The Impossibility of Simultaneous Time and I/O Optimality for The Planar Maxima and Convex Hull Problems

Peyman Afshani
Aarhus University, Denmark

Gerth Stølting Brodal
Aarhus University, Denmark

Nodari Sitchinava
University of Hawai'i at Mānoa, USA

Abstract

We prove that no deterministic output-sensitive algorithm for the planar convex hull and maxima problems can obtain both optimal time and I/O complexity, where the optimality is defined with respect to both the input and output sizes. This explains why the best previous algorithms achieved an optimal I/O bound at the cost of sub-optimal running time (Goodrich et al. [FOCS, 1993]). To the best of our knowledge, the impossibility of simultaneous optimality was only shown previously for the permutation problem by Brodal and Fagerberg [STOC, 2003]. Our results imply that no optimal deterministic output-sensitive cache-oblivious algorithm exists for either problem. In addition, we present simple deterministic algorithms that match our lower bounds and that provide a trade-off between time and I/Os. On the other hand, a simple modification of our deterministic algorithm results in a randomized algorithm that simultaneously achieves optimal (worst-case) time and optimal expected I/O bounds.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases External Memory model, cache-oblivious algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.115

Category Track A: Algorithms, Complexity and Games

Related Version *Full version*: https://arxiv.org/abs/2605.09464

Funding *Peyman Afshani*: Supported by DFF (Danmarks Frie Forskningsfond) of Danish Council for Independent Research under grant ID 10.46540/3103-00334B.
Gerth Stølting Brodal: Supported by Independent Research Fund Denmark grant 9131-00113B.
Nodari Sitchinava: Supported by NSF grant 2432018.

Acknowledgements The authors want to thank Ronitt Rubinfeld for suggesting this topic.

1 Introduction

We study the planar convex hull problem in the *external memory (EM)* [8, 41] and *cache-oblivious (CO)* [25] settings. The surveys on the EM model often mention that 2D convex hull can easily and optimally be solved in this model via sorting and scanning, including the output-sensitive variant [27, 40]. We report a very surprising discovery that the output-sensitive planar convex hull problem cannot be solved optimally both with respect to the external memory cost and the internal memory computational cost (i.e., running time), simultaneously, i.e., by the same algorithm! Consequently, this means that there is no optimal deterministic cache-oblivious algorithm for the output-sensitive planar convex hull problem. We also note that the existing I/O-efficient algorithms for the problem have substantially sub-optimal

Figure 1 Examples of a convex hull (left) and the maxima (right) of a planar point set. We let N and H denote the number of input and output points, respectively.

(internal memory) computational cost. We improve this sub-optimality to a very small factor (of an inverse Ackermann-like function) by giving a very simple algorithm, but with a non-trivial analysis. In addition, we prove lower bounds showing that such sub-optimality is necessary. Our lower bound applies even to the simpler problem of computing the maxima of a set of points in 2D [30], a fundamental problem in its own right. See Figure 1 for examples of the two problems.

To the best of our knowledge, this is only the second time that a "separation" between I/O and computational cost has been found, meaning, a problem for which optimal I/O cost and the optimal computational cost cannot be obtained simultaneously by the same deterministic algorithm but they can be achieved separately by different algorithms. Such a separation result immediately implies that an optimal cache-oblivious comparison-based algorithm cannot exist for such problems. This is because an optimal cache-oblivious algorithm performs optimal number of I/Os for all parameters M and B, including for $M = O(1)$ and $B = 1$. However, for these parameters, each I/O can result in at most $O(M) = O(1)$ useful (non-redundant) comparisons, i.e., comparisons that reveal new information. Since an optimal algorithm would not perform redundant computation, the lower bound on suboptimal computation cost (number of comparisons) implies suboptimal I/O complexity for parameters $M = O(1)$ and $B = 1$. There are very few instances of such impossibility result for cache-oblivious algorithms [7, 12, 16]. Below, we first present the definitions, as well as the relevant models of computation before presenting a more detailed statement of our results.

Models of computation. The *external memory (EM)* model (a.k.a. the *I/O* model) was introduced by Aggarwal and Vitter [8, 41] to model the cost of accessing memory in a machine with two levels of memory: a fast but limited *internal* memory of size M words and a slow but conceptually unlimited *external* memory. To perform computation, data must reside in the internal memory. Both memories are divided into *blocks* of B words and the transfer of data between the two memories are performed via reading or writing blocks. Each such transfer defines an *input/output (I/O)* operation and the *I/O complexity* of an algorithm is the number of I/Os performed during its execution. This captures the *cost of memory accesses* and ignores the computational steps performed in the internal memory (the *running time* in the standard algorithm analysis). For convenience we define $m = M/B$ and $n = N/B$.

However, modern systems consist of more than two levels of memory hierarchy, e.g., disk, DRAM, multiple levels of cache, and registers. One way to deal with deeper memory hierarchies is via *cache-oblivious (CO) algorithms* [25, 26] which are designed in the classical RAM model (i.e., without using M or B parameters), but they are analyzed in the *ideal cache* model, a variant of the EM model. In particular, during the analysis, it is assumed that there is a *cache* of m blocks and whenever the algorithm accesses an element, a block of size B containing that element is loaded into the cache via *an ideal offline paging algorithm*,

which makes the optimal choice when deciding which elements to evict from the cache. The ideal paging algorithm can be replaced by the *Least Recently Used (LRU)* algorithm as it offers a constant factor bicriteria approximation under a reasonable resource augmentation assumption [26]. Since the values of M and B are not known to a CO algorithm, if a CO algorithm achieves I/O optimality with respect to an arbitrary choice of parameters M and B, it is also optimal for every level of the memory hierarchy.

For many problems, optimal CO and EM algorithms are known. However, there are very few actual separation results between EM and CO computations and we mention them here. In the EM model, the problem of sorting a set of N comparable items has the tight bound of $\Theta(n \log_m n)$ [8,41]. However, Brodal and Fagerberg [16] showed that no CO comparison-based sorting algorithm can achieve the same optimal bound unless $M = \Omega\big(B^{1+\varepsilon}\big)$ for some constant $\varepsilon > 0$. This assumption is often known as the *tall cache* assumption. It is also known that CO data structures for range reporting require asymptotically more space than their EM counterparts [6, 7]. Finally, Arge and Thorup [12] specifically studied possible trade-offs between I/O complexity and time in the word-RAM model, highlighting that attaining optimality in both metrics at the same time can be challenging.

Prior results on convex hull computation. Given a set P of N points in the plane, the *planar convex hull* problem asks to compute the smallest convex polygon containing P. It can be solved in $O(N \log N)$ time using many techniques [9,13,28,33,34]. Graham's scan [28] is perhaps the most classical solution which involves sorting and a linear scan using a stack. The $\Omega(N \log N)$-time lower bound has been proven in various models [34,39,42] and holds even if the points on the convex hull can be returned in an arbitrary order [34]. When the size H of the convex hull is small (e.g., for uniformly random points in a fixed polygon [22]), *output-sensitive* algorithms improve the running time to $O(N \log H)$ [18,29]. Kirkpatrick and Seidel [29] proved a matching $\Omega(N \log H)$ lower bound in the algebraic decision tree model.

Obtaining an EM or CO algorithm for the planar convex hull is straightforward: sort the points using $O(n \log_m n)$ I/Os via one of the optimal EM or CO sorting algorithms [8,26,41], followed by the stack-based Graham Scan algorithm [28] that requires $O(n)$ I/Os.

An EM algorithm with the optimal output-sensitive I/O bound for convex hull was presented by Goodrich et al. [27] and it achieves $O\big(n \log_m \frac{H}{B}\big)$ I/Os but its running time (which was not mentioned) is $O(N \log H + N \log m)$, which is sub-optimal. Arge and Miltersen [11] showed a matching $\Omega\big(n \log_m \frac{H}{B}\big)$ I/O lower bound for the output-sensitive convex hull algorithms. But surprisingly, there has been no output-sensitive CO algorithm that matches this bound and the best known algorithm only achieves $O(n(\log_m H + \log\log m))$ I/Os [5].[1] Interestingly, the extra $\log\log m$ term comes from trying to "guess" the output size H and, thus, the algorithm can be made optimal if H is known.

Our contributions. Table 1 presents a summary of our and previous results. We develop deterministic CO algorithms for the planar maxima (Section 3) and convex hull (Section 5) problems that can trade-off optimality between time and I/O complexity via a parameter s. We prove that no *deterministic* algorithm can obtain both optimal time of $\Theta(N \log H)$ and optimal I/O complexity of $\Theta\big(n \log_m \frac{H}{B}\big)$ simultaneously for these two problems (Sections 4 and 6). Our lower bounds are proven in the comparison-based model for the maxima problem and in a model, where comparisons are generalized to geometric predicates on a constant

[1] Note that the tall cache assumption, on which the algorithm [5] relies, implies $\log_m H = \Theta\big(\log_m \frac{H}{B}\big)$.

Table 1 Previous results and our contributions. H is the size of the convex hull or the number of maxima points, s is an integer parameter, $A_s(\cdot)$ is an Ackermann-like function, $\alpha_s(\cdot)$ is its inverse, $\lambda_s(\cdot)$ is the s-th function in its inverse hierarchy (see Section 2.1 for details), and $O_E(\cdot)$ denotes an expected complexity bound. CO results hold under the tall cache assumption and all results apply to both 2D maxima and 2D convex hull problems.

Model	Time	I/O Complexity	Notes
RAM	$O(N \log N)$	-	Classic CH [9, 13, 28, 33, 34]
RAM	$\Theta(N \log H)$	-	[18, 29]
EM	$O(N \log N)$	$O(n \log_m n)$	Classic I/O [8, 26, 28]
EM	$O(N(\log H + \log m))$	$\Theta\left(n \log_m \frac{H}{B}\right)$	[11, 27]
CO	$O(N \log H)$	$O(n(\log_m H + \log\log m))$	[5]
CO	$O(N \log H)$	$O_E(n \log_m H)$	new, randomized
CO	$O(N(\log H + \log s))$	$O(n(\log_m sH + \alpha_s(\min\{H, m\})))$	new
CO	$O(N(\log H + \lambda_s(N)))$	$O(n(\log_m H + s))$	new
EM/CO	$O(N \cdot A_s(H))$	$\Omega\Big(n\big(\alpha_1\big(\min\big\{H, \sqrt{N/M}\big\}\big) - s\big)\Big)$	new

number of points (formally defined in Section 6), for the convex hull problem. The lower bound applies to any deterministic algorithm that spends up to $N \cdot A_s(H)$ time, where $A_s(N)$ is an Ackermann-like function (formally defined in Section 2.1) and s is a positive integer parameter. For example, it applies to all known output-sensitive algorithms which take $O(N \log H)$ time or $O(NH)$ time, and even to algorithms that take $O\left(N2^H\right)$ time, and it gives an $\Omega\Big(n \cdot \alpha_1\big(\min\big\{H, \sqrt{N/M}\big\}\big)\Big)$ I/O lower bound, by setting s to a fixed constant value that depends on the constant hidden in the $O(\cdot)$ notation. Finally, we show a randomized algorithm that can obtain both optimal worst-case running time and optimal *expected* I/O complexity (Section 7).

Our results imply that there are no optimal deterministic output-sensitive CO algorithms or algorithms that have both optimal I/O complexity and optimal running time for these two problems. The latter was shown only for one other problem before [16]. As far as we know, we show the first upper and lower bounds involving inverse Ackermann-like functions in the area of EM algorithm. Finally, our geometric embedding in Section 6 gives a technique that enables us to handle arbitrary geometric predicates involving a constant number of points in the context of the classical adversarial argument. We find this interesting and believe it could be of independent interest. For instance, we can get an alternative lower bound of $\Omega(N \log H)$ for the output-sensitive convex hull problem.

Summary of our approach. Our planar maxima algorithm (Section 3) is a very simple cache-oblivious recursive algorithm that aggressively prunes the points and uses the number of maxima points discovered in recursive subproblems so far to speed up the subsequent recursions. This ultimately leads to a non-trivial recurrence (Eq. (2) on page 8). In the full version [2] we show that it solves to an inverse Ackermann-like function. A similar approach works for the convex hull problem by replacing the pruning step. This is done by generalizing the "bridge finding" algorithm of Kirkpatrick and Seidel [29] to an algorithm that can find multiple bridges at the same time (Section 5). However, by randomly switching the order of the recursive calls, we can obtain an algorithm with both optimal running time and optimal expected I/O complexity (Section 7).

From a lower bound point of view, the problem is more complicated and requires more

involved techniques. For the maxima problem, we start in the classical comparison-based adversarial setting where the algorithm can only compare the coordinates of the points and the adversary decides on the results of the comparisons. However, we manage to apply this idea in the setting where the input resides in the external memory and only the coordinates of the points loaded into the internal memory of size M can be compared. During this argument, we show that the adversary can give extra information to the algorithm at specific intervals (*epochs*, defined on page 10) such that the behavior of the algorithm is simplified: among the available "subproblems" (captured by the concept of *top nodes* in the adversary's binary tree in Section 4) the algorithm can choose one subproblem to do one step of the recursion (which creates more subproblems). The main challenge is to show the lower bound regardless of the choices of the algorithm. To do that, we develop a potential function analysis and show that the best strategy for an algorithm is to recurse on the subproblems with the smallest potential (Section 4.3). This analysis leads to the following conclusion: when limited to an amortized budget of ζ I/Os per element and a budget of $N \cdot A_s(H)$ total comparisons, the algorithm can only "discover" $A_{s+2\zeta-1}(1)$ maxima points (Lemma 13). This in turn proves the lower bound (Theorem 14).

For the planar convex hull lower bound, we generalize this strategy to work for any geometric predicate: the algorithm can go beyond comparisons and choose a polynomial F and a set of σ points $p_1, \ldots, p_\sigma$ (for some fixed constant σ) and then ask for the sign of the evaluation of F on the coordinates of points $p_1, \ldots, p_\sigma$. To prove a lower bound in this case, we show that the adversarial argument for the maxima problem can be embedded geometrically in the plane. This requires some algebraic geometry ideas: we embed the points close to the $Y = X^\Delta$ curve, for some fixed constant Δ, and designate "squares" that represent the potential location of a group of points. The squares are conceptually arranged in a tree T of large fan out such that for any node $v \in T$ with a square $Q(v)$, the squares of children of v are doubly-exponentially smaller than $Q(v)$ and they are placed equally-spaced inside $Q(v)$ and centered on the curve $Y = X^\Delta$. By making the sizes of the square shrink at an appropriate doubly exponential rate, we show that the geometric predicates (captured by the polynomial F) do not provide too much information to the algorithm (Lemma 23). Thus, the lower bound for the maxima problem can be extended to the convex hull problem as well (Theorem 24).

2 Preliminaries

Let P be a set of N comparable elements. Scanning P while performing $O(1)$ work per element can be accomplished in $\Theta(N)$ time and $\Theta(n)$ I/Os. Comparison-based sorting of P can be performed cache-obliviously in $\Theta(n \log_m n)$ I/Os and $\Theta(N \log N)$ comparisons, assuming the cache is tall [26], i.e., $M \geq B^{1+\varepsilon}$ for some constant $\varepsilon > 0$. We define $\texttt{Scan}(N) = n$ and $\texttt{Sort}(N) = n \log_m n$.

Another fundamental problem is *distribution*, where given a value k, $1 \leq k \leq N$, the goal is to partition P into k subsets, $P_1, \ldots, P_k$, of roughly equal size, where each P_i has either $\lfloor \frac{N}{k} \rfloor$ or $\lceil \frac{N}{k} \rceil$ elements and all the elements in P_i are larger than or equal to all the elements in P_{i-1}. Each of the resulting subsets P_i should be stored in contiguous memory. In the classical comparison-based RAM model, distribution can be solved deterministically in $\Theta(N \log k)$ time, e.g., by recursive applications of a $\Theta(N)$-time median finding algorithm [14]. Distribution can be solved cache-obliviously in $\Theta(n \cdot \max\{1, \log_m k\})$ I/Os and $\Theta(N \log k)$ comparisons [23, 26], assuming a tall cache. We define $\texttt{Distr}(N, k) = n \cdot \max\{1, \log_m k\}$.

Observe that $\texttt{Distr}(N, k) = n \log_m k$ when $k \geq m$ and $\texttt{Distr}(N, k) = \texttt{Scan}(N)$ when

$k \leq m$. We exploit the following property of the `Distr` function, which follows from the concavity of logarithms: If $k = \sum_{i=1}^{t} k_i$ for some values $k_1, \ldots, k_t > m$ and $t \geq m$, then

$$\sum_{i=1}^{t} \texttt{Distr}\,(N, k_i) \leq \texttt{Distr}\left(tN, \frac{k}{t}\right) = \texttt{Distr}\,(tN, k) - \texttt{Distr}(tN, t). \tag{1}$$

2.1 Ackermann-like Functions and Their Inverses

To aid the exposition of the analysis, different papers present different definitions of the recursive functions that are referred to as the *Ackermann* functions [1,17,19,20,24,31,32,35–38]. While they differ from the original definition of Ackermann [1] and are not necessarily equivalent even asymptotically speaking, what they have in common is that they are extremely fast growing and their inverses are extremely slow growing. To aid our exposition, we will use the following functions.

Let $A_0, A_1, A_2, \ldots$ be an infinite sequence of functions, where $A_0(N) = N$, $A_1(N) = 2^N$ for any integer $N \geq 1$, and $A_{i+1}(N) = A_i^{(N+1)}(N)$, where the notation $f^{(k)}$ for a function f represents applying f to itself k times, e.g., $f^{(3)}(N) = f(f(f(N)))$.

The inverses of A are defined as two distinct functions $\lambda_i(x)$ and $\alpha_N(x)$, where $\lambda_i(x)$ is the smallest value N such that $A_i(N) \geq x$ and $\alpha_N(x)$ is the smallest value i such that $A_i(N) \geq x$. For example, $\lambda_1(x) = \Theta(\log x)$, $\lambda_2(x) = \Theta(\log^*(x))$ and in general, $\lambda_i(x)$ is roughly the number of times we need to apply function λ_{i-1} to x to get to a constant; therefore, λ_i can be thought of as the i-th function in the inverse hierarchy. In contrast, $\alpha_N(x)$ is a much slower growing function: $\alpha_1(x)$ grows slower than any of the functions $\lambda_i(x)$ for any fixed $i > 1$.

While there are many definitions of the Ackermann function, one of the more commonly accepted definitions is (the curried version of) the function by Péter [36] and Robinson [35], defined as $A_i(N) = A_i^{(N+1)}(1)$, with $A_0(N) = N + 1$ [37]. On the other hand, our definition is more similar to the definition of Cormen et al. [21], albeit with a different base case: they define $A_0(N) = N + 1$ and $A_i(N) = A_i^{(N+1)}(N)$ for all $i \geq 1$, which implies that $A_1(N) = 2N + 1$ and $A_2(N) = 2^{N+1}(N + 1) - 1$ [21, Chapter 19.4]. Our definition's base cases $A_0(N) = N$ and $A_1(N) = 2^N$ on the other hand do not have the additional offsets, which are not essential to the demonstration of how fast the Ackermann functions grow, and will make it easier to reason about our upper and lower bounds. Observe that asymptotically our function is slightly slower growing than that of Cormen et al. but faster growing that that of Péter and Robinson. Since our function is not quite the Ackermann function, we will refer to it as *Ackermann-like* function (Cormen et al. also refrain from calling their function Ackermann).

3 Upper Bound for Planar Maxima

Let P be the input set of N points in 2D, listed in an arbitrary order. The algorithm, which is presented in Algorithm 1, is initially invoked with an integral "seed" parameter $h \geq 1$ (not necessarily a constant). The choice of the seed provides a trade-off between the time and I/O complexity of the algorithm. At the subsequent recursive invocations, the parameter h will be equal to the initial seed, plus the number of maxima points discovered so far.

The seed h solves the challenge of not knowing H a priori. If we new H, we could distribute P into H buckets and it would be easy to show that the algorithm would achieve simultaneous optimality. However, distributing into too many buckets, e.g., $k = H^{\omega(1)}$ buckets, results in sub-optimal time of $\omega(N \log H)$. But if we choose too few buckets, we will

Algorithm 1 Algorithm for computing the maxima of a planar point set P with seed $h \geq 1$.

```
1:  procedure MAXIMA(P, h)
2:      if |P| ≤ 1                                          ▷ Base case
3:          Output P and return |P|
4:      (P_{2h}, ..., P_1) = DIST(P, 2h)        ▷ Distribute P into 2h buckets w.r.t. X-coor.
5:      PRUNE(P_1, ..., P_{2h})     ▷ For each P_i prune points dominated by points in ⋃_{j=1}^{i-1} P_j
6:      H' = 0
7:      for each P_j ∈ (P_1, ..., P_{2h})
8:          H_i = MAXIMA(P_j, h + H')
9:          H' = H' + H_i
10:     return H'
```

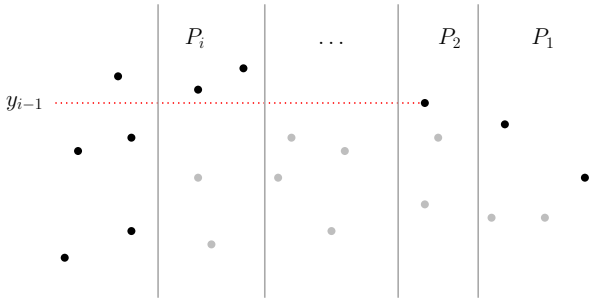

Figure 2 The pruning step: In P_i the points pruned (grey) are those with height at most the height y_{i-1} of the highest point to the of right P_i, i.e., the leftmost maximal point in $\bigcup_{j=1}^{i-1} P_j$.

have too many recursive levels, resulting in a sub-optimal I/O bound. The total number of discovered output points throughout the computation provides us with a lower bound on H, meaning, it is always safe to increase the number of buckets as we discover more points and the initial seed provides us with an initial "acceleration".

We distribute the points of P into $2h$ buckets of equal size, where P_1 contains the rightmost points and P_{2h} contains the leftmost ones. Next, we remove every point in P_i that is dominated by any of the points in $P_1, \ldots, P_{i-1}$ by a simple scan (see Figure 2): Process the buckets in order from right to left, and maintain the maximum y-coordinate, y_{i-1}, of the points in buckets $P_1, \ldots, P_{i-1}$; when processing the next bucket P_i, remove any point in P_i whose y-coordinate is smaller than or equal to y_{i-1}. Finally, recurse on each bucket, while increasing h by the number of newly discovered output points in each recursive call.

▶ **Theorem 1.** *For any integer $h \geq 1$, $\textsc{Maxima}(P, h)$ finds the H maxima points of the input set P of N points in $O(N \log H + N \log h)$ time and $O(n \log_m hH + n \cdot \alpha_h(\min\{H, m\}))$ I/Os.*

Proof. Observe that other than the recursive calls, the complexity of each level of the recursion of the algorithm is dominated by the cost of the distribution step, which is $O(\texttt{Distr}(N, 2h)) = O(\texttt{Distr}(N, h))$.

Let $T_H(N, h)$ denote the I/O complexity of $\textsc{Maxima}(P, h)$, where $|P| = N$ and H is the number of maxima points in P. Let H_i denote the number of maxima points of P_i (computed during the recursive call on P_i). Observe that the pruning step can always be performed with a simple scan and, thus, the distribution cost will always dominate the cost of all the

other operations at every recursive level. For any $h \geq 1$, we have the following recurrence:

$$T_H(N,h) \leq \begin{cases} c_0 \cdot \texttt{Distr}(N,h) & \text{if } h \geq H, \\ c_1 \cdot \texttt{Distr}(N,H) & \text{if } H > h \geq m, \\ \sum_{i=1}^{2h} T_{H_i}\left(\frac{N}{2h}, h + \sum_{j=1}^{i-1} H_j\right) + c \cdot \texttt{Distr}(N,h) & \text{if } h < \min(H,m), \end{cases} \tag{2}$$

where c_0, c_1, and c are some positive constants. We show that this recurrence solves to the claimed I/O bound in the full version [2]. We just note that the recurrence that bounds the time is exactly the same as the one described by Eq. (2), except the additive term $c \cdot \texttt{Distr}(N,h)$ is replaced by $cN \cdot \log h$ – the number of comparisons required to perform distribution into h buckets. But this is equivalent to setting $m = 2$ in the definition of $\texttt{Distr}(N,h)$. Observe that when $m = 2$, the third case in Eq. (2) cannot occur, meaning, the time can be upper bounded by simply adding up the bounds in the first two branches of Eq. (2), leading to the claimed time bound. ◀

Theorem 1 shows that we get almost optimal bounds, with the trade-off between optimal time and optimal I/Os being defined by the initial seed parameter. In Section 4 we prove that no algorithm can do asymptotically better.

▶ **Corollary 2.** *For any integer $s \geq 1$, the maxima problem on a planar set of N points can be solved with (i) optimal time of $O(N(\log H + \log s))$ and $O(n(\log_m sH + \alpha_s(\min\{H,m\})))$ I/O complexity cache-obliviously, or (ii) optimal I/O complexity of $O(n(\log_m H + s))$ and either $O(N(\log H + \lambda_s(N)))$ time cache-obliviously or $O(N(\log H + \lambda_s(m)))$ time cache-aware.*

Proof. The first bound follows from Theorem 1 by calling $\textsc{Maxima}(P,h)$ with the initial seed $h = s$. The second bound is obtained by calling it with the initial seed $h = \lambda_s(N)$ for a cache-oblivious algorithm, or $h = \lambda_s(m) \leq \lambda_s(N)$ if the cache parameters M and B are known. Observe that we can assume $N \geq m$ because otherwise $N < m \leq M$, i.e., the whole input fits in the internal memory. The claimed bounds follow because $\alpha_{\lambda_s(N)}(\min\{H,m\}) \leq \alpha_{\lambda_s(N)}(N) = s$. ◀

4 Lower Bound for Planar Maxima

Consider a set P of N points in the plane. In this section we show a lower bound for computing the maxima of P. We use the classical adversarial setting based on a rooted binary tree [15] for our lower bound. While this setting has been used in the past for proving lower bounds for a single metric of an algorithm (e.g., either for time [15] or for I/Os [10]), the biggest challenge for us is the need to balance the amount of I/Os performed by the algorithm versus the running time.

We prove a lower bound for I/O cost of any deterministic algorithm $\mathcal{A}$ that computes the maxima of any set of N points with H maxima points and uses at most $N \cdot A_s(H)$ comparisons, for an integer parameter $s \geq 1$. To simplify the presentation, we assume that the algorithm uses at most $NA_s(H)$ time, for a parameter $s \geq 3$. If the algorithm uses at most $cNA_s(H)$ time, for a fixed constant c, then we can bound $cNA_s(H) \leq NA_{s+c'}(H)$, for a fixed constant c' depending on c, and this only changes the constants in our lower bound.

We work in the classical comparison-based model. Comparisons between X- and Y-coordinates are the only way the algorithm can glean information about the relative position of the points but the algorithm has unlimited computational resources as well as unbounded capacity to recall all the previous comparisons. We consider two cost functions: the total

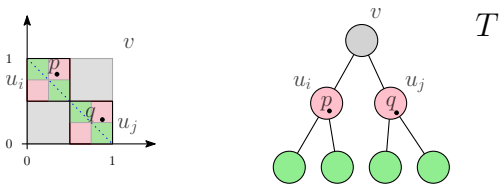

Figure 3 Conceptual regions (on the left) assigned to the nodes of T (on the right). Upon the completion of a comparison between two points $p, q \in v$, their assignment to nodes in the subtrees and, equivalently, to the corresponding regions maintains the consistency with future comparisons.

number of comparisons performed by the algorithm, which is a lower bound on the time, and the number of I/Os. We do not require the "blocked access" restriction of the I/O model, meaning, the algorithm can have the power of "random access" by being able to read or write any B locations on the external memory via one I/O. At the end, the algorithm terminates the computation and announces the set of maxima points and we obviously require the algorithm to be correct.

4.1 Resolving Comparisons

The adversary maintains a perfect binary tree T. Each node v of T is associated with a square region R_v on the plane: if v is the i-th node at depth $d \geq 0$ (for $i = 1, \ldots, 2^d$), it is associated with the square region $R_v = \left(\frac{i-1}{2^d}, \frac{i}{2^d}\right] \times \left(1 - \frac{i}{2^d}, 1 - \frac{i-1}{2^d}\right]$ (see Figure 3 for an example). That is, for the root r: $R_r = (0, 1] \times (0, 1]$; for an arbitrary node v, its left and right children are associated with the upper-left and lower-right quadrants of R_v, respectively.

Throughout the algorithm, the adversary will maintain an assignment of points to the nodes of T while maintaining the following invariant:

▶ **Invariant 3** (Tree invariant). *If a point p is assigned to a tree node v (denoted $p \in v$), then p can be placed anywhere within R_v consistent with the outcomes of all prior comparisons performed by the algorithm.*

▶ **Definition 4** (Ordered pairs). *A pair of points $p \in v$ and $q \in u$ is called an* unordered pair *if one of v or u is the ancestor of the other one (including $v = u$). Otherwise p and q is an* ordered pair*.*

▶ **Observation 5.** *There is only one way to consistently resolve a comparison between an ordered pair of points $p \in u_i$ and $q \in u_j$, because the regions R_{u_i} and R_{u_j} have non-overlapping X- and Y-ranges.*

Initially, all points are assigned to the root of T. Whenever two points are compared, the adversary produces the outcome of the comparison according to the following strategy and announces the outcome to the algorithm. A comparison between an ordered pair p and q is resolved according to the only consistent way, as per Observation 5. The next definition covers how unordered pairs are handled.

▶ **Definition 6** (Default strategy). *A comparison between a pair of unordered points $p \in v$ and $q \in u$ is resolved as follows. W.l.o.g., let v be an ancestor of u in T. If $v = u$, then p is moved to the left child of v and q is moved to the right child. If $v \neq u$, then p is moved to the child of v that is not the ancestor of u. In both cases, p and q become an ordered pair and the comparison is resolved according to Observation 5.*

▶ **Observation 7.** *The adversary's default strategy for resolving a comparison between points p and q maintains the tree invariant.*

4.2 Adversarial Strategy

In this subsection, we describe additional definitions and concepts used for our adversarial strategy. We will use the notation $T(v)$ to refer to the subtree of T rooted at v. We say a *t-descendant* of a node v is a node that lies at distance t below v (e.g., v is the 0-descendant of itself and the children of v are its 1-descendants). Each point p, is labeled as either a *deep point* or an *ordinary point*. We will explain how this labeling is performed during our adversarial strategy. Conceptually, deep points represent points that get involved in way too many comparisons and, thus, the vast majority of points will be ordinary points. We call a non-empty node $v \in T$ a *top node* if all ancestors of v are empty (i.e., contain no points). The tree invariant implies that the number of top nodes is a lower bound on the number of (current) maxima points. The *initial size of* v, denoted N_v, is the number of ordinary points in v the first time v becomes a top node.

Charges. We will maintain a non-negative integer *charge* with each ordinary point. Later, we show that the sum of charges across all ordinary points will be a lower bound on the total number of points accessed in the external memory during the execution of algorithm $\mathcal{A}$. This is done via the classical "amortization" technique where for every point accessed in the external memory, we transfer $\Omega(1)$ charge to some ordinary point. The adversary will maintain the following invariant:

▶ **Invariant 8** (Equality of charges). *For any top node v, all ordinary points in $T(v)$ have equal charges.*

Charges will be the way the adversary controls the termination of the algorithm. In particular, whenever ordinary points at a top node v accumulate ζ charges, for a parameter ζ to be chosen later, the adversary will use the following strategy.

▶ **Definition 9** (Node termination). *The adversary* terminates *a top node v by picking an arbitrary point $p \in v$ and fixing its coordinates to those of the northeast corner of R_v. For the remaining points at the nodes in $T(v)$ the adversary fixes their coordinates arbitrarily within the regions of their respective nodes. The coordinates of all points in $T(v)$ are then announced to the algorithm.*

First, observe that the adversary can perform node termination because by the tree invariant the points can be placed anywhere within the regions of their respective nodes and the relative order within each region is unknown to the algorithm. Second, this effectively reduces the size of the maxima among the points in $T(v)$ to 1, essentially pruning all the other points in $T(v)$, because p will dominate all of them. Finally, terminated nodes are still top nodes, and while the algorithm can still issue comparisons that involve the points in $T(v)$, these points are now ordered (since their coordinates are fixed and announced to the algorithm) and, thus, the results of these comparisons are already known.

Epochs. The adversary operates in *epochs* where during each epoch, the adversary resolves comparisons issued by $\mathcal{A}$ via the default strategy. Under some conditions, the adversary decides that the current epoch has to end, gives some extra information to the algorithm and then *transitions* to the next epoch. Crucially, the number of top nodes only changes during the transition. We use h_i to denote the number of top nodes at the start of epoch i. Initially, we start at epoch 1 when the algorithm $\mathcal{A}$ has issued no comparisons, the root r of T is the only top node, i.e., $h_1 = 1$, all points are ordinary, with charge zero, and they are placed in r. We will now present the details behind the transition process.

Consider an arbitrary epoch i. During the epoch the adversary resolves the comparisons issued by the algorithm (via the default strategy) until for some top node v, the number of ordinary points in v reduces to $N_v/2$ (recall that v started with N_v ordinary points). If the ordinary points of v had $\zeta - 1$ charge, we increase their charge to ζ and terminate v using Definition 9 and the epoch i continues. Otherwise, epoch i ends and the adversary transitions to epoch $i+1$ by performing the following. Define the function $d(x) = 2^{A_s(x)}$ and let $d_i = d(h_i)$. First, the adversary labels every point that is in a t-descendant of v for any $t > d_i$ *deep*. Next, consider every node u that is a t-descendant of v for $1 \le t \le d_i$. All points in u are moved to an arbitrary d_i-descendant of v that is also a descendant of u. The remaining $\frac{N_v}{4}$ points of v are then distributed equally among all d_i-descendants of v. These moves make a number of pairs among the moved points ordered. To do this, the adversary provides the information about the relative order of the newly ordered pairs (according to Observation 5) to the algorithm for free. The number of d_i-descendants of v is 2^{d_i} and, thus, each d_i-descendant will receive $\frac{N_v}{4 \cdot 2^{d_i}}$ points. At this point neither v, nor any of its d-descendants for $d < d_i$ contain any points. Therefore, the d_i-descendants of v become new top nodes and we call them *activated*. The charges of all ordinary points in the newly activated top nodes are incremented by one, which preserves Invariant 8.

If at any point, all nodes are terminated, then every top node contains exactly one point that dominates the others in its subtree. Since this information is available to the algorithm, it can announce the result and terminate. It is important to note that termination means that each ordinary point in $T(v)$ has received exactly ζ charge.

▶ **Observation 10.** *In each epoch $i > 1$: $h_i = h_{i-1} + 2^{d_{i-1}} - 1 = h_{i-1} + 2^{d(h_{i-1})} - 1$.*

▶ **Lemma 11.** *The initial size of every node v activated at the end of epoch $i-1$ is $N_v \ge \frac{N}{h_i^2}$.*

Proof. By induction on i (presented in [2]). ◀

▶ **Lemma 12.** *The resolution of v at the end of epoch i creates at most $\frac{N_v}{2^{h_i}}$ deep points, where N_v is the initial size of v at the time of its activation.*

Proof. Recall that $\mathcal{A}$ is limited to a budget of $N \cdot A_s(H)$ comparisons. Moreover, during each epoch i, if algorithm $\mathcal{A}$ performs more than $N \cdot A_s(h_i)$ comparisons, the adversary can use node termination on all top nodes and force $H = h_i$. Consequently, $\mathcal{A}$ is limited to a budget of $N \cdot A_s(h_i)$ comparisons in every epoch i. By Lemma 11 and monotonicity of h_is, we have $N \le N_v h_i^2$. Since each comparison moves at most 2 points one level lower, after $N \cdot A_s(h_i)$ comparisons in epoch i, the total number of points that can be deeper than d_i is at most $\frac{2N \cdot A_s(h_i)}{d(h_i)} \le \frac{2N_v h_i^2 A_s(h_i)}{d(h_i)} = \frac{2N_v h_i^2 A_s(h_i)}{2^{A_s(h_i)}} < \frac{N_v}{2^{h_i}}$, where the last inequality follows from the fact that for all integers $s \ge 3$ and $x \ge 1$: $2^{A_s(x)} > 2x^2 A_s(x) 2^x$. ◀

Our main technical lemma here is the following.

▶ **Lemma 13.** *The number of top nodes is upper bounded by $A_{s+2\zeta-1}(1)$.*

The proof is postponed to the next subsection but below we quickly show that this is sufficient to give us a lower bound.

▶ **Theorem 14.** *Let P be a planar point set of size N with at most H maxima points, and $s \ge 1$ be an integer parameter. Then any algorithm for computing the maxima of P that uses at most $N \cdot A_s(H)$ comparisons requires $\Omega\left(\frac{N}{B}(\alpha_1(\min\left\{H, \sqrt{\frac{N}{M}}\right\}) - s)\right)$ I/Os.*

Proof. We first claim that the adversary creates an instance with at most H maxima points. We choose $\zeta = \frac{\alpha_1(\ell)-s}{2}$ and let $\ell = \min\left\{H, \sqrt{\frac{N}{4M}}\right\}$. Then by the definition of the α function we have $A_{s+2\zeta-1}(1) < \ell$, i.e., by Lemma 13, the number of top nodes will always be smaller than $\ell \leq H$ and, thus, the claim holds.

Let V' be the set of nodes of T that have been resolved during the execution of $\mathcal{A}$ and let $Z = \sum_{v \in V'} Z_v$ be the total charge across all ordinary points by the end of $\mathcal{A}$, where Z_v is the increase in charges due to resolution of each $v \in V'$. Observe that by summing the bound given in Lemma 12, we get that at most a constant fraction of the points can be labeled deep, meaning, most of the points will be ordinary. Then, since the adversarial strategy ensures that each ordinary point receives a charge of ζ, we get $Z = \Omega(N\zeta)$. We now show that at least $Z/4$ points must have been accessed in the external memory to resolve comparisons performed by $\mathcal{A}$, implying $\frac{Z}{4B} = \Omega\left(\frac{N\zeta}{B}\right) = \Omega\left(\frac{N}{B}(\alpha_1(\ell) - s)\right)$ I/O lower bound.

Observe that in the final epoch j, the number of top nodes $h_j \leq \ell \leq \sqrt{\frac{N}{4M}}$. Since j is the final epoch, every top node v must have been activated prior to the end of some epoch $j' - 1 \leq j$ and, by Lemma 11 and monotonicity of h_is, starts with $N_v \geq \frac{N}{h_{j'}^2} \geq \frac{N}{h_j^2} \geq 4M$ points. By the time v is resolved, at least $N_v/2 \geq 2M$ points of v have been moved to the lower nodes due to comparisons. Thus, all these points could not be kept in the internal memory from activation till resolution of v, i.e., at least $N_v/2 - M \geq N_v/4$ of them must have been accessed in the external memory during this period. But we also know that at the end of the resolution of v, each ordinary point in $T(v)$ receives an additional charge, i.e., the overall charge in T is increased by $Z_v \leq N_v$. Summing over all nodes that have been resolved during the execution of $\mathcal{A}$, we get that at least $\sum_{v \in V'} \frac{N_v}{4} \geq \sum_{v \in V'} \frac{Z_v}{4} = \frac{Z}{4}$ vertices must have been accessed in the external memory during the execution of $\mathcal{A}$. ◀

4.3 Proof of Lemma 13

Thus, it remains to prove Lemma 13. The number of top nodes explodes very fast and moreover, this number also depends on the order in which the top nodes are resolved. To bound it, we use a potential function argument, for which we need to introduce two crucial notions. A *potential sequence (PS)* is a finite sequence of pairs of integers, e.g., $(t_1, \kappa_1), (t_2, \kappa_2), \ldots$, where $t_i \geq 0$ and $0 < \kappa_1 \leq \kappa_2 \leq \ldots$. We call κ_i the *potential* and a PS is allowed to be empty, denoted by $()$. A *status vector* is a pair $(h; S)$, where $h > 0$ is an integer and S is a PS. The status vectors will help us bound the number of top nodes we will ever get in the adversary argument. In particular, a status vector $W = (h; (t_1, \kappa_1), (t_2, \kappa_2), \ldots)$ represents the end of an epoch where we have at most h top nodes and we have at most t_i top nodes whose points have charge $\zeta - \kappa_i$. We make the following observations to simplify our mathematical manipulations of the status vectors.

▶ **Observation 15.** *Let $S_1 = (t_1, \kappa_1), \ldots (t_n, \kappa_n)$ and $S_2 = (t'_1, \kappa'_1), \ldots, (t'_m, \kappa'_m)$ be two potential sequences, such that $\kappa_n \leq \kappa'_1$. Then for any integer κ, $\kappa_n \leq \kappa \leq \kappa'_1$:*

- $(h; S_1, (0, \kappa), S_2) = (h; S_1, S_2)$, *and*
- $(h; S_1, (t, \kappa), S_2) = (h; (t', \kappa), (t - t', \kappa))$ *for any integer* $0 < t' < t$).

The first one states that since $(0, \kappa)$ represents 0 top nodes of charge $\zeta - \kappa$, they can be safely omitted from the status vector. The second one states that we can view a collection of t top nodes as two collections of t' and $t - t'$ nodes (with the same charges).

We claim that there is actually an explicit way to maximize the number of top nodes via function Φ defined as follows. Let $A(x) = A_{s+1}(x)$ and $S = (t_1, \kappa_1), (t_2, \kappa_2), \ldots$ be an

arbitrary PS. Then for all integers $t \geq 0$ and $0 < \kappa \leq \kappa_1$:

$$\Phi(h; ()) = h \qquad \text{and} \quad (3)$$

$$\Phi(h; (t, \kappa), S) = \begin{cases} \Phi(h; S) & \text{if } t = 0, \\ \Phi(A(h); (t-1, \kappa), S) & \text{if } t > 0 \text{ and } \kappa = 1, \\ \Phi(A(h); (A(h), \kappa - 1), (t-1, \kappa), S) & \text{if } t > 1 \text{ and } \kappa > 1. \end{cases} \quad (4)$$

The Φ function captures (the upper bound on) the number of top nodes of various potential in the algorithm if the top node with the least potential are resolved first. In Lemma 18, we will show that this resolution order produces the maximum number of top nodes. So consider an epoch i with at most h_i top nodes. By Observation 10, we can bound $h_{i+1} < A_{s+1}(h_i) = A(h_i)$. Eq. (3) captures the base case scenario where there are at most h top nodes and all the top nodes are terminated (represented by the empty status vector ()). Eq. (4) captures three possible values of the first term (t, κ) in the potential sequence of Φ:

- If $t = 0$, there are no nodes with potential κ, so this term can be ignored.
- If there are t nodes with potential $\kappa = 1$, i.e., the node contains items with charges $\zeta - 1$, one of those nodes is terminated, resulting in the activation of nodes with ζ charge which, in turn, can only reach the termination condition. Of course this still increases the number of top nodes to at most $A(h)$, of which $t - 1$ top nodes have potential $\kappa = 1$.
- In general, if all the top nodes have points with potential $\kappa > 1$, application of the resolution to one of the t top nodes with the smallest potential creates at most $A(h)$ nodes of potential $\kappa - 1$ and leaves at most $t - 1$ nodes of potential κ.

To prove that the Φ function is an upper bound on the number of top nodes, we need the following structural properties, which follow from the definitions of Φ and of the PS.

▶ **Lemma 16.** *Let $S_1 = (t_1, \kappa_1), \ldots, (t_n, \kappa_n)$ and $S_2 = (t'_1, \kappa'_1), \ldots, (t'_m, \kappa'_m)$ be two potential sequences, such that $\kappa_n \leq \kappa'_1$. Then, for any $h > 0$ and $\kappa_n \leq \kappa \leq \kappa'_1$: $\Phi(h; S_1, S_2) = \Phi(\Phi(h; S_1); S_2)$ and $\Phi(h; S_1, (0, \kappa), S_2) = \Phi(h; S_1, S_2)$.*

▶ **Lemma 17.** *For any integer $\kappa \geq 1$: $\Phi(h; (1, \kappa)) \leq A_{s+2\kappa-1}(h)$.*

Proof. *(By induction)* When $\kappa = 1$, for any $t > 0$: $\Phi(h; (t, 1)) = \Phi(A(h), (t-1, 1)) = A^{(t)}_{s+1}(h)$ and so $\Phi(h; (1, 1)) = A_{s+1}(h)$. Now assume that $\kappa \geq 2$ and that the claim is true for all positive integers $\kappa' < \kappa$.

$$\begin{aligned}
\Phi(h; (1, \kappa)) &= \Phi(A(h); (A(h), \kappa - 1), (0, \kappa)) = \Phi(A(h); (A(h), \kappa - 1)) && \text{by Eq. (4)} \\
&= \Phi(A(h); \underbrace{(1, \kappa - 1), \ldots, (1, \kappa - 1)}_{A(h) \text{ times}}) && \text{by Observation 15} \\
&= \Phi(\Phi(A(h); (1, \kappa - 1)); \underbrace{(1, \kappa - 1), \ldots, (1, \kappa - 1)}_{A(h) - 1 \text{ times}}) && \text{by Lemma 16} \\
&\leq \Phi(A_{s+2\kappa-3}(A(h)); \underbrace{(1, \kappa - 1), \ldots, (1, \kappa - 1)}_{A(h) - 1 \text{ times}}) && \text{by Inductive Hypothesis} \\
&\leq A^{(A(h))}_{s+2\kappa-3}(A(h)) && (*) \\
&< A_{s+2\kappa-2}(A_{s+1}(h)) < A_{s+2\kappa-2}(A_{s+2\kappa-2}(h)) && A(h) = A_{s+1}(h), \kappa \geq 2 \\
&\leq A_{s+2\kappa-1}(h), && A_i(A_i(x)) < A_{i+1}(x)
\end{aligned}$$

where (*) follows from a simple inductive argument. ◀

▶ **Lemma 18.** *Let $V = (h; (t_1, \kappa_1), (t_2, \kappa_2), \ldots, (t_m, \kappa_m)$ be a status vector that represents a state of the top nodes in the algorithm. Regardless of the order in which top nodes are resolved, the resulting number of top nodes is at most $\Phi(V)$.*

Proof. Assume inductively that the claim is true for any V' that is lexicographically greater than V. In the base case, the status vector is $(h'; ())$, i.e., the statement is vacuously true. Assume that the next node to get resolved has potential κ_i. Let V_i be the resulting status vector, i.e.,

$$V_i = (A(h); (t_1, \kappa_1), \ldots, (t_{i-1}, \kappa_{i-1}), (A(h), \kappa_i - 1), (t_i - 1, \kappa_i), (t_{i+1}, \kappa_{i+1}) \ldots, (t_m, \kappa_m)).$$

Observe that no matter which top node the algorithm chooses to resolve first, the number of top nodes at the next epoch will be upper bounded by $A(h) > h$, i.e, $V_i > V$ lexicographically. Thus, for every $1 \leq i \leq m$, inductively, $\Phi(V_i)$ is an upper bound for the number of top nodes. Observe that $\Phi(V) = \Phi(V_1)$, so to prove the claim it is sufficient to prove that $\Phi(V_i) \geq \Phi(V_{i+1})$ for every $1 \leq i < m$.

Consider an arbitrary $1 \leq i < m$. Observe that if $\kappa_i = \kappa_{i+1}$ then $V_i = V_{i+1}$ and we have nothing to prove. Thus, assume $\kappa_{i+1} > \kappa_i$. Define the following three potential sequences:

$$X = (t_1, \kappa_1), \ldots, (t_{i-1}, \kappa_{i-1}), \qquad Y = (A(h), \kappa_i - 1), (t_i - 1, \kappa_i),$$
$$Z = (t_{i+1} - 1, \kappa_{i+1}), (t_{i+2}, \kappa_{i+2}), \ldots, (t_m, \kappa_m).$$

Then, using Observation 15 we can rewrite V_i and V_{i+1} as follows:

$$\begin{aligned} V_i &= (A(h); X, Y, (1, \kappa_{i+1}), Z) \\ V_{i+1} &= (A(h); X, (t_i, \kappa_i), (A(h), \kappa_{i+1} - 1), Z) \\ &= (A(h); X, (t_i - 1, \kappa_i), (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z). \end{aligned}$$

Observe that Φ is a strictly increasing function of all of its parameters. Then

$$\begin{aligned} \Phi(V_{i+1}) &< \Phi(A(h); X, (A(h), \kappa_i - 1), (t_i - 1, \kappa_i), (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z) \\ &= \Phi(A(h); X, Y, (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z) \\ &= \Phi(\Phi(A(h); X, Y); (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z) \\ &= \Phi(\chi; (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z), \end{aligned}$$

where we set $\chi = \Phi(A(h); X, Y)$. On the other hand, we have

$$\begin{aligned} \Phi(V_i) &= \Phi(A(h); X, Y, (1, \kappa_{i+1}), Z) = \Phi(\Phi(A(h); X, Y); (1, \kappa_{i+1}), Z) = \Phi(\chi; (1, \kappa_{i+1}), Z) \\ &= \Phi(A(\chi); (A(\chi), \kappa_{i+1} - 1), Z). \end{aligned}$$

Observe that $\chi \geq A(h) > h$ which implies $A(\chi) > A(h)$ and thus $A(\chi) \geq A(h) + 1$. Thus,

$$\begin{aligned} \Phi(V_i) &= \Phi(A(\chi); (A(\chi), \kappa_{i+1} - 1), Z) \geq \Phi(\chi; (A(\chi), \kappa_{i+1} - 1), Z) \\ &\geq \Phi(\chi; (A(h) + 1, \kappa_{i+1} - 1), Z) = \Phi(\chi; (1, \kappa_{i+1} - 1), (A(h), \kappa_{i+1} - 1), Z) \\ &\geq \Phi(\chi; (1, \kappa_i), (A(h), \kappa_{i+1} - 1), Z) > \Phi(V_{i+1}), \qquad (*) \end{aligned}$$

where (*) follows from $\kappa_{i+1} - 1 \geq \kappa_i$, and Φ being an increasing function. ◀

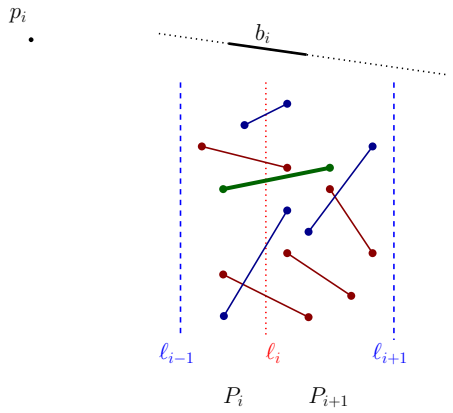

Figure 4 The collection C_i initially contains all the points between ℓ_{i-1} and ℓ_{i+1}. The bridge b_i intersects ℓ_i but could potentially extend beyond the collection in either directions which means the end points of b_i are not guaranteed to be in the collection C_i. The green (thicker) pair is the pair with the median slope, m_i, among the pairs in the collection. Red pairs have smaller slope than m_i, whereas blue pairs have larger slopes.

5 Upper Bound for Planar Convex Hull

In this section, we show that an algorithm similar to MAXIMA can be used to compute the output-sensitive convex hull of a planar point set. Recall that the algorithm starts with a (seed) parameter h. Observe that if $h \geq \sqrt{N}$, then we can simply switch to a worst-case $O(N \log N) = O(N \log h)$-time convex hull algorithm, e.g., Graham's scan, which can compute the convex hull by sorting and scanning. Thus, in the rest of this section, we assume that $h < \sqrt{N}$. The algorithm distributes the points into $2h$ buckets/subsets, $P_1, \ldots, P_{2h}$, of size roughly $\frac{N}{2h}$ and each pair of neighboring buckets P_i and P_{i+1} separated by a vertical line. In the maxima algorithm, the next step (line 5) is to eliminate all points in each bucket that are dominated by the points in the buckets to its right. In the context of the upper hull, this is slightly complicated and it involves pruning the points below the "bridges", which straddle the boundary between pairs of neighboring buckets P_i and P_{i+1}. We outline this process below.

To perform the pruning step, we adapt the strategy of Kirkpatrick and Seidel's algorithm for the convex hull [29]. Their main technique is to partition the input points into two equal sets with a vertical line ℓ and then find the "bridge", i.e., the edge of the upper hull intersected by ℓ in linear time. Here, we would like to generalize their technique to a setting that involves $2h$ subsets of points.

In our case, we have multiple buckets. Let ℓ_i be the vertical line separating (lies between) P_i and P_{i+1} and a bridge b_i is the edge of the upper hull that intersects ℓ_i (it separates P_i and P_{i+1}). Let Q be the set of all end points of all the bridges, $b_1, \ldots, b_{2h-1}$. Observe that pruning P_i is trivial if all the *bridges* intersecting ℓ_{i-1} and ℓ_i are already computed: simply remove the points that lie below the bridges and recurse on the remaining points in each subset P_i. Thus, the pruning step can be reduced to a *multi-bridge finding step* where the goal is to find all the bridges.

To find the bridges, we maintain $2h - 1$ *collections*: The i-th collection, C_i, is initialized as $P_i \cup P_{i+1}$ and observe that each point belongs to at most two collections. Then, we show that we can run the original Kirkpatrick and Seidel's algorithm on each collection. However, there are some details to take care of. During the algorithm we will be pruning a constant

fraction of each C_i until their total size becomes $O(N/\log N)$. We will maintain the invariant that at all times

$$Q \subset \bigcup_{i=1}^{2h-1} C_i. \tag{5}$$

The pruning strategy is as follows. Let $C = \cup_{i=1}^{2h-1} C_i$. First, we pair the points of C_i arbitrarily and compute the median slope, m_i, of the pairs using the linear time and I/O median finding algorithm [14]. Next, we show that for each m_i, we can find an extreme point p_i of C (not necessarily in C_i) in the direction orthogonal to m_i in $\texttt{Distr}(N, h)$ time.

▶ **Lemma 19.** *Given a set P of N points and a set of k slopes, for $k < \sqrt{N}$, we can compute the extreme points along each slope in $O(\texttt{Distr}(N, k))$ time.*

Proof. First, we sort (with an optimal CO algorithm) the slopes in decreasing order, $s_1, \ldots, s_k$. Since k is small this can be done in $O(\texttt{Scan}(N))$ time. Next, we group the points arbitrarily into $\frac{N}{k^2}$ batches of size k^2 and compute the upper hull of each batch using Graham Scan. This can be done in $O(\texttt{Distr}(N, k))$ time by simple sorting and scanning each batch.

Let u_i be the upper hull of the i-th batch. Then, for each u_i, we do the following: we scan all $s_1, \ldots, s_k$ at the same time as edges of u_i. Observe that the upper hull edges of u_i are also in the order of decreasing slopes. This means that the vertices of u_i that are extreme with respect to $s_1, \ldots, s_k$ can be found by just a forward scan of u_i. While scanning the slopes we update each slope with the most extreme point seen so far.

To analyze the I/O complexity, we need to consider two cases: First, consider the case when $k^2 \geq B$. In this case, scanning each u_i forward requires $O\big(\texttt{Scan}(k^2)\big)$ I/Os. As there are k slopes, the total cost per batch is $O\big(\texttt{Scan}(k^2) + 1 + \texttt{Scan}(k)\big) = O\big(\texttt{Scan}(k^2)\big)$. Over all $\frac{N}{k^2}$ batches this sums up to $O(\texttt{Scan}(N))$. Now assume $k^2 < B$ and let $t = \left\lfloor \frac{B}{k^2} \right\rfloor$. In this case, t consecutive batches are loaded with one I/O, and also all k slopes are loaded with one I/O. Thus, we can find all extreme points of t consecutive batches at the same time, resulting in overall $O(\texttt{Scan}(N))$ I/O complexity. ◀

Thus, in the rest of the proof we assume that p_i, an extreme point of C in the direction orthogonal to m_i has already been computed.

▶ **Lemma 20.** *Given p_i in each collection $C_i \subseteq C$, we can prune a fraction of the points in C_i in $O(\texttt{Scan}(|C|))$ I/Os. This process maintains Eq.* (5)*, meaning, the invariant still holds after pruning.*

Proof. W.l.o.g., assume p_i is to the left of ℓ_i (see Figure 4). Observe that slopes of the bridges $b_1, \ldots, b_{2h-1}$ is a decreasing sequence and since C contains Q, it follows that the slope of b_i is at most m_i. We now apply the argument of Kirkpatrick and Seidel [29]. Consider the pairs in C_i that have slopes larger than m_i (blue pairs in Figure 4). Each such pair is guaranteed to have a slope larger than or equal to the slope of b_i. We prune the left end points of each such pair as the left point cannot be an end point of b_i.

However, we also need to verify that we have not pruned any point in Q. To do that, consider a point $q \in Q$. By definition, there exists a collection C_i such that q is a vertex of the bridge b_i over ℓ_i. If p_i is to the left (resp. right) of ℓ_i, then it follows that b_i does not have larger (resp. smaller) slope than m_i. Then, we consider the matched pairs in C_i that have larger (resp. smaller) slope than m_i. For each matched pair of points (p, p'), where the point p has smaller X-coordinate than p', we pruned the point p (resp. p'). However, observe that q cannot be the pruned point. This shows that our invariant is maintained. ◀

After a pruning step, we go back to the beginning and pair the points in each C_i arbitrarily again and iterate. At each iteration, we will be pruning a fraction of the points in each collection. Thus, the size of the union, C, will be decreasing geometrically. Once the size decreases by a $\log N$ factor, we can simply compute the convex hull of the resulting subset and find the bridges explicitly in $O(\texttt{Sort}(N/\log N)) = O(\texttt{Scan}(N))$ I/Os.

6 Lower Bound for Planar Convex Hull

In this section, we show how to adapt the adversarial argument from the maxima lower bound to the planar convex hull problem.

Let Δ, σ be some constants. We will use the notations $\tilde{\Omega}(\cdot)$ and $\tilde{O}(\cdot)$ to hide factors that depend on Δ and σ; to avoid circular dependencies in the constants, we will use the notations to hide expressions that depend only on Δ and σ and fixed constants, i.e., constants that do not depend on other parameters in the construction. We call a polynomial F *suitable* if it has degree at most $\Delta - 1$, and it is defined over 2σ indeterminates with real-valued coefficients.

Predicates. At any moment, the algorithm can choose a suitable polynomial F and a sequence W of σ input points, i.e., $W = p_1, \ldots, p_\sigma$, and query the sign of the evaluation of F on the X- and Y-coordinates of points $p_1, \ldots, p_\sigma$, i.e., query $\textsc{Sign}(F(W))$. We call $\textsc{Sign}(F(W))$ a *geometric predicate*.

As with the previous lower bound, the adversary will decide the result of the predicate and then will inform the algorithm of the result. Based on the result of the predicate, the algorithm can choose another predicate to be evaluated on a potentially different set of points. This process will continue until the algorithm can correctly declare the points on the convex hull. Observe that a comparison is a simple predicate of degree 1 where $\Delta = 1$ and $\sigma = 2$ and it involves only either X- or Y-coordinates of the two points. A standard orientation test is the sign of a 3×3 matrix that is obtained by placing the X- and Y-coordinates of three points as the first two column and then adding a column of ones. Thus, in this case we have $\Delta = 2$ and $\sigma = 3$. In this section, we show a lower bound that applies to any algorithm that uses such geometric predicates.

A rough sketch of the proof. We show that the same set up that was used in Section 4 can be adapted to the convex hull problem as well. In particular, we define a tree T and the points will be placed on the nodes of T. However, predicates make this process much more complicated because they reveal much more geometric information about the position of the points than the relative order of the coordinates. Nonetheless, for the argument in Section 4 to go through, we only need to replace two main tools used by the adversary: one is the *default strategy* used to resolve comparisons (predicates here) and two, *node termination* which enables the adversary to essentially prune all but one point in a subtree of T. To do these, we borrow an idea due to Afshani and Cheng [3, 4] of embedding points very close to the curve $Y = X^\Delta$. This enables us to approximate any monomial X^iY^j as $X^{i+j\Delta}$ which in turn will convert any bivariate polynomial of degree less than Δ into a univariate polynomial of degree less than Δ^2 where distinct monomials in the bivariate polynomial are mapped to distinct monomials in the univariate polynomials. We follow this up by mapping every node v in T to small enough geometric regions (squares), $Q(v)$, close to the curve $Y = X^\Delta$ such that from the point of view of the algorithm any point $p \in v$ can be anywhere inside $Q(v)$, i.e., without violating any of the predicates chosen by the algorithm. Finally, we show that

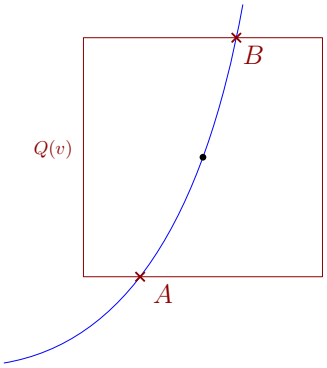

Figure 5 Square $Q(v)$ for node v.

the resolution of predicates can be done by moving points only a constant number of levels down the tree T. We now present the details.

The tree T. Let $f = \Delta^2 \cdot \Delta^{2\sigma} = \Delta^{2\sigma+2} = \tilde{O}(1)$. The adversary maintains a tree T with fan out f, similar to the lower bound in Section 4 but the adversary will also associate a square with every node of T. Let Q_0 be a unit square with its center on the point $(1, 1)$ which lies on the curve $Y = X^\Delta$. Q_0 will be associated with the root of T. We will shortly describe how the adversary assigns progressively smaller squares that are inside Q_0 to every node in T. We will use the notation $p \in v$ to denote that a point p is placed in a node v of T, the square associated with v is denoted by $Q(v)$ and the depth of v (in T) is denoted by $d(v)$.

Assigning squares. Consider a non-root node v at depth $d(v)$ in the tree T. The square $Q(v)$ will have side length $2^{-c^{d(v)}}$ where c is a large enough value that will depend on Δ and σ (as mentioned, we will not "hide" c in our asymptotic notations). Consequently, the squares associated with the children of v will have side length $2^{-c^{d(v)+1}}$ and they will be placed inside $Q(v)$, spaced equally across the X-axis and centered on the curve $Y = X^\Delta$ (see Figure 5 for an example). We claim that by picking c large enough, we can guarantee that the squares assigned to the children of v will be fully inside $Q(v)$. Let A and B be the intersection points of the boundary of $Q(v)$ with the curve $Y = X^\Delta$. Observe that the tangent to every point on the curve $Y = X^\Delta$, which is inside Q_0, has a slope between $2^{-\Delta}$ and 1.5^Δ which means that the difference between X- or the Y-coordinates of the points A and B is at least $\Omega\left(2^{-\Delta}2^{-c^{d(v)}}\right)$. By setting c large enough we can easily ensure that $f \cdot 2^{-c^{d(v)}} \geq 2^\Delta \cdot f \cdot 2^{-c^{d(v)+1}}$. This means that there is enough space inside $Q(v)$ to place the squares of the children of v.

As before, any non-empty node v whose ancestors do not contain any points is called a *top node*. The equivalent of Invariant 3 in this case is the following invariant.

▶ **Invariant 21** (The tree invariant)**.** *For every point $p \in v$, every placement of the point p inside the square $Q(v)$ is consistent with the results of all previous predicates returned by the adversary, meaning, if the algorithm in the past has queried the sign of a suitable polynomial F on σ points $p_1 \in v_1, , \ldots, p_\sigma \in v_\sigma$ and the adversary has returned a sign value z, then for every point $W \in Q(v_1) \times \cdots \times Q(v_\sigma)$, we have* $\textsc{Sign}(F(W)) = z$.

Node termination is captured with the following definition.

▶ **Definition 22** (Node termination). *The adversary can* terminate *a top node v by picking four points $p_1, p_2, p_3, p_4 \in u$, and declaring that they lie on the corners of $Q(v)$. The adversary also picks some arbitrary coordinates for the points in $T(v)$ within their corresponding squares and declares these to the algorithm.*

Note that similar to the previous proof, after the termination of a node v, the positions of all the points in the subtree of v will be known to the algorithm and thus we can assume that no predicates involves the coordinates of such points. We call these the *pruned points* and any other remaining point an *unpruned points*.

Thus, the heart of the construction is that the adversary can replace the *default strategy* with a similar strategy, captured in the following lemma.

▶ **Lemma 23.** *Consider a predicate F chosen by the algorithm on a set of σ points $p_1, \ldots, p_\sigma$ such that $p_i \in v_i$. The adversary can maintain Invariant 21 by moving each point p_i to a node u_i such that u_i is a t_i-descendant of v_1 for some $t_i \leq \sigma + 1$.*

In fact, we show something stronger. The adversary can create an infinite sequence of positive real values $\gamma_1, \gamma_2, \ldots$ such that the following holds: For every point $W \in Q(u_1) \times \cdots \times Q(u_\sigma)$, $|F(W)| \geq \gamma_d$ where $d = \max\{d(u_1), \ldots, d(u_\sigma)\}$.

Proof. The proof is an adaptation of an idea due to Afshani and Cheng [3, 4] of embedding the points very close to the curve $Y = X^\Delta$. The details are presented in [2]. ◀

Observe that as the function F is continuous, the latter claim in the above lemma not only implies that F is non-zero over $Q(u_1) \times \cdots \times Q(u_\sigma)$ but also that its magnitude is lower bounded by some fixed parameter that only depends on the depth of the deepest node among $u_1, \cdots, u_\sigma$.

We now show that these modifications are sufficient to obtain a lower bound for the convex hull problem.

▶ **Theorem 24.** *Consider a planar point set P of size N with H convex hull points and let $\Delta > 0$ and $\sigma > 0$ be integer constants. Consider an algorithm that computes the convex hull of P using predicates which are polynomials of degree less than Δ with 2σ indeterminates, where each predicate is applied to σ input points. If the algorithm uses $O(NA_s(H))$ predicates, then it requires $\Omega\Big(\frac{N}{B}(\alpha_1(\min\Big\{H, \sqrt{\frac{N}{M}}\Big\}) - s)\Big)$ I/Os.*

Proof. The proof is very similar to the proof of Theorem 14. The observation is that in the proof of Theorem 14, only two operations, the default strategy and node terminations, give information to the algorithm about the positions of the points and the rest of the proof does not really use the fact that we are computing the maxima other than that the adversary maintains a binary tree and each comparison moves the points involved in the comparison at most 1 level down the tree. In our case, we are maintaining a tree T of fanout $f = \Delta^{2\sigma+2}$ but it can be simulated with a binary tree T' where each level of T corresponds to $\log(f) = (2\sigma + 2)\log(\Delta) = \Theta(\sigma \log(\Delta))$ levels of T'. Lemma 23 directly replaces the *the default* strategy and it can be used by the adversary to resolve the predicates. However, the resolution of the predicates moves σ points, $\sigma + 1$ levels down the tree which in the corresponding binary tree T' corresponds to moving σ points $O(\sigma \log(\Delta))$ levels down.

Next, consider node termination. Terminating a node u ensures that the points in $T(u)$ will contribute at most four vertices to the convex hull. Consequently, if four times the number of top nodes will be an upper bound on the size of the convex hull which again changes only the argument in Theorem 14 by a constant factor.

Algorithm 2 Randomized algorithm for computing the maxima of a planar point set P with an integer seed $h \geq 2$.

```
1:  procedure RANDOM-MAXIMA(P, h)
2:      if |P| ≤ 1 then                                          ▷ Base case
3:          Output P and return |P|
4:      L = (P_{2h}, ..., P_1) = DIST(P, 2h)        ▷ Distribute P into 2h buckets w.r.t. X-coor.
5:      PRUNE(L)                  ▷ For each P_i prune points dominated by points in ⋃_{j=1}^{i-1} P_j
6:      Reverse L with probability 1/2                          ▷ Backwards for-loop with prob. 1/2
7:      H' = 0
8:      for each P_j ∈ L do
9:          H_i = RANDOM-MAXIMA(P_j, h + H')
10:         H' = H' + H_i
11:     return H'
```

The rest of the proof is, therefore, identical to the proof of Theorem 14, except any one predicate translates to $O(\sigma^2 \log \Delta) = \tilde{O}(1)$ comparisons, which only changes the internal memory work of the algorithm by a constant factor, and the same lower bound argument applies. ◀

7 Randomized Algorithm

In this section we present a randomized algorithm (Algorithm 2) for computing the maxima. The algorithm is extremely similar to the deterministic algorithm of Algorithm 1 with only two differences. In the original algorithm, first $2h$ subproblems are created and then they are pruned. Then, the algorithm does a recursion on the subproblems, in the order $P_1, \ldots, P_{2h}$. The first difference is that in the modified algorithm, with 50% probability, instead of that we will recurse in the order $P_{2h}, \ldots, P_1$. The second difference is that the algorithm is called with the seed $s = 2$ at the top level. Below, we show that these simple changes lead to an algorithm that has optimal (worst-case) time complexity and optimal expected I/O complexity, i.e., an optimal randomized cache-oblivious algorithm.

First observe that the time analysis of Algorithm 1 is still valid for the randomized algorithm because the order of recursive calls on the subproblems is not relevant to the internal computation time. This implies that the randomized algorithm has an optimal worst-case running time.

As before, let H be the total number of maxima points in the entire input set P, $N = |P|$, s be the initial seed given to the algorithm (i.e., at the top level, the algorithm is called with $h = s = 2$), and H_i denote the number of maxima points of P_i (computed during the recursive call on P_i). Define $H^{\oplus} = H + s$, and $G_i = h + H_1 + \cdots + H_{i-1}$ and $G'_i = h + H_{i+1} + \cdots + H_{2h}$. Observe that a forward **for** loop calls RANDOM-MAXIMA(P_i, G_i), while the backward **for** loop calls RANDOM-MAXIMA(P_i, G'_i). Each case happens with 50% probability, therefore, the expected I/O complexity of RANDOM-MAXIMA(P, h) is defined by the following recurrence relation (with the base cases being the same as in Eq. (2)):

$$\bar{T}_H(N,h) = \begin{cases} O(\texttt{Distr}(N,h)) & \text{if } h \geq H, \\ O(\texttt{Distr}(N,H)) & \text{if } H > h \geq m, \\ \frac{1}{2} \cdot \sum_{i=1}^{2h} \bar{T}_{H_i}\left(\frac{N}{2h}, G_i\right) + \frac{1}{2} \cdot \sum_{i=1}^{2h} \bar{T}_{H_i}\left(\frac{N}{2h}, G'_i\right) + O(\texttt{Distr}(N,h)) & \\ & \text{if } h < \min(H,m). \end{cases} \tag{6}$$

Let $\hat{\imath} = \operatorname{argmax}\{H_1, \ldots, H_{2h}\}$, i.e., the index of the largest $H_1, \ldots, H_{2h}$. The main observation here is that for any index $j \neq \hat{\imath}$, either $G_j \geq H_j$, or $G'_j \geq H_j$ (both can be true, and this might also hold for $H_{\hat{\imath}}$), i.e., the I/O complexity of at least one of the recursive calls RANDOM-MAXIMA(P_j, G_j) or RANDOM-MAXIMA(P_j, G'_j) is covered by the base cases. For instance, if $G_j \geq H_j$, $\bar{T}_{H_j}(N/(2h), G_j) = O(\texttt{Distr}(N/(2h), G_j)) = O(\texttt{Distr}(N/(2h), H^{\oplus}))$ (the last equality follows from $G_j \leq H^{\oplus}$). Similarly, when $G'_j \geq H_j$, $\bar{T}_{H_j}(N/(2h), G'_j) = O(\texttt{Distr}(N/(2h), H^{\oplus}))$. Then we can rewrite $\bar{T}_H(N, h)$ as follows by combining the recursive calls defined by the base cases.

$$\begin{aligned}
\bar{T}_H(N,h) &= \frac{1}{2} \cdot \left(\sum_{i=1}^{\hat{\imath}-1} \bar{T}_{H_i}\left(\frac{N}{2h}, G_i\right) + \bar{T}_{H_{\hat{\imath}}}\left(\frac{N}{2h}, G_{\hat{\imath}}\right) + \sum_{i=\hat{\imath}+1}^{2h} \bar{T}_{H_i}\left(\frac{N}{2h}, G_i\right) \right) \\
&\quad + \frac{1}{2} \cdot \left(\sum_{i=1}^{\hat{\imath}-1} \bar{T}_{H_i}\left(\frac{N}{2h}, G'_i\right) + \bar{T}_{H_{\hat{\imath}}}\left(\frac{N}{2h}, G'_{\hat{\imath}}\right) + \sum_{i=\hat{\imath}+1}^{2h} \bar{T}_{H_i}(\frac{N}{2h}, G'_i) \right) + O(\texttt{Distr}(N,h)) \\
&\leq \frac{1}{2} \cdot \left(\bar{T}_{H_{\hat{\imath}}}\left(\frac{N}{2h}, G_{\hat{\imath}}\right) + \bar{T}_{H_{\hat{\imath}}}\left(\frac{N}{2h}, G'_{\hat{\imath}}\right) \right) + \frac{1}{2} \cdot \sum_{i=1}^{2h} \bar{T}_{H_i}\left(\frac{N}{2h}, G^{\star}_i\right) + c \cdot \texttt{Distr}(N, H^{\oplus}),
\end{aligned}$$

where $G^{\star}_i$ is either G_i or G'_i, $c > 0$ is some constant, and the last inequality follows from $h \leq H^{\oplus}$. The main insight here is that we get geometrically decreasing series. In particular, we claim that $\bar{T}_H(N, h) \leq 4c \cdot \texttt{Distr}(N, H^{\oplus})$, which can easily be proven by induction:

$$\begin{aligned}
\bar{T}_H(N,h) &\leq \frac{1}{2} \cdot \left(4c \cdot \texttt{Distr}(\frac{N}{2h}, H^{\oplus}) + 4c \cdot \texttt{Distr}(\frac{N}{2h}, H^{\oplus}) \right) \\
&\quad + \frac{1}{2} \cdot \left(\sum_{i=1}^{2h} 4c \cdot \texttt{Distr}(\frac{N}{2h}, H^{\oplus}) \right) + c \cdot \texttt{Distr}(N, H^{\oplus}) \\
&= 4c(h+1) \cdot \texttt{Distr}(\frac{N}{2h}, H^{\oplus}) + c \cdot \texttt{Distr}(N, H^{\oplus}) \\
&\leq \frac{4c(h+1)}{2h} \cdot \texttt{Distr}(N, H^{\oplus}) + c \cdot \texttt{Distr}(N, H^{\oplus}) \\
&= \left(\frac{4(h+1)}{2h} + 1 \right) \cdot c \cdot \texttt{Distr}(N, H^{\oplus}) \\
&\leq 4c \cdot \texttt{Distr}(N, H^{\oplus}) \qquad \text{for any } h \geq 2.
\end{aligned}$$

Observe that the same applies to the convex hull algorithm. Thus, we have the following theorem.

▶ **Theorem 25.** *For a set P of N points in the plane, there exists a randomized cache-oblivious algorithm that finds the maxima of P or the convex hull of P in $O(N \log H)$ worst-case time and $O_E(n \log_m H)$ expected I/Os, where H is the size of the output and $m = M/B$ and $n = N/B$.*

References

1 Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, February 1928. doi:10.1007/BF01459088.

2 Peyman Afshani, Gerth Stølting Brodal, and Nodari Sitchinava. The impossibility of simultaneous time and I/O optimality for the planar maxima and convex hull problems, 2026. arXiv:2605.09464.

3 Peyman Afshani and Pingan Cheng. Lower bounds for intersection reporting among flat objects. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium*

on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA, volume 258 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SOCG.2023.3.

4 Peyman Afshani and Pingan Cheng. On semialgebraic range reporting. *Discretete Computational Geometry*, 71(1):4–39, 2024. doi:10.1007/S00454-023-00574-1.

5 Peyman Afshani and Arash Farzan. Cache-oblivious output-sensitive two-dimensional convex hull. In Prosenjit Bose, editor, *Proceedings of the 19th Annual Canadian Conference on Computational Geometry, CCCG 2007, August 20-22, 2007, Carleton University, Ottawa, Canada*, pages 153–155. Carleton University, Ottawa, Canada, 2007. URL: http://cccg.ca/proceedings/2007/07a3.pdf.

6 Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. In John Hershberger and Efi Fogel, editors, *Proceedings of the 25th ACM Symposium on Computational Geometry, Aarhus, Denmark, June 8-10, 2009*, pages 277–286. ACM, 2009. doi:10.1145/1542362.1542412.

7 Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. *Discret. Comput. Geom.*, 45(4):824–850, 2011. doi:10.1007/S00454-011-9347-7.

8 Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, September 1988. doi:10.1145/48529.48535.

9 A.M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979. doi:10.1016/0020-0190(79)90072-3.

10 Lars Arge, Mikael B. Knudsen, and Kirsten Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures, Third Workshop, WADS '93, Montréal, Canada, August 11-13, 1993, Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 1993. doi:10.1007/3-540-57155-8_238.

11 Lars Arge and Peter Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, volume 50 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 139–159. DIMACS/AMS, 1999. doi:10.1090/DIMACS/050/08.

12 Lars Arge and Mikkel Thorup. RAM-efficient external memory sorting. *Algorithmica*, 73(4):623–636, 2015. doi:10.1007/S00453-015-0032-8.

13 Jon Louis Bentley and Michael Ian Shamos. Divide and conquer for linear expected time. *Information Processing Letters*, 7(2):87–91, 1978. doi:10.1016/0020-0190(78)90051-0.

14 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Linear time bounds for median computations. In Patrick C. Fischer, H. Paul Zeiger, Jeffrey D. Ullman, and Arnold L. Rosenberg, editors, *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*, pages 119–124. ACM, 1972. doi:10.1145/800152.804904.

15 Allan Borodin, Leonidas J. Guibas, Nancy A. Lynch, and Andrew Chi-Chih Yao. Efficient searching using partial ordering. *Information Processing Letters*, 12(2):71–75, 1981. doi:10.1016/0020-0190(81)90005-3.

16 Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 307–315. ACM, 2003. doi:10.1145/780542.780589.

17 R. C. Buck. Mathematical induction and recursive definitions. *The American Mathematical Monthly*, 70(2):128–135, 1963. doi:10.1080/00029890.1963.11990055.

18 Timothy M. Chan. Fixed-dimensional linear programming queries made easy. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, SCG '96, page 284–290, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237218.237397.

19 Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000. doi:10.1145/355541.355562.

20 Bernard Chazelle and Burton Rosenberg. The complexity of computing partial sums offline. *International Journal of Computational Geometry & Applications*, 01(01):33–45, 1991. doi:10.1142/S0218195991000049.

21 Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. The MIT Press, 4nd edition, 2022.

22 Rex A. Dwyer. On the convex hull of random points in a polytope. *Journal of Applied Probability*, 25(4):688–699, 1988. doi:10.2307/3214289.

23 Arash Farzan. Cache-oblivious searching and sorting in multisets. Master's thesis, University of Waterloo, 2004. URL: http://hdl.handle.net/10012/1019.

24 M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 345–354, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/73007.73040.

25 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 285–298. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814600.

26 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

27 Michael T. Goodrich, Jyh-Jong Tsay, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory computational geometry. In *34th Annual Symposium on Foundations of Computer Science*, pages 714–723. IEEE Computer Society, 1993. doi:10.1109/SFCS.1993.366816.

28 R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

29 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. doi:10.1137/0215021.

30 H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975. doi:10.1145/321906.321910.

31 Seth Pettie. An inverse-Ackermann type lower bound for online minimum spanning tree verification. *Comb.*, 26(2):207–230, 2006. doi:10.1007/S00493-006-0014-1.

32 Seth Pettie. Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. *J. Graph Algorithms Appl.*, 19(1):375–391, 2015. doi:10.7155/JGAA.00365.

33 Franco P. Preparata. An optimal real-time algorithm for planar convex hulls. *Commun. ACM*, 22(7):402–405, July 1979. doi:10.1145/359131.359132.

34 Franco P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, February 1977. doi:10.1145/359423.359430.

35 Raphael M. Robinson. Recursion and double recursion. *Bull. Amer. Math. Soc.*, 54:987–993, 1948. doi:10.1090/S0002-9904-1948-09121-2.

36 Péter Rózsa. Konstruktion nichtrekursiver Funktionen. *Mathematische Annalen*, 111:42–60, December 1935. doi:10.1007/BF01472200.

37 Yngve Sundblad. The Ackermann function: A theoretical, computational, and formula manipulative study. *BIT Numerical Mathematics*, 11:107–119, 1971. doi:10.1007/BF01935330.

38 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.

39 Peter van Emde Boas. On the $\Omega(n \log n)$ lower bound for convex hull and maximal vector determination. *Information Processing Letters*, 10(3):132–136, 1980. doi:10.1016/0020-0190(80)90064-2.

40 Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CSUR)*, 33(2):209–271, 2001. doi:10.1145/384192.384193.

41 Jeffrey Scott Vitter. Algorithms and data structures for external memory. *Found. Trends Theor. Comput. Sci.*, 2(4):305–474, January 2008. doi:10.1561/0400000014.

42 Andrew Chi-Chih Yao. A lower bound to finding convex hulls. *J. ACM*, 28(4):780–787, October 1981. doi:10.1145/322276.322289.