot c(\mathsf{O}PT(H_i)) + O(\mathsf{segs}(H) \cdot \log n \cdot d(H) \\ &\leq \alpha \cdot \log n [\textstyle\sum_i c(\mathsf{O}PT(H_i)) + (\mathsf{segs}(H) - 1)] \qquad (5.7)\end{aligned}$$

provided $\alpha$ is large enough. (The last inequality used the fact that since $\mathsf{segs}(H) \geq 2, \mathsf{segs}(H) = O(\mathsf{segs}(H) - 1)$. But (5.7) is at most $\alpha \cdot \log n \cdot c(\mathsf{O}PT(H))$, by Lemma 9, which proves Theorem 7. □

## References

1. Knuth, D.E.: The art of computer programming. Volume 3: Sorting and searching. Addison-Wesley Publishing Co., Reading, Mass. (1973)
2. Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J., Raghavan, P., Sahai, A.: Query strategies for priced information. In: Proc. 32nd ACM STOC. (2000) 582–591
3. Gupta, A., Kumar, A.: Sorting and selection with structured costs. In: Proc. 42nd IEEE FOCS (2001) 416–425
4. Kannan, S., Khanna, S.: Selection with monotone comparison costs. In: Proc. 14th ACM-SIAM SODA (2003) 10–17
5. Bartal, Y.: Probabilistic approximations of metric spaces and its algorithmic applications. In: Proc. 37th IEEE FOCS. (1996) 184–193
6. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Proc. 35th ACM STOC (2003) 448–455
7. (Hartline, J., Hong, E., Mohr, A., Rocke, E., Yasuhara, K.) As reported in [3].
8. Kleinberg, J.: Detecting a network failure. Internet Math. **1** (2003) 37–55